Verification of Timeliness QoS Properties in Multimedia Systems

Behzad Bordbar[1] and Kozo Okano[2]

[1] University of Birmingham
B.Bordbar@cs.bham.ac.uk
[2] Osaka University
okano@ist.osaka-u.ac.jp

Abstract. One of the main challenges of the design of object-based Distributed Multimedia Systems is to address the performance related issues such as the Quality of Service (QoS). The specification of QoS is a crucial part of architectural object-based methods such as Open Distributed Processing (ODP). In the ODP, a QoS property assigned to an object is modelled via two clauses of *required* and *provided* QoS statements, which specify the level of QoS required/provided by an object from/to its environment, respectively. An over-demanding QoS statement can be beyond the physical limitation of the system and might result in inconsistencies. In particular, to produce a correct design, it is crucial to study the effect of QoS statements of components on the overall behaviour of the system in earlier stages of the design.
This paper develops a theory for the verification of *Timeliness QoS* properties such as Jitter, Throughput and Latency. The approach adopted is based on the idea of *Test Automata*. We shall present a formal definition of *Timeliness QoS* properties, which is used for the creation of Test Automata. Such Test Automata, which we shall refer to as *QoS Timed Automata*, can be used to verify the corresponding QoS Timeliness property. The method is illustrated by the verification of Throughput in a Video Player systems via the model checker UPPAAL.

Keywords: QoS, Network of Timed Automata, Real-time System, Verification, Model checker UPPAAL

1 Introduction

Since modern Distributed Multimedia systems are object-based, functional behaviour of such systems is encapsulated within multiple components. Quality of Service (QoS) properties, which can be seen as a set of contracts on the system, are end-to-end issues, i.e. a QoS requirement is related to the systems as a whole. As a result, a major challenge of the integration of QoS in the design process of object-based distributed systems is to specify suitable QoS characteristics for each component of the system such that; if the QoS characteristics of components are satisfied, then the QoS requirement of the whole system is satisfied. In particular, it is important to ensure that under the specification of the functional behaviour of the system the QoS is achievable.

J.S. Dong and J. Woodcock (Eds.): ICFEM 2003, LNCS 2885, pp. 523–540, 2003.

The current paper builds on earlier works [1, 11, 12], which present a method of specification of QoS in ODP [21] design of Distributed Multimedia Systems. Our aim is to present a method of verification of *Timeliness* QoS statements such a Jitter, Throughput and Latency, which are boolean functions on the set of sequences of time of occurrence of events. The adopted approach is based on the idea of *Test Automata* [2–4, 16]. Assume that the functional behaviour of the system is modelled via (a network of) Timed Automata [5, 9] $\mathscr{A}$. Starting from a Timeliness property ϕ related to the time of occurrence of external event $e_1, \ldots, e_K$ of $\mathscr{A}$, we shall present a network of Timed Automata $QTA(\phi, e_1, \ldots, e_K)$, called *QoS Timed Automata,* which will be used to verify the property ϕ on $\mathscr{A}$. The QoS Timed Automata is such that $\mathscr{A}$ satisfies the property ϕ if and only if $QTA(\phi, e_1, \ldots, e_K) \mid\mid \mathscr{A}$ does not reach to a global state with a coordinate **failure**, where the location **failure** of $QTA(\phi, e_1, \ldots, e_K)$ represents the violation of ϕ. In practice, using QTA transfers the problem of verifying a QoS statement of a distributed system into a reachability analysis in a network of Timed Automata, which can be carried out via model checkers. In this paper, we shall use UPPAAL [6, 9], which has been successfully applied to the verification of real-time systems [8, 14, 18].

The paper is organised as follows. Section 2 presents a brief introduction to Timed Automata and UPPAAL. Section 3, presents a formal definition for QoS Timeliness properties and QoS Timed Automata (QTA). Theorem 1, the main result of the paper, proves that a QTA is a Test Automata [2–4, 16]. Section 4 applies our approach to the verification of throughput in an example of a Video Player system. Section 5 presents a proof for **Theorem** 1. The final two sections discuss some related works and draw a conclusion.

2 Timed Automata with Data Variables and UPPAAL

In this section, we shall review a variation of Timed Automata model proposed by Alur and Dill [5], which is used in UPPAAL [6, 9, 16, 17], a tool for the verification of behavioural properties of real-time systems.

