Specification and Refinement of Soft Real-Time Requirements Using Sequence Diagrams

Atle Refsdal[1], Knut Eilif Husa[1,2], and Ketil Stølen[1,3]

[1] Department of Informatics, University of Oslo, Norway
[2] Ericsson, Norway
[3] SINTEF ICT, Norway

Abstract. Soft real-time requirements are often related to communication in distributed systems. Therefore it is interesting to understand how UML sequence diagrams can be used to specify such requirements. We propose a way of integrating soft real-time requirements in sequence diagram specifications by adding probabilities to timed sequence diagrams. Our approach builds on timed STAIRS, which is an approach to the compositional and incremental development of sequence diagrams supporting specification of mandatory as well as potential behavior.

1 Introduction

A soft real-time requirement is a time requirement that needs to be met only by a certain percentage of the relevant behavior. A hard real-time requirement can be seen as a special case of a soft real-time requirement; it is a soft real-time requirement that needs to be met in 100% of the cases. When a delay depends on factors that are hard to measure, highly complex or outside our control, a soft real-time requirement is often more appropriate than a hard constraint.

Time constraints are often related to some kind of communication scenario. Therefore it is important to be able to express soft real-time constraints in sequence diagrams. Sequence diagrams show how a task is performed by sending messages between lifelines.

In this paper we enable specification of soft real-time constraints with sequence diagrams by extending STAIRS presented in [HS03], [HHRS05a] and [HHRS05b] with the possibility of assigning probabilities. The probabilities are added independently from the time constraints, so our approach supports probabilistic specifications in general.

The rest of this paper is organized as follows: Section 2 introduces a specification to illustrate aspects of probabilistic STAIRS throughout the paper. Section 3 defines events, traces and some basic operators. Timed STAIRS is introduced in section 4, while section 5 discusses the relation between mandatory choice and probabilities. Probabilistic STAIRS is introduced in section 6, and section 7 shows how this enables the addition of a soft real-time requirement to the example specification. In section 8 the refinement relation is defined. Section 9 demonstrates refinement of the example specification. We discuss some related work in section 10 before concluding in section 11.

P. Pettersson and W. Yi (Eds.): FORMATS 2005, LNCS 3829, pp. 32–48, 2005.

2 The Automatic Teller Machine Example

We use as example a scenario where a customer withdraws money from an automatic teller machine (atm). This section gives a brief and informal explanation of the example. Figure 1 shows the first version of the specification. It serves two purposes. Firstly, it introduces the basic UML sequence diagram notation. Secondly, it allows us to characterize the need for more expressiveness. We come back to this example in later sections to illustrate our approach. Since our main concern is demonstration of real-time specifications we have omitted some details that would belong in a real-life scenario, such as the entering of a PIN code. The scenario describes the case where the transaction succeeds.

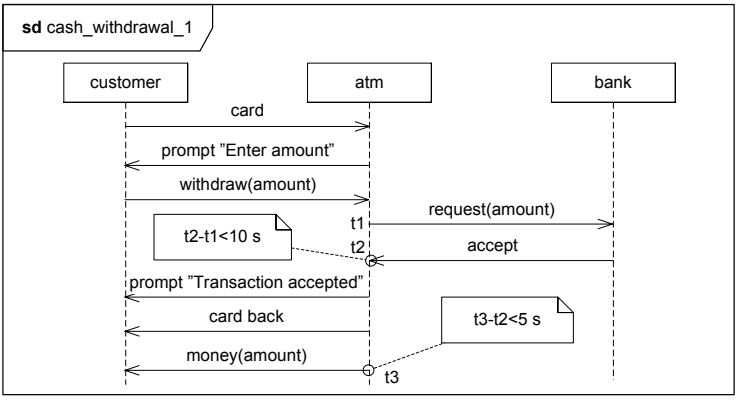

Fig. 1. A cash withdrawal scenario

It is an interaction between three *lifelines*: the customer, the atm and the bank. Lifelines represent the entities taking part in the interaction. The intuition behind the specification is the following: First the customer inserts her/his card, and the atm displays the text "Enter amount". The customer then enters the desired amount, and the atm sends a request to the bank asking whether the transaction is acceptable. A hard real-time requirement has been placed on the reply from the bank, stating that it should take no more than 10 seconds from the atm sends its request to the reply is received.[1] After the atm receives a positive reply from the bank, it displays the text "Transaction accepted", returns the card, and finally delivers the desired amount of money. A second hard real-time requirement has been put on the delivery of money stating that the delay from the atm receives a positive reply from the bank to the money is delivered should be less than five seconds.

UML sequence diagrams describe *traces* representing execution histories, and categorize traces as positive (valid) or negative (invalid). Positive traces represent

[1] We have chosen to use a different notation for real-time requirements than in UML 2.0, since we find our notation more suitable when the requirement crosses an operator boundary, as will happen in later specifications. Graphical (concrete) syntax is not a main issue in this paper.

acceptable executions, while negative traces represent unacceptable executions. All other traces are inconclusive, meaning that the specification does not say whether they are acceptable [OMG04–p. 526]. According to the specification in Figure 1, the positive traces are those where 1) messages are sent in the order shown in the diagram and 2) both real-time requirements are fulfilled. The negative traces are those that fulfill 1) but not 2).

The delay from the request is sent from the atm to a reply is received may depend on several complex factors, so we might want to replace the hard real-time requirement with a soft real-time requirement. Timed STAIRS gives a formal semantics to (a subset of) UML sequence diagrams with hard real-time requirements. Specifying soft real-time constraints, however, is not possible. Enabling the specification of soft real-time requirements within the framework of timed STAIRS is the aim of this paper.

