STOCHASTICALLY ENHANCED TIMED AUTOMATA

Lynne Blair, Trevor Jones, Gordon Blair
Computing Department, Lancaster University, U.K.
(email: lb@comp.lancs.ac.uk)

Abstract: There is currently considerable interest in the formal specification of distributed multimedia systems, with the majority of research in this area considering the use of timed formal languages for the specification of such systems. In contrast, however, there has been less research on the specification of stochastic behaviour, and yet this style of behaviour is predominant in this class of system. We therefore present stochastically enhanced timed automata, as a solution to the problem. The paper also introduces an associated tool suite that supports the editing, composition and simulation of such automata. Finally, a multimedia example is presented, illustrating the use of the enhanced automata and the associated tool suite.

1. INTRODUCTION

In our previous work, we have been interested in the development of methodologies and techniques for the specification and verification of distributed multimedia systems, with particular emphasis on object and component-based systems [Blair98a]. This has led us to develop an *aspect-oriented* style of specification that follows the principles of aspect-oriented programming [AOP00][Murphy99]. Essentially, we decompose a system specification into four aspects: functional, non-functional, management and requirements. Motivations for this choice can be found in [Blair99c]. In the (aspect) *programming* world, once written, the aspects must be woven together using a set of identified join-points. There is a nice parallel here with aspects in a *specification* world: once written, our aspects are composed together (using standard parallel composition rules) and the join-points

become the set of actions (events) on which synchronisation occurs. Prior to this paper, our common model has been taken to be timed automata, along with their underlying timed labelled transition system semantics. Within this framework, we support a multi-paradigm approach, where (if appropriate) different aspects of a system may be specified in different formal languages. A detailed description of the semantics of translation and composition in this approach was presented in [Blair99a] and details of a tool suite developed to support our approach can be found in [Jones99a].

Using this approach, we have been able to capture the real-time information of object-based distributed multimedia systems and their associated quality of service (QoS) parameters such as latency, throughput and jitter. However, up to now, we have not been able to represent stochastic information. So, whilst we can specify that a communication takes n units of time (latency) and may vary by $\pm\delta$ time units (jitter), we cannot allow for the possibility of, for example, extreme delays (due to congestion, or client/server down-time). This paper presents a stochastic extension to our approach, permitting the specification of the *distribution* of event timings (typically associated with some distributed communication, though not necessarily). Clearly, we would like to adopt a technique that will fit within our aspect-oriented and multi-paradigm framework and that can be supported by an extension to our tool suite.

2. STOCHASTIC MODELLING TECHNIQUES

2.1 Overview of techniques

There exists a wealth of literature on stochastic process models such as Markov chains and generalised semi-Markov processes [Glynn89], graphical representations of such processes using stochastic Petri nets (including the generalised stochastic Petri nets of [Marsan95]) and more abstract notations such as queuing networks [Gross85]. Recently, much interest in stochastic process algebras has lead to the development of a number of formalisms such as EMPA [Bernardo96], GSMPA [Bravetti99], PEPA [Hillston96], SPADES [D'Argenio99] and TIPP [Hermanns96]. Our particular interest lies in techniques that fit our multi-paradigm framework and can be supported by extending our current tool suite. As mentioned above, our framework uses timed automata based on a timed labelled transition systems. The work of [D'Argenio99] (and [D'Argenio98]) is particularly interesting in that, in addition to describing the stochastic process algebra SPADES, it introduces a stochastic automata model based on probabilistic transition systems with general distributions. The process algebraic model has already

been analysed in the context of a simple multimedia system [Bryans99] and, as such, we are confident that it could be used within our framework and domain of distributed multimedia systems. However, in order to fit our framework, we need to adopt a new common model based on stochastic automata. Consequently, we focus further on this technique below.

2.2 Focus on SPADES

We are primarily interested in the stochastic automata of [D'Argenio99] in this paper (rather than the higher-level process algebra). Stochastic automata are finite state automata extended with clocks. Instead of these clocks being used to model deterministic timing, they are defined to take a random value set according to a given probability distribution. Once set, clocks count down (unlike timed automata where they count up). When the clocks reach zero, they enable certain transitions. Depending on required interactions with the environment, transitions need not fire immediately, but may wait until all clocks controlling an interaction reach zero.

To illustrate the firing of transitions, consider the two stochastic automata in figure 1, both involved in an interaction on a *transmit* event. The timing of this event in the first automaton is governed by clock x sampled from a distribution $F_x(t)$, whilst the timing of the event in the second automata is governed by clock y sampled from a distribution $F_y(t)$. The setting (and resetting) of clocks is depicted by the inclusion of a clock name within a state, as in the initial state of each automata below. Since SPADES supports general distributions (not just exponential distributions as in many stochastic techniques), there is no restriction on what distributions are used.

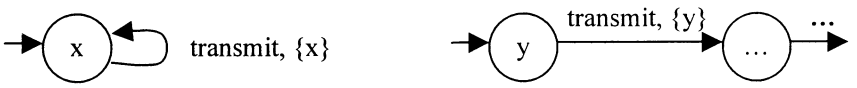

Figure 1. Stochastic Automata in SPADES

When composed, both clocks are initialised (i.e. set to a value sampled from the associated distribution). The automaton containing the clock that reaches zero first must wait to synchronise with the other automaton (wait until the other clock reaches zero). This results in a situation where you can think of clocks as being allowed to go negative, with a transition being enabled when all involved clocks have values of zero (or below).

2.3 Stochastic versus timed automata

The behaviour described in the stochastic automaton above is subtlely different from a traditional (deterministic) timed automata model, where clocks count up and a combination of guards, resets and invariants are used to control the timing of events (although we do not use the latter in figure 2 below). In order to compare models, we present a similar example to above in timed automata. We assume the existence of two constants, c1 and c2, used as lower bounds in the guards of our automata.

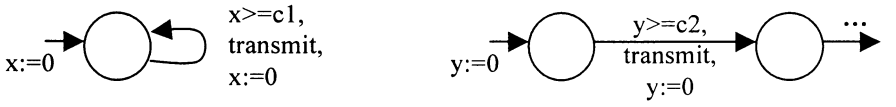

Figure 2. Timed automata

When composed, both clocks are explicitly initialised and the automata then idle in their respective initial states until one of the clocks reaches its desired bound (c1 or c2). The first automaton to reach this point continues to idle until the other automaton is ready to participate in the event.

