Specification and Refinement by Probabilistic Sequence Diagrams

{Knut Eilif Husa[1,2], Atle Refsdal[1]} [1]Department of Informatics, University of Oslo ,[2]Ericsson
{knutehu, atler}@ifi.uio.no

***Abstract*— We propose probabilistic sequence diagrams as a specification language for describing interactions where probabilities are relevant. The language can be seen as a generalization of a subset of UML2.0 sequence diagrams. A formal semantics is defined based on traces with probabilities. To facilitate an incremental development process a refinement relation is defined on the semantics. There are many ways in which probabilistic sequence diagrams might prove useful: formulating Service Level Agreements, describing security properties, specifying games, abstracting behavior of complex systems and describing the behavior of a system's environment.**

I. Introduction

Sequence diagrams show how messages are sent between lifelines to perform a task. They are used in a number of different situations. They are for example used by an individual designer to get a better grip of a communication scenario or by a group to achieve a common understanding of the situation. Sequence diagrams are also used during the more detailed design phase where the precise inter-process communication must be set up according to formal protocols. The lifelines in a sequence diagram can represent many different kinds of entities, such as objects, components, systems and humans. What they all have in common is that they are able to interact with other entities through sending of messages.

Sequence diagrams qualitatively specify traces, i.e. they say something about what behaviors a system should possess and not possess. E.g. the STAIRS approach ([HS03], [HHRS04]) gives a formal semantics to UML2.0 sequence diagrams where traces are categorized as positive, negative or inconclusive. The approach presented in this paper is in many ways inspired by STAIRS. However, such a categorization of traces puts certain limits to what can be expressed. In some situations it is not enough to say that a certain behavior is wanted or unwanted; we need to say *with what probability* the behavior should occur. There are several situations where probabilistic descriptions are suitable:

- Probabilistic behavior. Some applications behave in a probabilistic way. Games often involve some random element where each outcome should have a certain probability. If we want to model the behavior of a dice, it is not enough to say that all outcomes 1 to 6 are acceptable, and all others are not. We need to specify that the probability should be 1/6 for each of the acceptable outcomes. Also in a security setting we often need to put restrictions on probabilities for different behavior. Suppose we want to specify a program that produces passwords. We may then want to specify that the probability that the program produces any *particular* password should be less than 0.0001%.
- Abstraction. If a certain component/sub-system makes complex calculations/decisions based on details that we do not care about in a given context, then we may want to abstract from these details. Often it will be sufficient to say something like "the system will respond with message m in x% of the cases". A Service Level Agreement is an example of a context where such descriptions may be appropriate.
- Modelling the environment. Being able to model the environment of a system is important for analysis purposes. Often our knowledge of the environment is of a probabilistic nature, based on empirical data. For example, we might know that when users log in to a service, they type a wrong password in 10% of the cases, or that 3% of the items running on a conveyor belt will have some kind of defect. By including such knowledge in a formal model we are able to analyze how overall system performance will be affected.
- Prioritizing resources. By quantifying scenarios we gain insight into where resources should be allocated. Bottlenecks affecting common behavior should normally be dealt with before the others.

In this paper we propose the language of probabilistic sequence diagrams (psd). This involves introducing a new operator - the `palt` - to facilitate the description of probabilistic behavior. Probabilistic sequence diagrams are meant as a generalization of sequence diagrams as presented in [OMG03]. Our goal is a language suitable for expressing what is the acceptable (range of) probability for a given behavior, as well as positive and negative behavior in the UML2.0 sense.

Reaching a detailed system specification is typically the outcome of a process of learning. Starting at a rough sketch one would iteratively elaborate the interactions and in the end reach a specification specific enough for implementation. This means that a precise interpretation of the various steps in incremental system development is needed. Therefore our formalism should capture the notion of refinement and formalize incremental development.

The remainder of this paper is divided into four sections. Section II defines the formal syntax and semantics of probabilistic interactions, and demonstrates the use of the language. In section III we introduce the notion of probabilistic refinement. Section IV presents some related work, while section V show how our goals have been met and outline some ideas for further work.

II. Formal Foundation

In the following we define the notion of probabilistic sequence diagram. In particular, we formalize the meaning of probabilistic sequence diagrams in denotational trace semantics.

