Stepwise refinement of sequence diagrams with soft real-time constraints

Atle Refsdal[a,*], Ragnhild Kobro Runde[b], Ketil Stølen[a,b]

[a] *SINTEF ICT, P.O.Box 124 Blindern, 0314 Oslo, Norway, Phone: +47 22067300, Telefax: +47 22067350*

[b] *Department of Informatics, University of Oslo, P.O.Box 1080 Blindern, 0316 Oslo, Norway, Phone: +47 22840144, Telefax: +47 22852401*

Abstract

UML sequence diagrams and similar notations are much used to specify and analyze computer systems and their requirements. Probabilities are often essential, in particular for capturing soft real-time constraints. It is also important to be able to specify systems at different levels of abstraction. Refinement is a means to relate abstract specifications to more concrete specifications in such a way that constraints and analysis results are preserved through the transition. In order to allow soft real-time constraints to be included as an integral part of sequence diagram specifications, this paper presents an approach to extend UML 2.x sequence diagrams to capture probabilistic choice in general and soft real-time constraints in particular. The approach is supported by formal semantics and pragmatic refinement relations with mathematical properties that allow stepwise and modular development of specifications. An example focusing on communication is provided to demonstrate the language and refinement relations.

Keywords: Soft real-time specification, sequence diagram, refinement, probabilistic choice

1. Introduction

UML 2.x sequence diagrams and similar notations are used to specify dynamic or behavioral aspects of computer systems. Sequence diagrams are particularly suited to model communication, which is an essential aspect of most computer systems today. According to [1] and [2], sequence diagrams (and use case diagrams) are the most popular UML languages for modeling the dynamic aspects of a system. Sequence diagrams are used, for example, as specifications for programmers or as maintenance documentation, to verify and validate

*Corresponding author

Email addresses: `Atle.Refsdal@sintef.no` (Atle Refsdal), `ragnhild.runde@ifi.uio.no` (Ragnhild Kobro Runde), `Ketil.Stolen@sintef.no` (Ketil Stølen)

constraints with client representatives, and to clarify understanding of the application among technical members of a project team [1]. [3] shows how MSC (which are very similar to UML sequence diagrams) can be used in a number of different parts that occur in most software development methods.

Probabilities sometimes play a major role when specifying and analyzing computer systems. In particular, soft real-time constraints are often important when specifying communication scenarios. By soft real-time constraints we mean constraints such as "when sending a request, the probability of receiving a reply within 5 seconds should be at least 0.95". In other words, soft real-time constraints are real-time constraints that need only be fulfilled with a certain (usually high) probability. Soft constraints are important because the corresponding hard constraints may be impossible or too costly to fulfill.

Refinement relations define what it means for one specification to be a more concrete or detailed representation of another specification. A suitable refinement relation should facilitate abstraction in an intuitive manner and ensure that constraints are preserved in the transition from the more abstract to the more concrete specification. Furthermore, an analysis of constraints performed at an abstract specification should remain valid for the more refined specifications. An important aspect of abstraction is the concept of underspecification. Underspecification means that some choices are left to those responsible for refining the specification or implementing the system, and is an essential feature of specification languages. Reducing the amount of underspecification brings a specification closer to an actual implementation, and is therefore an important form of refinement.

Motivated by the suitability and popularity of sequence diagrams for specifying behavior and communication, the importance of soft real-time constraints in communication, and the usefulness of being able to specify systems at different levels of abstraction, this paper presents an approach that allows soft real-time constraints to be expressed in sequence diagram specifications. More specifically, the approach extends UML 2.x sequence diagrams to capture soft real-time and to support underspecification with respect to probabilities as well as behaviors. We refer to this approach as probabilistic STAIRS, or pSTAIRS. The approach is based on STAIRS [4, 5, 6], and extends STAIRS with the expressive power to capture soft real-time constraints as well as other probabilistic constraints. This is achieved through extensions to ordinary sequence diagram notation that are, in our opinion, small and intuitive. The main contribution of this paper is the consolidated theory of modular refinement for soft real-time facilitating underspecification with respect to behavior as well as probability. Two different relations capturing refinement with respect to both kinds of underspecification are offered, each relation targeting a different part of a development process. Both relations have essential transitivity and monotonicity properties allowing the refinement to be conducted in a stepwise and modular fashion. Although proving these properties for all the composition operators and refinement relations has been a major effort in developing pSTAIRS, we believe that it is the results, rather than their proofs, that are of interest to most readers. Therefore we have confined ourselves to providing proofs of selected key results in an appendix. References to formal proofs in technical reports are provided for all results. For the main part of the paper, our emphasis is on providing a detailed, stepwise and example-driven explanation of the semantics of the operator for probabilistic choice, as well as refinement.

The rest of the paper is organized as follows: In Section 2 we introduce the pSTAIRS approach. We focus on demonstrating the suitability of probabilistic sequence diagrams to capture soft real-time constraints, as well as the stepwise application of the refinement relations. For this purpose we use an example addressing soft real-time constraints in a communication scenario. Section 3 presents the formal definition of probabilistic choice. In Section 4, we formally define the two refinement relations and present theoretical results of practical importance. In Section 5 we present related work before concluding in Section 6. There are also three appendices: Appendix A provides the formal semantics of the composition operators referred to (except for the operator for probabilistic choice, which is covered by Section 3). Appendix B presents a system model and defines what it means for a system to correctly implement a specification. Finally, Appendix C presents proofs for the main results.

2. Probabilistic sequence diagrams – an informal introduction

In this section we demonstrate the suitability of probabilistic sequence diagrams as defined in pSTAIRS to capture interaction scenarios with hard and soft real-time constraints, as well as the use and usefulness of the refinement relations. Only informal explanations are given at this point; formal definitions are provided in Section 3 and Section 4. The presentation is based on a scenario describing SMS-based interaction between a web portal and a mobile phone. We start with a simple diagram, which is further elaborated and refined in subsequent diagrams. This simple diagram is shown in Figure 1. It describes that a user sends an SMS message to a mobile phone from a web portal and is based on [7]. We expand on the original specification in order to illustrate more features of the specification language. The following examples are adopted from specifications developed in cooperation with a representative from a large Norwegian telecom operator and presented in [8].

There are four entities taking part in the interaction, each represented by a vertical dashed line called a lifeline. The lifelines :User, :WebPortal, :SendSMS-WS, and :MobilePhone represent the user, the web portal, the web service, and the mobile phone, respectively. Messages sent between the lifelines are represented by arrows. Each message gives rise to two events; the arrow tail represents the transmission of the message and the arrow head represents reception. All communication is assumed to be asynchronous. Events on each lifeline are ordered from top to bottom. In addition, for each message, the transmission event occurs before the reception event. There are no other ordering constraints, unless there are real-time constraints in the diagram.

The first message enterTextAdr represents the user entering the SMS message text and address (phone number) to the web portal. Next, the sendTextAdr message represents the call to the web service from the web portal. The web service

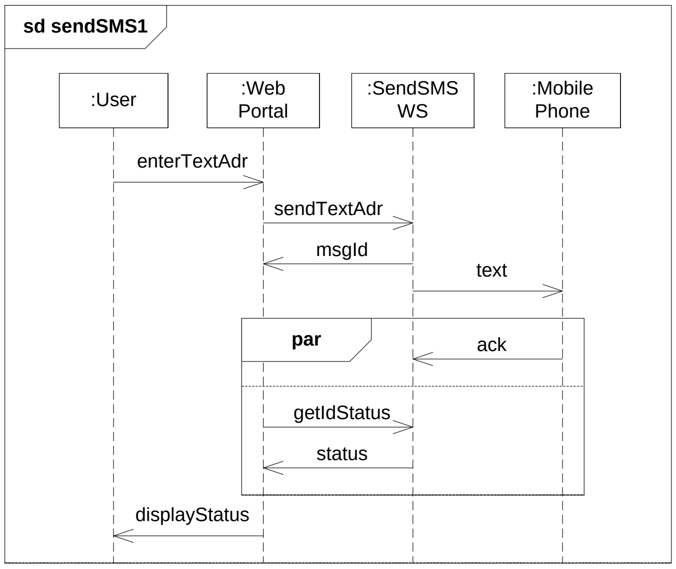

Figure 1: A scenario where an SMS message is sent from a web portal to a mobile phone.

responds by sending an SMS message identifier to the web portal, as shown by the msgId message, before sending the SMS text to the mobile phone, as shown by the text message. According to the above ordering rules, the reception of this message may occur before or after the reception of the msgId message on the web portal.

Next, the UML 2.x par operator for parallel composition is used to specify two sub-scenarios (interaction fragments) that may happen in parallel, i.e. with their events interleaved. The first sub-scenario states that after receiving the text, the mobile phone acknowledges the SMS by sending an ack message. In the second sub-scenario, after receiving the message id, the web portal asks the web service for the current status of the SMS message, as shown by the getIdStatus message. The web service then responds by sending a status message to the web portal. The par operator allows the events of its operators to be interleaved in any manner, as long as the ordering rules are followed for each operator. In our example, this means that the web service will still receive the getIdStatus message before sending the status message, but the ack message may be received at any time with respect to these two other events.

After receiving the message status, the web portal then displays it to the user, as shown by the displayStatus message. The status of an SMS message can be, for example "MessageWaiting" (the message is still queued for delivery), "DeliveredToNetwork" (the message has been successfully delivered to the network), or "DeliveredToTerminal" (the message has been successfully delivered to the mobile phone). However, we do not go further into this.

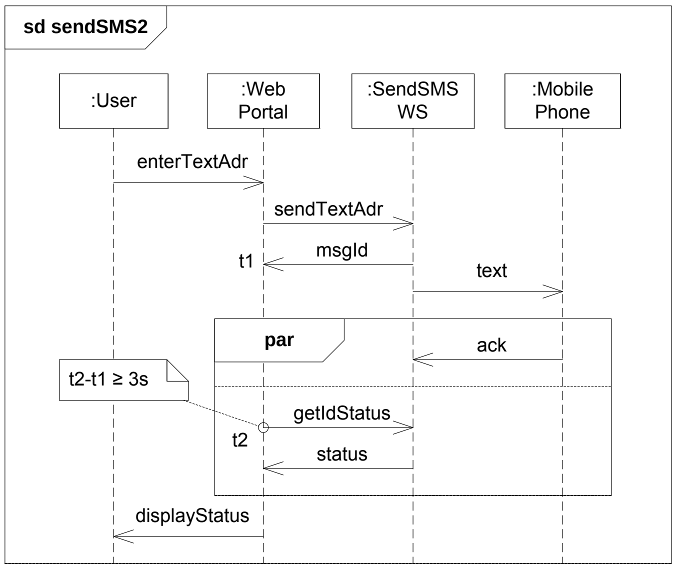

Figure 2: A hard real-time constraint has been introduced, stating that there should be a delay of at least 3 seconds from the msgId message is received by the web portal to the getIdStatus is sent.

The diagram sendSMS1 in Figure 1 captures the scenario in an intuitive and comprehensible manner. We now demonstrate how a refinement step may introduce a hard real-time constraint, i.e. a real-time constraint that should always be fulfilled. In order to allow the web service some time to send the SMS message and receive an acknowledgment from the mobile phone before it is asked for the status of the message, we introduce a constraint stating that there should be a delay of at least 3 seconds from the msgId message is received by the web portal to the getIdStatus message is sent. Figure 2 shows a specification where this constraint has been added. The real-time constraint is expressed in terms of so-called timestamp tags that are assigned to the events. In this case the timestamp tag t1 has been assigned to the reception of the msgId message and the timestamp tag t2 has been assigned to the transmission of the getIdStatus message. The real-time constraint is expressed by a note associated to the transmission of the getIdStatus message that contains a predicate over timestamp tags.[1] In this case the predicate is $\mathsf{t2\text{-}t1} \geq \mathsf{3s}$. All events have a timestamp tag, but only the timestamp tags referred to in a predicate are shown explicitly in the diagram.

There are no real-time constraints in Figure 1. Hence, all time delays are

[1]We prefer to use this notation rather than the standard UML notation for real-time constraints, as this makes it easier to specify real-time constraints in cases where the relevant events occur in operands of operators.

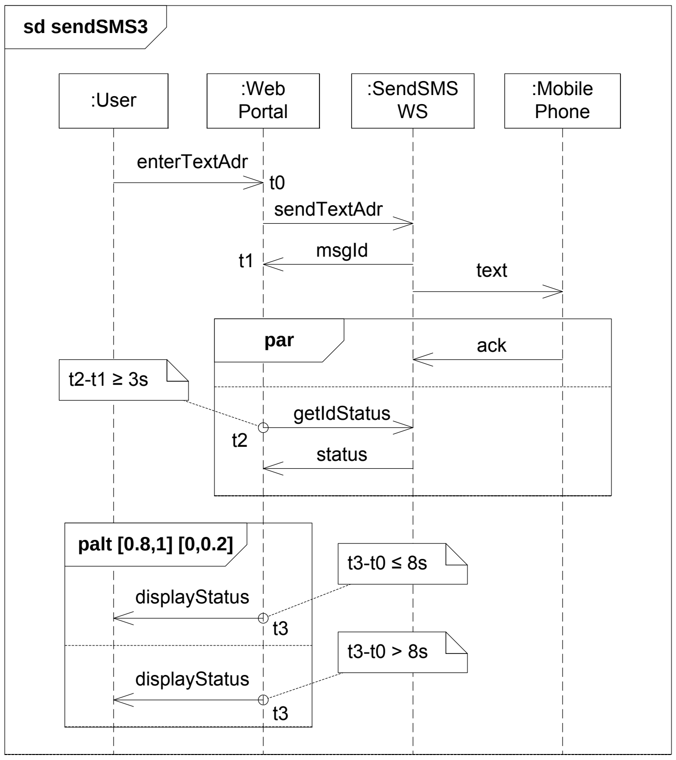

Figure 3: A soft real-time constraint has been added, stating that the probability of sending the displayStatus message on the web portal within 8 seconds after receiving the enterTextAdr message should be at least 0.8.

acceptable according to Figure 1, as long as the ordering of events is obeyed. By introducing the real-time constraint in Figure 2, we narrow the range of acceptable behavior by rejecting cases where it takes less than 3 seconds from the msgId message is received by the web portal to the getIdStatus message is sent. This kind of refinement, where previously positive behavior is redefined as negative, for example through introduction of new or stronger constraints, we call *narrowing*.

Next, we show how the specification can be further refined by the introduction of a soft real-time constraint. In pSTAIRS, soft real-time constraints are captured by the use of the operator palt for probabilistic choice. Figure 3 shows a diagram where we have imposed the soft real-time constraint that the probability of sending the displayStatus message from the web portal within 8 seconds after receiving the enterTextAdr message should be at least 0.8. We use the palt operator to achieve this. Both its operands contain the displayStatus message, but only the first operand fulfills the desired real-time constraint. Sets of acceptable probabilities are assigned to the operands after the operand name in the upper left-hand corner of the operator frame in the same order as the operands occur, starting from the top. Hence, the alternative where the predicate t3-t0$\leq$ 8s is fulfilled should occur with a probability in the interval $[0.8, 1]$, whereas the alternative with the complimentary predicate t3-t0$>$8s should occur with a probability in $[0, 0.2]$. The assignment of sets of acceptable probabilities to alternatives, rather than exact probabilities, represents underspecification with respect to probability. Without a means to represent underspecification with respect to probability, we would not be able to capture soft real-time constraints, as soft real-time constraints impose limits on acceptable probabilities, rather than exact probabilities.

Note that the introduction of the soft real-time constraint in the diagram sendSMS3 in Figure 3 is another example of narrowing refinement, as sendSMS3 does not allow behavior where the probability of a delay of more than 8 seconds from transmission of the enterTextAdr to transmission of the displayStatus message is higher than 0.2. This behavior was positive in sendSMS2 in Figure 2.

In the next refinement step, we add an optional report-message to be sent in the cases where the sending of displayStatus takes more than 8 seconds from the reception of enterTextAdr. This message may report the reason for the delay, e.g. server problems or network congestion. The resulting diagram is sendSMS4 in Figure 4. In the second palt-operand, the reportMsg is enclosed by and opt operator. This operator means that the content of its operand (in this case transmission and reception of the reportMsg message) is optional. Essentially, it describes two alternatives: one where the content of the operand is executed and one where it is not. These alternatives represent underspecification with respect to system behavior, as they constitute an implementation or design choice left to those responsible of implementing or further refining the specification. In the first palt-operand, the reportMsg is enclosed by a veto operator, used to express that the behavior of its operand is not acceptable. In this case, the veto operator means that a report-message should *not* be sent when the displayStatus is sent within 8 seconds of the reception of sendTextAdr.