Our two models differ *if* we assume that the automata provide a complete model of our system (i.e. no outside influences affect the timing of events). This is also referred to as a *closed* system. If this is the case for the stochastic automata model in figure 1, the *transmit* event will fire immediately the second clock reaches zero. This is known as a *maximum progress* condition. However, in the timed automata model depicted in figure 2, the clocks may continue to idle; there is no urgency for the *transmit* event to (ever) occur.

Timed automata have traditionally addressed the issue of urgency by associating *invariants* with states. These determine the length of time that an automaton may remain in a given state before it must *progress*. The use of invariants has sparked off many debates, particularly with respect to the likelihood of time-locks arising in a composed system [Bowman99]. This itself has prompted the development of other timed automata models that include a notion of urgency but avoid invariants, such as timed automata with deadlines (TAD) [Bornot98]. However, these issues are aside from our purpose in this paper, so we continue our example using timed automata with invariants.

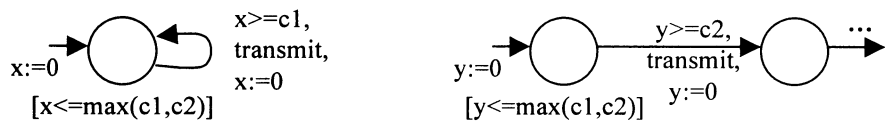

Figure 3. Timed automata with invariants

In this system, once max(c1,c2) is reached, neither automaton may remain in the same state. Since both have an enabled transition, the *transmit* event, this *must* fire immediately (before time can progress further). It should be obvious that the solution that we have presented in figure 3 is not ideal since the invariant of one automaton relies on "global" information, namely the lower bound of the other automaton's guard. However, using only "local" information increases the potential for time-locks.

To conclude this section, the stochastic automata of SPADES provide an elegant low-level representation of stochastic behaviour, where the timing of events can be governed by any distribution. This approach fits in well with our desire to find a suitable low-level common model for our aspect-oriented and multi-paradigm approach. However, the downward counting of clocks differs from our existing model and, as such, would require substantial re-definition of our (multi-paradigm) framework and re-implementation of our tool suite. Furthermore, we make heavy use of untimed variables in our work (e.g. counters, variables representing desired rates of transmission, permitted error rates, rates of throughput, etc). These are not defined for SPADES, although this should not be a difficult extension (at least at the automata level). Consequently, we define our own timed automata extension, inspired by the constants used in figures 2 and 3.

3. EXTENDING TIMED AUTOMATA

The timed automata clocks in figures 2 and 3 counted up to deterministic values represented by constants. For our distributed multimedia examples, we require a model whereby these constants can be sampled from an appropriate distribution. This is straightforward from a syntactic perspective, but requires more care with respect to the semantic relationship between the clocks and stochastic variables, and also the semantics of re-sampling. We must also move from our underlying timed transition system semantics to (timed) probabilistic transition system semantics. We explain the relationship between different semantic models below.

3.1 Underlying semantic models

Labelled transition systems (LTS). We define a labelled transition system over Act, the set of (atomic) actions, to have the following form:

$$LTS = \langle S, s_0, \longrightarrow \rangle$$

where S is a (non-empty) finite set of states,
$s_0 \in S$ is the initial state, and
$\longrightarrow \subseteq S \times Act \times S$ is a transition relation.

Intuitively, we now have a model with actions, states and transitions (labelled with actions) between states. As in LOTOS, we permit actions to have associated data (currently only integers in our tool suite) and denote output by "a!value" and input by "a?var:var_sort". The rules governing the synchonisation of events carrying data follow those of LOTOS ([ISO89]).

Timed labelled transition systems (TLTS). To extend labelled transition systems with time, we follow the approach of [Larsen95] and permit two sorts of actions: atomic actions (Act) and delay actions (Δ) whose elements are denoted by $\varepsilon(d) \in \Delta$ (for $d \in 3^+$). We denote the set of labels L as $Act \cup \Delta$. Our transition relation from the LTS definition above must now also permit delay actions, and must satisfy the usual time determinism and time additivity rules [Larsen95].

We now have a timed semantic model where transitions are labelled by an action or a delay. In the first case, the set of action-labelled transitions is:

$$\{s \xrightarrow{a} s' \mid a \in Act \text{ and } s, s' \in S\}$$

In the second case, the set of delay-labelled transitions (for $\varepsilon(d) \in \Delta$) is:

$$\{s \xrightarrow{\varepsilon(d)} s' \mid d \in 3^+ \text{ and } s, s' \in S\}$$

Unfortunately, at this low semantic level, the model yields an infinite structure. For example, an action occurring at a time less than 3 would (in a continuous domain) be represented by an infinite number of delay transitions each labelled with a time t ($0 \le t < 3$), and each followed by the action-labelled transition. This infinite structure is clearly unmanageable as a modelling language, hence the development of higher-level models such as automata.

Probabilistic transition systems (PTS). We can further extend transition systems to include probabilistic information. Suppose we have a transition occurring at time f where f is sampled from a distribution F, $s \xrightarrow{\varepsilon(f)} s'$. Again, there are an infinite number of possible (delay) transitions, this time with the probability of each determined by the distribution F. Clearly, the sum of probabilities of all possible transitions should equal one.

Probabilistic transition systems can be formally defined. However, their definition is mathematically complex and we believe that the above intuitive

definition is sufficient for this paper. We thus move on to define our timed automata model, and refer those wishing to see a complete formal definition of probabilistic transition systems to [D'Argenio98].

3.2 Timed automata

Notation. The purpose of our timed automata model is to provide a higher-level, more manageable model than that provided by TLTS. We start with the timed automata of UPPAAL [Larsen95], but extend the model to permit data-carrying events as described above. Timed automata are finite state automata extended with clocks and variables; let C be a finite set of real-valued clocks and V be a finite set of real-valued variables. The subset $D \subseteq V$ will be used to represent the set of data variables used to attach data to events (we currently distinguish between D and V since we restrict data to integers in our tool suite). We will let $Var = C \cup V$.

Our timed automata model also permits *guards* (conditions on clocks or variables) on transitions and *invariants* on states (that must remain true whilst in that state). Let the set of constraints (guards) over Var be represented by G(Var). Note that the values of clocks and variables can be compared with constants and/or *reset* on transitions.