A. Events

An event may be of two kinds; a transmission event tagged by ! or a consumption event tagged by ?. A message m is a triple (s, re, tr) of a signal s, a receiver re and a transmitter tr. M denotes the set of all messages. The receivers and transmitters are lifelines. L denotes the set of all lifelines. The semantics of an event is a pair $(k, m) \in \{!, ?\} \times M$. Syntactically we represent an event by $!m$ or $?m$. For any event e, we define $k.e$, $m.e$, $re.e$ and $tr.e$ to denote its kind, message, receiver and transmitter, respectively. We let E denote the set of all event symbols, while $Events$ denotes the set of all events. This means that $Events = \{!, ?\} \times M$.

B. Basic composition of trace sets

A trace is a finite sequence of events such that a transmit event of a message occurs before its corresponding receive event. We write $\langle e_1 ... e_2 \rangle$ for the trace starting with event e_1 and ending with event e_2, and let U denote the universe of traces.

Before the composition operators are defined, some new notation need to be introduced. For concatenation and filtering of traces, we have the functions $\frown$ and Ⓢ, respectively. Concatenating two traces implies gluing them together. Hence, $a_1 \frown a_2$ denotes a trace that is prefixed by a_1. By $b Ⓢ a$ we denote the trace obtained from the trace a by removing all events in a that are not in the set b. π_2 is a projection operator returning the second element of a tuple. In order to resolve the non-determinism in interleaving of traces, we make use of an oracle o, an infinite sequence of trace identifiers. It determines the order in which events from two different traces are sequenced. The set of events that may take place on the lifeline l is denoted by $e.l$. Formally:

$$e.l \stackrel{def}{=} \{e \in Events \mid (k.e = ! \wedge tr.e = l) \vee (k.e = ? \wedge re.e = l)\}$$

We are now ready to define the two basic composition operators on traces, namely weak sequencing and parallel execution denoted by $\gtrsim$ and $\parallel$, respectively:

$$\begin{aligned} t_1 \parallel t_2 \quad &\stackrel{def}{=} \quad \{t \in U \mid \exists o \in \{1,2\}^\infty : \\ &\quad \pi_2((\{1\} \times Events) Ⓢ (o, t)) = t_1 \\ &\quad \wedge \pi_2((\{2\} \times Events) Ⓢ (o, t)) = t_2\} \\ t_1 \gtrsim t_2 \quad &\stackrel{def}{=} \quad \{t \in t_1 \parallel t_2 \mid \\ &\quad \forall l \in L : e.l Ⓢ t = e.l Ⓢ t_1 \frown e.l Ⓢ t_2\} \end{aligned}$$

The definitions of $\parallel$ and $\gtrsim$ are extended to trace sets in the following way: For trace sets T_1 and T_2 we have $T_1 \parallel T_2 \stackrel{def}{=} \bigcup_{t_1 \in T_1} \bigcup_{t_2 \in T_2} t_1 \parallel t_2$ and $T_1 \gtrsim T_2 \stackrel{def}{=} \bigcup_{t_1 \in T_1} \bigcup_{t_2 \in T_2} t_1 \gtrsim t_2$.

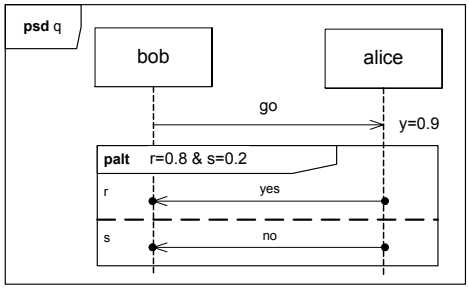

Fig. 1. Graphical representation of a sequence diagram

Example: let $T_1 = \{\langle !a\ ?a \rangle\}$ and $T_2 = \{\langle !b\ ?b \rangle\}$ where $tr.a = tr.b = l_1$, $re.a = re.b = l_2$ and $l_1 \neq l_2$. Then $T_1 \parallel T_2 = \{\langle !a\ ?a\ !b\ ?b \rangle, \langle !a\ !b\ ?a\ ?b \rangle, \langle !b\ ?b\ !a\ ?a \rangle, \langle !b\ !a\ ?b\ ?a \rangle\}$ and $T_1 \gtrsim T_2 = \{\langle !a\ ?a\ !b\ ?b \rangle, \langle !a\ !b\ ?a\ ?b \rangle\}$.