Comparing sendSMS4 with the previous specification sendSMS3 in Figure 3, we see that all of the original behavior is still included in the diagram, but some more positive and negative behaviors are added. Adding new behavior in this way is a kind of refinement that we refer to as *supplementing*. Supplementing is primarily aimed at the early stages of development, where alternative ways of fulfilling a scenario are explored. To understand why we consider supplementing a form of refinement, it is important to remember that sequence diagrams, unlike most specification languages, give only partial descriptions of behavior. By this we mean that a sequence diagram does not categorize *all* behavior as either positive (acceptable) or negative (unacceptable). In most specification languages, such as state machines, the positive behavior is described explicitly, and all behavior that is not described explicitly is considered to be negative. Sequence diagrams, on the other hand, allow behavior to be described explicitly as positive or negative through dedicated operators. The remaining behavior, which is not described as either positive or negative, is considered to be inconclusive, in the sense that is has not (yet) been decided whether this behavior

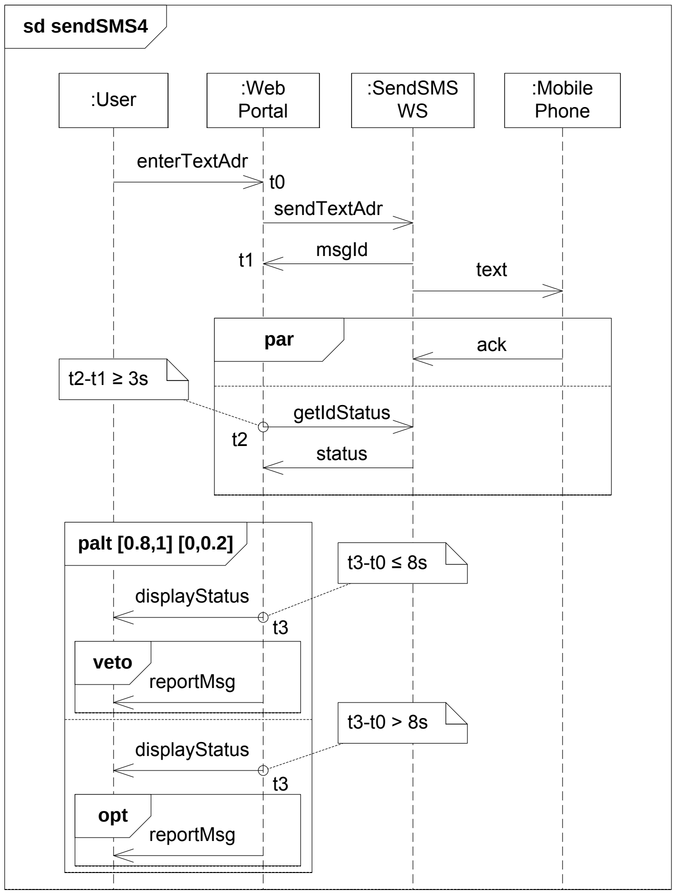

Figure 4: Adding the possibility of sending a report-message if (and only if) the sending of displayStatus takes more than 8 seconds from the reception of enterTextAdr.

is acceptable. In Figure 4 we have reduced the set of inconclusive behavior by specifying some of the behavior that was inconclusive according to Figure 3 as positive or negative. Hence, supplementing is a bit similar to broadening the scope of a specification by weakening the pre-condition in the classical pre-post specification paradigm [9], with the following correspondence: Positive behavior in pSTAIRS corresponds to fulfilling both the pre-condition and the post-condition. Negative behavior corresponds to fulfilling the pre-condition, but not the post-condition. Inconclusive behavior corresponds to not fulfilling the pre-condition. Weakening the pre-condition means that more cases fulfill the pre-condition, thus becoming positive or negative, depending on whether the post-condition is fulfilled or not.

At a certain point in the development process, we may decide that all relevant behavior has been identified and captured by the specification. From this point onwards, the task is to bring the specification closer to an implementation by reducing underspecification (i.e. making design/implementation choices) and possibly strengthening real-time constraints. Hence, supplementing with new behavior is no longer acceptable, and we therefore adopt a more strict definition of refinement hereafter.

We end the example by conducting a further narrowing of the diagram sendSMS4 in Figure 4. The resulting diagram is sendSMS5 in Figure 5. Two changes have been made, each of which constitutes a narrowing. In order to save space we have included them both in a single diagram, rather than presenting them in a stepwise fashion. First, the maximum acceptable probability for the last palt-operand (with t3-t0>8s) has been reduced from 0.2 to 0.1. Second, the first palt-operand has been split in two, requiring that the time delay between reception of the enterTextAdr and transmission of the displayStatus message should be at most 6 seconds with a probability of at least 0.8. Together, these two changes constitute a strengthening of the soft real-time constraint that was introduced in Figure 4. Clearly, any system that fulfills the constraints of sendSMS5 also fulfills the constraints of sendSMS4. Thus the constraints of the earlier specifications have also been preserved in this final refinement step.

We round off this informal presentation of pSTAIRS by a simple illustration of how the result of an analysis of constraints remains valid through refinement. Assume we are interested in how long the described scenario may take for the web portal. That is, we ask the question "how long is the delay x from the enterTextAdr message is received by the web portal to the displayStatus message is transmitted?". The specification sendSMS1 does not include any real-time constraints, so the only result we get from analyzing this specification is the trivial observation that "x is at least 0 seconds". However, in sendSMS2 we added the constraint that it should take at least 3 seconds from reception of msgId to transmission of getIdStatus. As events are ordered from top to bottom on the :WebPortal lifeline, this means that the delay is at least this long also between the first and final events on this lifeline, even if this is not stated explicitly as a real-time constraint. Hence, from analyzing sendSMS2 we get "x is at least 3 seconds", which is consistent with the previous result, but more specific. After introduction of the soft real-time constraint in sendSMS3,

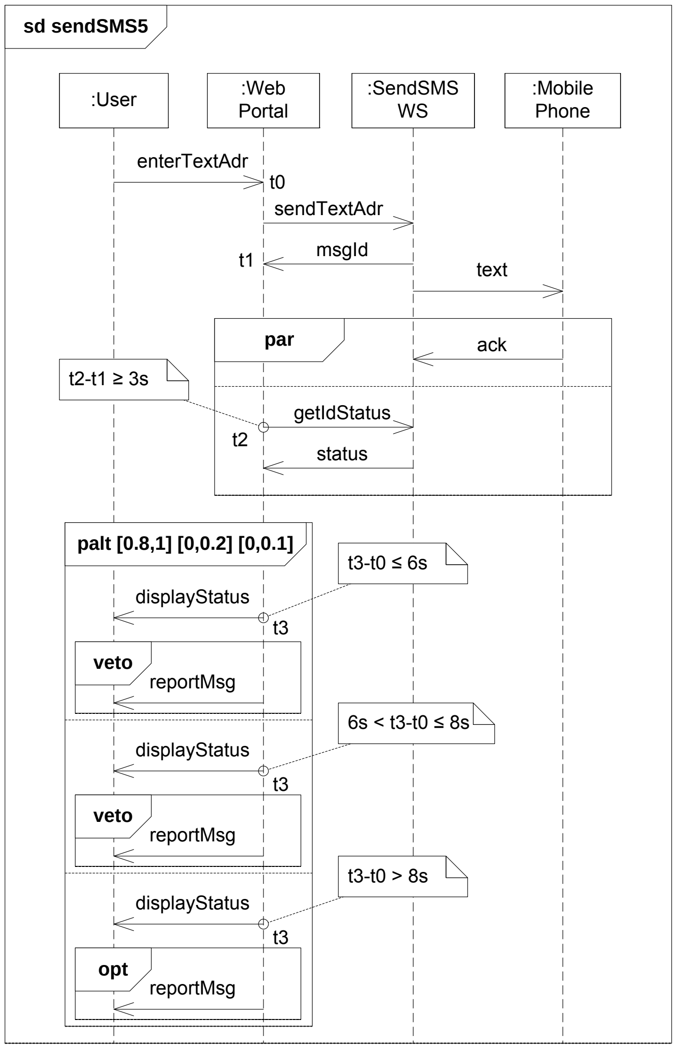

Figure 5: Strengthening of the soft real-time constraint. The probability of sending the displayStatus message on the web portal within 6 seconds after receiving the enterTextAdr message should be at least 0.8, and the probability of sending it within more than 8 seconds should be at most 0.1.

we are able to obtain an even more specific result: "x is at least 3 seconds, and the probability that x is less than or equal to 8 seconds is at least 0.8". The specification sendSMS4 does not add any new information with respect to real-time constraints, and the result of analyzing sendSMS4 with respect to our question remains the same as for sendSMS3. However, in sendSMS5 the soft real-time constraint has been strengthened, allowing us to conclude that "x is at least 3 seconds, and the probability that x is less than or equal to 6 seconds is at least 0.8". Again, this is consistent with the previous analysis, but more specific. Hence, for each refinement of the original specification we have obtained an analysis result that is consistent with all earlier results, but possibly more specific.

3. Probabilistic choice – its formal definition

In this section we first introduce the semantic domain of pSTAIRS, before providing and explaining the formal definition of the operator for probabilistic choice.

3.1. Semantic domain: probability obligations

As is done in STAIRS, we base our approach on the semantic model for sequence diagrams taken from the UML 2.4 standard [10]. Here, behavior is represented by traces, which are sequences of event occurrences. Sequence diagrams focus on communication, therefore we are primarily interested in events representing transmission or reception of messages. Since sequence diagrams give only a partial description of behavior, as discussed in the previous section, UML categorizes the traces described by a sequence diagram as either valid (positive) or invalid (negative); the traces not described are the inconclusive ones as explained above. Hence, the semantics of a UML sequence diagram is represented by a pair of trace sets (p, n) where p denotes the set of positive traces and n denotes the set of negative traces. The inconclusive ones are implicitly represented as the rest of the trace universe. In a sequence diagram, the same trace may be categorized as both positive and negative, in which case we will have $p \cap n \neq \emptyset$. For practical purposes (e.g. in relation to refinement), such traces will be treated in the same manner as the other negative traces.

In pSTAIRS we extend the semantic model above in order to capture probability. The semantics of a pSTAIRS specification is given as a set of *probability obligations*, or p-obligations for short. A p-obligation is a pair (o, Q) where $o = (p, n)$ is a pair of a set of positive traces p and a set of negative traces n, and Q is a set of probabilities, i.e. $Q \subseteq [0, 1]$. We use the name *interaction obligation* for pairs (p, n) of sets of positive and negative traces. Every interaction obligation represents an obligation for the specified system; the system is obliged to produce behavior that represents the interaction obligation. An interaction obligation still leaves freedom, as the interaction obligation does not in general require any particular trace to be produced. Instead it allows a number of traces from which the implementer may choose. In this way, an interaction obligation represents implementation freedom or underspecification with respect to system behavior.

Thus, an interaction obligation represents an "abstract piece of behavior" that may be implemented by a number of different traces. From the specifier's point of view, all these traces are considered to be equivalent. Conceptually, it is therefore natural to assign probabilities to interaction obligations, rather than to traces. This also gives a clean separation between underspecification with respect to traces (system behavior) and probabilities; underspecification with respect to traces is captured by the interaction obligation, while the assignment of a *set* of probabilities Q rather than a single probability to each interaction obligation captures underspecification with respect to probability, as the implementer is free to implement the p-obligation with any probability in Q.

3.2. The palt *operator for capturing probabilistic choice*

The palt operator describes the probabilistic choice between two or more alternative operands. Each operand is assigned a set of probabilities, and each operand should be chosen with a probability in its probability set. The only constraint is that the probabilities selected from each operand add up to 1. Using sets of probabilities rather than a single probability for each operand allows us to capture underspecification with respect to probabilities. In particular, it allows us to specify a minimum probability for an operand, which is essential to capture soft real-time constraints.

Note that an operand may have $[0, 1]$ as its probability set. A palt where operands all have $[0, 1]$ as probability set leaves the selection of probabilities completely open.

Any specification without a palt operator contains exactly one p-obligation, and the probability set of this p-obligation is $\{1\}$. Thus, for specifications without palt operators, the semantic representation according to pSTAIRS corresponds to the semantic representation according to UML, except that the former assigns probability 1 to the pair of positive and negative trace sets. The definition of the palt semantics involves some new operators on p-obligations and probability sets. We therefore develop this definition in a stepwise manner. First we give two preliminary definitions and explain why these do not work as desired. The preliminary definitions are (3) and (5). Then we present Definition (10), which is the final one. Definition (5) is a strengthening of (3), and (10) is a strengthening of (5).

For operators other than palt, we explain their semantics only to the extent necessary for the examples to be understandable, in order to avoid cluttering the presentation with details that are not important for the main issue of this paper. For the full definitions of these operators, we refer to Appendix A.

We start by introducing the notion of probability decoration, which is used to assign probabilities to each operand of the palt operator. A probability decoration must occur in every operand of a palt, and cannot occur anywhere else. It is denoted by d;Q, where d is a closed sequence diagram in the sense that both the transmitter and receiver lifelines are included for all messages, and Q is a set of probabilities. Intuitively, $d;Q$ states that the sequence diagram operand d should be selected with a probability in Q. Semantically, probability decoration is defined by:

$$[\![\ d;Q'\ ]\!] \stackrel{\mathsf{def}}{=} \{(o, Q * Q') \mid (o, Q) \in [\![\ d\ ]\!]\} \tag{1}$$

where multiplication of probability sets is pointwise:

$$Q_1 * Q_2 \stackrel{\mathsf{def}}{=} \{q_1 * q_2 \mid q_1 \in Q_1 \wedge q_2 \in Q_2\} \tag{2}$$

In the textual syntax, a palt operator and its operands is represented by palt$(d_1;Q_1, \ldots, d_n;Q_n)$. This can be read as "at least one of the operands $d_1, \ldots, d_n$ should be selected; operand d_1 should be selected with a probability in Q_1 and ... and the operand d_n should be selected with a probability in Q_n". It would be intuitively tempting to define the palt semantics as a simple union of its operands. This would give the following definition:

$$[\![\ \mathsf{palt}(d_1;Q_1, \ldots, d_n;Q_n)\ ]\!] \stackrel{\mathsf{pre}}{=} \bigcup_{j=1}^{n} [\![\ d_j;Q_j\ ]\!] \tag{3}$$

where we use $\stackrel{\mathsf{pre}}{=}$ to highlight that this is a preliminary definition. However, Definition (3) is not satisfactory. The reason is that the definition does not ensure that the probabilities of the operands are chosen so that they add up to one.

To see this, assume we want to specify a soft real-time constraint where there is an upper limit to the time delay that should not be exceeded in any run. Diagram paltEx in Figure 6 shows such a specification. Intuitively, the paltEx specification requires that the probability is at least 0.8 that the displayStatus message is transmitted from the web portal at most 8 seconds after reception of enterTextAdr, and that the delay is between 8 and 12 seconds in the remaining cases. Hence, the traces described by paltEx can be divided into three trace sets s_1, s_2, s_3, where s_1 denotes the set of traces where the delay between reception of enterTextAdr and transmission of displayStatus is at most 8 seconds, s_2 denotes the set of traces where the delay is more than 8 seconds but no more than 12 seconds, and s_3 denotes the set where the delay is more than 12 seconds. With Definition (3), we get

$$[\![\ \mathsf{paltEx}\ ]\!] = \{((s_1, s_2 \cup s_3), [0.8, 1]), ((s_2, s_1 \cup s_3), [0, 0.2])\}$$

where the first p-obligation represents the first palt operand, and the second p-obligation represents the second palt operand.

But this semantics does not ensure that probabilities are chosen from each p-obligation in such a way that they add up to 1. For example we may select 0.8 as probability for the first operand, 0.1 for the second, and still have room for a refinement step adding a third operand with probability 0.1 allowing delays of more than 12 seconds.

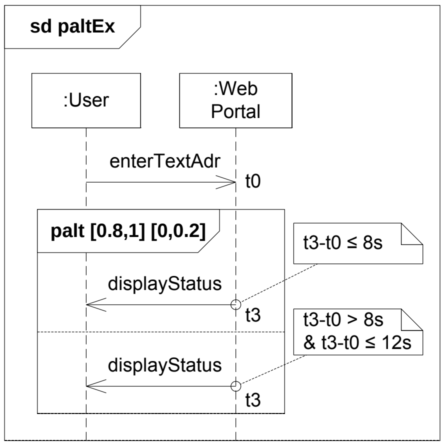

Figure 6: A specification of a soft real-time constraint with an absolute upper limit for the time delay.

To ensure that the chosen probabilities of the operands add up to 1, we strengthen the palt semantics with an additional p-obligation representing the combination of all the p-obligations we obtain from the operands. The only acceptable probability for this combined p-obligation is 1. This formalizes that one of the operands must be chosen; i.e. the probabilistic choice will be made among the specified operands. For the paltEx specification this means that we add a p-obligation $((p, n), \{1\})$ representing the combination of the two alternatives. The positive and negative traces of this combined p-obligation are determined by the interaction obligations of the original p-obligations $((s_1, s_2 \cup s_3), [0.8, 1])$ and $((s_2, s_1 \cup s_3), [0, 0.2])$. If a trace is positive in one of these then it is acceptable for the system to produce this trace. Therefore, if a trace is positive in at least one p-obligation (and not inconclusive in any p-obligation) then it is positive in the combined p-obligation. For the paltEx specification this means that traces in $s_1 \cup s_2$ are positive. If a trace is negative in all the original p-obligations then this means that it should not be produced at all. Hence it is also negative in the combined p-obligation. For the paltEx specification this means that traces in s_3 are negative. If a trace is inconclusive in at least one of the original p-obligations then it has not been considered for all alternatives. It is therefore considered to be inconclusive also in the combined p-obligation.

The interaction obligation of the combined p-obligation is formalized by the $\oplus$ operator, whose operand is a set of p-obligations:

$$\oplus S \overset{\text{def}}{=} \Big(\big(\bigcup_{((p,n),Q) \in S} p \big) \cap \big(\bigcap_{((p,n),Q) \in S} p \cup n \big), \bigcap_{((p,n),Q) \in S} n \Big) \tag{4}$$

As explained, a trace is negative only if it is negative in all p-obligations; a trace is inconclusive if it is inconclusive in at least one p-obligation; otherwise it is

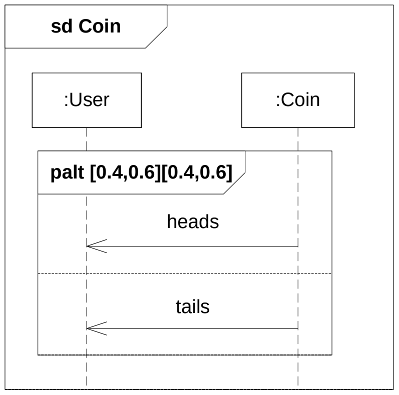

Figure 7: A specification of a coin with different sets of inconclusive traces for the two palt-operands.

positive. In the paltEx specification the interaction obligation of the combined p-obligation is

$$\oplus\{((s_1, s_2 \cup s_3), [0.8, 1]), ((s_2, s_1 \cup s_3), [0, 0.2])\} = (s_1 \cup s_2, s_3)$$

To include the combined p-obligation in the palt semantics we add another line to the palt definition:

$$[\![\ \mathsf{palt}(d_1;Q_1, \ldots, d_n;Q_n)\ ]\!] \overset{\mathsf{pre}}{=} \tag{5}$$

$$\bigcup_{j=1}^{n} [\![\ d_j;Q_j\ ]\!] \cup \tag{a}$$

$$\{(\oplus \bigcup_{j=1}^{n} [\![\ d_j;Q_j\ ]\!], \{1\})\} \tag{b}$$

In the paltEx specification the set of inconclusive traces is the same for each p-obligation — from a practical point of view this is normally advisable. Specifying probabilistic alternatives does not make much sense unless they are mutually exclusive in the sense that the positive traces in one operand are negative in the other. For an example where this is not the case, consider the specification of a coin toss in Figure 7, where heads and tails each should occur with a probability between 0.4 and 0.6. However, as tails is inconclusive in the heads alternative, a refinement step may result in tails being the only positive trace in both palt-operands, with heads (and every other trace) negative in both operands. This is obviously not the intention behind the specification, and can be prevented by ensuring that the set of inconclusive traces for each palt-operand is the same. When this is the case, we say that the palt is well-balanced.