3 Events, Traces and Basic Operators

In this section we define the notions of events and traces. We also introduce a number of helpful operators. Most of the definitions and explanations in this section are taken from [HHRS05a].

For any set A, A^ω denotes the set of finite as well as infinite sequences of elements of A. $\mathbb{N}$ denotes the set of natural numbers, while $\mathbb{N}_0$ denotes the set of natural numbers including 0. We define the functions

$$\#\_ \in A^\omega \to \mathbb{N}_0 \cup \{\infty\}, \quad \_[\_] \in A^\omega \times \mathbb{N} \to A, \quad \_ \frown \_ \in A^\omega \times A^\omega \to A^\omega,$$
$$\_|\_ \in A^\omega \times \mathbb{N}_0 \to A^\omega, \quad \_ \circledS \_ \in \mathbb{P}(A) \times A^\omega \to A^\omega$$

to yield the length of a sequence, the nth element of a sequence, the concatenation of two sequences, truncation of a sequence and the filtering of a sequence. Hence, $\#a$ yields the number of elements in a, and $a[n]$ yields a's nth element if $n \leq \#a$. To concatenate two sequences means to glue them together. Therefore, $a_1 \frown a_2$ denotes a sequence of length $\#a_1 + \#a_2$ that equals a_1 if a_1 is infinite, and is prefixed by a_1 and suffixed by a_2 otherwise. For any $0 \leq i \leq \#a$, $a|_i$ denotes the prefix of a of length i. By $B \circledS a$ we denote the sequence obtained from the sequence a by removing all elements in a that are not in the set B.

We also need filtering of pairs of sequences. The filtering function

$$\_ \circledT \_ \in \mathbb{P}(A \times B) \times (A^\omega \times B^\omega) \to A^\omega \times B^\omega$$

can be understood as a generalization of $\circledS$. For any set of pairs of elements P and pairs of sequences t, $P \circledT t$ denotes the pair of sequences obtained from t by

- truncating the longest sequence in t at the length of the shortest sequence in t if the two sequences are not of equal length;
- for each $j \in [1...k]$, where k is the length of the shortest sequence in t, selecting or deleting the two elements at index j in the two sequences, depending on whether the pair of these elements is in the set P.

For example, we have that

$$\{(1,f),(1,g)\} \circledcirc (\langle 1,1,2,1,2\rangle, \langle f,f,f,g,g\rangle) = (\langle 1,1,1\rangle, \langle f,f,g\rangle)$$

For a formal definition of $\circledcirc$, see [BS01].

π_i is a projection operator returning element number i of a tuple.

3.1 Events

A message is a triple (s, re, tr) of a signal s, a receiver re and a transmitter tr. $\mathcal{M}$ denotes the set of all messages. The receiver and transmitter are lifelines. $\mathcal{L}$ denotes the set of all lifelines.

An event may be of two kinds; a transmission event tagged by "!" or a reception event tagged by "?".[2] Every event occurring in a sequence diagram has a timestamp tag. $\mathcal{T}$ denotes the set of timestamp tags. We use logical formulas with timestamp tags as free variables to impose constraints on the timing of events. By $\mathbb{F}(v)$ we denote the set of logical formulas whose free variables are contained in the set of timestamp tags v.

An event is a triple $(k, m, t) \in \{!, ?\} \times \mathcal{M} \times \mathcal{T}$ of a kind, a message and a timestamp tag. $\mathcal{E}$ denotes the set of all events. We define the functions

$$k.\_ \in \mathcal{E} \to \{?, !\},\ m.\_ \in \mathcal{E} \to \mathcal{M},\ t.\_ \in \mathcal{E} \to \mathcal{T},\ tr.\_ \in \mathcal{E} \to \mathcal{L},\ re.\_ \in \mathcal{E} \to \mathcal{L}$$

to yield the kind, message, timestamp tag, transmitter and receiver of an event, respectively. Since we are primarily interested in communication scenarios, we do not give a semantic interpretation to events, except that the timestamp tag is assigned a timestamp in form of a real number. $\mathbb{R}$ denotes the set of timestamps. The set $[\![\ \mathcal{E}\ ]\!]$ of event interpretations is therefore defined by

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

$t \mapsto r$ means that timestamp r is assigned to timestamp tag t. We also define the function

$$r.\_ \in [\![\ \mathcal{E}\ ]\!] \to \mathbb{R}$$

to yield the timestamp of an event interpretation. In the following, we use "event" and "event interpretation" interchangeably.

3.2 Traces

A trace $h \in [\![\ \mathcal{E}\ ]\!]^\omega$ is a finite or infinite sequence of events. Traces represent executions of the system under specification, and must satisfy a number of well-formedness conditions. Firstly, we require the events of h to be ordered by time:

$$\forall\, i, j \in [1..\#h] : i < j \Rightarrow r.h[i] \leq r.h[j] \tag{2}$$

[2] Note that in timed STAIRS [HHRS05a] "?" represents consumption. We have chosen to use "?" for reception since we do not concider consumption events in this paper.

Note that two events may occur at the same time.