C. Probabilistic sequence diagrams

We use predicates with probability variables $x, y, ... \in [0, 1]$ as free variables to impose probability constraints. By $\mathbb{F}(v)$ we denote the set of logical formulas whose free variables are contained in the set v. The set of syntactically correct sequence diagrams D is defined inductively:

- $\{e; C(x) \mid e \in E\} \subset D$, where $C(x) \in \mathbb{F}(\{x\})$.
- $d_1, d_2 \in D \implies d_1 \text{ seq } d_2 \in D \wedge d_1 \text{ par } d_2 \in D \wedge d_1, x \text{ palt } d_2, y; C(x, y) \in D$, where $C(x, y) \in \mathbb{F}(\{x, y\})$.

For simplicity the formal definition of probabilistic sequence diagrams is given in terms of the textual representation. Figure 1 shows a simple example of a graphical representation. In the graphical representation we use the following conventions for probabilities attached to events: If an event is not annotated with a probability predicate C, then we assume an implicit probability variable x_n with default predicate $0 \leq x_n \leq 1$. A black dot is a shorthand for the probability predicate $x_n = 1$, in other words an event with probability one.

The diagram in figure 1 can be represented textually as
$((!go; 0 \leq x_1 \leq 1) \text{ seq } (?go; y = 0.9)) \text{ seq}$
$(((!yes; x_2 = 1) \text{ seq } (?yes; x_3 = 1)), r$
palt
$((!no; x_4 = 1) \text{ seq } (?no; x_5 = 1)), s$
$; r = 0.8 \wedge s = 0.2)$

D. Semantics of probabilistic sequence diagrams

This section defines a denotational semantics for probabilistic sequence diagrams. The semantics is defined in terms of a function $[\![\_]\!]$ that for any diagram d yields a pair $[\![d]\!] = (T, F)$ where T is a set of traces and F is a set of functions $f : U \to [0, 1]$ assigning probabilities to every trace. A trace with a probability assignment of zero is never allowed to happen, while a trace with probability assignment of one must always occur. The universe U is divided in two sets of traces: the *described traces* T and the *undescribed* traces $U \backslash T$. The rationale for letting the domain of f be U is that we want f to be a probability distribution, i.e. $\sum_{t \in U} f(t) = 1$. In the following let $[\![d_1]\!] = (T_1, F_1)$ and $[\![d_2]\!] = (T_2, F_2)$.

1) Events
$[\![(e; C(x))]\!] \stackrel{def}{=} (T, F)$ where

$$T = \{\langle e \rangle\}$$
$$F = \{f \mid \exists p \in [0,1] : C(p) \wedge f(\langle e \rangle) = p \wedge \sum_{t \in U} f(t) = 1\}$$

if $e \in E$.

The semantics of an event and its probability predicate $C(x)$ is given by a unary trace and all the functions assigning a probability p to the trace such that the predicate $C(p)$ is true. The designer is free to specify the probability predicate; for example it is possible to assign probability zero to a send event and probability one to its corresponding receive event. The reason for this is that probabilities are understood in the context they appear. Consider the diagram in Figure 2. This diagram means simply that the probability of producing the trace $\langle !m, ?m \rangle$ is $0.6 * 0.9$. It does *not* mean that the probability of producing the traces $\langle !m \rangle$ or $\langle ?m \rangle$ is 0.6 or 0.9, respectively.

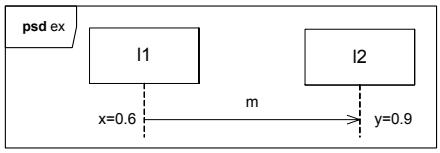

Fig. 2. Sending of a message.