Timed automata. We now define a timed automaton formally as:

$$TA = \langle S, s_0, \longrightarrow, I \rangle$$

where S is a finite set of states, $s_0 \in S$ is the initial state, $\longrightarrow$ is a transition relation, and $I : S \rightarrow G(Var)$ is an invariant assignment function.

Our transitions are more complex than for the timed transition systems above since they can now incorporate guards and resets. The format of a transition is thus the tuple:

$$\langle l, g, a, r, l' \rangle$$

where l, l' are nodes (locations) and $a \in L$ (as above),
and g is a constraint (guard) s.t. $g ::= x \sim c$, and
I(l) is an invariant (on location l) s.t. $I(l) ::= [\, x \sim c \,]$, where
$\sim ::= < \mid = \mid > \mid \leq \mid \geq$, $x \in$ Var and c is a constant,
r is a reset (or reassignment) function.

The semantics of timed automata can now be defined in terms of timed transition systems. Each state in an automaton must now consist of the tuple:

$$s = (l, u_c, u_v)$$

where
l is a node (or location)
u_c is a clock variable assignment function s.t.for clocks $X \subseteq C$, $u_c : X \rightarrow 3$

u_v is a variable assignment function s.t. for variables $Y \subseteq V$, $u_v : Y \rightarrow 3$
and for data variables $Z \subseteq D$, $u_v : Z \rightarrow 9$

The initial state is $s_0 = (l_0, u_{c0}, u_{v0})$, where l_0 is the initial node, u_{c0} initialises all clock variables [C α0] and u_{v0} initialises all variables [V α 0].

Intuitively, we now have a model containing a set of states where, at each state, we can query the value of clock variables (respectively variables) through the clock variable (respectively variable) assignment function.

Finally, the semantics of the transition relation can be defined as follows:

$(l, u_c, u_v) \xrightarrow{g,a,r} (l', u_c', u_v')$ if g is satisfied by u_c and u_v,
$u_c' = [r \alpha r']u_c$ and $u_v' = [r \alpha r']u_v$,

i.e. All guards must be satisfied, and those variables being reset are reset whilst the rest remain unchanged.

$(l, u_c, u_v) \xrightarrow{d(d)} (l', u_c', u_v')$ if $(l = l')$, $u_c' = u_c+d$ and $u_v' = u_v$, and
I(l') is satisfied by u_c' and u_v'.

i.e. Time can pass whilst in a given state, providing the invariant still remains true after the passage of time.

3.3 Stochastically enhanced timed automata

In order to allow variables to assume stochastically sampled values, we define the set F of distribution functions and $F_i \in F$. We use $f_i \blacktriangleleft F_i$ to denote that variable f_i is sampled from distribution F_i. Such variables are denoted *sampling variables*, the set of which is $SVar \subseteq Var$ (all $f_i \in SVar$). We now require an extended tuple to represent the semantics of an automaton's state:

$$s = (l, u_c, u_v, f)$$

where f is a distribution sampling function s.t. for sampling variables $SV \subseteq SVar$, $f : SV \rightarrow 3$.

The initial state is as before, except that it now contains f_{init}. This ensures that all sampling variables referred to in the initial state are initialised to samples from their associated distributions ($\forall f_i \in sv(l_0)$, $f_i \blacktriangleleft F_i$). The current value of a sampling variable can be referenced either on guards or state invariants (similar to our use of constants in figures 2 and 3), such that:

g has the form $x \sim f_i \mid f_i \sim c$, and
I(l) has the form $[\, x \sim f_i \,] \mid [f_i \sim c\,]$where
f_i is sampled from distribution F_i (denoted $f_i \blacktriangleleft F_i$).

We also need to make reference to the set of all guards on outgoing transitions of location l (denoted GL_l). We define the function g(l) over the domain G(Var) to return the set GL_l. We also define the function sv(l) over SVar to return a set of all sampling variables appearing in invariants or

outgoing transitions of l (SL_l). We now require (in addition to the semantics for the transition relation for timed automata above):

$(l, u_c, u_v, f) \xrightarrow{g,a,r} (l', u_c', u_v', f')$ if for all $f_i \in sv(l'), f_i \blacktriangleleft F_i$

i.e. All sampling variables referred to in invariants or outgoing transitions of the new location are re-sampled on entry into the new state. An example illustrating this is presented in figure 4.

$(l, u_c, u_v, f) \xrightarrow{\varepsilon(d)} (l', u_c', u_v', f')$ if $f' = f$

i.e. Values of sampling variables are unchanged by the passage of time.

Finally, composition of our extended timed automata is governed by parallel composition rules, details of which are included in Appendix A. We now have a timed automata model with clocks that can count upwards towards values sampled from stochastic distributions. As can be seen from above, this requires minimal changes to the timed automata model. By taking this approach, we have also retained our standard (non-clock) variables and our data variables that we use to model data-carrying events.

4. TOOL SUPPORT: LUSCETA

LUSCETA is a tool suite that we have developed at Lancaster to provide support for the analysis of our stochastically enhanced timed automata (Lancaster University tool suite for Simulating, Composing and Editing Timed Automata). A first version of this tool suite offered support for the composition of different aspects written in different paradigms, and their subsequent simulation or model checking, but did not offer any support for modelling stochastic behaviour [Blair99b]. In addition, the tool suite supported the generation of timed automata from temporal logic and enabled interoperability with existing tools such as Eucalyptus [Garavel96] and UPPAAL [Larsen95]. For reasons of evolution, a large proportion of our original tool suite has now been rewritten using component technology, i.e. Java Beans [Jones99a]. Importantly, the tool suite has recently been extended further to support stochastically enhanced timed automata, as described by the semantics above. The use of component technologies in this area is noteworthy given the monolithic and closed nature of many existing formal tools. It is hoped that, by adopting this approach, LUSCETA can evolve rapidly to meet new requirements and that we can also promote the re-use of key parts of tool suites such as model checking algorithms.