Secondly, we allow the same event to occur only once in the same trace:

$$\forall\, i, j \in [1..\#h] : i \neq j \Rightarrow h[i] \neq h[j] \tag{3}$$

Thirdly, 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, and for any possible timestamp t there must exist an l-event in h whose timestamp is greater than t:

$$\forall\, l \in \mathcal{L} : (\#e.l \circledS h = \infty \Rightarrow \forall\, t \in \mathbb{R} : \exists\, i \in \mathbb{N} : r.(e.l \circledS h)[i] > t) \tag{4}$$

where $e.l$ denotes the set of events that may take place on the lifeline l. Formally:

$$e.l \stackrel{\text{def}}{=} \{e \in [\![\ \mathcal{E}\ ]\!] \mid (k.e =! \wedge tr.e = l) \vee (k.e =? \wedge re.e = l)\} \tag{5}$$

We also require that for any single message, transmission happens before reception. But we need to take into account that the transmitter or receiver of a certain message might not be included in the sequence diagram. Thus we get the following well-formedness requirement on traces, stating that if at any point in the trace we have a transmission event, up to that point we must have had at least as many transmissions as receptions of that particular message:

$$\forall\, i \in [1..\#h] : k.h[i] =! \Rightarrow \tag{6}$$

$$\#(\{!\} \times \{m.h[i]\} \times U) \circledS h|_i > \#(\{?\} \times \{m.h[i]\} \times U) \circledS h|_i$$

where $U \stackrel{\text{def}}{=} \{t \mapsto r \mid t \in \mathcal{T} \wedge r \in \mathbb{R}\}$.

$\mathcal{H}$ denotes the set of well-formed traces. Traces are written as a sequence of events enclosed by $\langle\rangle$, for example $\langle e_1, e_2, e_3\rangle$.

4 Syntax and Semantics for Timed STAIRS

In the following we explain how a timed sequence diagram can be represented by a *specification pair* (p, n) where p is a set of positive traces and n is a set of negative traces. (This is a simplification of timed STAIRS, where a sequence diagram is represented by a set of specification pairs.) $\mathcal{O}$ denotes the set of specification pairs. A specification pair (p, n) is contradictory if $p \cap n \neq \varnothing$. $[\![\ d\ ]\!]$ denotes the specification pair representing sequence diagram d.

4.1 Textual Syntax for Timed Sequence Diagrams

The set of syntactically correct sequence diagrams, $\mathcal{D}$, is defined inductively as the least set such that: [3]

[3] Timed STAIRS [HHRS05a] also include the operators loop, assert and xalt. We have omitted these operators to save space. There is also a formal requirement stating that if we have a message in the diagram and both the transmitter and the receiver lifelines of that message are present in the diagram, then both the transmit and the receive event of that message must be present in the diagram as well. This requirement is also omitted here to save space.

- $\mathcal{E} \subset \mathcal{D}$
- $d \in \mathcal{D} \Rightarrow \mathsf{neg}\ d \in \mathcal{D}$
- $d_1, d_2 \in \mathcal{D} \Rightarrow d_1\ \mathsf{par}\ d_2 \in \mathcal{D} \wedge d_1\ \mathsf{seq}\ d_2 \in \mathcal{D} \wedge d_1\ \mathsf{alt}\ d_2 \in \mathcal{D}$
- $d \in \mathcal{D} \wedge C \in \mathbb{F}(tt.d) \Rightarrow d\ \mathsf{tc}\ C \in \mathcal{D}$

where $tt.d$ yields the set of timestamp tags occurring in d. The base case implies that any event is a sequence diagram. Any other sequence diagram is constructed from the basic ones through the application of operations for negation, potential choice (alternative), weak sequencing, parallel execution and time constraint.

4.2 Denotational Semantics for Timed STAIRS

Event. The semantics of an event is the specification pair whose positive set consists of infinitely many unary positive traces – one for each possible assignment of a timestamp to its timestamp tag. The negative set is empty.

$$[\![\ (k, m, t)\ ]\!] \stackrel{\mathsf{def}}{=} (\{\langle (k, m, t \mapsto r) \rangle \mid r \in \mathbb{R}\}, \varnothing) \quad \text{if } (k, m, t) \in \mathcal{E} \tag{7}$$

Negation. Undesired behavior is defined by the use of the neg construct. To negate a specification means to move every positive trace to the negative set. Negative traces remain negative. The empty trace is defined as positive to enable positive traces in a composition. Negation of a specification is defined by

$$[\![\ \mathsf{neg}\ d\ ]\!] \stackrel{\mathsf{def}}{=} \neg\ [\![\ d\ ]\!] \tag{8}$$

where

$$\neg\ (p, n) \stackrel{\mathsf{def}}{=} (\{\langle\rangle\}, n \cup p) \tag{9}$$

Parallel Execution. The operator for parallel execution is represented semantically by $\parallel$. Ignoring for the time being the sets of negative traces, a parallel execution defines the set of traces we get by merging one trace from one (positive) set with one trace from the other (positive) set. Informally, for sets of traces s_1 and s_2, $s_1 \parallel s_2$ is the set of all traces such that:

- all events from one trace in s_1 and one trace in s_2 are included (and no other events),
- the ordering of events from each of the traces is preserved.

Formally:

$$\begin{aligned} s_1 \parallel s_2 \quad \stackrel{\mathsf{def}}{=} \quad & \{h \in \mathcal{H} \mid \exists\, or \in \{1, 2\}^\infty : \\ & \pi_2((\{1\} \times [\![\ \mathcal{E}\ ]\!])\, ⓣ\, (or, h)) \in s_1 \wedge \\ & \pi_2((\{2\} \times [\![\ \mathcal{E}\ ]\!])\, ⓣ\, (or, h)) \in s_2\} \end{aligned} \tag{10}$$

In this definition we make use of an oracle, the infinite sequence or, to resolve the non-determinism in the interleaving. It determines the order in which events from traces in s_1 and s_2 are sequenced.

The semantics of parallel execution may then be defined as

$$[\![\ d_1\ \mathsf{par}\ d_2\ ]\!] \stackrel{\mathsf{def}}{=} [\![\ d_1\ ]\!] \parallel [\![\ d_2\ ]\!] \tag{11}$$

where

$$(p_1, n_1) \parallel (p_2, n_2) \stackrel{\mathsf{def}}{=} (p_1 \parallel p_2, (n_1 \parallel (p_2 \cup n_2)) \cup (p_1 \parallel n_2)) \tag{12}$$

Note that the merging of a negative trace with another (positive or negative) trace always results in a negative trace.