2) Sequential composition
$[\![d_1 \text{ seq } d_2]\!] \stackrel{def}{=} (T, F)$ where

$$\begin{aligned} T &= T_1 \precsim T_2 \\ F &= \{f \mid \exists f_1 \in F_1, f_2 \in F_2, \forall t_1 \in T_1, t_2 \in T_2 : \\ &\quad \sum_{\{(t_3,t_4) \in T_1 \times T_2 \mid (t_3 \precsim t_4) = (t_1 \precsim t_2)\}} f_1(t_3) * f_2(t_4) \\ &\quad \leq \sum_{t \in t_1 \precsim t_2} f(t) \leq \\ &\quad \sum_{\{(t_3,t_4) \in T_1 \times T_2 \mid (t_3 \precsim t_4) \cap (t_1 \precsim t_2) \neq \varnothing\}} f_1(t_3) * f_2(t_4) \\ &\quad \wedge \sum_{t \in U} f(t) = 1\} \end{aligned}$$

The seq construct defines weak sequencing which is the implicit composition mechanism combining constructs of a sequence diagram. Sequencing of two sets T_1 and T_2 of traces is done by sequencing every trace t_1 from T_1 with every trace t_2 from T_2. Deriving probabilities for the resulting traces is complicated by two factors. Firstly, sequencing of two traces generally yields a set of traces, and not a single trace. Secondly, a trace that is in the set of traces we get by sequencing t_1 and t_2 may also be in the set we would get by sequencing two traces t_3 and t_4, even if t_1 is different from t_3 and/or t_2 is different from t_4. In other words, we may have $(t_1 \precsim t_2) \cap (t_3 \precsim t_4) \neq \varnothing$ or $(t_1 \precsim t_2) = (t_3 \precsim t_4)$ even if $t_1 \neq t_3$ and/or $t_2 \neq t_4$. So how should probabilities of traces in $t_1 \precsim t_2$ be derived based on the probabilities for the traces in T_1 and T_2? We have chosen to place an upper and a lower bound for the sum of probabilities for the traces in $t_1 \precsim t_2$. The lower bound is given by the sum of the product of the probabilities for all traces $t_3 \in T_1$ and $t_4 \in T_2$ such that $(t_1 \precsim t_2) = (t_3 \precsim t_4)$. If t_1 and t_2 are the only traces to fulfill this last condition (when $t_1 = t_3$ and $t_2 = t_4$), the lower bound will be $f_1(t_1) * f_2(t_2)$. The upper bound is given by the sum of the product of the probabilities for all traces $t_3 \in T_1$ and $t_4 \in T_2$ such that $(t_1 \precsim t_2) \cap (t_3 \precsim t_4) \neq \varnothing$. If t_1 and t_2 are the only traces to fulfill this last condition, the upper bound will be equal to the lower bound.

3) Parallel composition
$[\![d_1 \text{ par } d_2]\!] \stackrel{def}{=} (T, F)$ where T and F are defined as for seq, except that $\precsim$ is replaced by $\|$ every place it occurs.

The par construct represents a parallel merge or a permutation of sequences such that the ordering of the events from each operand is maintained in the resulting traces. Parallel merge of two traces may result in a set of traces. The considerations regarding probabilities are exactly the same as for seq.

4) Probabilisic alternative
$[\![(d_1, x \text{ palt } d_2, y; C(x,y))]\!] \stackrel{def}{=} (T, F)$ where

$$\begin{aligned} T &= T_1 \cup T_2 \\ F &= \{f \mid \exists f_1 \in F_1, f_2 \in F_2, \\ &\quad p_1 \in [0,1], p_2 \in [0,1] : \\ &\quad (C(p_1,p_2) \wedge \\ &\quad \forall t_1 \in T_1 \backslash T_2 : f(t_1) = p_1 * f_1(t_1) \wedge \\ &\quad \forall t_2 \in T_2 \backslash T_1 : f(t_2) = p_2 * f_2(t_2) \wedge \\ &\quad \forall t_3 \in T_1 \cap T_2 : f(t_3) = p_1 * f_1(t_3) + p_2 * f_2(t_3)) \\ &\quad \wedge \sum_{t \in U} f(t) = 1\} \end{aligned}$$

The palt construct defines probabilities of traces described in the operands. The predicate C with the probability variables x, y as free variables determines the probabilities. If the same trace is described in both operands, the resulting trace probability is defined as the sum of the individual trace probabilities.

Example of semantics: Applying the definitions of the semantics for the operators we obtain the following semantics for diagram in figure 1: $[\![q]\!] = (T, F)$, where $T = \{t_1, t_2\}$, $t_1 = \langle !go\ ?go\ !yes\ ?yes \rangle$, $t_2 = \langle !go\ ?go\ !no\ ?no \rangle$ and $F = \{f \mid 0 \leq f(t_1) \leq 0.72 \wedge 0 \leq f(t_2) \leq 0.18\}$. The reason why $f(t_1)$ and $f(t_2)$ can range from zero to an upper bound is that the $?go$ event can have any probability from zero to one.