The editor in LUSCETA allows us to enter stochastic information relating to our automaton in the form of a header. Typically, we define a list of the required distribution functions (e.g. Uniform, Normal, Binomial, Poisson) with their appropriate parameters (e.g. range, mean, variance). We then associate sampling variables with these distributions. For example, let F

be a Normal distribution, F = N(10, 6.5) and f be an associated sampling variable, f := F. Note that this corresponds to the sampling process referred to as f ◄ F above. We can now use these directly in the guards and/ or invariants of our automaton (respectively, f ~ c and [f ~ c]) or, assuming we define t as a (clock) variable, we can compare the variable's value with the sampled value (respectively, t ~ f and [t ~ f]). To illustrate this, consider the following simple automaton that makes use of a sampling variable f taken from the normal distribution F, as defined above.

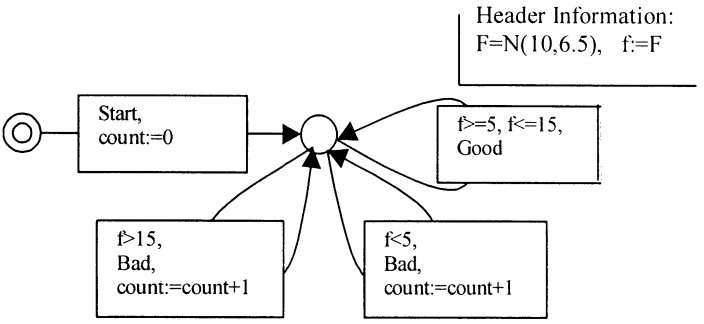

Figure 4. Simple automata to demonstrate stochastically generated values

Using our tool suite we can now simulate this automaton, sampling new values for f every time we enter the second state (according to our semantics). As can be seen, "good" actions occur when the sampled value is between 5 and 15 (which, for the stated mean and variance, corresponds to a 95% confidence interval) and "bad" actions at other values. Table 1 shows the results of the count variable after varying numbers of simulation steps. As expected, the error count roughly corresponds to 5% of the steps.

Table 1. Simulation results from the above automaton

Number of steps	Error count	Time taken (seconds)
10	1	1
50	2	2
100	6	4
500	21	18
1000	48	35

Two further tool components are a *graphical component* that produces a graphical representation of the simulation steps (only appropriate for a small-medium number of steps) and a *distribution component* that produces a visualisation of the sampled values and how they fit the sampling distribution. All components are described in detail in [Jones99a]. We now consider the application of our automata and tool suite to a larger example.

5. EXAMPLE: SYNCHRONISATION PROTOCOL

5.1 Overview

Many different algorithms have been proposed in the literature to handle the synchronisation of multiple media streams under varying assumptions/ conditions. For example, [Blakowski96] and other papers in the same journal cover a variety of different techniques. The algorithm that we will use in this example has been published recently in [Biersack99]. In their paper, they present a synchronisation protocol for *stored* media by "stepwise refinement". This means that a separate model is used to represent three incremental stages of the protocol design, as will be described below. The protocol assumes that the media units (mu) are distributed in a round robin fashion across distributed servers (see figure 5).

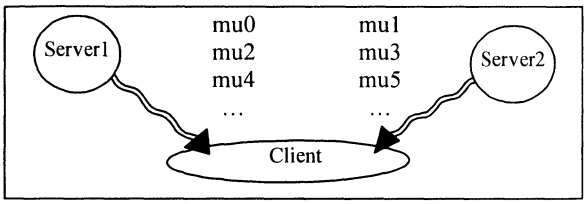

Figure 5. Two distributed servers showing stream connections to a client

5.2 The three models in detail

The *initial model* of [Biersack99] addresses the problem of start-up synchronisation under the assumptions of constant delay and zero jitter. The main problem addressed by this model is compensating for potentially different latency values for each server-client connection due to the distributed nature of the servers. Traditionally, delaying media units at the client by buffering data compensates for this. However, with large differences in latency this implies large buffers are required. Note that since the protocol is intended for stored (c.f. live) data makes this possible.

This model is split into two phases: an evaluation phase and a synchronisation phase. The role of the evaluation phase is to obtain the earliest possible play-out time for the first media unit and, correspondingly, the required start time for each server. Also calculated is the required buffer space at the client. The synchronisation phase involves the sending of a message from the client to each server after which each server starts sending its data at the specified time. Earlier work has considered the modelling of these phases using an underlying model of timed automata [Blair99d].

The *second model* relaxes the restriction on jitter by permitting bounded end-system jitter and network jitter (termed the maximum jitter strategy). The effect of jitter is that the temporal relationship within (and between) media stream(s) is destroyed, even though the media units may have been sent in a timely manner. Consequently, to smooth out the effects of jitter, buffering at the sink must occur. In order to illustrate our stochastically enhanced automata, we make a slight amendment to the maximum jitter strategy. Instead of having hard upper and lower bounds, we allow the jitter to be sampled from a Normal distribution. In a similar way to the example presented above, we select the parameters of the distribution such that we are 95% confident that the jitter is within our allowable bounds.

The evaluation phase is the same as for the first model, but the synchronisation phase requires a further property to be satisfied before play-out of a stream of media units (see [Biersack99]). The *last* sub-stream (one per server) to satisfy this property determines the play-out deadline for the stream. Different jitter bounds per sub-stream complicate the situation (by giving different play-out deadlines and buffer requirements). We determine the necessary buffer requirements from the sub-stream with the largest jitter bound. Due to jitter, these will clearly be higher than for the first model.

The *third model* is more general still and can cope with alterations in the average latency, clock drifts and server drop-outs. In our example, we address the first two models, but not the third.

5.3 Specification of the example

Using timed automata to model the client and server. For the purposes of this paper, we choose to focus on the synchronisation phase. Note that automata for the evaluation phase can be found in [Jones99b]. Since we have chosen to relax the bounded jitter assumption, we add a transition to detect when the maximum jitter strategy fails (i.e. the jitter bounds are exceeded). If this occurs, we allow the play-out to continue as follows: if a media unit has failed to arrive by its play-out deadline, we increment an error counter and wait for the length of time it takes to play one media unit before trying again. Note that we should also check if the maximum buffer size has been exceeded, although this has not yet been included in the current version.

Figure B.1 (in Appendix B) shows the timed automaton representing the second phase of the client. After initializing some constants and variables, we send out play requests. We then wait until we have at least one media unit from each server and, once the play-out deadline has been reached, we start playing media units. We continue to receive media units from each server and play them on schedule (or generate an error) until all our media units have been played. On finishing, we analyse the number of errors in

order to decide whether our play-out was successful or not. If there have been errors, we can analyse the log to check that these have arisen from jitter outside the acceptable bounds (see following section).