Consider a set of *Completed Actions*, denoted by CA, which specify internal actions of a component of the system modelled via Timed Automata. In the UPPAAL model, Timed Automata (components) communicate via simple CCS [20] style point-to-point communication. As a result, consider a set of *Half Actions*, $HA = \{x?, x! \mid x \in CA\}$. Let $\mathbb{A}$ denotes the set of *all* actions of the system consist of all half actions and complete actions, i.e. $\mathbb{A} = HA \cup CA$. $x \in CA$. Underlying actions are defined via the function $\downarrow\colon \mathbb{A} \to \mathbb{A}$ defined by $\downarrow (x!) = \downarrow (x?) = \downarrow (x) = x$ for all $x \in CA$. If there is no fear of confusion, we shall sometimes drop parentheses and write $\downarrow x!, \downarrow x?$ or $\downarrow x$. Moreover, for $A \subset \mathscr{A}$, $\downarrow A = \{\downarrow y \mid y \in A\}$.

Suppose that $\mathscr{C}$ is a set of clock variables, with values in $\mathbb{R}^{\geq 0}$ and $\mathscr{D}$ is a set of data variables, with integer values. Let $c(\mathscr{C} \cup \mathscr{D})$ denotes the conjunction of boolean expressions over atomic formulae of the form $x \sim q$ or $x - y \sim q$ or $i \sim n$, where $x, y \in \mathscr{C}$, $i \in \mathscr{D}$, q is a rational number, $n \in \mathbb{N} = \{0, 1, \ldots\}$ a

natural number and $\sim \in \{\leq, \geq, =, <, >\}$. In what follows the term *variable* refers to both data and clock variables.

A *valuation* (*variable assignment*) is a map $v : c(\mathscr{C} \cup \mathscr{D}) \to \mathbb{R}^{\geq 0} \cup \mathbb{N}$, which assigns to each clock a non-negative real-number and to a data variable a natural number. For a valuation v, a delay $d \in \mathbb{R}^{\geq 0}$, which is denoted by $v+d$, is defined as $(v+d)(x) = v(x)+d$, if x is a clock and $(v+d)(i) = v(i)$, if i is a data variable. In other words, all clocks operate with the same speed and data variables are time-insensitive. If $A \neq \emptyset$ is a set of variables, i.e. $A \subset \mathscr{C} \cup \mathscr{D}$, the set of valuations on A is denoted by $\mathscr{V}(A)$. For nonempty sets of variables A, B and valuations $v_1 \in \mathscr{V}(A)$ and $v_2 \in \mathscr{V}(B)$ if $v_1(x) = v_2(x)$ for all $x \in A \cap B$, we define $v_1 \cup v_2 \in \mathscr{V}(A \cup B)$ by $v_1 \cup v_2(x) = v_1(x)$ if $x \in A$ and $v_1 \cup v_2(y) = v_2(y)$ if $y \in B$.

The value of clock or data variable can be *reset*. A reset statement is of the form $x := e$, where x is a clock or a data variable and e is an expression. In the current version of UPPAAL, for a clock, e must be a natural number, and for a data variable, e must be in the form of $cy + c'$, where c and c' are constant integer and y is a data variable. A set of reset statements is called a *reset-set* or *reset* if each variable is assigned at most once. The result of applying a reset r to a valuation v is denoted by the valuation $r(v)$. If a variable x is such that no assignment of r changes its value then $v(x) = r(v)(x)$. Let $\mathscr{R}$ denotes the set of all resets. If $r_1, r_2 \in \mathscr{R}$, then $r_1 \cup r_2 \in \mathscr{R}$, if $r_1 \cup r_2$ assigns at most one value to each variable. A *Timed Automaton* $\mathscr{A}$ is a 6-tuple (L, l_0, T, I, C, D, A) such that