Weak Sequencing. Weak sequencing is the implicit composition mechanism combining constructs of a sequence diagram. The operator for weak sequencing is represented semantically by $\gtrsim$. We again temporarily ignore the sets of negative traces, and let s_1 and s_2 be trace sets. Since lifelines are independent, the constraint for the ordering of events applies to each lifeline; events that occur on different lifelines are interleaved. For $s_1 \gtrsim s_2$ we therefore have the constraint that events on one lifeline from one trace in s_1 should come before events from one trace in s_2 on the same lifeline:

$$\begin{aligned} s_1 \gtrsim s_2 \quad \stackrel{\mathsf{def}}{=} \quad & \{h \in \mathcal{H} \mid \exists\, h_1 \in s_1, h_2 \in s_2 : \\ & \forall\, l \in L : e.l \circledS h = e.l \circledS h_1 \frown e.l \circledS h_2\} \end{aligned} \tag{13}$$

The semantics of weak sequencing may then be defined as

$$[\![\ d_1\ \mathsf{seq}\ d_2\ ]\!] \stackrel{\mathsf{def}}{=} [\![\ d_1\ ]\!] \gtrsim [\![\ d_2\ ]\!] \tag{14}$$

where

$$(p_1, n_1) \gtrsim (p_2, n_2) \stackrel{\mathsf{def}}{=} (p_1 \gtrsim p_2, (n_1 \gtrsim (p_2 \cup n_2)) \cup (p_1 \gtrsim n_2)) \tag{15}$$

Weak sequencing involving at least one negative trace results in a negative trace.

Time Constraint. Time requirements 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

$$s \wr C \stackrel{\mathsf{def}}{=} \{h \in s \mid h \models C\} \tag{16}$$

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

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

then $h \models C$ if $r_3 < r_1 + 5$.

To apply a time requirement to a specification means to define failure to meet the requirement as negative behavior. The positive traces of the operand that do not fulfill the requirement become negative. The semantics of a time constraint is defined as

$$[\![\ d\ \mathsf{tc}\ C\ ]\!] \stackrel{\mathsf{def}}{=} [\![\ d\ ]\!] \wr C \tag{17}$$

where

$$(p, n) \wr C \stackrel{\text{def}}{=} (p \wr C, n \cup (p \wr \neg C)) \tag{18}$$

Potential Choice. The alt construct is used to express underspecification by grouping together traces that from the specifier's point of view serve the same purpose. This means that they are seen as equally desirable (for positive traces) or undesirable (for negative traces). For two trace sets where both are positive or both are negative, this can be represented semantically simply by taking the union of the sets. Hence, potential choice corresponds to the pairwise union of the positive sets and the negative sets. Formally, the semantics of the alt is defined by

$$[\![d_1 \text{ alt } d_2]\!] \stackrel{\text{def}}{=} [\![d_1]\!] \uplus [\![d_2]\!] \tag{19}$$

where

$$(p_1, n_1) \uplus (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \cup p_2, n_1 \cup n_2) \tag{20}$$

5 Mandatory Choice and Probabilities

In STAIRS the alt operator as formally defined above enables underspecification, what we also refer to as potential choice. Underspecification means to leave some freedom of choice to the developers that will eventually implement (or further refine) the specification. This is for example useful when different design alternatives fulfill a function equally well from the specifier's point of view.

STAIRS supports also the specification of mandatory choice. For this purpose the STAIRS specific xalt operator is used. Mandatory choice means that all alternatives must be possible. It is often needed within security, for example in relation to information flow [Ros95]. When specifying a password generator, for instance, it is vital that all alternatives remain possible in the final implementation – otherwise in the extreme case we might end up with an implementation that always generates the same password.

Mandatory choice is also useful for other purposes. Sometimes non-determinism is employed to model the behavior of the environment of the system under specification. The mandatory choice operator is then used to represent alternative inputs from the environment that the designer has considered. If some of these alternatives are removed from the final specification, the implementation will not be able to handle the relevant input as intended.

Sometimes an application is non-deterministic by nature, for example in games. If we want to specify a dice, we obviously need to ensure that all alternatives, one through six, are possible outcomes in the implementation.

In probabilistic STAIRS we generalize the xalt operator into an operator for the specification of probabilities called palt. We may then also specify with what probability the different alternatives should occur. In the dice example, the probability of every outcome should be exactly $\frac{1}{6}$. Of course, if an alternative has an exact probability greater than zero, then this alternative must be a possible outcome of a valid implementation. For this reason, probabilistic choice can be

viewed as a special case of mandatory choice. This view is consistent with the one presented in [MM99].

If an alternative is assigned a *set* of acceptable probabilities, then this set represents underspecification. Such underspecification is usually present in soft real-time requirements. A specification might say that the probability of a certain delay being less than 10 seconds should be 0.8 or more. This amounts to saying that the set of acceptable probabilities is $[0.8, .., 1.0]$. According to this specification, an implementation that gives a probability of 0.9 is certainly valid; the developer only needs to achieve one of the acceptable probabilities.