E. Example

The scenario described in Figure 3 is inspired by an example taken from [Jan03] and describes an interaction between a client, an automatic teller machine (atm) and a bank. It starts with the atm prompting "please insert card" to the client. The client may then insert her/his card, and the atm prompts "please enter PIN". In this initial phase of the interaction we make no assumptions regarding probabilities of the client's behavior,

therefore we have put no explicit predicates on the first three events on this lifeline. After the ”please enter PIN” message has been received there are two alternative continuations of the scenario. The first is given by the first operand of the outermost `palt` and starts with the client giving the correct PIN. According to the specification the probability of this continuation being chosen should be higher than 0.9. (Note that this probability applies to the whole operand, not only to the first event.) The second alternative is that the client enters a wrong PIN, and nothing more happens. The probability for this should be less than 0.1. If the first scenario is chosen, then the atm responds to the correct PIN by asking how much money the user wants to withdraw. Again two alternatives are given. The first has a probability of between 0.2 and 0.4. It begins with the client choosing 20 euros (or whatever the unit might be), and the atm then passes this message along to the bank. What happens next is described in the referenced diagram in Figure 4. The second alternative has a probability from 0.4 to 0.7 and only differs from the first in that the chosen sum is 50 instead of 20. In Figure 4 we see that after the bank has received a request it might either accept it (with probability > 0.8) or reject it (with probability < 0.2). This is an abstraction in the sense that we do not go into the details for why the bank does one or the other. If the bank rejects the request it will perhaps in most cases be because the client does not have enough money in her/his account. But there could also be other reasons, such as the account being temporarily closed due to a lost card, or some maintenance problems at the bank.

Note that for the innermost probabilistic alternative in Figure 3 there is a possibility that the sum of probabilities for the given alternatives is less than one. This allows undescribed situations to happen, for example withdrawal of other amounts.

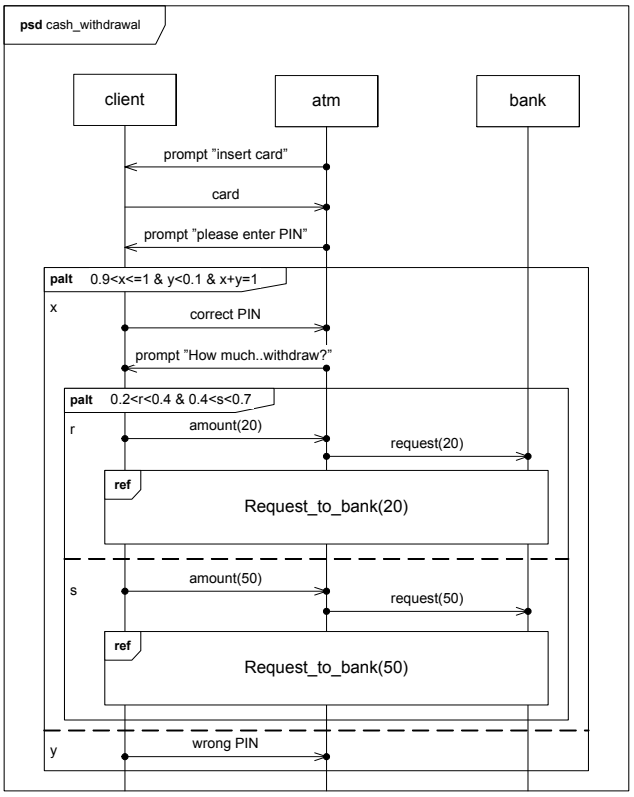

Fig. 3. A cash withdrawal scenario.

III. Refinement

Refinement of a specification means to reduce underspecification by adding information such that the specification becomes closer to an implementation. In our setting this can be done by reducing the acceptable variability for trace probabilities or by describing new traces.