- L is a finite set of *locations* and $l_0 \in L$ is a designated location called the *initial location.*
- $C \subset \mathscr{C}$, $D \subset \mathscr{D}$ and $A \subset \mathbb{A}$ are finite sets of clock variables, data variables and actions, respectively.
- $T \subset L \times A \times c(C \cup D) \times \mathscr{R} \times L$ is a transition relation. An element of T is of the form of (l_1, a, g, r, l_2), where $l_1, l_2 \in L$ are locations of Timed Automaton, $a \in \mathbb{A}$ is an action, $g \in c(C \cup D)$ is called a *guard*, and $r \in \mathscr{R}$ is a set of reset statement. We sometimes write $l_1 \xrightarrow{e,g,r} l_2$ to depict that $\mathscr{A}$ evolves from a location l_1 to a new location l_2, if the guard g is evaluated *true*, the action a is performed and clocks and data variables are reset according to r.
- $I : L \to c(C \cup D)$ is a function that assigns to each location an *invariant.* Intuitively, a timed automata can stay in a location while its invariants are satisfied. The default invariant for a location is *true* $(x \geq 0)$.

For each Timed Automaton $\mathscr{A}$, we shall write $Location(\mathscr{A})$, $Clock(\mathscr{A})$, $Data(\mathscr{A})$ and $Act(\mathscr{A})$ to denote the set of *locations*, *clocks*, *data variables* and *actions* of $\mathscr{A}$, respectively.

The semantics of Timed Automata can be interpreted over transition systems, i.e. triple $(S, s_0, \Rightarrow)$, where

- $S \subset L \times \mathscr{V}$ is the set of *states*, i.e. each state is a pair (l, v), where l is a location and v is a valuation
- $s_0 \in S$ is an *initial state*, and
- $\Rightarrow \subset S \times (Act(\mathscr{A}) \cup \mathbb{R}^{\geq 0}) \times S$ is a *transition relation.*

A transitions can be either a *discrete transitions*, e.g. (s_1, e, s_2), where $e \in Act(\mathscr{A})$ or a *time transitions*, *e.g.* (s_1, d, s_2), where $d \in \mathbb{R}^{>0}$ and denotes the passage of d time units. Transitions are written: $s_1 \overset{e}{\Rightarrow} s_2$ and $s_1 \overset{d}{\Rightarrow} s_2$, respectively, and are defined according of the following inference rules:

$$\frac{l_1 \xrightarrow{e,g,r} l_1', g(v)}{(l_1, v) \overset{e}{\Rightarrow} (l_2, r(v))} \qquad \frac{\forall d' \leq d \quad I(l)(v + d')}{(l, v) \overset{d}{\Rightarrow} (l, v + d)}$$

A direct results of the above definition is the Time Additivity Axiom [23].

Time Additivity Axiom: For every $s_1, s_2 \in S$ and $d_1, d_2 \in \mathbb{R}^{\geq 0}$, $s_1 \overset{d_1+d_2}{\Rightarrow} s_2$ if and only if there is a state s_3 such that $s_1 \overset{d_1}{\Rightarrow} s_3$ and $s_3 \overset{d_2}{\Rightarrow} s_2$.

To model concurrency and synchronisation between Timed Automaton, CCS [20] style parallel composition operators are introduced, which synchronise over half actions. Suppose that $\mathscr{A}_1, \ldots, \mathscr{A}_n$ are Timed Automata, the parallel composition $\mathscr{A} := \mathscr{A}_1 \mid\mid \mathscr{A}_2 \mid\mid \cdots \mid\mid \mathscr{A}_n$ is referred to as a *network of Timed Automata* [6, 9, 16, 17]. The semantics of a network of Timed Automata can be expressed via a transition system $(S, s_0, \Rightarrow)$. A state $s \in S$ is of the form $s = (\boldsymbol{l}, v)$ where $\boldsymbol{l} = (l_1, \ldots, l_n)$, in which each l_i is location of $\mathscr{A}_i$ and v is a valuation on $\cup_i (Clock(\mathscr{A}_i) \cup Data(\mathscr{A}_i))$. $s_0 = (\boldsymbol{l}_0, v_0)$ is the *initial location*, where $\boldsymbol{l}_0$ is the vector of initial location of the components and v_0 is the a valuation compatible with the initial valuation of the components, i.e. $u_0 \mid_{Clock(\mathscr{A}_i) \cup Data(\mathscr{A}_i)}$ is the initial valuation of the i-th component.