Formally, $\mathsf{palt}(d_1;Q_1,\ldots,d_k;Q_k)$ is well-balanced if

$$\forall((p_i,n_i),Q_i),((p_j,n_j),Q_j) \in \bigcup_{l=1}^{k} [\![\ d_l\ ]\!] : p_i \cup n_i = p_j \cup n_j \tag{6}$$

Every palt occurring in this paper except from Figure 7 is well-balanced, and well-balancedness may easily be imposed syntactically by using the following macro operator:

$$\begin{aligned}&[\![\ \mathsf{expalt}(d_1;Q_1,\ldots,d_n;Q_n)\ ]\!] \overset{\mathsf{def}}{=} \\ &\quad [\![\ \mathsf{palt}((d_1 \ \mathsf{alt}\ \mathsf{refuse}(d_2\ \mathsf{alt}\ \ldots\ \mathsf{alt}\ d_n));Q_1, \\ &\qquad \ldots \\ &\qquad (d_n\ \mathsf{alt}\ \mathsf{refuse}(d_1\ \mathsf{alt}\ \ldots\ \mathsf{alt}\ d_{n-1}));Q_n)\ ]\!]\end{aligned} \tag{7}$$

where the alt operator specifies alternative traces representing underspecification, and the refuse operator specifies negative traces. For the fragment d_1 alt refuse(d_2 alt ... alt d_n), the combined use of alt and refuse specifies that the only positive traces are those that are positive in d_1, while the negative traces include both those that are negative in d_1 and also all the positive and negative traces in the other operands $(d_2,\ldots,d_n)$. In this paper, we use palt in the examples in order to illustrate the basics of probabilistic choice and soft real-time constraints in pSTAIRS. For all practical purposes, we recommend using expalt and not palt.

Note that line (b) in Definition (5) implies that nesting of palt operators is significant, as the sum of probabilities for the operands of each particular palt operator should add up to 1. This means that a specification with nested palt operators cannot in general be rewritten into an equivalent specification with only a single palt operator. As an example, consider the specifications nested and flat in Figure 8. The ref constructs are references to other diagrams whose specifications are of no significance here and therefore left out. The specification nested is stricter than flat, because nested requires that the probability of selecting one of d3 and d4 is a value in Q. The reason is that the probability set Q is assigned to the third operand of the outermost palt, which contains the choice between d3 and d4 represented by the innermost palt. This constraint is not present in flat. For example, let $Q = [\frac{1}{4},\frac{1}{2}]$, which gives $Q * Q = [\frac{1}{16},\frac{1}{4}]$. According to Flat, it would be acceptable to select d1 with probability $\frac{1}{2}$, d2 with probability $\frac{3}{8}$, d3 with probability $\frac{1}{16}$ and d4 with probability $\frac{1}{16}$. According to nested, this is not acceptable, since the probability of selecting one of d3 and d4 is then $\frac{1}{8}$, which is not a value in $[\frac{1}{4},\frac{1}{2}]$. This illustrates the extra expressiveness obtained by including line (b) in Definition (5).

But Definition (5) is also not fully satisfactory. The reason is that when the palt has three (or more) operands, two of the operands may unintentionally be interpreted to correspond to the same probabilistic choice in the implementation, thereby giving the implementation room to include additional behavior not intended by the specification. As an example, consider the specification of

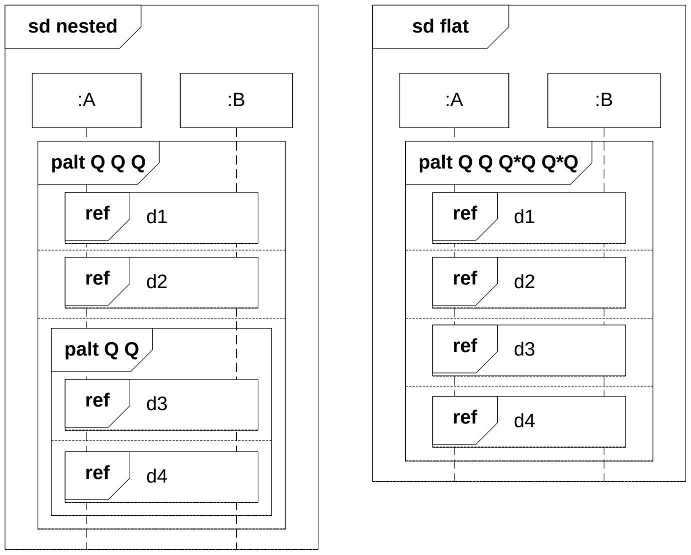

Figure 8: Specification nested, which contains a palt operator within a palt operand, is stricter than specification flat. Q represents probability sets.

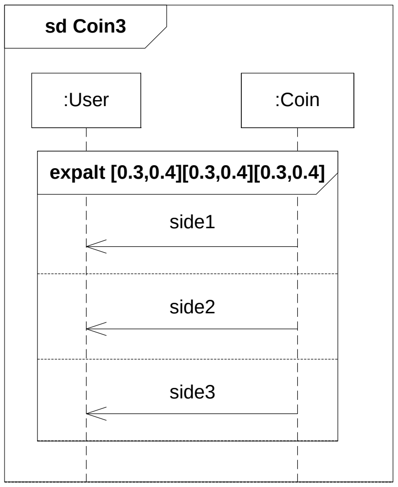

Figure 9: A 3-sided coin, using expalt to ensure well-balancedness.

a three-sided coin in Figure 9. Here, expalt is used to ensure that the positive traces of one operand are negative in the others. According to Coin3, each of the three sides should occur with a probability between 0.3 and 0.4. Intuitively, each side should *not* be produced with a probability higher than 0.4, since the side is negative in the two other operands which together has a minimum probability of $0.3 + 0.3 = 0.6$. However, this is not ensured by Definition (5), which does not consider constraints imposed by combining a subset of the palt-operands.

To avoid this problem we strengthen the semantics of palt with p-obligations representing the combined sum of *any subset* of p-obligations from the original specification. Two p-obligations can no longer be implemented (only) by the same probabilistic choice since the implementation must also offer a choice corresponding to their combination. As before, the interaction obligation of a combined p-obligation is produced by the $\oplus$ operator. But since each new combination represents only a subset of the original p-obligations, we cannot use 1 as the only acceptable probability. Instead we use the sum of the probability sets of each p-obligation of the subset. The combined sum operator $\bar{\oplus}$ combines an indexed set $\{(o_i, Q_i)\}_{i \in N}$ of p-obligations into a single p-obligation as follows:

$$\bar{\oplus}(\{(o_i, Q_i)\}_{i \in N}) \stackrel{\mathsf{def}}{=} (\oplus\{(o_i, Q_i) \mid i \in N\}, \sum_{i \in N} Q_i) \tag{8}$$

Summation of probability sets is pointwise by choosing one value from each set and then adding those combinations that do not exceed 1. Formally, summation of n probability sets is defined by:

$$\sum_{i=1}^{n} Q_i \stackrel{\mathsf{def}}{=} \{\mathsf{min}(\sum_{j=1}^{n} q_j, 1) \mid \forall j : q_j \in Q_j\} \tag{9}$$

Note that $\bar{\oplus}\{(o, Q)\} = (o, Q)$ for any $Q \subseteq [0, 1]$.

The following definition of palt, in which line (a) in Definition (5) has been replaced, ensures that all possible combinations of p-obligations coming from the operands of the palt are included:

$$\begin{aligned}
&[\![\ \mathsf{palt}(d_1;Q_1, \ldots, d_n;Q_n) \ ]\!] \stackrel{\mathsf{def}}{=} && (10)\\
&\quad \{\bar{\oplus}(\{po_i\}_{i \in N}) \mid N \subseteq \{1, \ldots, n\} \land N \neq \emptyset \land \forall i \in N : po_i \in [\![\ d_i;Q_i \ ]\!]\} \cup && \text{(a)}\\
&\quad \{(\oplus \bigcup_{j=1}^{n} [\![\ d_j;Q_j \ ]\!], \{1\} \cap \sum_{j=1}^{n} Q_j)\} && \text{(b)}
\end{aligned}$$

Note that the set of p-obligations we get from (10a) is a superset of the set we get from (5a), since $po = \bar{\oplus}\{po\}$ for any p-obligation po. The palt operator represents a complete probabilistic choice, in the sense that the sum of the probabilities chosen for each operand can not be less than 1. If this cannot be achieved, then no system should comply with the specification. We ensure this by substituting $\{1\}$ in (5b) with $\{1\} \cap \sum_{j=1}^{n} Q_j$. An implemented computer system is assumed to be represented by a set of p-obligations whose probability sets contain exactly one probability, as there is no underspecification

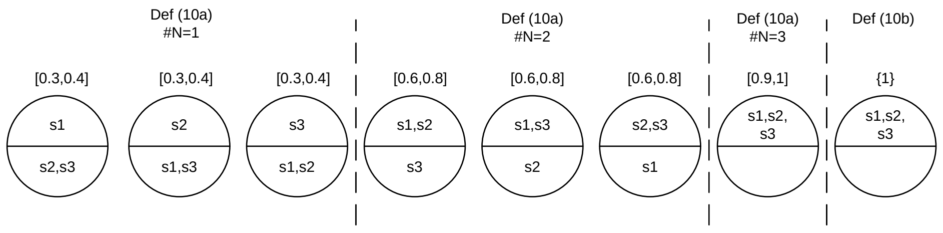

Figure 10: The semantics of Figure 9.

with respect to probability in an implementation. Therefore, no system can comply with a specification whose semantics contains a p-obligation with an empty probability set.

Using Definition (10), the semantics of the three-sided coin in Figure 9 may be illustrated as in Figure 10. Here, each p-obligation is illustrated by a circle representing the interaction obligation and a probability set. The upper part of the circle contains the positive traces and the lower part contains the negative traces. For simplicity, we have used s1, s2 and s3 as abbreviations for the traces representing the three sides of the coin. The three left-most obligations are those we get from considering each palt-operand in isolation, the four next p-obligations are the results of combining two or more p-obligations, while the final p-obligation is the result of line (b) in Definition (10).

4. Refinement of probabilistic sequence diagrams

Refinement means to add more information or stronger constraints to the specification in order to bring it closer to a real system. In this section we define the notion of refinement of probabilistic sequence diagrams formally. Two different refinement relations, aimed at different stages of the development process, are provided.

Our strategy for defining refinement is the following: In Section 4.1 we first define what it means for one interaction obligation to refine another interaction obligation, and then we lift this definition from interaction obligations to p-obligations. In Section 4.2 we use these definitions to define refinement of pSTAIRS specifications, which are represented semantically by sets of p-obligations. Note that this means that we use the term refinement for the binary relations that are defined in Section 4.1 between single elements of the semantic representations of specifications, as well as for the relations that are defined in Section 4.2 between full specifications.

After giving the definitions, we present important theoretical results in Section 4.3. Throughout the section, we also explain how the definitions and theoretical results may be applied to verify that each of the example diagrams in Section 2 is a refinement of the previous one.

4.1. Refinement relations for single interaction obligations and p-obligations

There are two ways in which an interaction obligation may be refined. Firstly, the incompleteness of an interaction obligation may be reduced by supplementing it with more positive and/or negative traces. This reduces the set of inconclusive traces, thereby giving a more complete description of the behavior. Secondly, underspecification with respect to behavior may be reduced by redefining positive traces as negative. This corresponds to making a design decision by rejecting some of the alternative ways of fulfilling a task that are described in the more abstract specification. These two possibilities are combined in the refinement relation $\rightsquigarrow_r$ (where r stands for "refinement"), which is defined as follows:

$$(p, n) \rightsquigarrow_r (p', n') \quad \overset{\mathsf{def}}{=} \quad n \subseteq n' \land p \subseteq p' \cup n' \tag{11}$$

A refinement relation that allows a specification to be supplemented with new positive or negative behavior, as well as reducing underspecification, is suitable in the early stage of the development process. At this stage, alternative ways of fulfilling the scenario in question (or failing to do so) are explored. Hence, adding new positive or negative behavior to the specification should be allowed.

Later, we may reach a point where all behavior we consider to be relevant and interesting has been described. This includes normal behavior, exceptional behavior and erroneous behavior. At this point, we may decide that supplementing (introducing new traces) is no longer allowed. However, the specification may still include underspecification in the form of several positive traces in the interaction obligation, as some design decisions may still be open. Hence, narrowing the specification by redefining some of the positive traces as negative should still be allowed. This is captured by the refinement relation $\rightsquigarrow_{nr}$ (where nr stands for "narrowing refinement"), defined by:

$$(p, n) \rightsquigarrow_{nr} (p', n') \quad \overset{\mathsf{def}}{=} \quad (p, n) \rightsquigarrow_r (p', n') \land p \cup n = p' \cup n' \tag{12}$$

We now lift the two definitions of refinement for interaction obligations to p-obligations. A p-obligation is refined by either refining its interaction obligation, or by reducing its set of acceptable probabilities, thus reducing underspecification with respect to probability. This gives the following refinement relations for p-obligations (where p stands for "probabilistic"):

$$((p, n), Q) \rightsquigarrow_{pr} ((p', n'), Q') \quad \overset{\mathsf{def}}{=} \quad (p, n) \rightsquigarrow_r (p', n') \land Q' \subseteq Q \tag{13}$$

$$((p, n), Q) \rightsquigarrow_{pnr} ((p', n'), Q') \quad \overset{\mathsf{def}}{=} \quad (p, n) \rightsquigarrow_{nr} (p', n') \land Q' \subseteq Q \tag{14}$$

As an example of refinement of a single p-obligation, consider the two sequence diagrams in Figure 11, where sendSMS1* is a subdiagram of sendSMS1 in Figure 1 and sendSMS2* is a subdiagram of sendSMS2 in Figure 2. As neither diagram uses the palt operator, the semantics of each diagram contains a single p-obligation with probability set $\{1\}$. sendSMS1* has an empty set of negative

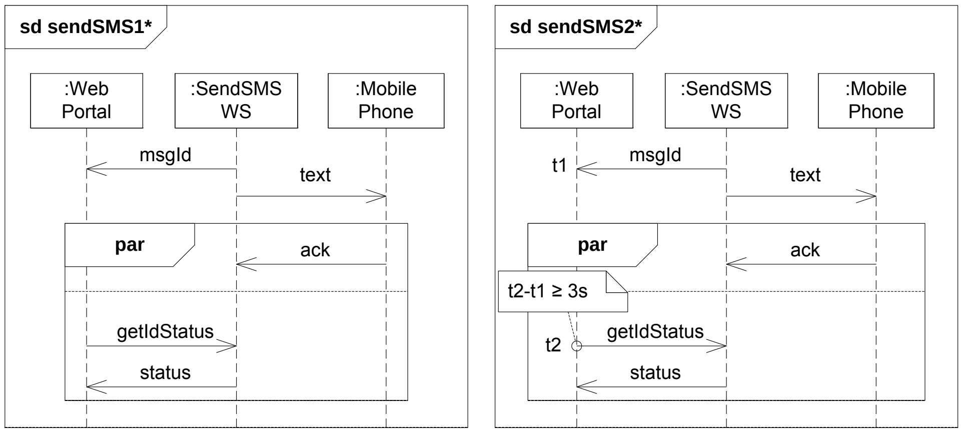

Figure 11: Subdiagrams of sendSMS1 and sendSMS2.

traces, while sendSMS2* has both positive and negative traces due to the use of a time constraint.

The set p of positive traces for sendSMS1* consists of all traces where weak sequencing is used to combine the messages msgId and text with the traces of the par-fragment. Listing all these traces here would be tedious. For the purposes of this section, it is sufficient to note that since sendSMS1* does not contain time constraints, it allows any possible assignment of timestamps to the individual events in its set of positive traces.[2]

The difference between sendSMS1* and sendSMS2*, is the addition of the time constraint t2-t1 $\geq$ 3s, requiring that the message getIdStatus is sent at least 3 seconds after the msgId message has been received. Semantically, this has the consequence of splitting the positive trace set p for sendSMS1* into two sets p' and n', where traces that are in accordance with this time constraint belong to the positive trace set p', and traces where the time between the two events is less than 3 seconds belong to the set n' of negative traces.

As $p = p' \cup n'$, it is easy to see that the p-obligation $((p', n'), \{1\})$ (for sendSMS2*) is a narrowing refinement of the p-obligation $((p, \emptyset), \{1\})$ (for sendSMS1*) according to definition (14).

More examples of supplementing and narrowing of single obligations may be found in previous work, e.g. [11] and [12].

[2]As long as there are no time constraints, the only constraint is that the events in a trace are be ordered by time (but two events may happen at the same time). Traces where the timestamps are assigned such that the events are not ordered by time are ill-formed, and not included in the semantic domain (i.e. an ill-formed trace is neither positive, negative, nor inconclusive).

4.2. Refinement relations for pSTAIRS specifications

For general refinement of specifications, we define two relations $\rightsquigarrow_{pg}$ and $\rightsquigarrow_{png}$ (where g stands for "general"). The refinement relation $\rightsquigarrow_{pg}$ is based on the relation $\rightsquigarrow_{pr}$ for p-obligations and intended for the early stage of the development process, while $\rightsquigarrow_{png}$ is based on the relation $\rightsquigarrow_{pnr}$ for p-obligations and intended for the late stage of the development process.

As for the definition of palt, we develop the definitions of refinement for specifications in a stepwise manner. We first give a preliminary definition (15) and explain why this is insufficient. Definition (16) is the final definition.

As explained in Section 3.1, the semantics of a pSTAIRS specification is given as a set of p-obligations. Intuitively, each p-obligation represents an abstract class of similar behaviors of which at least one representative is required of the system. Consequently, it is tempting to define refinement of a specification by requiring that every p-obligation at the abstract level should be refined by a p-obligation at the concrete level. This would give the following definition:

$$[\![\ d\ ]\!] \rightsquigarrow_x [\![\ d'\ ]\!] \quad \overset{\mathsf{pre}}{=} \quad \forall po \in [\![\ d\ ]\!] : \exists po' \in [\![\ d'\ ]\!] : po \rightsquigarrow_y po' \tag{15}$$

where $(x, y) \in \{(pg, pr), (png, pnr)\}$.