6 Syntax and Semantics for Probabilistic STAIRS

In the following we explain how a probabilistic sequence diagram can be represented by a multiset of *probability obligations* (also called *p-obligations*). A p-obligation $((p, n), Q)$ consists of a specification pair (p, n) and a set of probabilities Q, with the following interpretation: The traces implementing (p, n) should occur with a probability greater than or equal to a probability in Q. Only traces in $\mathcal{H} \setminus n$ are allowed to implement (p, n). The probability for these traces may be greater than the values in Q only if some or all of the traces are also positive or inconclusive according to some other p-obligation. We use a multiset instead of just a set because multiple occurrences of the same p-obligation (which may result from the composition operators) means that the specification pair should occur with a probability greater than or equal to the sum of probabilities from each instance of the p-obligation. For example, if the p-obligation $((p, n), [0.3, ..., 0.4])$ occurs twice in a specification, then this means that the traces implementing (p, n) should occur with a probability greater than or equal to a value in $[0.6, ..., 0.8]$. $\mathcal{P}$ denotes the set of p-obligations. In probabilistic STAIRS we may have underspecification with respect to traces and with respect to probabilities. Underspecification with respect to traces is captured by the fact that we may choose among the non-negative traces within a specification pair. Underspecification with respect to probabilities is modeled by the possibility of selecting among the probabilities within a p-obligation.

6.1 Textual Syntax for Probabilistic Sequence Diagrams

The set of syntactically correct sequence diagrams $\mathcal{D}$ is defined simply by adding the following case to the inductive definition in 4.1:

- $d_1, d_2 \in \mathcal{D} \wedge Q_1, Q_2 \subseteq [0...1] \Rightarrow d_1; Q_1 \ \mathsf{palt} \ d_2; Q_2 \in \mathcal{D}$

6.2 Denotational Semantics for Probabilistic STAIRS

Event. Probabilities can be assigned only by the use of the palt. The traces specified by a sequence diagram without occurrences of palt must occur with probability 1 in their relevant context. Therefore the set of probabilities associated with an event is $\{1\}$.

$$[\![\ (k,m,t)\ ]\!] \stackrel{\text{def}}{=} \{((\{\langle (k,m,t \mapsto r)\rangle \mid r \in \mathbb{R}\},\varnothing),\{1\})\} \quad \text{if } (k,m,t) \in \mathcal{E} \tag{21}$$

Negation and Time Constraint. Negation and time constraint are not affected by probabilities. They are defined by

$$[\![\ \mathsf{neg}\ d\ ]\!] \stackrel{\text{def}}{=} \{(\neg\ o, Q) \mid (o,Q) \in [\![\ d\ ]\!]\} \tag{22}$$

$$[\![\ d\ \mathsf{tc}\ C\ ]\!] \stackrel{\text{def}}{=} \{(o \wr C, Q) \mid (o,Q) \in [\![\ d\ ]\!]\} \tag{23}$$

Parallel Execution and Weak Sequencing. When executing two specifications in parallel or sequentially, we get the mulitset of p-obligations obtained from choosing one p-obligation from the first and one p-obligation from the second and composing them in parallel or sequentially. Choosing the two p-obligations to be composed is seen as two independent probabilistic choices; therefore the sets of probabilities are multiplied. Formally, parallel execution and weak sequencing is defined by

$$[\![\ d_1\ \mathsf{par}\ d_2\ ]\!] \stackrel{\text{def}}{=} \{(o_1 \parallel o_2, Q_1 * Q_2) \mid (o_1,Q_1) \in [\![\ d_1\ ]\!] \wedge (o_2,Q_2) \in [\![\ d_2\ ]\!]\} \tag{24}$$

$$[\![\ d_1\ \mathsf{seq}\ d_2\ ]\!] \stackrel{\text{def}}{=} \{(o_1 \gtrsim o_2, Q_1 * Q_2) \mid (o_1,Q_1) \in [\![\ d_1\ ]\!] \wedge (o_2,Q_2) \in [\![\ d_2\ ]\!]\} \tag{25}$$

where multiplication of probability sets is defined by

$$Q_1 * Q_2 \stackrel{\text{def}}{=} \{q_1 * q_2 \mid q_1 \in Q_1 \wedge q_2 \in Q_2\} \tag{26}$$

Potential Choice. The alt construct captures underspecification with respect to traces (and not with respect to probabilities). When combining two p-obligations the probabilities that are not in both probability sets are removed. Otherwise, we might realize the composed specification with traces from only the first operand with a probability allowed only by the second operand.

$$[\![\ d_1\ \mathsf{alt}\ d_2\ ]\!] \stackrel{\text{def}}{=} \{(o_1 \uplus o_2, Q_1 \cap Q_2) \mid (o_1,Q_1) \in [\![\ d_1\ ]\!] \wedge (o_2,Q_2) \in [\![\ d_2\ ]\!]\} \tag{27}$$

Probabilistic Choice. The palt construct expresses probabilistic choice (and therefore mandatory choice). Before defining the semantics of the palt we introduce the notion of probability decoration. Probability decoration is used to assign the probabilities associated with the operands of a palt. It is defined by

$$[\![\ d;Q'\ ]\!] \stackrel{\text{def}}{=} \{(o, Q * Q') \mid (o,Q) \in [\![\ d\ ]\!]\} \tag{28}$$

The palt operator is meant to describe the probabilistic choice between two alternative operands whose joint probability should add up to one. Formally, the palt is defined by

$$[\![\ d_1\ \mathsf{palt}\ d_2\ ]\!] \stackrel{\text{def}}{=} [\![\ d_1\ ]\!] \cup [\![\ d_2\ ]\!] \cup \{(\oplus([\![\ d_1\ ]\!] \cup [\![\ d_2\ ]\!]),\{1\})\} \tag{29}$$

Note that the syntactic restrictions ensure that d_1 and d_2 are of the form $d;Q$ (see section 6.1). The p-obligation in the multiset to the right in 29 requires the

probabilities of the two operands to add up to one. $\oplus$ characterizes the traces allowed by the two operands together: A trace t is positive if it is positive according to at least one p-obligation and not inconclusive according to any; t is negative only if it is negative according to all p-obligations; traces that are inconclusive according to at least one p-obligation remain inconclusive. Formally, the operator $\oplus$ for combining the specification pairs of a multiset S of p-obligations into a single specification pair is therefore defined by

$$\oplus S \stackrel{\text{def}}{=} ((\bigcup_{((p,n),Q)\in S} p) \cap (\bigcap_{((p,n),Q)\in S} p \cup n), \bigcap_{((p,n),Q)\in S} n) \tag{30}$$

7 Adding a Soft Real-Time Requirement to the Atm

We now replace the first hard real-time requirement in the atm example with a soft real-time requirement. Consider the sequence diagram in Figure 2.