Let $\mathit{II}_T.f \in T \to [0,1]$ be the projection of f on T, and let $\mathit{II}_T.F = \{\mathit{II}_T.f \mid f \in F\}$. We say that (T', F') is a refinement of (T, F) iff

1) $T \subseteq T'$ and
2) $\mathit{II}_T.F' \subseteq \mathit{II}_T.F$

Point 1) says that all traces that are described remain described through a refinement. However, traces from the undescribed domain may be described and hence moved from the undescribed set to the described set. Point 2) says that for all functions in the refined specification there must be a function in the original specification that assigns the same value for every trace in the original specification.

A. Example of refinement

Figure 5 shows a refinement of the scenario given in Figure 3. First of all some new traces have been added. A new operand has been added to the innermost `palt` that describes

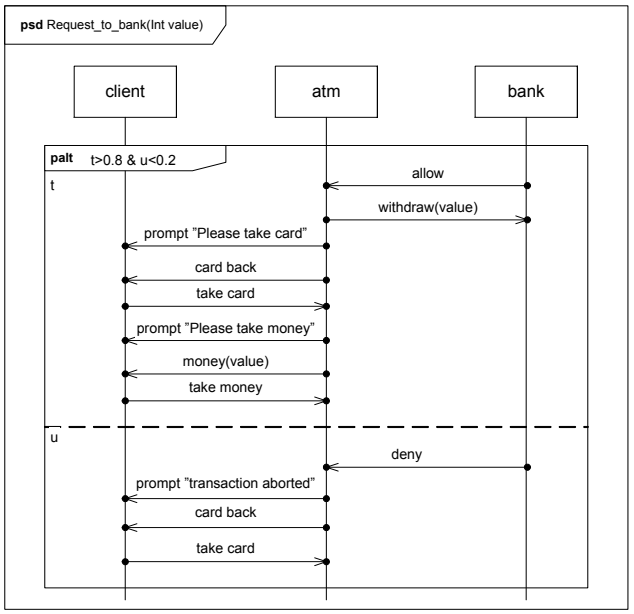

Fig. 4. Cash withdrawal scenario continued.

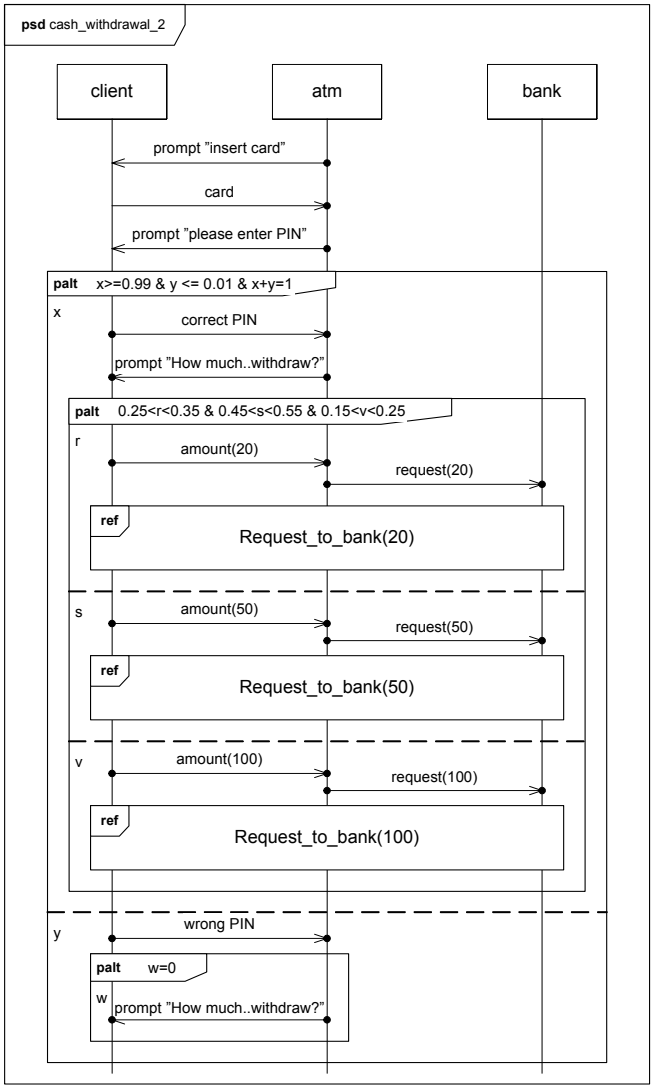

Fig. 5. A refinement of the cash withdrawal scenario.

a situation where the client withdraws 100 euros. The probabilities for the resulting new traces add up to between 0.15 and 0.25. We have also added a new message "How much do you want to withdraw" enclosed by a palt and given a probability of zero in the second operator of the outermost `palt`. This means that we have added a trace where the atm sends this question to the client after the client has entered an incorrect PIN. The probability for this trace should of course be zero.