The server is much simpler in design, (figure B.2, Appendix B) receiving play requests and sending each media unit, in turn, at the required time. This enables the client to piece together each media unit with its complementary parts, ready for play-out. In our example, we assume each server contains 10 media units (represented by variable N in the server automaton).

Using stochastically enhanced automata to model the medium. The transition between the server and client (through a medium) is now quite complicated. The automaton in figure 6 models transmissions from one server and allows up to 4 media units to be in transit from any one server at a time. The automaton is duplicated for transmissions from the second server.

After the play request has been sent, our transition automaton accumulates media units (by synchronising on the server's *snd_mu1* event and the transition automaton's *add_mu1* event). The array-like notation AA[1,2,3,4] and multiple arrows have been used to simplify the presentation; they correspond to the 4 media units allowed to be in transit at a given time. The first media unit is used to decide when play-out may begin. To determine the transmission delay, we sample jitter from a Normal distribution, using different parameters for each server's sub-stream. In the transition automaton above, *sample1* is used to denote the sampling variable from the distribution associated with server 1's sub-stream. This information is specified as a header to our automaton, e.g. F1=N(6, 0.26), sample1:=F1. When the sampled value is added to the client's local time (variable *lt*) we obtain the time at which the client receives each media unit. This involves synchronisation between the transition automaton's *out_mu1* event and the client's *in_mu1* event. The media unit is now removed from the transition automaton's buffer and buffered at the client until the play-out deadline.

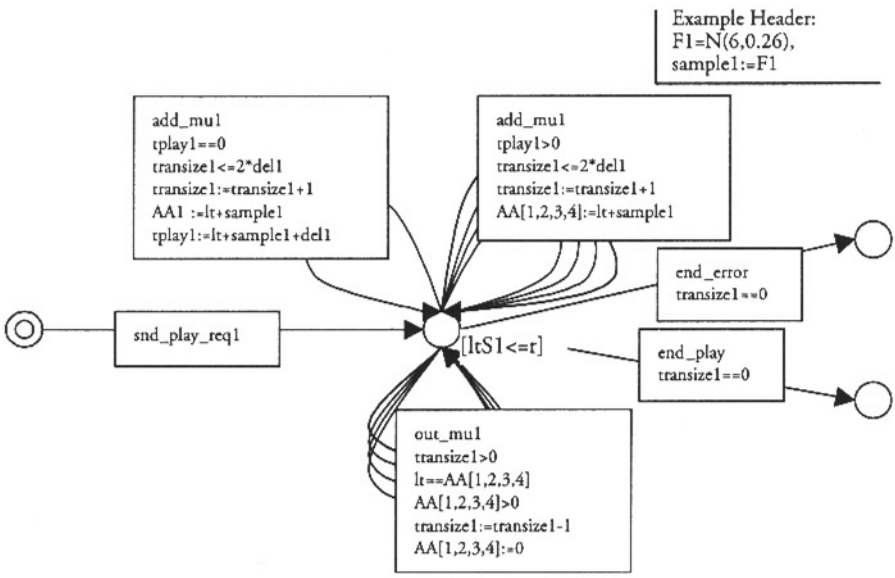

Figure 6. The transition automaton

6. ANALYSIS

We believe that our stochastically enhanced timed automata provide the extra expressive power required for our domain and also fit elegantly into our existing framework. Furthermore, the formal semantics offer a precise description of the model for implementation purposes (and have shown up several flaws in our original design). We now focus on the use of this tool suite to analyse the behaviour of the synchronisation protocol.

Firstly, it is important to point out that the bounded jitter algorithm presented in [Biersack99] is, by design and mathematical analysis, correct with respect to synchronised play-out and buffering requirements under the assumption of bounded jitter. Thus, it has *not* been our aim to find errors in this algorithm. Instead, the approach we have taken is to relax the bounded jitter assumption and sample the transmission delays from a Normal distribution such that 95% of the samples should lie within the required bounds. This permits us to demonstrate our stochastic enhancements to automata and to evaluate our tool suite.

Having specified all automata required for this example, our first step is to compose them together using the standard parallel composition rules presented in Appendix A. Each transition automaton is composed with the corresponding server and then added to the client in turn. We then add the initial evaluation phase to give us the overall composition of the system.

Each stage of the composition takes of the order of 1 second or less to complete and our final automaton contains 348 transitions.

In our analysis, we have simulated our system repeatedly (100 times) and collated results. Note that, for a single run, we have 10 media units per server and 2 servers, hence 20 media units in total. With an expected 5% of these giving errors, we would *statistically* expect 1 error per run. Of course, in practice our sampled values may give us a different number of errors. Consequently, for each simulation run, we were particularly interested in the number of errors in that run and the cause of the error (i.e. checking the log to ensure that the error arose due to a sampled time outside the required bounds). After running the simulation 100 times, we noted errors in 78 of the runs, with a total of 93 errors. Each run typically consists of approximately 100 steps and the log for each one of these takes approximately 5 seconds to generate. An annotated screen-dump of a portion of the log from one of the simulation runs resulting in an error is shown in figure 7.

At each step of the simulation, the log provides us with a list of possible transitions. Some of these may be *unavailable* due to guards not being satisfied. All *available* actions are listed on the main interface. At this point in the log, transition 14 is the only one listed as available, hence we must take this transition. The last line shows the current values of all our variables and, importantly, shows our error count equal to one. This indicates that in the simulation, one media unit failed to arrive by its play-out deadline.