This specification is modeled semantically by three p-obligations, we call these po_1, po_2 and po_3. The result of choosing the first palt operand is modeled semantically by po_1. The positive traces of po_1 are only those in which it takes less than 10 seconds before the reply arrives from the bank and it takes less than five seconds from the reply arrives to the money is delivered. Traces where one or both of these constraints are not met are negative in po_1. The acceptable range of probability for this p-obligation is $[0.8, ..., 1]$.

The result of choosing the second palt operand is modeled semantically by po_2. The positive traces of po_2 are all traces where it takes 10 seconds or more before the reply arrives from the bank and it takes less than five seconds from

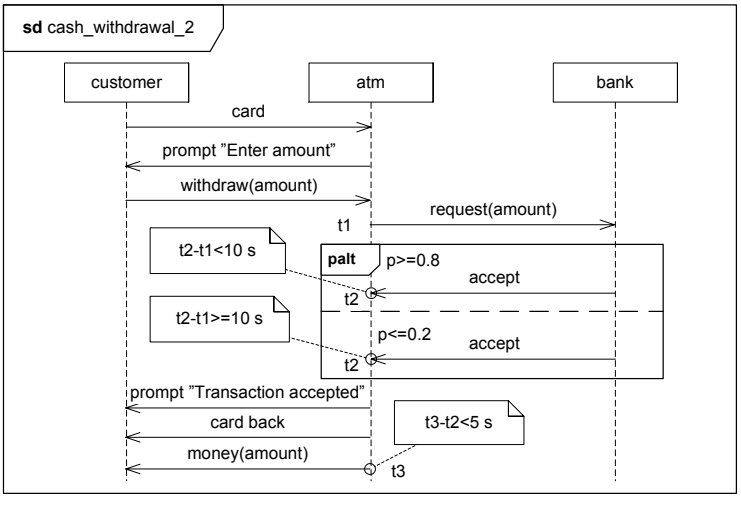

Fig. 2. Cash withdrawal with soft real-time constraint.

the reply arrives to the money is delivered. Traces where one or both of these constraints are not met are negative in po_2. The acceptable range of probability for this p-obligation is $[0, ..., 0.2]$.

The last p-obligation, po_3, models the combination of the two operands, which means that $po_3 = (\oplus\{po_1, po_2\}, \{1\})$. This means that the positive traces of po_3 are all traces where it takes less than five seconds to get money after the reply is received from the bank, regardless of how long it takes to get the reply. The negative traces are only those where it takes five seconds or more to get the money.

Traces where messages are not exchanged between the customer, the atm and the bank as described by Figure 2 (but ignoring the time requirements) are inconclusive according to po_1, po_2 and po_3.

8 Refinement

Refinement of a specification means to reduce underspecification by adding information so that the specification becomes closer to an implementation. Semantically, in our setting this can be done at the level of p-obligations or at the level of multisets of p-obligations. We first define refinement semantically for p-obligations. Then we lift this definition to specifications that are represented semantically by multisets of p-obligations.

8.1 Refinement of P-obligations

As in [HHRS05b], a specification pair is refined by moving positive traces to the set of negative traces or by moving traces from the set of inconclusive traces to either the positive or the negative set. STAIRS [HHRS05b] refers to the first option as narrowing and the second option as supplementing. As argued in [HHRS05b], narrowing reduces the set of positive traces to capture new design decisions or to match the problem more accurately. Supplementing categorizes (to this point) inconclusive behavior as either positive or negative recognizing that early descriptions normally lack completeness.

A p-obligation is refined by either refining its specification pair or reducing its set of probabilities. Formally, a p-obligation $((p', n'), Q')$ is a refinement of a p-obligation $((p, n), Q)$, written $((p, n), Q) \rightsquigarrow ((p', n'), Q')$, iff

$$n \subseteq n' \wedge p \subseteq p' \cup n' \wedge Q' \subseteq Q \tag{31}$$

8.2 Refinement of Specifications

All p-obligations at the given (more abstract) level represent a mandatory alternative. Therefore each p-obligation needs to be represented by a p-obligation also at the refined (more concrete) level. However, there are three additional considerations to take into account when defining the refinement relation.

Firstly, if a p-obligation has 0 as an acceptable probability, this means that it does not need to be implemented. Therefore p-obligations with 0 in the probability set need not be represented at the refined level.

Secondly, we need to ensure that any combination of p-obligations at the abstract level is refined at the concrete level. For example, if a p-obligation po occurs several times at the abstract level it is not enough just to ensure that there is a single p-obligation po' at the concrete level such that $po \rightsquigarrow po'$.

Thirdly, it should be possible to refine a p-obligation at the abstract level by the combination of a submultiset of p-obligations at the concrete level, as long as the combination of these p-obligations is a refinement of the abstract p-obligation. Suppose now that we have two specifications d_1 and d_2 such that $((\{t_1, t_2\}, \{t_3\}), [0.4, ..., 0.6]) \in [\![\ d_1\ ]\!]$ and such that there is no p-obligation $((p, n), Q)$ in $[\![\ d_2\ ]\!]$ such that $Q \subseteq [0.4, ..., 0.6]$. Then it should still be possible for d_2 to be a refinement of d_1, as long as there is a submultiset S of $[\![\ d_2\ ]\!]$ such that the combination of all p-obligations in S is a refinement of $((\{t_1, t_2\}, \{t_3\}), [0.4, ..., 0.6])$. This is the case for example if $[\![\ d_2\ ]\!]$ contains the p-obligations $((\{t_1\}, \{t_2, t_3\}), \{0.3\})$ and $((\{t_2\}, \{t_1, t_3\}), \{0.2\})$. Taken together, these two p-obligations are certainly a valid refinement of $((\{t_1, t_2\}, \{t_3\}), [0.4, ..., 0.6])$.