However, Definition (15) is not satisfactory for soft real-time constraints. Consider the constraint "a request should be followed by a response within 5 seconds with a probability of at least 0.9". A specification replacing this constraint with the corresponding hard real-time constraint (requiring the system to *always* produce a response within 5 seconds) would certainly preserve the original soft real-time constraint, and should therefore be considered a valid refinement, even if the alternative where it takes more than 5 seconds is not represented.

In the refinement definition, this is captured by adding an exception stating that only p-obligations not having 0 as an acceptable probability need to be represented at the concrete level:

$$\begin{aligned} &[\![\ d\ ]\!] \rightsquigarrow_x [\![\ d'\ ]\!] \ \overset{\mathsf{def}}{=} \\ &\quad \forall po \in [\![\ d\ ]\!] : 0 \notin \pi_2.po \Rightarrow \exists po' \in [\![\ d'\ ]\!] : po \rightsquigarrow_y po' \end{aligned} \tag{16}$$

where $(x, y) \in \{(pg, pr), (png, pnr)\}$ and $\pi_2.po$ denotes the second element of a p-obligation, i.e. its probability set.

As a first example of refinement according to Definition 16, consider the two diagrams sendSMS2- and sendSMS3- given in Figure 12. These are simplified versions of sendSMS2 (in Figure 2) and sendSMS3 (in Figure 3), respectively, showing only the communication between the :User and :WebPortal lifelines. In sendSMS2- the :User lifeline first sends the enterTextAdr to the :WebPortal, which then sends the dispayStatus message back. No time constraints are given in sendSMS2-. In sendSMS3-, a soft real-time constraint is added, requiring that the probability of the web portal transmitting the displayStatus message at most 8 seconds after receiving the enterTextAdr message, should be at least 0.8.

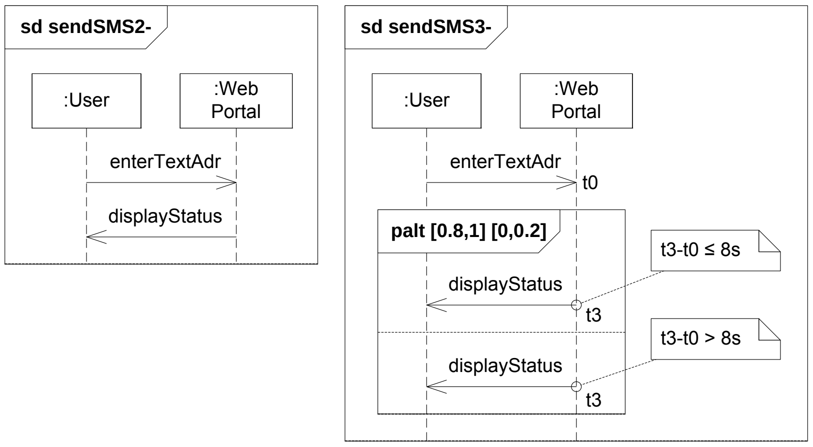

Figure 12: Simplified versions of parts of sendSMS2 and sendSMS3.

Adding a soft real-time constraint like this should constitute a narrowing refinement as the two diagrams describe the same behavior, with the difference that some of the behavior that was valid in sendSMS2- (i.e. behavior where the probability is higher than 0.2 for a delay of more than 8 seconds between the reception of enterTextAdr and the transmission of displayStatus) is not allowed in sendSMS3-.

To see that this is indeed a narrowing refinement, let s_1 denote the set of traces where the delay between reception of enterTextAdr and transmission of displayStatus is at most 8 seconds, while s_2 is the set of traces where the delay is more than 8 seconds.

As sendSMS2- does not include any palt operators, time constraints or other constructs defining negative behavior, it is easy to see that its semantics is a single p-obligation with positive trace set $s_1 \cup s_2$, i.e. $[\![\text{ sendSMS2}-]\!] = ((s_1 \cup s_2, \emptyset), \{1\})$. For sendSMS3-, we get a total of four p-obligations:

$$\begin{array}{ll} po_1 = ((s_1, s_2), [0.8, 1]) & po_2 = ((s_2, s_1), [0, 0.2]) \\ po_3 = ((s_1 \cup s_2, \emptyset), [0.8, 1]) & po_4 = ((s_1 \cup s_2, \emptyset), \{1\}) \end{array}$$

where po_1 and po_2 represent each of the two palt operands, resulting from using Definition (10), part a, with $N = 1$, po_3 represents their combination (using $N = 2$), while po_4 is the final combined p-obligation resulting from Definition (10), line (b).

As po_4 is identical to the single p-obligation for sendSMS2-, Definition (16) is satisfied for both $\rightsquigarrow_{pg}$ and $\rightsquigarrow_{png}$.

As another example, this time involving supplementing, consider diagram sendSMS4- in Figure 13. Compared to sendSMS3- in Figure 12, the difference is that sendSMS4- also includes traces where displayStatus is followed by reportMsg. To see that this constitutes a supplementing refinement, let s_1 and s_2 be as in

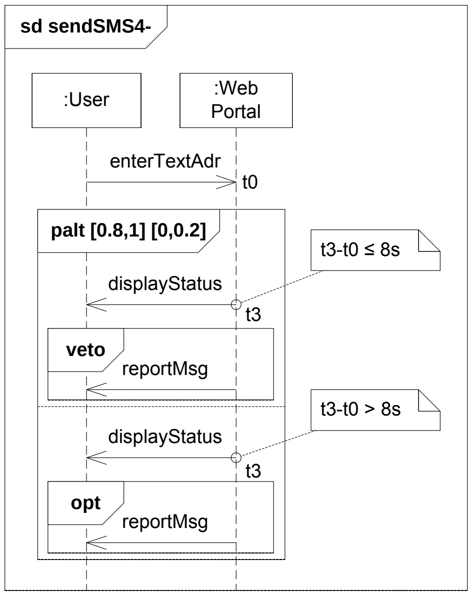

Figure 13: Simplified version of sendSMS4.

the previous example, s_3 be the traces with a delay of at most 8 seconds and including reportMsg, and s_4 be the traces with a delay of more than 8 seconds and including reportMsg. The traces in s_3 and s_4 are all inconclusive in all of the p-obligations for sendSMS3-. For sendSMS4-, we get the following four p-obligations:

$$po'_1 = ((s_1, s_2 \cup s_3 \cup s_4), [0.8, 1]) \quad po'_2 = ((s_2 \cup s_4, s_1 \cup s_3), [0, 0.2])$$
$$po'_3 = ((s_1 \cup s_2 \cup s_4, s_3), [0.8, 1]) \quad po'_4 = ((s_1 \cup s_2 \cup s_4, s_3), \{1\})$$

We see that for each p-obligation po_i in the semantics of sendSMS3-, it is refined by po'_i in the semantics of sendSMS4- according to $\rightsquigarrow_{pr}$ (but not $\rightsquigarrow_{pnr}$), and we may conclude that $[\![$ sendSMS3$-$ $]\!] \rightsquigarrow_{pg} [\![$ sendSMS4$-$ $]\!]$.

As a final example, we demonstrate how the diagram sendSMS5- in Figure 14 is a narrowing refinement of the diagram sendSMS4- in Figure 13. The semantics of the two diagrams are given in Figure 15. Here, u1 is the set of traces where $t3 - t0 \leq 6$ and reportMsg does not occur, while r1 is the set of traces where $t3 - t0 \leq 6$ and reportMsg does occur. Similarly, u2 and r2 are the sets of traces with $6 \leq t3 - t0 \leq 8$, respectively with and without reportMsg, and u3 and r3 are the set of traces with $t3 - t0 \geq 8$, with and without reportMsg. The upper row represents $[\![$ sendSMS4$-$ $]\!]$ and the lower row represents $[\![$ sendSMS5$-$ $]\!]$. Each p-obligation is illustrated by a circle representing the interaction obligation, with the probability set above the circle.

From Figure 15, we see that each p-obligation for sendSMS4- that does not include 0 in its probability set is refined by an p-obligation in sendSMS5-. In

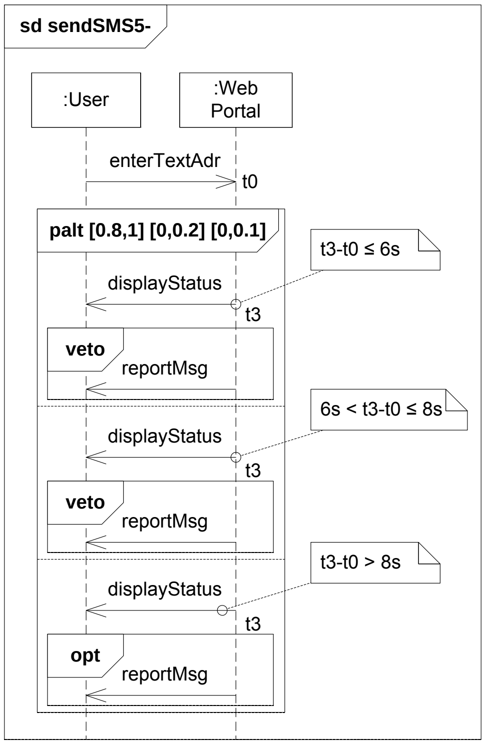

Figure 14: Simplified version of sendSMS5.

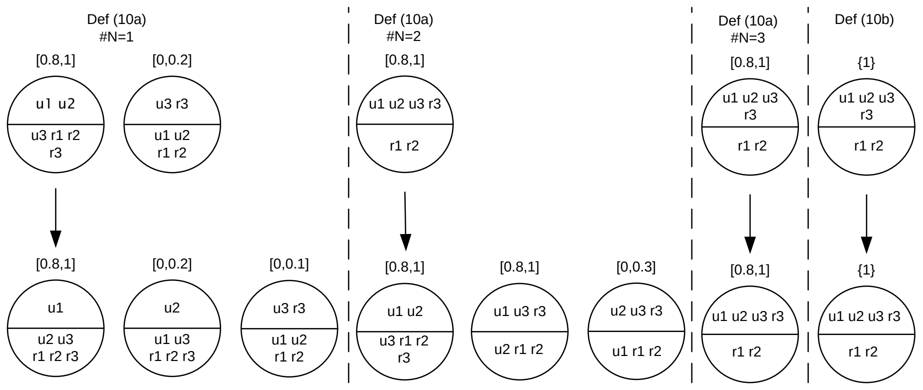

Figure 15: sendSMS4- is refined by sendSMS5-.

each case, it is a narrowing refinement, as the set of inconclusive traces remains the same in the refinement. This means that Definition (16) is fulfilled, and we conclude that sendSMS5- is a narrowing refinement of sendSMS4-. There are also other refinements between the p-obligations in Figure 15, but as these are not required by Definition (16), they are not shown here.

4.3. Properties of refinement – theoretical results

In the refinement examples in Sections 4.1 and 4.2, we considered subparts of the diagrams from Section 2 and demonstrated how the local modifications constituted valid refinements of those subparts. In this section, we present essential properties of our refinement definitions, and show how these may be used to establish refinement of the complete diagrams, without calculating the full semantics for each diagram. For proofs we refer to Appendix C.

Since practical specifications may be large, it is important that different parts of a sequence diagram may be refined separately, without considering the rest of the diagram. These are the mathematical properties of monotonicity and modularity, which are studied in Section 4.3.1.

When performing a series of refinement steps, it is also important that the end result refines not only the previous specification, but also the original one. This is the property of transitivity, which is studied in Section 4.3.2.

4.3.1. Monotonicity and modularity

A binary sequence diagram operator op_{sd} is monotonic with respect to refinement if the following holds: If d_1 is refined by d'_1 and d_2 is refined by d'_2 then $d_1\ op_{sd}\ d_2$ is refined by $d'_1\ op_{sd}\ d'_2$ (see e.g. [13, p. 281]). Formally, an operator op_{sd} with n operands is monotonic with respect to a refinement relation $\rightsquigarrow$ if the following holds:

$$(\forall i \leq n : [\![\ d_i\ ]\!] \rightsquigarrow [\![\ d'_i\ ]\!]) \Rightarrow op_{sd}(d_1, \ldots, d_n) \rightsquigarrow op_{sd}(d'_1, \ldots, d'_n) \tag{17}$$

Except for palt, all of the common sequence diagram operators used in this paper (and formalized in Appendix A) are monotonic with respect to the refinement relations defined in Section 4.2.

Theorem 1. *The operators* refuse, veto, par, seq, alt, loop *and* tc *are monotonic with respect to the refinement relations $\rightsquigarrow_{pg}$ and $\rightsquigarrow_{png}$.*

In the case of palt, there exist some syntactically correct specifications that do not fulfill the monotonicity requirement, so we do not have full monotonicity [14, p. 33]. However, we have a slightly weaker modularity result that, under certain conditions that will normally hold for practical specifications, allows us to ensure that any implementation of the more concrete specification will also be an implementation of the more abstract specification by considering the operands one by one.

For any given refinement relation $\rightsquigarrow_x$, we assume that a corresponding "implements" relation $\mapsto_x$ is defined in such a way that $d \mapsto_x I$ holds if and only if

1. every p-obligation in $\llbracket\ d\ \rrbracket$ whose set of acceptable probabilities does not contain 0 is implemented by I, and
2. for any p-obligations po, po', if $po \rightsquigarrow_y po'$ and po' is implemented by I then po is also implemented by I,

where $(x, y) \in \{(pg, pr), (png, pnr)\}$ and I is an implementation. As the focus of this paper is on sequence diagram specifications and refinement, we do not go into more detail here. An implementation model is presented in Appendix B.

We say that a sequence diagram operator op_{sd} with n operands is modular with respect to the pair $(\rightsquigarrow, \mapsto)$ if the following holds:

$$(\forall i \leq n : \llbracket\ d_i\ \rrbracket \rightsquigarrow \llbracket\ d_i'\ \rrbracket \wedge d_i' \mapsto I_i) \Rightarrow op_{sd}(d_1, \ldots, d_n) \mapsto op_{imp}(I_1, \ldots, I_n) \quad (18)$$

where op_{imp} is a composition operator for implementations corresponding to the sequence diagram operator op_{sd}. For example, if $op_{sd} = \mathsf{par}$ then op_{imp} will be an operator for parallel execution of implementations. This does not mean that each I_i must be constructed in exactly the same way as d_i or d_i' was built. Definition (18) is just a rephrasing of monotonicity in the context of an implementation relation.

For all operators except from palt, modularity follows directly from monotonicity and the assumption that the 'implements' relation is preserved through abstraction. In the case of palt we have modularity under three conditions that from a practical point of view are entirely reasonable:

1. The palt is well-balanced.
2. Each operand d_i is safe in the sense that the positive behavior of d_i at infinite time is completely determined by the behavior of d_i at finite time.
3. The composition operator for implementations that corresponds to palt is *trace preserving* in the sense that the traces produced by the composed implementation I equals the union of the traces produced by the implementations I_i of each alternative.

Theorem 2. *The operator* palt *is modular with respect to* $(\rightsquigarrow_{pg}, \mapsto_{pg})$ *and* $(\rightsquigarrow_{png}, \mapsto_{png})$ *if it is well-balanced, each of its operands is safe and the composition operator for implementations corresponding to* palt *is trace preserving.*

It is normally advisable to make sure that every palt is well-balanced and this condition may be imposed syntactically. The safety condition (formally defined in Appendix A.2.2) restricts us from expressing liveness properties. In a real-time notation like probabilistic STAIRS, progress may be expressed directly as time constraint. Hence, in practice there is no need to express liveness properties. As argued by Olderog and Dierks in [15, p. 6]: "Such liveness property is not strong enough in the context of realtime systems. Here one would like to see a time bound when the good state occurs."

We now argue why each of the diagrams in Section 2 is a valid refinement of the previous one.

The diagram sendSMS1 in Figure 1 may be seen as consisting of three subdiagrams, the first (topmost) four messages above the par operator, the par fragment itself, and the message displayStatus, all composed by the implicit weak sequencing operator seq. The part of sendSMS1 that is changed by sendSMS2 in Figure 2 is given by sendSMS1* and sendSMS2* in Figure 11. In Section 4.1, sendSMS2* was shown to be a valid narrowing refinement of sendSMS1*. By monotonicity of seq, it follows that sendSMS2 is a valid narrowing refinement of sendSMS1, as the other parts are the same in both diagrams.

By an argument similar to the one given for sendSMS2- and sendSMS3- in Section 4.2, it is possible to show that sendSMS3 is a valid narrowing refinement of sendSMS2. In this case, considering only parts of the diagrams and then applying the monotonicity results is not possible, as the time constraints introduced in sendSMS3 stretches across most of the diagram.

For sendSMS4 and sendSMS5, the only changes made are with respect to the palt-fragment. These changes are valid refinements as shown in Section 4.2. By monotonicity of seq, it follows that sendSMS4 is a valid refinement of sendSMS3, and that sendSMS5 is a valid refinement of sendSMS4.

4.3.2. Transitivity

A refinement relation $\rightsquigarrow$ is transitive if the following holds: If d_1 is refined by d_2 and d_2 is refined by d_3, then d_1 is refined by d_3. Formally:

$$[\![\ d_1\ ]\!] \rightsquigarrow [\![\ d_2\ ]\!] \wedge [\![\ d_2\ ]\!] \rightsquigarrow [\![\ d_3\ ]\!] \Rightarrow [\![\ d_1\ ]\!] \rightsquigarrow [\![\ d_3\ ]\!] \tag{19}$$

Theorem 3. *The refinement relations $\rightsquigarrow_{pg}$ and $\rightsquigarrow_{png}$ are transitive.*

In the previous subsection, we argued why each of the example diagrams in Section 2 is a refinement of the previous one. By transitivity, we get that each diagram is a refinement of all of the previous diagrams, and in particular that the final specification in sendSMS5 is a refinement of the original specification in sendSMS1.

5. Related work

This section on related work is organized in two subsections. First, in Section 5.1, we address related work on UML 2.x sequence diagrams and similar notations, including Message Sequence Charts (MSC). The few UML/MSC-related approaches including probability-related constraints are discussed in Section 5.2. As little work has been done on sequence diagrams with probability, we address related work with respect to expressing probabilistic constraints in other kinds of specification languages in Section 5.3.