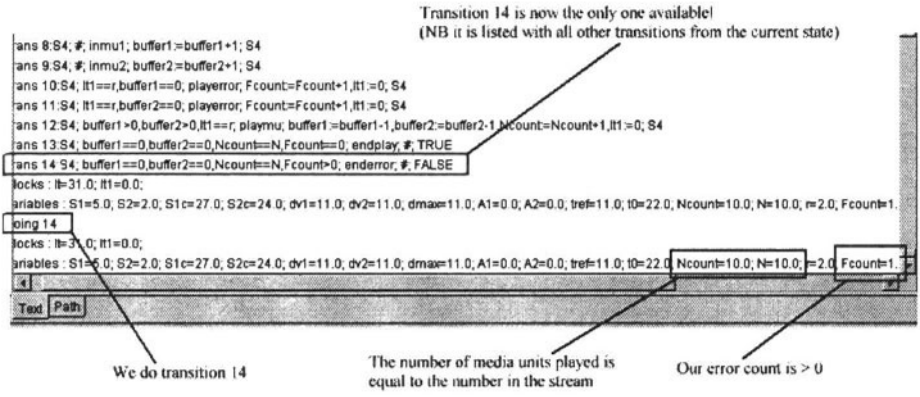

Figure 7. A simulation run resulting in 1 error

To see where this error occurred, we look further back in the log (figure 8). We find an *available* transition (numbered 11) about to be taken. Note that the previous transition (10) is *unavailable* since the guard *buffer1==0* is false. Also observe that the client's play-out deadline (2 time units) has been reached by variable lt1, and that buffer2 is still empty. Hence, although a

media unit from server 1 has already arrived (*buffer1=1*), we cannot play-out since the media unit from server 2 has not yet arrived. We must thus delay the play-out further, report a *playerror* event and increment our error count. No other errors were generated on this run since our final error count was 1.

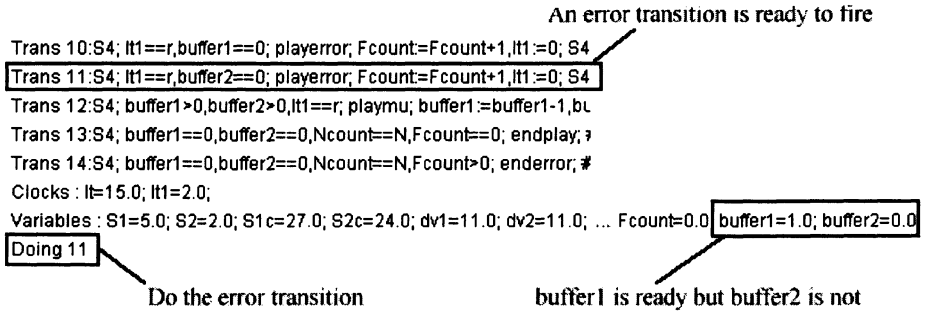

Figure 8. Part of the log showing a *playerror* transition

One point that was not detected in our analysis, however, was that of buffer overflow. Because our buffering requirements were based on the maximum jitter assumption, relaxing this assumption leads to new (larger) buffer requirements; if these requirements are not met, the possibility of buffer overflow exists. Our analysis was based only on a sample of 20 media units which, on simulation, was not sufficient to generate a buffer overflow. We clearly need to greatly increase the number of media units and insert a break-point capability into our tool (allow it to run continually until a certain condition is met). Alternatively, we could look at more complex simulation strategies based on, e.g., visiting each state (or transition) at least once.

To summarise, our relaxation of the maximum jitter bounds has allowed us to demonstrate our stochastic enhancement to timed automata. In the analysis, we have benefited greatly from the use of our tool suite, although we have identified certain desirable extensions. We should also point out that we have focused on real-time analysis and simulation. The former can be extended to probabilistic analysis by incorporating semantics based on probabilistic transition systems, e.g. to allow questions such as what is the probability of reaching a given state. The latter can be extended by including other forms of verification, such as (stochastic) model checking (see below).

7. CONCLUSIONS AND FUTURE WORK

Previously, we have developed an aspect-oriented and multi-paradigm framework for the specification of distributed multimedia systems. This

proved to be successful in a number of case studies, but clearly did not support formal reasoning about the stochastic behaviour of such systems. This is a significant problem, given the predominance of such behaviours in this field. We have therefore extended our previous work, inspired by research into stochastic automata, particularly SPADES. By developing stochastically enhanced timed automata, we have retained compatibility with our framework. We have also described a component-based tool suite, LUSCETA, to support our methodology. LUSCETA currently enables the editing, composition and subsequent simulation of stochastically enhanced timed automata. Being component-based, the architecture is naturally extensible and further developments are planned (see below). With a small amount of additional implementation, the new model will enable us to support our more general multi-paradigm specification architecture, with the stochastically enhanced timed automata providing a common underlying model. The paper has also presented a multimedia example illustrating the use of the enhanced automata model and associated tool suite.

One limitation with the tool suite is that composition is performed before simulating the system, thus leading to the inevitable state-space explosion problem. On-the-fly techniques are currently being considered to improve the ability of the tool suite to handle large systems. A further limitation is that we have only addressed the simulation of our extended automata. Ongoing work is addressing other forms of verification such as stochastic model-checking (incorporating probabilistic transition systems). This work is being carried out in collaboration with the University of Kent at Canterbury (see acknowledgements). In addition, our collaborators have considered the mapping of stochastic automata to timed automata (actually timed automata with deadlines), to enable existing real-time model checking tools and techniques to be exploited [Bryans99]. We plan to extend our tool suite with such techniques as details are finalised. We also plan to investigate the timed automata with deadlines model as a means of eliminating time-locks in a composed system.

ACKNOWLEDGEMENTS

We would like to acknowledge EPSRC's financial support (reference GR/L28890) and the contribution of our V-QoS partners: Howard Bowman, John Derrick and Jeremy Bryans (University of Kent at Canterbury).

REFERENCES

[AOP00] "Aspect-Oriented Programming Home Page", Xerox PARC, 2000. http://www.parc.xerox.com/csl/projects/aop/

[Bernardo96] M. Bernardo, R. Gorrieri, "Extended Markovian Process Algebra", In Proc. CONCUR'96, U. Montanari, V. Sassone (eds), Pisa, Italy, LNCS 1119, pp 314-330, Berlin: Springer, 1996.

[Biersack99] E. Biersack, W. Geyer, "Synchronized Delivery and Playout of Distributed Stored Multimedia Streams", Multimedia Systems, 7(1), pp 70-90, Berlin: Springer, 1999.

[Blair98a] G.S. Blair, L. Blair, H. Bowman, A. G. Chetwynd, "Formal Specification of Distributed Multimedia Systems", London: UCL Press, 1998.

[Blair99a] L. Blair, G.S. Blair, "Composition in Multi-Paradigm Specification Techniques", Proc. 3rd International Workshop on Formal Methods for Open Object-based Distributed Systems (FMOODS'99), P. Ciancarini, A. Fantechi, R. Gorrieri (eds), Florence, Italy, Massachusetts: Kluwer, 1999.