Each p-obligation represents a probabilistic choice. The probability for a combination of choices is the sum of probabilities for each choice. Let $\{Q_1, ..., Q_n\}$ be a multiset of probability sets. We then define

$$\sum_{i=1}^{n} Q_i \stackrel{\mathsf{def}}{=} \{\mathsf{min}(1, \sum_{i=1}^{n} q_i) \mid q_i \in Q_i\} \tag{32}$$

Note that the upper limit of probabilities is 1.

We are now ready to define refinement of specifications. Formally, a specification d' is a refinement of a specification d, written $d \rightsquigarrow d'$, iff

$$\forall\, S \subseteq [\![\ d\ ]\!] : 0 \notin \pi_2.\bar{\oplus} S \Rightarrow \exists\, S' \subseteq [\![\ d'\ ]\!] : \bar{\oplus} S \rightsquigarrow \bar{\oplus} S' \tag{33}$$

where the operator $\bar{\oplus}$ characterizes the combination of all p-obligations in a multiset S into a single pair $((p, n), Q)$. Formally, $\bar{\oplus}$ is defined by

$$\bar{\oplus} S \stackrel{\mathsf{def}}{=} (\oplus S, \sum_{po \in S} \pi_2.po) \tag{34}$$

9 Refining the Atm Specification

Figure 3 shows a refinement of the specification in Figure 2.

The change that has been made to "cash_withdrawal_2" is to impose an upper limit to the acceptable response time from the bank also in the second operand, stating that the reply should be received within 20 seconds. In addition we have narrowed the acceptable range of probability for both operands. It is now required that the reply from the bank should be received within 10 seconds in at least 90% of the cases, instead of just 80%.

The specification "cash_withdrawal_3" is modeled semantically by three p-obligations, we call these po'_1, po'_2 and po'_3. The p-obligation po'_1 represents

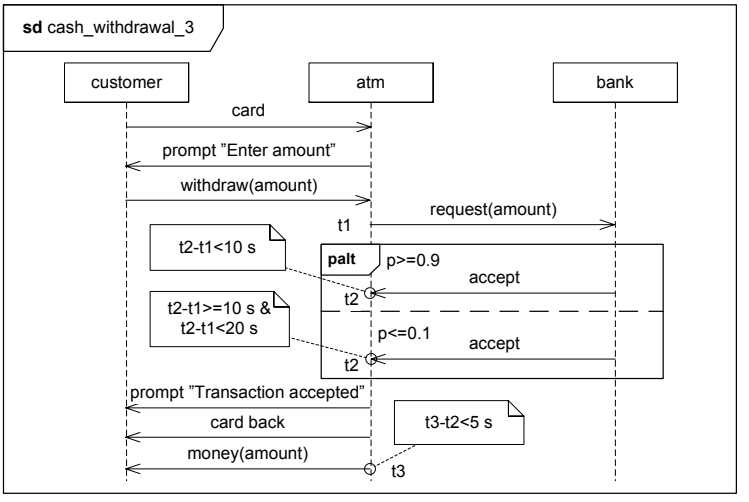

Fig. 3. A refinement of Figure 2

the result of choosing the first operand of the palt. The positive and negative traces of po_1' are the same as for po_1, while the set of acceptable probabilities for po_1' is $[0.9, ..., 1]$, which is a subset of the probability set of po_1. This means that $po_1 \rightsquigarrow po_1'$.

The result of choosing the second palt operand is modeled semantically by po_2'. The positive and negative traces of po_2' are the same as for po_2, except that traces where it takes more than 20 seconds to get a reply from the bank are positive in po_2 and negative in po_2'. Since the probability set of po_2', $[0, ..., 0.1]$, is a subset of the probability set of po_2, we get $po_2 \rightsquigarrow po_2'$.

The last p-obligation, po_3', models the combination of the two operands, which means that $po_3' = (\oplus\{po_1', po_2'\}, \{1\})$. According to po_3' the positive traces are all traces where it takes less than 20 seconds to get an answer from the bank and less than five seconds to get money after the reply is received from the bank. The negative traces are those where it takes 20 seconds or more to get a reply or five seconds or more to get the money. Since the probability sets of po_3 and po_3' are both $\{1\}$ and the only difference with respect to traces is that the traces where it takes 20 seconds or more to get a reply from the bank are positive in po_3 and negative in po_3', we get $po_3 \rightsquigarrow po_3'$.

The above shows that condition 33 is fulfilled for every singleton set that is a submultiset of the semantics of "cash_withdrawal_2". It is easy to verify that $\bar{\oplus}\{po_1, po_2\} \rightsquigarrow \bar{\oplus}\{po_1', po_2'\}$, $\bar{\oplus}\{po_2, po_3\} \rightsquigarrow \bar{\oplus}\{po_2', po_3'\}$, $\bar{\oplus}\{po_1, po_3\} \rightsquigarrow \bar{\oplus}\{po_1', po_3'\}$ and $\bar{\oplus}\{po_1, po_2, po_3\} \rightsquigarrow \bar{\oplus}\{po_1', po_2', po_3'\}$. This means that condition 33 is fulfilled, so the specification "cash_withdrawal_3" is a refinement of "cash_withdrawal_2".