5.1. Sequence diagrams and similar notations

Probabilistic STAIRS is built on the STAIRS approach as presented in [4, 5]. STAIRS assigns formal semantics to sequence diagrams and defines refinement relations similar to the ones presented here. Time is introduced in STAIRS in [6]. An operational semantics for STAIRS is given in [16, 17], equivalent to the denotational one. However, STAIRS does not have the expressive power to capture constraints that depend on probabilities. The purpose of probabilistic STAIRS is to extend STAIRS in order to be able to capture also soft real-time constraints and other kinds of probabilistic constraints in the formal specifications.

A recent survey of different semantics for UML 2.x sequence diagrams is given by Micskei and Waeselynck in [18]. Probabilities are not covered, as they are not included in the UML 2.x standard. In [12], we have compared STAIRS and its refinement relations to related work on sequence diagrams, including the most relevant from [18], as well as other similar notations, refinement and nondeterminism. The discussion in this section is therefore restricted to aspects concerning time and probabilities.

Of all the approaches discussed in [18] and [12], the one most closely related to STAIRS is the trace-based semantics proposed by Cengarle and Knapp in [19]. Based on that approach, [20] develops a semantics for UML 2.0 Interactions with support for (hard) real-time constraints. A refinement relation for constraints is defined. As is the case in our approach, constraint introduction is shown to be monotonic with respect to refinement. Unlike our approach, real-time constraints or other forms of probabilistic constraints are not considered.

For Message Sequence Charts (MSC), a semantics for basic MSCs with time is given in [21, 22]. The events of a basic MSC are assigned timestamps using a timing function, and timing constraints are used to specify minimum and maximum time intervals between events. In addition, algorithms are given for checking the realizability of MSCs and the existence of a timing function that is consistent with the timing constraints of an MSC. This issue has not been addressed in our approach. In [23], a semantics with time for MSC-2000 is given, based on partially ordered sets. Time is represented by a function mapping each event in a diagram to a set of time values, giving the absolute time interval in which the event should occur. Relative timing constraints are expressed by a function mapping pairs of events to intervals of time values. Again, probabilities are not considered, and soft real-time constraints can not be captured.

Live Sequence Charts (LSC) [24, 25] is an extension of MSC also discussed in [12]. that allow a distinction between possible and necessary behavior, as well as give explicit conditions under which the constraints of the diagram applies. In [26], a time extension to LSCs is presented. Here, a clock variable *Time* is added to the formalism so that time can be treated as data and time constraints can be expressed by means of ordinary variables. [25] includes a construct for probabilistic choice (which they call nondeterministic choice), where an exact probability is assigned to each alternative. Unlike our approach, refinement is not formally defined. However, [24] gives an example of refinement where a system is described in increasing level of detail, and states that refinement can easily be defined from the semantics of LSC.

5.2. Expressing probabilities in UML/MSC

Probabilities are not included in the UML 2.x standard for sequence diagrams, and consequently most approaches relating to UML and other kinds of sequence diagrams does not include probabilities of any kind.

Performance Message Sequence Chart (PMSC) [27, 28] extends MSC with syntactic constructs for expressing performance constraints. The aim is to integrate performance characteristics, such as response time and throughput, in functional specifications. Of particular interest to us is the new operator altprob for probabilistic choice that is introduced in [28]. This operator allows exact probabilities to be assigned to the alternatives represented by its operands. This means that, unlike our palt operator, underspecification with respect to probability can not be captured by this operator. Apart from mentioning instance decomposition, refinement is not discussed, and no definition is given of what it means for a system to comply with a PMSC specification. The semantics of PMSC is explained at a purely intuitive level.

The UML Profile for Schedulability, Performance, and Time Specification [29] and the UML Profile for Modeling and Analysis of Real-Time Embedded Systems (MARTE) [30] extend UML by adding stereotypes and annotations for defining values for performance measures, and allow specification of probability-related constraints, such as soft real-time constraints. The profiles can be used to construct models that can be analyzed with respect to performance, such as Markov chains. [31] presents a technique for constructing a Markov chain model from a sequence diagram that is annotated with constructs from the profile, thereby creating a model that can be analyzed by a suitable performance analysis tool.

5.3. Expressing probabilities in other languages

We are not aware of any approach where probabilities are fully integrated in a formal semantics of sequence diagrams. However, there are a number of other languages where probabilities are fully integrated into the semantics. We now review some of these. Since underspecification with respect to probability and refinement are central issues in our work, we focus specifically on how other approaches deal with these issues. It should be noted that all the approaches differ from ours in that there is no concept of inconclusive behavior.

Two probabilistic variants of the process algebra CSP (Communicating Sequential Processes) [32], called $PCSP_0$ and $PCSP$, are presented in [33]. For both $PCSP_0$ and $PCSP$, an axiomatic characterization of the operators is offered, thus supporting algebraic reasoning about processes. In addition, a satisfaction relation is defined between a specification expressed as a predicate R over traces and a process P expressed in $PGCL$, as follows: A process P satisfies a specification R if $R(t)$ holds for every trace t of P. In both $PCSP_0$ and $PCSP$, the operator $\sqcap$ of CSP is replaced with an operator ${}_p\sqcap$ for probabilistic choice, where p is the probability of choosing the left-hand process and $1 - p$ is the probability of choosing the right-hand process. Underspecification with respect to probabilities cannot be expressed, as only exact probabilities can be assigned to the operator for probabilistic choice, and there is no operator for letting the system choose nondeterministically between two different probabilistic choices.

The language *pGCL* [34, 35] is based on Dijkstra's Guarded Command Language (*GCL*) [36]. *GCL* and *pGCL* can be seen as programming languages where states are represented by variable assignments. Both languages contains an operator $\sqcap$ for nondeterministic (demonic) choice[3]. This operator allows the notion of abstraction, and therefore, also refinement, to be captured. A nondeterministic choice between alternatives gives the union of the alternatives, and refinement is defined as reverse inclusion: *prog′* refines *prog* if the behavior of *prog′* is included in the behavior of *prog*. Hence, the nondeterminism expressed by $\sqcap$ represents underspecification.

In addition to the *GCL* operators, *pGCL* contains an operator ${}_p\oplus$ for probabilistic choice. Underspecification with respect to probability can be expressed by a nondeterministic choice between two probabilistic choices whose alternatives are identical, where the probability values represent the upper and lower bounds on the acceptable probability. Hence, this differs from our approach in that underspecification with respect to probability is captured through a special combination of two operators, rather than allowing sets of probabilities directly on the probabilistic alternatives. A nice feature of *pGCL* is that it also offers a program logic that allows us to discover properties of the system via syntactic manipulations on the *pGCL* program. For example, if S is a set of desirable states, then we may find the probability that execution of the program will end in a state in S.

In [37] it is shown how probabilistic reasoning can be applied to predicative programs and specifications. The semantics of a standard (non-probabilistic) predicative program is given in terms of first-order logic. For example, the program statement `if` b `then` $x := H$ `else` $x := T$ is interpreted as $(b \wedge x = H) \vee (\neg b \wedge x = T)$. This approach is generalized to the probabilistic case by considering Booleans to be a subset of real numbers, where $\top = 1$ and $\bot = 0$. A probabilistic choice can then be expressed with the `if`...`then`...`else`... construct. For example, an unfair coin biased toward the tails outcome can be represented by the program statement `if` 0.4 `then` $x := H$ `else` $x := T$. Nondeterminism is disjunction, and equivalent to an `if`...`then`...`else`... construct where the condition is a variable of unknown value (probability); $P \vee Q$ is equivalent to $\exists p \in [0, 1]$: `if` p `then` P `else` Q. Nondeterminism gives freedom to the implementer, who is intuitively free to choose p. A nondeterministic choice can be refined either by a probabilistic choice (by ensuring that $0 < p < 1$) or by a deterministic choice (by ensuring that $p = 1$ or $p = 0$). As in [34], and unlike our approach, underspecification with respect to probabilities can be expressed

[3]An operator $\sqcup$ for angelic choice is also introduced. Intuitively, in a *pGCL* program, a demonic choice is made by a demon who seeks to minimize the probability of reaching the state under consideration, while an angelic choice is made by an angel who seeks to maximize the same probability.

by a nondeterministic choice between two probabilistic choices.

Probabilistic automata [38, 39] are extensions of labeled transition systems designed to address the problem of modeling and verification of randomized distributed algorithms. Unlike ordinary automata, probabilistic automata allow probabilistic choice to be represented in the form of probabilistic transitions. A probabilistic transition is a transition from a state to a discrete distribution over pairs consisting of a label and a state. This means that, unlike the pSTAIRS approach, each probabilistic alternative (here: pair of a label and state) can only be assigned one exact probability, rather than a set of probabilities. Nondeterminism is represented by the fact that there may be several outgoing (probabilistic) transitions from any state. Underspecification with respect to probabilities can be represented by nondeterministic choices between probabilistic transitions that are identical, except for the probability values of the distribution. [38] proposes hierarchical verification techniques based on either preorders of trace distributions (set inclusion) or on simulation.

A trace-based model for systems with both probabilistic and nondeterministic choice is presented in [40]. Here, a trace is a sequence of states, rather than messages as is the case in pSTAIRS. A state is an assignment of values to a set of variables. Semantically, a system is represented by a set of probability distributions on traces, which are called bundles. The fact that the model contains a set of bundles instead of a single bundle is due to nondeterministic choices. As in, for example, [38], nondeterminism is resolved by a scheduler, so underspecification with respect to probabilities can be expressed in a similar way as in [38]. However, unlike [38] and other earlier work, [40] allows multiple schedulers for the resolution of the nondeterminism within a system in order to achieve deep compositionality. This means that the semantics (set of trace bundles) of a composite system can be obtained from the semantics of its component systems, which is also the case in pSTAIRS, as all composition operators are defined only in terms of the semantics of its operands. For each scheduler, the set of variables it may affect and the set of variables that is visible to the scheduler (the variables upon which the scheduler's choice may depend) must be specified by a so-called *atom*. As the atoms determine probabilistic dependence between variable values, merging of atoms may increase the behavior (bundles) of a system. Atoms form a part of the semantic representation and are taken into consideration (by taking their union) when composing systems. Atoms also play a role with respect to refinement; refinement is basically bundle containment, with the additional requirement that the concrete system (implementation) cannot exhibit more variable dependencies than the abstract system (specification).

In [41], Jonson and Larsen present a specification formalism in the form of probabilistic (unlabeled) transition systems. A probabilistic transition system is a transition system where transitions are assigned sets of allowed probabilities, in the same way as sets of probabilities are assigned to the operands of a palt in pSTAIRS. The use of sets of allowed probabilities instead of exact probabilities represents underspecification with respect to probability, and is motivated partly by the need to specify soft constraints, such as 'the probability of losing a message in a communication channel should be no more than 0.01'. Two different refinement relations between specifications are proposed. The stronger criterion is based on the idea of simulation between specifications. The idea is that a transition in one specification can be simulated by a set of transitions in the other specification, as long as the combined probability of the transitions in this set is an acceptable probability of the original transition. A similar idea can also be reflected in pSTAIRS, as illustrated by the introduction of the palt operator in Figure 4 as a valid refinement step. The weaker criterion views a specification as a definition of a set of processes. Refinement is then defined as set inclusion of processes. A specification S is refined by a specification S' if all the processes of S' are also processes of S. A process (unlike a specification) has exactly one probability assigned to each transition.

In [42, 43], labeled transition systems with both nondeterministic and probabilistic choice are used for specifying systems. Nondeterministic choice is used to represent underspecification, and refinement corresponds to restricting the possible behavior. Refinement relations are defined based on testing. A test is a labeled transition system with both nondeterministic and probabilistic choice, where a subset of the states is defined as success states. A testing system $P \parallel T$ is the parallel composition of a process P and a test T, and from this we can obtain the set of possible probabilities of reaching a success state. Based on this, the concepts of may-refinement and must-refinement are defined as follows: A process P_2 is a may-refinement of a process P_1 if for every test T the highest probability of reaching a success state in $P_2 \parallel T$ is not higher than in $P_1 \parallel T$. P_2 is a must-refinement of P_1 if for every test T the lowest probability of reaching a success state in $P_2 \parallel T$ is not lower than in $P_1 \parallel T$. P_2 is a refinement of P_1 if it is both a may-refinement and a must-refinement of P_1. Intuitively, this means that P_2 refines P_1 if for every test, the interval of probabilities of reaching a success state is made smaller. In other words, a refinement step may reduce underspecification with respect to probability, as is also the case in pSTAIRS. [43] shows that the refinement relations are compositional in the sense that if P_1 is refined by P_2, then $P_1 \parallel P$ is refined by $P_2 \parallel P$ for any process P.

6. Conclusion

In this paper, we have presented probabilistic STAIRS (pSTAIRS), an approach to extend UML 2.x sequence diagrams to capture soft real-time constraints as well as probabilistic choice in general. Soft real-time constraints are expressed by the combined use of operators for probabilistic choice (palt) and real-time constraints (tc). Having separate operators for real-time constraints and probabilistic choice allows us to capture all kinds of probabilistic choice in the same manner, whether related to time or not. In order to obtain a simpler notation for soft real-time constraints, one could easily introduce a macro operator for the combination of palt and tc used to capture such constraints. We have chosen not to introduce such a macro operator, as we wish to emphasize the similarity between soft real-time constraints and general probabilistic choice in the underlying theory. Probabilistic choice is used to capture constraints where there for each alternative is given a set of probabilities for how often the alternative may occur. Soft real-time constraints are a special case, where the difference between the alternatives is the timing constraints.

Probabilistic STAIRS makes it possible to capture underspecification with respect to probability as well as with respect to behavior/traces. Underspecification with respect to probability is essential in order to capture real-time constraints. It is also highly useful for other kinds of probabilistic choices, as the specifier will typically be satisfied as long as a given alternative occurs with a probability within a given interval, rather than with an exact probability. Moreover, achieving an exact probability in the final implementation can be very hard, meaning that specifications requiring exact probabilities may be almost impossible to comply with. Underspecification with respect to behavior/traces can be captured independently of underspecification with respect to probability. This enables refining a (sub-)specification with respect to behavior/traces without worrying about probabilities and vice versa.

Taking into account the incomplete nature of sequence diagrams, a formal semantics consistent with the semi-formal trace semantics of UML 2.x sequence diagrams has been provided for pSTAIRS. Based on this formal semantics, two alternative refinement relations targeting different parts of a development process have been presented. The first refinement relation ($\rightsquigarrow_{pg}$), suitable early in the development process, is used to reduce the amount of incompleteness and/or underspecification in the specification. At a later stage, general narrowing refinement ($\rightsquigarrow_{png}$) may be more suitable, assuming that all relevant behavior is specified so that the only relevant refinement step is reducing underspecification with respect to behavior/traces and/or probabilities. Both refinement relations have been shown to have the mathematical properties of transitivity and monotonicity, which makes it possible to develop and analyze specifications in a stepwise and modular manner. The practical use of the refinement relations and the exploitation of their mathematical properties in order to simplify the analysis have been demonstrated on a scenario from the telecom industry.

Acknowledgements The research on which this paper reports has been partly carried out within the DIGIT project (180052/S10), funded by the Research Council of Norway, and the SecureChange and NESSoS projects, both funded from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreements FP7-231101 and FP7-256980, respectively. We are grateful for the anonymous reviewers for a very thorough reading of the paper and for suggesting a number of improvements.

References

[1] B. Dobing and J. Parsons. How UML is used. *Communications of the ACM*, 49(5):109–113, 2006.

[2] T. Weigert. *UML 2.0 RFI Response Overview.* Object Management Group, document: ad/00-01-07 edition, 1999.

[3] S. Mauw, M. A. Reniers, and T. A. C. Willemse. Message sequence charts in the software engineering process. In *Handbook of Software Engineering and Knowledge Engineering*, pages 437–463. World Scientific, 2001.

[4] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. STAIRS towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, 22(4):349–458, 2005.

[5] R. K. Runde, Ø. Haugen, and K. Stølen. Refining UML interactions with underspecification and nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.

[6] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. In *Proc. Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 1–25. Springer, 2005.

[7] ETSI. *Open Service Access (OSA); Parlay X Web Services; Part 4: Short Messaging (Parlay X 2)*, 2006.

[8] A. Refsdal and S. Jacobsson. Adding soft real-time requirements in a stepwise development process. *Telektronikk*, 105(1):69–80, 2009.

[9] C. B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall, 1986.

[10] Object Management Group. *OMG Unified Modeling Language (OMG UML) Superstructure Version 2.4.1*, document: formal/2011-08-06 edition, 2011.

[11] A. Refsdal, R. K. Runde, and K. Stølen. Underspecification, inherent nondeterminism and probability in sequence diagrams. In *Proc. 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, volume 4037 of *LNCS*, pages 138–155. Springer, 2006.

[12] R. K. Runde, A. Refsdal, and K. Stølen. Relating computer systems to sequence diagrams: the impact of underspecification and inherent nondeterminism. *Formal Aspects of Computing*, 25:159–187, 2013.

[13] R.-J. Back and J. von Wright. *Refinement calculus - a systematic introduction.* Undergraduate texts in computer science. Springer, 1999.

[14] A. Refsdal, K. E. Husa, and K. Stølen. Specification and refinement of soft real-time requirements using sequence diagrams. Technical Report 323, Department of Informatics, University of Oslo, 2007.

[15] E.-R. Olderog and H. Dierks. *Real-time systems - formal specification and automatic verification.* Cambridge University Press, 2008.

[16] M. S. Lund and K. Stølen. A fully operational semantics for UML 2.0 sequence diagrams with potential and mandatory choice. In *Proc. 14th International Symposium on Formal Methods*, volume 4085 of *LNCS*, pages 380–395. Springer, 2006.

[17] M. S. Lund. *Operational Analysis of Sequence Diagram Specifications.* PhD thesis, University of Oslo, 2008.

[18] Z. Micskei and H. Waeselynck. The many meanings of UML 2 sequence diagrams: a survey. *Software and System Modeling*, 10(4):489–514, 2011.