[Blair99b] L. Blair, T. Jones, G.S. Blair, "A Tool Suite for Multi-Paradigm Specification", Short paper, Proc. Fundamental Approaches to Software Engineering (FASE'99), J.-P. Finance (ed), Amsterdam, LNCS 1577, pp 234-238, Berlin: Springer, 1999.

[Blair99c] L. Blair, "A Tool Suite to Support Aspect-Oriented Specification", Aspect-Oriented Programming Workshop at ECOOP'99, Lisbon, June 1999. Workshop report in "Object-Oriented Technology: ECOOP'99 Workshop Reader", A. Moreira, S. Demeyer (eds), LNCS 1743, pp 288-313, Berlin: Springer, 1999. Paper available from http://www.comp.lancs.ac.uk/computing/users/lb/v-qos.html.

[Blair99d] L. Blair, "The Role of Temporal Logic and Timed Automata in Distributed Multimedia Systems", AAAI'99 workshop on Temporal and Modal Logics for the Planning of Open Networked Multimedia Systems (PONMS'99), Cape Cod, MA, 1999. Available from http://www.comp.lancs.ac.uk/computing/users/lb/v-qos.html

[Blakowski96] G. Blakowski, R. Steinmetz, "A Media Synchronization Survey: Reference Model, Specification and Case Studies", IEEE Journal on Selected Areas in Communications, 14(1), Special Issue on Synchronization Issues in Multimedia Communications, pp 5-35, IEEE, 1996.

[Bornot98] S. Bornot, J. Sifakis, S. Tripakis, "Modelling Urgency in Timed Systems", Proc. Compositionality (COMPOS'97), LNCS 1536, Berlin: Springer, 1998.

[Bowman99] H. Bowman, "Modelling Timeouts without Timelocks", In Proc. Formal Methods for Real-Time and Probabilistic Systems (ARTS'99), J.-P. Katoen (ed), Bamberg, Germany, LNCS 1601, pp 334-353, Berlin: Springer, 1999.

[Bravetti99] M. Bravetti, R. Gorrieri, "Interactive Generalized Semi-Markov Processes", In Proc. 7th Int. Workshop on Process Algebras and Performance Modelling (PAPM'99), J. Hillston and M. Silva (eds), pp 83-98, Zaragoza, Spain, Sept. 1999.

[Bryans99] J. Bryans, J. Derrick, "Stochastic Specification and Verification", Proc. 3rd Irish Workshop on Formal Methods, Electronic Workshops in Computing pp20, Springer, 1999.

[D'Argenio98] P.R. D'Argenio, J.-P. Katoen, E. Brinksma, "An Algebraic Approach to the Specification of Stochastic Systems (extended abstract)", In Proc. Programming Concepts and Methods (Procomet'98), D. Gries, W.-P. de Roever (eds), Shelter Island, New York, pp 126-147, Chapman & Hall, 1998.

[D'Argenio99] P.R. D'Argenio, J.-P. Katoen, E. Brinksma, "Specification and Analysis of Soft Real-Time Systems: Quantity and Quality", Proc. 20th IEEE Real-time Systems Symposium, Phoenix, Arizona, pp 104-114, IEEE, December 1999.

[Garavel96] H. Garavel, "An Overview of the Eucalyptus Toolbox", Proc. International Workshop on Applied Formal Methods in System Design, pp 76-88, Maribor, Slovenia, June 1996. See http://www.inrialpes.fr/vasy/Publications/Garavel-96.html.

[Glynn89] P.W. Glynn, "A GSMP Formalism for Discrete Event Simulation", In Proc. of the IEEE, 77(1), pp 14-23, IEEE, 1989.

[Gross85] D. Gross, C.M. Harris, "Fundamentals of Queuing Theory (2nd edition)", New-York: Wiley, 1985.

[Hermanns96] H. Hermanns, V. Mertsiotakis, M. Rettlebach, "A Construction and Analysis Tool based on the Stochastic Process Algebra TIPP", In Proc. TACAS'96, LNCS 1055, Berlin: Springer p. 427-430, 1996.

[Hillston96] J. Hillston, "A Compositional Approach to Performance Modelling", Distinguished Dissertation in Computer Science, Cambridge University Press, 1996.

[ISO89] ISO, "Information Processing Systems - Open Systems Interconnection - LOTOS, A Formal Description Technique based on the Temporal Ordering of Observational Behaviour", ISO8807, International Organisation for Standardisation, Geneva, 1989.

[Jones99a] T. Jones, L. Blair, "A tool-suite for simulating, composing and editing timed automata (LUSCETA: Users manual release 1.0)", Internal Report MPG-99-24, 1999. Available at http://www.comp.lancs.ac.uk/computing/users/jonest/default.htm.

[Jones99b] T. Jones, "Case study: Synchronized delivery and playout of distributed stored multimedia streams", Powerpoint presentation, 1999. Available from http://www.comp.lancs.ac.uk/computing/users/jonest/default.htm

[Larsen95] K.G. Larsen, P. Pettersson, W. Yi, "Diagnostic Model-Checking for Real-Time Systems", Proc. 4th DIMACS Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995.

[Marsan95] M.A. Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, "Modelling with Generalized Stochastic Petri Nets", John Wiley & Sons, 1995.

[Murphy99] G.C. Murphy, R.J. Walker, E.L.A. Baniassad, "Evaluating Emerging Software Development Technologies: Lessons Learned from Assessing Aspect-Oriented Programming", IEEE Transactions on Software Engineering (Special Section on Empirical Software Engineering), 25(4), pp 438-455, IEEE, July/August 1999.