In addition to adding new traces we have also reduced the acceptable probability intervals for the original traces. For example, the situation where a wrong PIN was given originally had a probability from zero to (but not including) 0.1. In the refined version the probability has to be lower than 0.01. We might imagine that the designer has decided that the high probability of wrong PINs will be unacceptable. A satisfactory probability could perhaps be reached by improving the user interface of the atm or by changing the PIN code format. Semantically this change means that all functions assigning probabilities from 0.01 to 0.1 to the corresponding trace have been removed.

IV. Related Work

Sequence diagrams have been used informally for several decades. The first standardization of sequence diagrams came in 1992 [ITU93] — often referred to as MSC-92. Later we have seen several dialects and variations. The sequence diagrams of UML 1.4 [OMG00] were comparable to those of MSC-92, while the recent UML 2.0 [OMG03] has upgraded sequence diagrams to conform well to MSC-2000 [ITU99].

STAIRS [HS03] is an approach to the compositional development of sequence diagrams supporting the specification of mandatory as well as potential behavior. A new language construct `xalt` is introduced to describe mandatory behavior. Traces are categorized as positive, negative or inconclusive. Basic increments in system development are structured into three types: Supplementing means moving traces from the inconclusive set to a positive or negative set. Narrowing means moving traces from a positive set to a negative set. Detailing means introducing a more detailed description without significantly altering the externally observable behavior. We see our work as a generalization of the work in STAIRS. Our notion of refinement is based on that found in STAIRS.

Live Sequence Charts [DH01] is an extension of MSC allowing the distinction between possible and necessary behavior. The designer is allowed to selectively designate a chart or part of a chart as universal (necessary, mandatory) or existential (possible, provisional, optional). We view the distinction between universal and existential behavior as a special case of probability - universal behavior corresponds to behavior with probability one, while existential behavior can have probability in the interval $[0, 1\rangle$. Therefore, to a certain degree our work can be seen as a generalization of LSC. However, [DH01] also introduce explicit criteria for when a chart applies in the form of pre-charts: Whenever the system exhibits the communication behavior of its pre-chart its own behavior must conform to that prescribed by the chart. In our approach we have nothing similar to pre-charts. Instead we rely on an implied assumption of alignment, just like UML sequence diagrams.

V. Conclusion

We have proposed probabilistic sequence diagrams as a language for expressing probabilistic behavior. A formal semantics is given based on traces with probabilities. The semantics is accompanied by a refinement relation, thereby supporting incremental development. The language is a generalization of a subset of UML2.0 sequence diagrams. Positive and negative behavior are expressed as a special cases of probabilistic behavior; positive traces are traces that may have probability greater than zero, while negative traces have probability zero.

We intend to continue the work presented in this paper. One of the things we plan to do is to include time in the language. The combination of time and probabilities on sequence diagrams should give a very powerful language for specifying system behavior and properties. One main focus for us will be to explore how availability properties can be expressed within such a language. A natural next step would also be to explore the relationship between probabilistic sequence diagrams with time and other formalisms including time and probabilities, for example stochastic state-charts as presented in [Jan03].

To our knowledge no other language has fully integrated probabilities with sequence diagrams. The expressive power of this combination allows a wide area of applications.

VI. Acknowledgments

The research on which this paper reports has been carried out within the context of the IKT-2010 project SARDAS (15295/431) funded by the Research Council of Norway. We would like to thank Øystein Haugen and Ketil Stølen for helpful comments.

References

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

[HHRS04] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. Technical Report ISBN 82-7368-261-7, University of Oslo, 2004.

[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.

[ITU93] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1993.

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

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

[OMG00] Object Management Group. *Unified Modeling Language, Version 1.4*, 2000.

[OMG03] Object Management Group. *Unified Modeling Language: Superstructure*, OMG ad/03-04-01 edition, 2003.