[19] M. V. Cengarle and A. Knapp. UML 2.0 interactions: Semantics and refinement. In *Proc. 3rd International Workshop on Critical Systems Development with UML (CSDUML)*, Technical report TUM-I0415, pages 85–99. Institut für Informatik, Technische Universität München, 2004.

[20] D. Calegari, M. V. Cengarle, and N. Szasz. UML 2.0 interactions with OCL/RT constraints. In *Proc. Forum on specification and Design Languages (FDL)*, pages 167–172, 2008.

[21] R. Alur, G. J. Holzmann, and D. Peled. An analyser for message sequence charts. In *Proc. Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop (TACAS '96)*, volume 1055 of *LNCS*, pages 35–48. Springer, 1996.

[22] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. *IEEE Transactions on Software Engineering*, 29(7):623–633, 2003.

[23] T. Zheng, F. Khendek, and L. Hélouët. A semantics for timed MSC. *Electronic Notes in Theoretical Computer Science*, 65(7), 2002.

[24] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[25] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine.* Springer, 2003.

[26] D. Harel and P. S. Thiagarajan. Message sequence charts. In *UML for Real: Design of Embedded Real-Time Systems*, pages 77–105. Kluwer Academic Publishers, 2003.

[27] N. Faltin, L. Lambert, A. Mitschele-Thiel, and F. Slomka. An annotational extension of message sequence charts to support performance engineering. In *SDL'97, Time For Testing*, SDL Forum, pages 307–322. Elsevier, 1997.

[28] L. Lambert. PMSC for performance evaluation. In *Proc. 1. Workshop on Performance and Time in SDL/MSC*, pages 70–80, 1998.

[29] Object Management Group. *UML Profile for Schedulability, Performance, and Time Specification (Version 1.1)*, document: formal/05-01-02 edition, 2005.

[30] Object Management Group. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems (Version 1.1)*, document: formal/2011-06-02 edition, 2011.

[31] A. A. Abdulatif and R. Pooley. A computer assisted state marking method for extracting performance models from design models. *International Journal of Simulation Systems, Science and Technology*, pages 36–46, September 2007.

[32] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[33] K. Seidel. Probabilistic communicating processes. *Theoretical Computer Science*, 152(2):219–249, 1995.

[34] A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Springer, 2005.

[35] A. McIver and C. Morgan. Developing and reasoning about probabilistic programs in pGCL. In *Proc. First Pernambuco Summer School on Software Engineering, Revised Lectures (PSSE)*, volume 3167 of *LNCS*, pages 123–155. Springer, 2006.

[36] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[37] E. C. R. Hehner. Probabilistic predicative programming. In *Proc. 7th International Conference on Mathematics of Program Construction*, number 3125 in LNCS, pages 169–185. Springer, 2004.

[38] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Massachusetts Institute of Technology, 1995.

[39] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[40] L. de Alfaro, T. A. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In *Proc. 12th International Conference on Concurrency Theory (CONCUR)*, pages 351–365. Springer, 2001.

[41] B. Jonsson and K. G. Larsen. Specification and refinement of probabilistic processes. In *Proc. 6th Annual IEEE Symposium on Logic in Computer Science*, pages 266–277, 1991.

[42] B. Jonsson, C. Ho-Stuart, and W. Yi. Testing and refinement for nondeterministic and probabilistic processes. In *Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863, pages 418–430. Springer, 1994.

[43] B. Jonsson and W. Yi. Compositional testing preorders for probabilistic processes. In *Proc. 10th Annual IEEE Symposium on Logic in Computer Science*, pages 431–443. IEEE Computer Society, 1995.

[44] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: FOCUS on Streams, Interfaces, and Refinement.* Monographs in Computer Science. Springer, 2001.

[45] R. K. Runde, Ø. Haugen, and K. Stølen. How to transform UML neg into a useful construct. In *Proc. Norsk Informatikkonferanse*, pages 55–66. Tapir, 2005.

[46] A. Refsdal, R. K. Runde, and K. Stølen. Relating computer systems to sequence diagrams with underspecification, inherent nondeterminism and probabilistic choice, Part 2. Technical Report 347, Department of Informatics, University of Oslo, 2007.

[47] R. M. Dudley. *Real Analysis and Probability.* Cambridge studies in advanced mathematics. Cambridge, 2002.

[48] A. Refsdal, R. K. Runde, and K. Stølen. Stepwise refinement of sequence diagrams with soft real-time requirements. Technical Report A19749, SINTEF ICT, 2011.

Appendix A. Formal semantics of composition operators

In this section we give a more detailed explanation of the semantic domain of specifications, and provide formal definitions of the remaining composition operators.

Appendix A.1. Events and traces

A *trace* is a sequence of *events* representing a system run. An event is a triple (k, m, t) consisting of a kind k, a message m, and a timestamp tag t. The kind k can be either !, denoting a transmission event, or ?, denoting a reception event. A message is a triple (s, tr, re) consisting of a signal s, a transmitter lifeline tr, and a receiver lifeline re. Every timestamp tag is assigned a timestamp, which is a positive real number, to indicate the time of occurrence for the event. Constraints on the timing of events are imposed by the use of logical formulas with timestamp tags as free variables. We let $\mathcal{E}$ denote the set of all events.

The empty trace $\langle\rangle$ and all traces with only one event are well-formed. For a trace h with two or more events to be well-formed, we require that:

- for all messages, if both the transmitter and receiver lifelines are present in the trace (meaning that they occur as the transmitter or receiver of at least one event in the trace), then both the transmit and receive events are present in the trace, and the transmit event is ordered before its corresponding receive event;
- no event occurs more than once in the trace;
- if event e_1 occurs before event e_2 in the trace, then the timestamp assigned to the timestamp tag of e_2 is greater than or equal to the timestamp assigned to the timestamp tag of e_1.

We let $\mathcal{H}$ denote the set of all traces that are well-formed. In addition, for an infinite trace to be well-formed we require that time will eventually progress beyond any finite point in time. The following constraint states that for each lifeline l represented by infinitely many events in the trace h, there exists an event in h that takes place on l and whose timestamp is greater than t for any possible timestamp t:

$$\forall l \in \mathcal{L} : (\#e.l \circledS h = \infty \Rightarrow \forall t \in \mathbb{R} : \exists i \in \mathbb{N} : r.(e.l \circledS h)[i] > t) \tag{A.1}$$

where $\mathcal{L}$ denotes the set of all lifelines; $e.l$ is the set of events that may take place on the lifeline l, i.e. all transmission events where l is the transmitter and all receive events where l is the receiver, $e.l \circledS h$ is the trace obtained from h by removing all events not in $e.l$ (meaning that all events on $e.l \circledS h$ take place on l); and $r.(e.l \circledS h)[i]$ is the timestamp of the i'th event of $e.l \circledS h$. In other words, $e.\_$ is a function returning the set of all events that may take place on a given lifeline and $r.\_$ is a function returning the timestamp of a given event. $\circledS$ is a filtering operator for traces, i.e. $E \circledS h$ is the trace obtained from the trace h by removing from h all events that are not in the set of events E. For instance, assuming $E = \{a_i \mid i \in \mathbb{N}\} \cup \{b_i \mid i \in \mathbb{N}\}$ we have that

$$E \circledS \langle a_1, c_1, c_2, b_1, b_2, d_1, a_2, c_3\rangle = \langle a_1, b_1, b_2, a_2\rangle$$

A formal definition of $\circledS$ can be found in [44].

Appendix A.2. Semantics of composition operators

In this section we define the semantics of the composition operators. But first we present the abstract/textual syntax of probabilistic sequence diagrams.

Appendix A.2.1. Abstract/textual syntax

The set of syntactically correct sequence diagrams, $\mathcal{D}$, is defined inductively as the least set such that[4]:

- $\mathcal{E} \subset \mathcal{D}$
- $\mathsf{skip} \in \mathcal{D}$
- $d \in \mathcal{D} \Rightarrow \mathsf{refuse}\ d \in \mathcal{D} \wedge \mathsf{veto}\ d \in \mathcal{D} \wedge \mathsf{opt}\ d \in \mathcal{D}$
- $d_1, d_2 \in \mathcal{D} \Rightarrow d_1\ \mathsf{par}\ d_2 \in \mathcal{D} \wedge d_1\ \mathsf{seq}\ d_2 \in \mathcal{D} \wedge d_1\ \mathsf{alt}\ d_2 \in \mathcal{D}$
- $d_1, \ldots, d_m \in \mathcal{D} \wedge Q_1, \ldots, Q_m \subseteq [0,1] \Rightarrow \mathsf{palt}(d_1;Q_1, \ldots, d_m;Q_m) \in \mathcal{D} \wedge \mathsf{expalt}(d_1;Q_1, \ldots, d_m;Q_m) \in \mathcal{D}$
- $d \in \mathcal{D} \wedge I \subseteq \mathbb{N}_0 \cup \{\infty\} \Rightarrow \mathsf{loop}\ I\ d \in \mathcal{D}$
- $d \in \mathcal{D} \wedge C \in \mathbb{F}(tt.d) \Rightarrow d\ \mathsf{tc}\ C \in \mathcal{D}$

[4]We sometimes use infix notation also for palt when there are only two operands.

where $\mathbb{F}(tt.d)$ denotes the set of logical formulas whose free variables are contained in the set $tt.d$ of all timestamp tags occurring in the diagram d.

The first two cases imply that any event, as well as the empty diagram, is a sequence diagram. Any other sequence diagram is constructed by the use of operators for negative behavior (refuse and veto), optional behavior (opt), parallel execution (par), weak sequencing (seq), potential choice/underspecification (alt), probabilistic choice (palt and expalt), time constraint (tc) or loop (loop). The semantics of these operators will be explained in Appendix A.2.2, except from probabilistic choice, which was covered in the main section. Note that the seq operator occurs implicitly in the graphical diagrams.

Appendix A.2.2. Parallel composition, sequential composition, underspecification, negative behavior, time constraints and loop

We now define the operators that allow us to express parallel composition, sequential composition, underspecification with respect to behavior, negative behavior, and time constraints. But first we need to introduce some basic operators on traces, trace sets, interaction obligations, and p-obligations.

Parallel composition ($\|$) of two trace sets corresponds to point-wise interleaving of their individual traces. The ordering of events within each trace is maintained in the result. For sequential composition ($\succsim$) we require in addition that for events on the same lifeline, all events from the first trace are ordered before the events from the second trace. Formally:

$$\begin{aligned} s_1 \parallel s_2 \quad &\overset{\mathsf{def}}{=} \quad \{h \in \mathcal{H} \mid \exists p \in \{1,2\}^\infty : \pi_2((\{1\} \times \mathcal{E}) \circledT (p,h)) \in s_1 \\ &\qquad \wedge \pi_2((\{2\} \times \mathcal{E}) \circledT (p,h)) \in s_2\} \end{aligned} \tag{A.2}$$

$$\begin{aligned} s_1 \succsim s_2 \quad &\overset{\mathsf{def}}{=} \quad \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \\ &\qquad \forall l \in \mathcal{L} : e.l \circledS h = e.l \circledS h_1 \frown e.l \circledS h_2\} \end{aligned} \tag{A.3}$$

where π_2 is a projection operator returning the second element of a pair and $\frown$ is the concatenation operator for sequences. The operator $\circledT$ is a generalization of $\circledS$ filtering pairs of traces with respect to pairs of elements such that, for instance

$$\{(1,e_1),(1,e_2)\} \circledT (\langle 1,1,2,1,2\rangle, \langle e_1,e_1,e_1,e_2,e_2\rangle) = (\langle 1,1,1\rangle, \langle e_1,e_1,e_2\rangle)$$

A formal definition of $\circledT$ can be found in [44].

Time constraints are imposed by the use of a time constraint, denoted by $\wr C$, where C is a predicate over timestamp tags. When a time constraint is applied to a trace set, all traces not fulfilling the constraint are removed. Formally, time constraint for a trace set s is defined as

$$s \wr C \overset{\mathsf{def}}{=} \{h \in s \mid h \models C\} \tag{A.4}$$

where $h \models C$ holds if for all possible assignments of timestamps to timestamp tags done by the trace h, there is an assignment of timestamps to the remaining timestamp tags in C (possibly none) such that C evaluates to true. For example, assume that

$$h = \langle (k_1, m_1, t_1 \mapsto r_1), (k_2, m_2, t_2 \mapsto r_2), (k_3, m_3, t_3 \mapsto r_3) \rangle \text{ and } C = t_3 < t_1 + 5$$

where $t_i \mapsto r_j$ denotes that timestamp r_j is assigned to timestamp tag t_i. Then $h \models C$ if $r_3 < r_1 + 5$.

For interaction obligations, parallel composition ($\|$), sequential composition ($\succsim$), inner union ($\uplus$), refusal ($\dagger$), and time constraint ($\wr$) are defined by:

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

$$(p_1, n_1) \succsim (p_2, n_2) \overset{\text{def}}{=} (p_1 \succsim p_2, (n_1 \succsim p_2) \cup (n_1 \succsim n_2) \cup (p_1 \succsim n_2)) \quad (A.6)$$

$$(p_1, n_1) \uplus (p_2, n_2) \overset{\text{def}}{=} (p_1 \cup p_2, n_1 \cup n_2) \quad (A.7)$$

$$\dagger(p_1, n_1) \overset{\text{def}}{=} (\emptyset, p_1 \cup n_1) \quad (A.8)$$

$$(p, n) \wr C \overset{\text{def}}{=} (p \wr C, n \cup (p \wr \neg C)) \quad (A.9)$$

Notice that for $\|$ and $\succsim$, composing a positive and a negative trace always yields a negative trace, while the result of composing an inconclusive trace with a positive or negative trace is always inconclusive.

Inner union $\uplus$ represents underspecification with respect to behavior/traces. The idea is that two different interaction obligations represent behavior that from the specifier's point of view are equivalent with respect to their positive and negative traces. Hence the interaction obligations can be combined into a single interaction obligation, thus allowing us to introduce new positive or negative traces in an interaction obligation.

Finally, time constraint $\wr$ defines traces that do not fulfill the constraint as negative, while traces that fulfill the constraint are positive.

Definitions (A.5) to (A.9) for interaction obligations are extended to p-obligations as follows:

$$(o_1, Q_1) \parallel (o_2, Q_2) \overset{\text{def}}{=} (o_1 \parallel o_2, Q_1 * Q_2) \quad (A.10)$$

$$(o_1, Q_1) \succsim (o_2, Q_2) \overset{\text{def}}{=} (o_1 \succsim o_2, Q_1 * Q_2) \quad (A.11)$$

$$(o_1, Q_1) \uplus (o_2, Q_2) \overset{\text{def}}{=} (o_1 \uplus o_2, Q_1 * Q_2) \quad (A.12)$$

$$\dagger(o, Q) \overset{\text{def}}{=} (\dagger o, Q) \quad (A.13)$$

$$(o, Q) \wr C \overset{\text{def}}{=} (o \wr C, Q) \quad (A.14)$$

where we write o_i for an interaction obligation (p_i, n_i). The multiplication of probability sets when composing two p-obligation with $\|$, $\succsim$ or $\uplus$ is motivated by the fact that such compositions always occur in the context of composing specifications represented by sets of p-obligations, where each p-obligation in the resulting composed specification is obtained by choosing independently one p-obligation from each set. In other words, the composition of two sets of p-obligations is the set we may obtain by choosing one p-obligation from each set and composing these two p-obligations. Thus, the definitions of parallel composition ($\|$), sequential composition ($\succsim$), inner union ($\uplus$), refusal ($\dagger$), and time constraint ($\wr$) are lifted from p-obligations to sets of p-obligations in a straightforward manner:

$$O_1 \; op \; O_2 \overset{\mathsf{def}}{=} \{po_1 \; op \; po_2 \mid po_1 \in O_1 \wedge po_2 \in O_2\} \tag{A.15}$$

$$\dagger O \overset{\mathsf{def}}{=} \{\dagger po \mid po \in O\} \tag{A.16}$$

$$O \wr C \overset{\mathsf{def}}{=} \{po \wr C \mid po \in O\} \tag{A.17}$$

where op is one of $\|$, $\succsim$ and $\uplus$.

We are now ready to define the semantics of the pSTAIRS operators. The semantics of an event $(k, m, t) \in \mathcal{E}$ is the interaction obligation whose positive set consists of infinitely many unary positive traces – one for each possible assignment of a timestamp to the timestamp tag of the event. The negative set is empty, and 1 is the only allowed probability:

$$[\![\; (k, m, t) \;]\!] \overset{\mathsf{def}}{=} \{((\{\langle(k, m, t \mapsto r)\rangle \mid r \in \mathbb{R}\}, \emptyset), \{1\})\} \tag{A.18}$$

The empty diagram denotes an empty trace:

$$[\![\; \mathsf{skip} \;]\!] \overset{\mathsf{def}}{=} \{((\{\langle\rangle\}, \emptyset), \{1\})\} \tag{A.19}$$

The operators seq, par, alt and refuse are defined as follows:

$$[\![\; d_1 \; \mathsf{seq} \; d_2 \;]\!] \overset{\mathsf{def}}{=} [\![\; d_1 \;]\!] \succsim [\![\; d_2 \;]\!] \tag{A.20}$$

$$[\![\; d_1 \; \mathsf{par} \; d_2 \;]\!] \overset{\mathsf{def}}{=} [\![\; d_1 \;]\!] \parallel [\![\; d_2 \;]\!] \tag{A.21}$$

$$[\![\; d_1 \; \mathsf{alt} \; d_2 \;]\!] \overset{\mathsf{def}}{=} [\![\; d_1 \;]\!] \uplus [\![\; d_2 \;]\!] \tag{A.22}$$

$$[\![\; \mathsf{refuse} \; d \;]\!] \overset{\mathsf{def}}{=} \dagger[\![\; d \;]\!] \tag{A.23}$$

$$[\![\; d \; \mathsf{tc} \; C \;]\!] \overset{\mathsf{def}}{=} [\![\; d \;]\!] \wr C \tag{A.24}$$

The macro operators veto and opt are defined by:

$$\mathsf{veto} \; d \overset{\mathsf{def}}{=} \mathsf{skip} \; \mathsf{alt} \; \mathsf{refuse} \; d \tag{A.25}$$

$$\mathsf{opt} \; d \overset{\mathsf{def}}{=} \mathsf{skip} \; \mathsf{alt} \; d \tag{A.26}$$

Notice that the two operators refuse and veto are used instead of the UML operator neg. The reason for this is explained in [45]. Space restrictions prevent us from including the formal semantics of loop. Intuitively, any finite loop corresponds to a finite number of seq operators, while the semantics of an infinite loop is made up of p-obligations whose traces, when projected on each lifeline, constitute the least upper bound with respect to prefixing of a sequence of

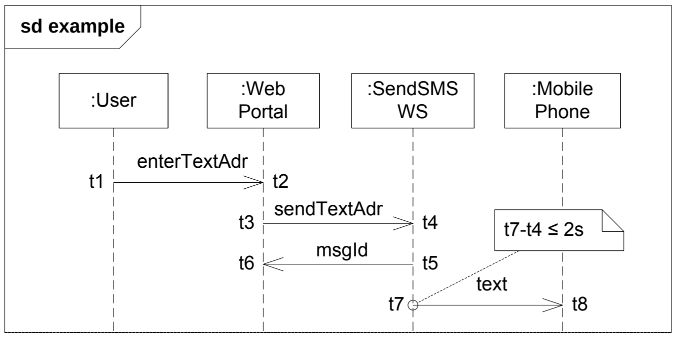

Figure A.16: Example scenario for illustration of semantics.

traces obtained through any finite number of sequential compositions. The formal definition can be found in [14].