APPENDIX A. COMPOSITION

Composition of our stochastically enhanced timed automata follow the parallel composition rules defined below. SA is the set of actions on which the execution of each automaton should synchronise, and $sf(a_1,a_2)$ is the synchronisation function governing the synchronisation of events a_1 and a_2 (see [ISO89]). Let $s_1=(l_1,u_{c1},u_{v1},f_{s1})\in TA_1$ and $s_2=(l_2,u_{c2},u_{v2},f_{s2})\in TA_2$, where $TA_1=\langle S_1,s_{01},\rightarrow,I\rangle$ and $TA_2=\langle S_2,s_{02},\rightarrow,I\rangle$ are stochastically enhanced timed automata. Let $L = Act \cup \Delta$ and $a\in L$. Composition rules for TA_1 and TA_2 are:

Table 2. Composition rules

$\dfrac{s_1 \xrightarrow{g,a,r} s_1',\ a \notin SA}{s_1 \|\|_{SA} s_2 \xrightarrow{g,a,r} s_1' \|\|_{SA} s_2}$	1
$\dfrac{s_2 \xrightarrow{g,a,r} s_2',\ a \notin SA}{s_1 \|\|_{SA} s_2 \xrightarrow{g,a,r} s_1 \|\|_{SA} s_2'}$	2
$\dfrac{s_1 \xrightarrow{g1,a1,r1} s_1', s_2 \xrightarrow{g2,a2,r2} s_2',\ a \in SA}{s_1 \|\|_{SA} s_2 \xrightarrow{g,a,r} s_1' \|\|_{SA} s_2'}$ where $sf(a1,a2) = a$, $g = g1 \wedge g2$ and $r = r1 \cup r2$	3
$\dfrac{s_1 \xrightarrow{\varepsilon(d)} s_1', s_2 \xrightarrow{\varepsilon(d)} s_2'}{s_1 \|\|_{SA} s_2 \xrightarrow{\varepsilon(d)} s_1' \|\|_{SA} s_2'}$	4

The resulting composition can be defined as:

$$Comp = \langle S, s_0, \longrightarrow, I \rangle$$

where S is a finite set of states s.t. $s_1 \|\|_{SA} s_2 \in S \Leftrightarrow s_1 \in S_1$ and $s_2 \in S_2$,

$s_0 \in S$ is the initial state s.t. $s_0 = s_{01} \|\|_{SA} s_{02}$,

$\longrightarrow$ is the transition relation governed by rules 1-4 above,

$I(s_1 \|\|_{SA} s_2) = I(s_1) \wedge I(s_2)$

With respect to the resampling of variables, recall that the function sv(l) returned all sampling variables contained in the outgoing guards or invariant of location l. Consequently, for rule 1: $\forall f_i \in sv(l_1)$, $f_i \blacktriangleleft F_i$, for rule 2: $\forall f_i \in sv(l_2)$, $f_i \blacktriangleleft F_i$, for rule 3: $\forall f_i \in sv(l_1) \cup sv(l_2)$, $f_i \blacktriangleleft F_i$, and finally for rule 4: $f' = f$, i.e. no sampling variables are re-sampled.

APPENDIX B. AUTOMATA FROM SECTION 5

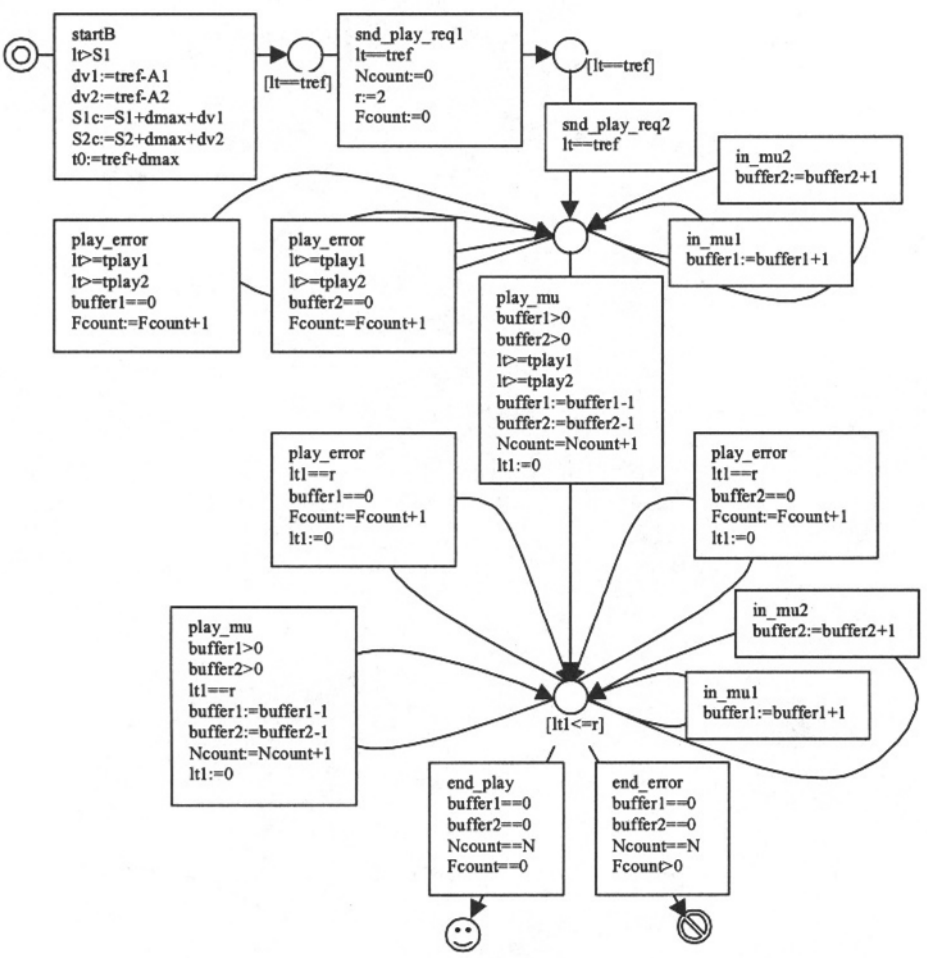

Figure B.1. Client automaton for the second phase

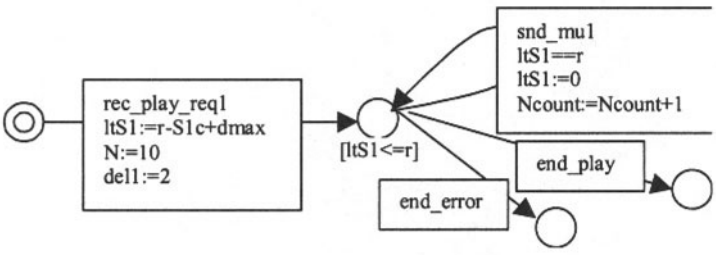

Figure B.2. Server automaton for the second phase