Let for a vector of location $\boldsymbol{l} = (l_1, \ldots, l_n)$, $\boldsymbol{l}[l'_i/l_i]$ denotes the vector of location created by replacing l_i with l'_i, then $\Rightarrow$ is defined via the following inference rules:

- For a completed action a which belongs to a component $\mathscr{A}_i$, i.e. $a \in Act(\mathscr{A}_i) \cap CA$, $(\boldsymbol{l}, v) \overset{a}{\Rightarrow} (\boldsymbol{l}[l'_i/l_i], r_i(v))$, if $l_i \xrightarrow{a,g_i,r_i} l'_i$ and $g_i(v)$ [1]
- Suppose that $x!$ and $x?$ are half actions of $\mathscr{A}_i$ and $\mathscr{A}_j$ where $i \neq j$. $(\boldsymbol{l}, v) \overset{x}{\Rightarrow} (\boldsymbol{l}[l'_i/l_i, l'_j/l_j], r_i \cup r_j(v))$, if $l_i \xrightarrow{x!,g_i,r_i} l'_i$, $l_j \xrightarrow{x?,g_j,r_j} l'_j$, $g_i(v)$, $g_j(v)$ and $r_i \cup r_j \in \mathscr{R}$.
- For $d \in \mathbb{R}^{\geq 0}$, $(\boldsymbol{l}, v) \overset{d}{\Rightarrow} (\boldsymbol{l}, v + d)$, if $I(l_i)(v + d')$ for all i, and all $d' \leq d$.

Presence of urgent channels and committed locations may overrule the above transitions as follows. In a state where two components may synchronise of an urgent channel, no further delay is allowed. If in a state, one of the components is in a location labelled as being committed, no delay is allowed to occur and any discrete transition *must* invoke. In this paper, we shall not use any urgent action or committed state.

Assume that $\mathscr{A}$ is a network of Timed Automata. A *run* σ of $\mathscr{A}$ is a finite/infinite sequence of transitions of the form $s_0 \overset{\lambda_1}{\Rightarrow} s_1 \overset{\lambda_2}{\Rightarrow} s_2 \cdots$ where s_0 is the initial state and $\lambda_i \in \mathrm{Act}(\mathscr{A}) \cup \mathbb{R}^{\geq 0}$. For each state s_j of the run σ, define

[1] Note that g_i is a function defined on valuations of the a component timed automata $\mathscr{A}_i$. As a result, $g_i(v)$ is an abbreviation for $g_i(v \mid_{Clock(\mathscr{A}_i) \cup Data(\mathscr{A}_i)})$.

$TimeStamp(\sigma, s_j) = \sum_{i=1}^{j-1}\{\lambda_i \mid \lambda_i \in \mathbb{R}^{\geq 0}\}$. Similarly, for an action λ_j define $TimeStamp(\sigma, \lambda_j) := TimeStamp(\sigma, s_j)$, which denotes the time of occurrence of λ_j. We shall denote the set of all runs of $\mathscr{A}$ with $Run(\mathscr{A})$. Assume that σ_1 and $\sigma_2 \in Run(\mathscr{A})$ are such that $\sigma_1 := s_0^1 \stackrel{\lambda_1}{\Rightarrow} s_1^1 \stackrel{\lambda_2}{\Rightarrow} s_2^1 \cdots$ and $\sigma_2 := s_0^2 \stackrel{\mu_1}{\Rightarrow} s_1^2 \stackrel{\mu_2}{\Rightarrow} s_2^2 \cdots$. We say σ_1 and σ_2 are *identical* if they have the same length, i.e. either both are infinite length or both have the same length, and for each i, $s_i^1 = s_i^2$ and $\lambda_i = \mu_i$. However, by Time Additivity Axiom, two runs $\sigma_1 := s_0 \stackrel{\frac{1}{2}}{\Rightarrow} s_1 \stackrel{\frac{1}{2}}{\Rightarrow} s_2$ and $\sigma_2 := s_0 \stackrel{1}{\Rightarrow} s_2$, although not identical, are equal. In this paper, two runs are called **equal,** if they are equal up to Time Additivity Axiom, i.e. applying Time Additivity Axiom to one of them results in an identical run with the other. If l is a location of $\mathscr{A}$, we say a run $\sigma \in Run(\mathscr{A})$ *meets* the location l, if there is a state $s_i = (l, v)$ in σ.