Having defined the specification language, we now define formally what it means for a specification to be safe, which was described intuitively in Section 4.3.1. A *chain* c is an infinite sequence of traces such that $\forall j \in \mathbb{N} : c[j] \sqsubseteq c[j+1]$, where $c[j]$ denotes the jth element of c. Any chain has a least upper bound with respect to $\sqsubseteq$ denoted by $\sqcup c$.

A specification d is *safe* iff the following holds for any chain c:

$$(\forall j \in \mathbb{N} : \exists t \in \mathcal{H} \backslash \pi_2.\oplus[\![\, d \,]\!] : c[j] \sqsubseteq t \wedge \#\sqcup c = \infty) \Rightarrow \sqcup c \in \mathcal{H} \backslash \pi_2.\oplus[\![\, d \,]\!] \quad \text{(A.27)}$$

where $\# \sqcup c$ denotes the length of $\sqcup c$.

Appendix A.2.3. Example

We now illustrate the definitions of weak sequencing and time constraints with an example. Consider the diagram example in Figure A.16, where we have chosen to show the timestamp tags for all events explicitly. The semantics of this specification contains only a single p-obligation, as the palt operator must be employed to introduce more p-obligations (this is explained later). Hence, we get $[\![\, \mathsf{example} \,]\!] = \{((p, n), \{1\})\}$, where p is the set of traces that is described as positive by example and n is the set of traces that is described as negative by example. We now explain which traces are contained in p and n.

The traces described by the diagram follows the restrictions on the ordering of events imposed by weak sequencing. The first event to occur can only be transmission of enterTextAdr on the :User lifeline, as the first event on all other lifelines are reception events that cannot occur before their corresponding transmission events. For the same reason, the second event can only be reception of enterTextAdr on the :WebPortal lifeline, followed by transmission of sendTextAdr on the same lifeline and reception of sendTextAdr on the :SendSMSWS lifeline. Again, there is only one event that can occur next at this point, namely transmission of msgId on the :SendSMSWS lifeline. However, after transmission of this message there are two possibilities for the next event: either the msgId message is received by the web portal or the text message is sent from the web service. As these events occur on different lifelines (and transmission of msgId has already occurred), weak sequencing does not impose a particular order between these two events. In the case where the transmission event of text occurs before the reception event of msgId, there are again two alternatives for the next event: either reception of msgId occurs before reception of text, or vice versa. This means that there are three alternative orderings that differ with respect to the following: 1) Reception of msgId occurs before transmission of text. 2) Transmission of text occurs before reception of msgId, and reception of msgId occurs before reception of text. 3) Transmission of text occurs before reception of msgId, and reception of text occurs before reception of msgId.

Before showing what the resulting traces look like we introduce the following shorthand notation for messages:

$$\begin{aligned}
et &= (\mathsf{enterTextAdr}, \mathsf{User}, \mathsf{WebPortal})\\
st &= (\mathsf{sendTextAdr}, \mathsf{WebPortal}, \mathsf{SendSMSWS})\\
mi &= (\mathsf{msgId}, \mathsf{SendSMSWS}, \mathsf{WebPortal})\\
te &= (\mathsf{text}, \mathsf{SendSMSWS}, \mathsf{MobilePhone}).
\end{aligned}$$

Ignoring for the moment the difference between positive and negative traces, the set s of all traces described by example equals the union $s = s_1 \cup s_2 \cup s_3$ of the trace sets we get by ordering the events according to the alternatives described above. The trace sets s_1, s_2, s_3 are then given by the following:

$$\begin{aligned}
s_1 = \{ & \langle (!, et, t_1 \mapsto r_1), (?, et, t_2 \mapsto r_2), (!, st, t_3 \mapsto r_3), (?, st, t_4 \mapsto r_4),\\
& (!, mi, t_5 \mapsto r_5), (?, mi, t_6 \mapsto r_6), (!, te, t_7 \mapsto r_7), (?, te, t_8 \mapsto r_8) \rangle\\
& \mid \forall i < 8 : r_i \leq r_{i+1} \}\\
s_2 = \{ & \langle (!, et, t_1 \mapsto r_1), (?, et, t_2 \mapsto r_2), (!, st, t_3 \mapsto r_3), (?, st, t_4 \mapsto r_4),\\
& (!, mi, t_5 \mapsto r_5), (!, te, t_7 \mapsto r_7), (?, mi, t_6 \mapsto r_6), (?, te, t_8 \mapsto r_8) \rangle\\
& \mid \forall i < 8 : r_i \leq r_{i+1} \}\\
s_3 = \{ & \langle (!, et, t_1 \mapsto r_1), (?, et, t_2 \mapsto r_2), (!, st, t_3 \mapsto r_3), (?, st, t_4 \mapsto r_4),\\
& (!, mi, t_5 \mapsto r_5), (!, te, t_7 \mapsto r_7), (?, te, t_8 \mapsto r_8), (?, mi, t_6 \mapsto r_6) \rangle\\
& \mid \forall i < 8 : r_i \leq r_{i+1} \}
\end{aligned}$$

Note that s_1, s_2, and s_3 are infinite sets due to the fact that there are infinitely many ways of assigning timestamps to the timestamp tags.

The positive traces according to example are the traces of s that also fulfills the time constraint attached to the transmission event of text, while the negative traces are those that do not fulfill this constraint. Hence, we have

$$\begin{aligned}
p &= \{h \in s \mid r_7 - r_4 \leq 2\}\\
n &= \{h \in s \mid \neg(r_7 - r_4 \leq 2)\}
\end{aligned}$$

Appendix B. Implementation model

Based on our work in [46] we present an implementation model and define what it means for a system to implement a probabilistic sequence diagram. To do this we need 1) a mathematical representation of the computer system, and 2) a characterization of the relation between a specification and the mathematical representation of the system. We present this implementation model in order to provide examples (in Appendix B.2) of implementation relations fulfilling the criteria in Section 4.3.1. However, there are many other possible implementation models, and the modularity proofs in Appendix C are independent of the exact implementation model used.

Appendix B.1. Representation of a system

In order to build an implementation model for a system I, we need to know the set of traces $traces(I)$ that the system I is able to produce, as well as information about probabilities. A common mathematical model of a probabilistic process is a probability space, i.e. a triple $(\Omega, \mathcal{F}, f)$ where

- Ω is a *sample space*, i.e. a set of outcomes.
- $\mathcal{F}$ is a *σ-field* on Ω, i.e. a set of subsets of Ω that is closed under complement and countable union, and that contains Ω.
- f is a *probability measure* on $\mathcal{F}$, i.e. a function from $\mathcal{F}$ to $[0,1]$ assigning probabilities to the sets in $\mathcal{F}$ such that $f(\Omega) = 1$ and for any sequence $\omega_1, \omega_2, \ldots$ of disjoint sets from $\mathcal{F}$, the following holds: $f(\bigcup_{i=1}^{\infty} \omega_i) = \sum_{i=1}^{\infty} f(\omega_i)$.

We assume that I is represented by a probability space $(traces(I), \mathcal{F}_I, f_I)$. In other words, $traces(I)$ is the sample space of our probability space. However, we are not interested in an arbitrary $\mathcal{F}_I$. We want to ensure that the probability space gives the necessary information with respect to probabilities. $\mathcal{F}_I$ contains all sets for which the probability is known, and we could for example let $\mathcal{F}_I$ be the σ-field $\{\emptyset, traces(I)\}$. But a probability space with this σ-field would tell us nothing about probabilities, except that the probability of producing a trace in $traces(I)$ is 1. To ensure that the necessary information about probabilities is contained in the probability space we require $\mathcal{F}_I$ to be the *cone-σ-field* of $traces(I)$.

The cone-σ-field is the smallest σ-field [47, p. 86] generated from the set C_I of *cones* we obtain from $traces(I)$. The cone c_t of a finite trace t is the set of all traces with t as a prefix, formally defined by:

$$c_t \overset{\mathsf{def}}{=} \{t' \in traces(I) \mid t \sqsubseteq t'\} \tag{B.1}$$

where $\sqsubseteq$ is the standard prefix operator on traces. The set of cones C_I contains the cone of every finite trace that is a prefix of a trace in $traces(I)$, formally defined by:

$$C_I \overset{\mathsf{def}}{=} \{c_t \mid \#t \in \mathbb{N}_0 \wedge \exists t' \in traces(I) : t \sqsubseteq t'\} \tag{B.2}$$

One may ask why we did not simply require $\mathcal{F}_I$ to be the power set of $traces(I)$. This would ensure that a representation of a system would contain information about the probability of *every* subset of $traces(I)$. The answer is that not all processes can be represented by a probability space whose σ-field is the power set of its sample space. For example, assume I is a process that flips a fair coin infinitely many times. Then the set $traces(I)$ is uncountable, and the probability of each single trace is 0. According to the continuum hypothesis – which states that there is no set whose size is strictly between that of the integers and that of the real numbers – the cardinality of $traces(I)$ then equals the cardinality of the real numbers, and hence of $[0, 1]$. The following theorem taken from [47, Appendix C] by Banach and Kuratowski then implies that there is no measure f_I on $\mathbb{P}(traces(I))$ such that $f_I(\{t\}) = 0$ for each $t \in traces(I)$ and $f_I(traces(I)) = 1$:

> Assuming the continuum hypothesis, there is no measure μ defined on all subsets of $\Omega = [0, 1]$ with $\mu(\Omega) = 1$ and $\mu(x) = 0$ for each $x \in \Omega$.

Our decision to use a cone-based probability space to represent probabilistic systems is inspired by [38]. In [38, p. 52] probability spaces whose σ-fields are cone-σ-fields are used to represent fully probabilistic automata, i.e. automata with probabilistic choice but without nondeterminism. This is done in order to define formally how to compute probabilities for trace sets. A cone-based probability space is a suitable representation of a probabilistic system, since it gives maximum information about probabilities while still allowing processes such as an infinite coin toss to be represented.

Note that for any trace t in $traces(I)$ we have $\{t\} \in \mathcal{F}_I$ (which is proved in [46]). As $\mathcal{F}_I$ is closed under countable union, this means that for any countable $s \subseteq traces(I)$ we have $s \in \mathcal{F}_I$. Consequently, the probability of every finite subset of $traces(I)$ is included in the system representation, and if $traces(I)$ is finite then $\mathcal{F}_I = \mathbb{P}(traces(I))$.

In order to check whether a system implements a pSTAIRS specification we represent the system in the same way as a specification, i.e. as a set of p-obligations. To ensure that all information from the cone-σ-field is contained in the representation we generate one p-obligation from every trace set in $\mathcal{F}_I$. The pSTAIRS representation $\langle I \rangle_d$ of the system I is defined by:

$$\langle I \rangle_d \quad \overset{\mathsf{def}}{=} \quad \{((s, \mathcal{H}^d \setminus s), \{f_I(s)\}) \mid s \in \mathcal{F}_I \wedge s \neq \emptyset\} \tag{B.3}$$

The subscript d means that the representation is related to the specification d, i.e. that only traces where all events occur exclusively on lifelines in d are included as negative. $\mathcal{H}^d$ denotes the set of all traces where this holds.

Appendix B.2. Implementation relations

Having established a system representation on the same form as our specifications, we may now define what it means for a system I to implement a specification d in the same way as we defined refinement. Again, we start by defining implementation relations for single p-obligations:

$$(o,Q) \mapsto_{pr} (o',Q') \overset{\mathsf{def}}{=} o \mapsto_{r} o' \wedge Q' \subseteq Q \tag{B.4}$$

$$(o,Q) \mapsto_{pnr} (o',Q') \overset{\mathsf{def}}{=} o \mapsto_{nr} o' \wedge Q' \subseteq Q \tag{B.5}$$

The implementation relations for interaction obligations are defined by:

$$(p,n) \mapsto_{r} (p',n') \overset{\mathsf{def}}{=} n \subseteq n' \wedge p \subseteq p' \cup n' \tag{B.6}$$

$$(p,n) \mapsto_{nr} (p',n') \overset{\mathsf{def}}{=} (p,n) \mapsto_{r} (p',n') \wedge p' \subseteq p \tag{B.7}$$

Note that the implementation relation $\mapsto_{nr}$ relaxes the requirement $p \cup n = p' \cup n'$ from its corresponding refinement relation. The reason is that, unlike the p-obligations of $[\![\ d\ ]\!]$, the p-obligations of $\langle I \rangle_d$ will always contain all traces in $\mathcal{H}^d$.

A system I implements a sequence diagram d if every p-obligation in $[\![\ d\ ]\!]$ where 0 is not an acceptable probability is implemented by at least one p-obligation in $\langle I \rangle_d$:

$$[\![\ d\ ]\!] \mapsto_{x} \langle I \rangle_d \overset{\mathsf{def}}{=} \forall po \in [\![\ d\ ]\!] : 0 \notin \pi_2.po \Rightarrow \exists po' \in \langle I \rangle_d : po \mapsto_{y} po' \tag{B.8}$$

where $(x,y) \in \{(pg,pr),(png,pnr)\}$.

Appendix C. Proofs

In this section we present proofs for selected key results. The remaining proofs are similar. Full proofs of Theorem 1 can be found in [46] (refuse, veto, par, seq, alt), [48] (tc) and [14] (loop).[5] The proof of Theorem 2 can be found below, while full proofs of Theorem 3 can be found in [48].

Appendix C.1. Monotonicity of $\rightsquigarrow_{pg}$ with respect to seq.

PROOF. To prove monotonicity of $\rightsquigarrow_{pg}$ with respect to seq, we assume

$$[\![\ d_1\ ]\!] \rightsquigarrow_{pg} [\![\ d_1'\ ]\!] \wedge [\![\ d_2\ ]\!] \rightsquigarrow_{pg} [\![\ d_2'\ ]\!] \tag{C.1}$$

We need to prove

$$[\![\ d_1 \ \mathsf{seq}\ d_2\ ]\!] \rightsquigarrow_{pg} [\![\ d_1' \ \mathsf{seq}\ d_2'\ ]\!]$$

i.e. that

$$\forall po \in [\![\ d_1 \ \mathsf{seq}\ d_2\ ]\!] : 0 \notin \pi_2.po \Rightarrow \exists po' \in [\![\ d_1' \ \mathsf{seq}\ d_2'\ ]\!] : po \rightsquigarrow_{pr} po' \tag{C.2}$$

[5]In [14] we prove monotonicity of loop with respect to $\rightsquigarrow_{pg}$. The proof can be straightforwardly adapted to $\rightsquigarrow_{png}$.

Assume that

$$po \in [\![\ d_1 \ \mathsf{seq}\ d_2\ ]\!] \wedge 0 \notin \pi_2.po \tag{C.3}$$

Then, by Definition (A.20) and Definition (2), there exists $po_1 \in [\![\ d_1\ ]\!], po_2 \in [\![\ d_2\ ]\!]$ such that

$$po = po_1 \precsim po_2 \wedge 0 \notin \pi_2.po_1 \wedge 0 \notin \pi_2.po_2 \tag{C.4}$$

Hence, by assumption (C.1), there exists $po'_1 \in [\![\ d'_1\ ]\!], po'_2 \in [\![\ d'_2\ ]\!]$ such that

$$po_1 \rightsquigarrow_{pr} po'_1 \wedge po_2 \rightsquigarrow_{pr} po'_2 \tag{C.5}$$

A straightforward application of Definition (A.11) then gives

$$po_1 \precsim po_2 \rightsquigarrow_{pr} po'_1 \precsim po'_2 \tag{C.6}$$

From $po'_1 \in [\![\ d'_1\ ]\!]$ and $po'_2 \in [\![\ d'_2\ ]\!]$ we get

$$po'_1 \precsim po'_2 \in [\![\ d'_1 \ \mathsf{seq}\ d'_2\ ]\!] \tag{C.7}$$

This means that $po'_1 \precsim po'_2$ is the po' we are looking for.