We also have that the original specification "cash_withdrawal_1" with its hard real-time constraint is a refinement of "cash_withdrawal_2". To see this,

note that the specification "cash_withdrawal_1" is represented semantically by $\{(\pi_1.po_1, \{1\})\}$, and that $(\pi_1.po_1, \{1\})$ is a valid refinement of both po_1 and po_3. Since $0 \in \pi_2.po_2$, this shows that condition 33 is fulfilled for every singleton set that is a submultiset of the semantics of "cash_withdrawal_2". In addition, we see that $(\pi_1.po_1, \{1\})$ is a valid refinement of $\bar{\oplus}\{po_1, po_2\}$, $\bar{\oplus}\{po_2, po_3\}$ and $\bar{\oplus}\{po_1, po_2, po_3\}$. Condition 33 is therefore fulfilled. A similar argument shows that "cash_withdrawal_1" is also a refinement of "cash_withdrawal_3".

10 Related Work

[Seg95] uses probabilistic automata to address the problem of verification of randomized distributed algorithms. The analysis includes timed systems, so that real-time properties can be investigated in a probabilistic setting. [Jan03] introduces a stochastic extension to statecharts called StoCharts to allow the quantification of the time between events according to a stochastic distribution, and defines a formal semantics that can be analyzed by tools. [JL91] presents a formalism for specifying probabilistic transition systems where transitions have sets of allowed probabilities, and defines two refinement relations on such systems. These formalisms address many of the same issues as we do, but rely on complete specifications of the communicating entities since the models are automata and statecharts.

Various dialects of sequence diagrams have been used informally for several decades. The latest versions of the most known variants are UML 2.0 [OMG04] and MSC-2000 [ITU99].

Live Sequence Charts [DH01], [HM03] is an extension of MSC where (a part of) a chart may be designated as universal (mandatory) or existential (optional). Explicit criteria in the form of pre-charts are given for when a chart applies: Whenever the system exhibits the communication behavior of its pre-chart its own behavior must conform to that prescribed by the chart. Timing constraints are included and alternatives may be assigned exact probabilities.

The UML Profile for Schedulability, Performance and Time [OMG05] extends UML by adding stereotypes and annotations for defining values for performance measures such as response time and CPU demand time. The profile is envisaged to be used with a suitable modeling tool based on for example schedulability analysis, Petri Nets or stochastic process algebra. The profile enables specification of a wide range of time-related requirements, including soft real-time requirements. However, no formal semantics is defined for the language.

Most closely related to the work presented in this paper is of course timed STAIRS as presented in [HHRS05a]. Here the notions of positive and negative behavior, mandatory choice and refinement are formalized in relation to sequence diagrams. Timed STAIRS has a more fine-grained analysis of refinement than presented here. This is partly due to a richer semantical model for events and traces. Events in timed STAIRS can be of three different types: transmit, receive and consume. This enables the distinction between two forms of refinement: glass-box refinement, which take the full semantics into account, and black

box refinement, which only considers externally visible changes. The approach presented in this paper can easily be generalized to take this into account. Timed STAIRS does not address probabilities.

11 Conclusion

We have extended the work presented in [HHRS05a]. Our contribution is to generalize the approach to handle probabilities. This enables specification of soft real-time constraints as well as probabilistic specifications in general. The resulting approach, which we call probabilistic STAIRS, offers a powerful language for specifying a wide range of communicating systems, underpinned by a formal semantics that allows analysis of functional and non-functional properties, as well as formal definition of incremental development. The full report [RHS05] on which this paper is based contains additional composition operators (loop, assert, palt with n operands), a discussion on how probability spaces relate to probabilistic STAIRS specifications and proofs of various properties such as transitivity of refinement. In the future we intend to explore the relationship between probabilistic STAIRS and state machines with time and probabilities.

Acknowledgments

The research on which this paper reports has been carried out within the context of the IKT-2010 project SARDAS (15295/431) and the IKT SOS project ENFORCE (164382/V30), both funded by the Research Council of Norway. We thank Rolv Bræk, Øystein Haugen, Birger Møller Pedersen, Mass Soldal Lund, Judith Rossebø, Ragnhild Kobro Runde, Manfred Broy, Ina Schieferdecker, Thomas Weigert and the anonymous reviewers for helpful feedback.

References

[BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.

[DH01] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[HHRS05a] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo, 2005.

[HHRS05b] Ø. Haugen, K.E. Husa, R.K. Runde, and K. Stølen. STAIRS towards formal design with sequence diagrams. *Software and System Modeling*, 00:1–13, 2005.

[HM03] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer, 2003.

[HS03] Ø. Haugen and K. Stølen. STAIRS — Steps to analyze interactions with refinement semantics. In *Sixth International Conference on UML*, number 2863 in Lecture Notes in Computer Science, pages 388–402. Springer, 2003.

[ITU99] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1999.

[Jan03] D. N. Jansen. *Extensions of Statecharts with Probability, Time, and Stochastic Timing*. PhD thesis, University of Twente, 2003.

[JL91] B. Jonsson and K. G. Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 266–277, Amsterdam, The Netherlands, 1991.

[MM99] C Morgan and A McIver. pGCL: Formal reasoning for random algorithms. *South African Computer Journal*, 22:14–27, 1999.

[OMG04] Object Management Group. *UML 2.0 Superstructure Specification*, ptc/04-10-02 edition, 2004.

[OMG05] Object Management Group. *UML Profile for Schedulability, Performance and Time Specification*, version 1.1 formal/05-01-02 edition, jan 2005.

[RHS05] 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, 2005.

[Ros95] A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 114–127, Washington, DC, USA, 1995. IEEE Computer Society.

[Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.