Appendix C.2. Conditional monotonicity of $\rightsquigarrow_{pg}$ with respect to palt

Proof. To prove monotonicity of $\rightsquigarrow_{pg}$ with respect to palt, we assume

$$\forall i \leq n : [\![\ d_i\ ]\!] \rightsquigarrow_{pg} [\![\ d'_i\ ]\!] \wedge Q'_i \subseteq Q_i \tag{C.8}$$
$$\forall i \leq n : \oplus[\![\ d_i\ ]\!] \rightsquigarrow_{r} \oplus[\![\ d'_i\ ]\!] \tag{C.9}$$

We need to prove that this implies[6]

$$\mathsf{palt}(d_1;Q_1,\ldots,d_n;Q_n) \rightsquigarrow_{pg} \mathsf{palt}(d'_1;Q'_1,\ldots,d'_n;Q'_n)$$

i.e. that

$$\begin{aligned} &\forall po \in [\![\ \mathsf{palt}(d_1;Q_1,\ldots,d_n;Q_n)\ ]\!] : \\ &\quad 0 \notin \pi_2.po \Rightarrow \exists po' \in [\![\ \mathsf{palt}(d'_1;Q'_1,\ldots,d'_n;Q'_n)\ ]\!] : po \rightsquigarrow_{pr} po' \end{aligned} \tag{C.10}$$

Let $po \in [\![\ \mathsf{palt}(d_1;Q_1,\ldots,d_n;Q_n)\ ]\!]$ such that $0 \notin \pi_2.po$. We need to show that

$$\exists po' \in [\![\ \mathsf{palt}(d'_1;Q'_1,\ldots,d'_n;Q'_n)\ ]\!] : po \rightsquigarrow_{pr} po' \tag{C.11}$$

As $po \in [\![\ \mathsf{palt}(d_1;Q_1,\ldots,d_n;Q_n)\ ]\!]$, we know that po comes from either line (a) or (b) of Definition (10). We first look at the first case, i.e. the case where $po = \bar{\oplus}(\{po_i\}_{i\in N})$ for some $N \subseteq \{1,\ldots,n\}$ such that $N \neq \emptyset \wedge \forall i \in N : po_i \in [\![\ d_i;Q_i\ ]\!]$. Let $N' = N \setminus \{i \mid 0 \in \pi_2.po_i\}$. Since $N' = \emptyset$ would imply that 0 was included in the probability set of every po_i such that $i \in N$, and hence $0 \in \pi_2.po$, we get

[6]Condition (C.9) is an extra condition that is required for the result to hold. Later we use this result to prove modularity of palt.

$N' \neq \emptyset$. For each $i \in N'$ choose $po'_i \in [\![d'_i;Q'_i]\!]$ such that $po_i \rightsquigarrow_{pr} po'_i$. The existence of such p-obligations is ensured by assumption (C.8), as $0 \notin \pi_2.po_i$ for all $i \in N'$. From $N' \subseteq N$ it follows (Lemma 29 in [46]) that

$$\pi_1.po \rightsquigarrow_r \oplus\{po_i\}_{i\in N'} \tag{C.12}$$

Now it can be shown by induction on the number of elements in N' that

$$\oplus\{po_i\}_{i\in N'} \rightsquigarrow_r \oplus\{po'_i\}_{i\in N'} \tag{C.13}$$

From (C.12) and (C.13) it follows that

$$\pi_1.po \rightsquigarrow_r \oplus\{po'_i\}_{i\in N'} \tag{C.14}$$

By the construction of N', which ensures that 0 is included in the probability set of every p-obligation po_j such that $j \in N \setminus N'$, together with Definition (9) it follows that

$$\sum_{i\in N'} Q_i \subseteq \sum_{i\in N} Q_i \tag{C.15}$$

From (C.14) and (C.15) we get

$$po \rightsquigarrow_{pr} \bar{\oplus}\{po'_i\}_{i\in N'} \tag{C.16}$$

From line (a) of Definition (10) it follows that $\bar{\oplus}(\{po'_i\}_{i\in N'}) \in [\![\mathsf{palt}(d'_1;Q'_1,\ldots,d'_n;Q'_n)]\!]$. This means that (C.11) can be fulfilled by letting $po' = \bar{\oplus}(\{po'_i\}_{i\in N'})$.

We now look at the case where po comes from line (b) of Definition (10), i.e. the case where $po = (\oplus \bigcup_{j=1}^{n} [\![d_j;Q_j]\!], \{1\} \cap \sum_{j=1}^{n} Q_j)$. Through a series of set theoretic steps it can be shown that the following holds for all sets of p-obligations S_1, S'_1, S_2, S'_2 (Lemma 6 in [14]):

$$\oplus S_1 \rightsquigarrow_r \oplus S'_1 \wedge \oplus S_2 \rightsquigarrow_r \oplus S'_2 \Rightarrow \oplus(S_1 \cup S_2) \rightsquigarrow_r \oplus(S'_1 \cup S'_2) \tag{C.17}$$

Now, from assumption (C.9) and (C.17) it can be shown by induction over n that

$$\oplus \bigcup_{i=1}^{n} [\![d_i;Q_i]\!] \rightsquigarrow_r \oplus \bigcup_{i=1}^{n} [\![d'_i;Q'_i]\!] \tag{C.18}$$

Furthermore, by the assumption that $Q'_i \subseteq Q_i$ for all i we have that

$$\{1\} \cap \sum_{i=1}^{n} Q'_i \subseteq \{1\} \cap \sum_{i=1}^{n} Q_i \tag{C.19}$$

Together, (C.18) and (C.19) mean that

$$(\oplus \bigcup_{i=1}^{n} [\![d_i;Q_i]\!], \{1\} \cap \sum_{i=1}^{n} Q_i) \rightsquigarrow_{pr} (\oplus \bigcup_{i=1}^{n} [\![d'_i;Q'_i]\!], \{1\} \cap \sum_{i=1}^{n} Q'_i) \tag{C.20}$$

From Definition (10) it is clear that

$$(\oplus \bigcup_{i=1}^{n} [\![\ d'_i;Q'_i\ ]\!], \{1\} \cap \sum_{i=1}^{n} Q'_i) \in [\![\ \mathsf{palt}(d'_1;Q'_1, \ldots, d'_n;Q'_n)\ ]\!] \tag{C.21}$$

Hence, we ensure that (C.11) is fulfilled by letting $po' = (\oplus \bigcup_{i=1}^{n} [\![\ d'_i;Q'_i\ ]\!], \{1\} \cap \sum_{i=1}^{n} Q'_i)$.

Appendix C.3. Modularity of palt *w.r.t.* $(\rightsquigarrow_{pg}, \mapsto_{pg})$

We now prove the modularity result (Theorem 2) which shows that the extra requirement (C.9) for monotonicity of refinement w.r.t. palt does not have significant practical consequences as explained in Section 4.3.1. The proof is general and applies to all implementation relations satisfying the criteria given in Section 4.3.1. It does not rely on the special features of the implementation model given as an example in Appendix B.

PROOF. Let

$$d \quad = \quad \mathsf{palt}(d_1;Q_1, \ldots, d_k;Q_k) \tag{C.22}$$

$$d' \quad = \quad \mathsf{palt}(d'_1;Q'_1, \ldots, d'_k;Q'_k) \tag{C.23}$$

Assume that

$$\forall j \leq k : [\![\ d_j\ ]\!] \rightsquigarrow_{pg} [\![\ d'_j\ ]\!] \wedge Q'_j \subseteq Q_j \tag{C.24}$$

$$\forall j \leq k : d_j \text{ is safe as defined in (A.27)} \tag{C.25}$$

$$d' \text{ is well-balanced} \tag{C.26}$$

$$d' \mapsto_{pg} I \wedge \forall j \leq k : d'_j \mapsto_{pg} I_j \tag{C.27}$$

where the implementation I is composed of the implementations I_j, i.e. $I = op_{imp}(I_1, \ldots, I_k)$ for some suitable composition operator op_{imp} for implementations that is trace preserving, meaning that the set of traces produced by I is the union of the traces produced by each I_j. We need to prove that

$$d \mapsto_{pg} I \tag{C.28}$$

If $[\![\ d\ ]\!] \rightsquigarrow_{pg} [\![\ d'\ ]\!]$ then (C.28) follows immediately from the first conjunct of assumption (C.27) together with preservation of $\mapsto_{pg}$ through abstraction. In the following we therefore assume

$$[\![\ d\ ]\!] \not\rightsquigarrow_{pg} [\![\ d'\ ]\!] \tag{C.29}$$

This means that there exists a p-obligation $((p_1, n_1), Q_1) \in [\![\ d\ ]\!]$ such that

$$\forall ((p', n'), Q') \in [\![\ d'\ ]\!] : ((p_1, n_1), Q_1) \not\rightsquigarrow_{pr} ((p', n'), Q') \tag{C.30}$$

As $d = \mathsf{palt}(d_1;Q_1, \ldots, d_k;Q_k)$, it is clear that the p-obligation $((p_1, n_1), Q_1)$ comes either from line (a) or line (b) of Definition (10). Assume it comes from line (a). This means that $((p_1, n_1), Q_1) = \bar{\oplus} M$, where M is a set of p-obligations obtained by selecting at most one p-obligation po_j from each operand $d_j;Q_j$ of the palt. Then we can obtain a set of p-obligations M' by selecting corresponding p-obligations po'_j from the corresponding operands of $[\![\ d'\ ]\!]$ such that $\forall j \leq k$: $po_j \rightsquigarrow_{pr} po'_j$. But this means that $\bar{\oplus} M' \in [\![\ d'\ ]\!]$ and $((p_1, n_1), Q_1) \rightsquigarrow_{pr} \bar{\oplus} M'$, which contradicts (C.30). Hence, $((p_1, n_1), Q_1)$ must come from line (b) of Definition (10), which means that

$$((p_1, n_1), Q_1) = (\oplus \bigcup_{j=1}^{k} [\![\ d_j;Q_j\ ]\!], \{1\} \cap \sum_{j=1}^{k} Q_j) \tag{C.31}$$

Since p-obligations with an empty probability set are not implementable, it follows from the first conjunct of assumption (C.27) that

$$\forall ((p, n), Q) \in [\![\ d'\ ]\!] : Q \neq \emptyset \tag{C.32}$$

From this together with assumption (C.24) it then follows that

$$\forall ((p, n), Q) \in [\![\ d\ ]\!] : Q \neq \emptyset \tag{C.33}$$

From this, (C.31) and the fact that $\oplus \bigcup_{j=1}^{k} [\![\ d_j;Q_j\ ]\!] = \oplus [\![\ d\ ]\!]$ it then follows that

$$((p_1, n_1), Q_1) = (\oplus [\![\ d\ ]\!], \{1\}) \tag{C.34}$$

From (C.32) and Definition (10) it follows that

$$(\oplus [\![\ d'\ ]\!], \{1\}) \in [\![\ d'\ ]\!] \tag{C.35}$$

Together with (C.30), this implies that

$$\oplus [\![\ d\ ]\!] \not\rightsquigarrow_r \oplus [\![\ d'\ ]\!] \tag{C.36}$$

This means that we have either $\pi_2. \oplus [\![\ d\ ]\!] \not\subseteq \pi_2. \oplus [\![\ d'\ ]\!]$ or $\pi_1. \oplus [\![\ d\ ]\!] \not\subseteq \pi_1. \oplus [\![\ d'\ ]\!] \cup \pi_2. \oplus [\![\ d'\ ]\!]$. From assumptions (C.24) (first conjunct) and (C.26) it follows that the latter alternative is not possible, as this would mean that there exists a trace that is included in all p-obligations in $[\![\ d\ ]\!]$, but not in all p-obligations in $[\![\ d'\ ]\!]$. As d' is well-balanced, this would imply that this trace is not included in any p-obligation in $[\![\ d'\ ]\!]$. This would, however, contradict the first conjunct of assumption (C.24), since a p-obligation where a trace t is included in either the positive or negative set cannot be refined by a p-obligation where t is inconclusive, which follows from (11). Hence, the first alternative must hold, i.e. $\pi_2. \oplus [\![\ d\ ]\!] \not\subseteq \pi_2. \oplus [\![\ d'\ ]\!]$. This means that there exists a trace t such that

$$t \in \pi_2. \oplus [\![\ d\ ]\!] \wedge t \notin \pi_2. \oplus [\![\ d'\ ]\!] \tag{C.37}$$

From the second conjunct of (C.37) it follows that there is at least one palt-operand d'_i of d' that contains a p-obligation whose negative set does not include

t. From the first conjunct it follows that the negative sets of all p-obligations of all palt operands of d contain t. This means that there exists an $i \leq k$ such that

$$t \in \pi_2.\oplus[\![\ d_i\ ]\!] \wedge t \notin \pi_2.\oplus[\![\ d'_i\ ]\!] \tag{C.38}$$

From (C.33) it follows that $\forall((p,n),Q) \in [\![\ d_i\ ]\!] : Q \neq \emptyset$, and hence that there exists a p-obligation $((p,n),\{1\})$ such that

$$((p,n),\{1\}) \in [\![\ d_i\ ]\!] \tag{C.39}$$

To see this, observe that if d_i is of the form $\mathsf{palt}(\ldots)$, then (C.39) follows from the second line of Definition (10), as the probability set of the resulting p-obligation is either $\emptyset$ or $\{1\}$. In all other cases, the composition operators ensure that if all operands contain at least one p-obligation whose probability set is $\{1\}$, then the composition also contains such a p-obligation, as composition of p-obligations involves either multiplication or (in the case of unary operators) preservation of probability sets (see definitions (A.10)-(A.17)). Therefore, (C.39) can be proved by induction over the syntactic structure of d_i, with the one-event diagram (Definition (A.18)) and the empty diagram (Definition (A.19)) as base cases.

Together, (C.39), (C.32) and (C.24) imply that there exists a p-obligation $((p',n'),\{1\})$ such that

$$((p',n'),\{1\}) \in [\![\ d'_i\ ]\!] \wedge (p,n) \rightsquigarrow_r (p',n') \tag{C.40}$$

From the first conjunct of (C.38) together with (C.39) we get

$$t \in n \tag{C.41}$$

Together, (C.41) and (C.40) imply that

$$t \in n' \tag{C.42}$$

From (C.40) and (C.42) we have that t is negative in a p-obligation in $[\![\ d'_i\ ]\!]$ with 1 as the only allowed probability. This means that in any implementation of d'_i, t can only be produced with probability 0. Now assume for contradiction that I_i is able to produce t. No execution that terminates can occur with probability 0, as such an execution can only result from a finite number of choices, where the probability of each selected alternative is greater than 0. Since any trace in the semantics of a probabilistic STAIRS specification by definition is either infinite or represents an execution that terminates, it follows that

$$\#t = \infty \tag{C.43}$$

From the first conjunct of (C.38) it follows that

$$t \notin \mathcal{H} \setminus \pi_2.\oplus[\![\ d_i\ ]\!] \tag{C.44}$$

From (C.25) it follows that d_i is safe. Together with (C.43), (C.44) and definition (A.27), this means that there exists $m \in \mathbb{N}$ such that

$$\forall t' \in \mathcal{H} \setminus \pi_2.\oplus[\![\ d_i\ ]\!] : t|_m \not\sqsubseteq t' \tag{C.45}$$

To see this, assume that (C.45) did not hold, i.e. that $\forall j \in \mathbb{N} : \exists t' \in \mathcal{H} \setminus \pi_2. \oplus [\![\ d_i \ ]\!] : t|_j \sqsubseteq t'$. As t is the least upper bound for the chain defined by $\forall j \in \mathbb{N} : c[j] = t|_j$, (C.25) would then imply that $t \in \mathcal{H} \setminus \pi_2. \oplus [\![\ d_i \ ]\!]$, which contradicts (C.44).

Let S be the set of all traces with $t|_m$ as a prefix, i.e. $S = \{t' \in \mathcal{H} \mid t|_m \sqsubseteq t'\}$. Note that since $t|_m$ is finite it follows that either I_i does not produce any traces in S at all, or the probability that I_i will produce a trace in S is greater than 0. From (C.45) it follows that

$$(\mathcal{H} \setminus \pi_2. \oplus [\![\ d_i \ ]\!]) \cap S = \emptyset \tag{C.46}$$

From (C.39) it follows that

$$\pi_2. \oplus [\![\ d_i \ ]\!] \subseteq n \tag{C.47}$$

Together with the second conjunct of (C.40), (C.47) implies that

$$\pi_2. \oplus [\![\ d_i \ ]\!] \subseteq n' \tag{C.48}$$

From (C.46) we get

$$S \subseteq \pi_2. \oplus [\![\ d_i \ ]\!] \tag{C.49}$$

Together, (C.49) and (C.48) imply that

$$S \subseteq n' \tag{C.50}$$

From the definition of S it follows that $t \in S$, which means that I_i is able to produce a trace in S, and hence that the probability that I_i produces a trace in S is greater than 0. But together with (C.50) this means that the probability of producing n' is greater than 0, which contradicts the first conjunct of (C.40). Hence, the assumption that I_i is able to produce t cannot hold.

From the left conjunct of (C.37) it follows that $t \in \pi_2. \oplus [\![\ d_j \ ]\!]$ for any $j \leq k$. Hence, what we have shown from (C.36) and onwards is essentially that for any trace t that breaks the condition for $\oplus [\![\ d \ ]\!] \rightsquigarrow_r \oplus [\![\ d \ ]\!]$ to hold there is no $j \leq k$ such that t is produced by I_j, even in the cases where $t \notin \pi_2. \oplus [\![\ d'_j \ ]\!]$. Hence, t is not produced by I. But this means that if I implements $(\oplus [\![\ d' \ ]\!], \{1\})$, it also implements $(\oplus [\![\ d \ ]\!], \{1\})$. Since it follows from (C.35) and the first conjunct of (C.27) that I implements $(\oplus [\![\ d' \ ]\!], \{1\})$, it then follows that I implements $(\oplus [\![\ d \ ]\!], \{1\})$. As this is the only p-obligation in $[\![\ d \ ]\!]$ not refined by any p-obligation in $[\![\ d' \ ]\!]$ (which is clear from the fact that (C.34) could be derived from (C.30)), this means that (C.28) must also hold.

Appendix C.4. Modularity of palt *w.r.t.* $(\rightsquigarrow_{png}, \mapsto_{png})$

The proof is similar to the proof in Appendix C.3, with $\rightsquigarrow_{pg}$ everywhere replaced by $\rightsquigarrow_{png}$, $\rightsquigarrow_r$ everywhere replaced by $\rightsquigarrow_{nr}$ and $\mapsto_{pg}$ everywhere replaced by $\mapsto_{png}$. Apart from this, the only difference is that there is an extra alternative $\pi_1. \oplus [\![\ d \ ]\!] \cup \pi_2. \oplus [\![\ d \ ]\!] \neq \pi_1. \oplus [\![\ d' \ ]\!] \cup \pi_2. \oplus [\![\ d' \ ]\!]$ that must be considered after step (C.36). However, from the assumptions that d' is well-balanced and that $\forall j \leq k : [\![\ d_j \ ]\!] \rightsquigarrow_{png} [\![\ d_j \ ]\!]$, it is clear that $\pi_1. \oplus [\![\ d \ ]\!] \cup \pi_2. \oplus [\![\ d \ ]\!] \neq \pi_1. \oplus [\![\ d' \ ]\!] \cup \pi_2. \oplus [\![\ d' \ ]\!]$ cannot hold. We may therefore continue the proof from step (C.37) in the same manner as in Appendix C.3.