3 Verification of QoS Timeliness Properties

Assume that e is an action of the system, a Timeliness property for e is defined to be a property related to the time of occurrence of e [10]. For example, if the action e marks the dispatch of frames from a communication channel, the throughput of 25 frames per sec. can be seen as a property of the time sequence $\{t_1, t_2, \ldots\}$ of the time of the occurrence of e such that

$$\forall n \quad \mid t_{n+25} - t_n \mid < 1000, \tag{1}$$

where time is measured in msec.

In general, the sequence of time of occurrence of events are finite or infinite sequences of non-decreasing, non-negative real numbers.

Definition 1. *For $n \in \mathbb{N}$, let $\Gamma^n = \{\{t_i\}_{i=1}^n \mid 0 \leq t_1 \leq t_2 \leq \cdots \leq t_n\}$.*
Let $\Gamma = \bigcup_{n=1}^{\infty} \Gamma^n \cup \{\emptyset\}$, where $\emptyset$ is the empty set.

Suppose that $\mathscr{A}$ is a a network of Timed Automata and $\sigma = s_0 \stackrel{\lambda_1}{\Rightarrow} \cdots \stackrel{\lambda_n}{\Rightarrow} s_n \cdots$ is a finite/infinite run of $\mathscr{A}$, where for each i, s_i is a state and $\lambda_i \in act(\mathscr{A}) \cup \mathbb{R}^{\geq 0}$.

Definition 2. *For each action $e \in Act(\mathscr{A})$, if e is an event occurring as $\{\lambda_i\}$, let $Time(\sigma, e, n)$ denotes the time of n-th occurrence of e in the run σ, i.e. $Time(\sigma, e, n) = TimeStamp(\sigma, s_j) = \sum_{i=1}^{j-1}\{\lambda_i \mid \lambda_i \in \mathbb{R}^{\geq 0}\}$. Let the sequence $Time(\sigma, e) := \{Time(\sigma, e, n) \mid n \in \mathbb{N}\}$.*

Clearly, for each run σ and each action $e \in \mathscr{A}$, $Time(\sigma, e) \in \Gamma$. In particular, if e does not appear in σ, then $Time(\sigma, e) = \emptyset (\in \Gamma)$ is the empty sequence. Now, we shall present a formal definition of Timeliness properties as boolean functions on the set of time sequences.

Definition 3. *A Timeliness property of degree $K \geq 1$, is a function $\phi : \Gamma^K \longrightarrow \{\mathbf{T}, \mathbf{F}\}$, where $\Gamma^K = \overbrace{\Gamma \times \Gamma \times \cdots \times \Gamma}^{K}$.*

Example 1. The throughput of 25 frames per sec. for e can be expressed via the Timeliness property ϕ of degree 1, defined by

$$\phi(t) = \begin{cases} \mathbf{T} & \forall n \mid t_{n+25} - t_n \mid < 1000 \\ \mathbf{F} & \text{otherwise} \end{cases}.$$

As a result, throughput is a Timeliness property of degree 1. It can be seen QoS statements such as various types of Jitter [13] are Timeliness properties of degree 1, whereas latency is a Timeliness property of degree 2. In general, it seems that, any property related to the relative time of occurrence of K events can be evaluated via a Timeliness property function of degree K. Assume that the functional behaviour of system is modelled via a network of Timed Automata. For a property to satisfy, it must satisfy for all runs of the network of Timed Automata.

Definition 4. *Assume that $\mathscr{B}$ is a Timed Automaton such that $e_1, \ldots, e_K \in Act(\mathscr{B})$. Suppose that ϕ is a Timeliness property of the degree K. $\mathscr{B}$ satisfies ϕ for $e_1, \ldots, e_K$ iff for each run σ of $\mathscr{B}$, $\phi(\text{Time}(\sigma, e_1), \ldots, \text{Time}(\sigma, e_K)) = \mathbf{T}$. In this case, we say σ satisfies ϕ.*

The main focus of this paper is on Timeliness properties which express QoS statements. The outline of our approach is as follows. We start from a Timeliness statement ϕ and create a network of Timed Automata such that all its runs that do not meet a state called **failure**, satisfy ϕ. Moreover, all runs of the network of Timed Automata that *meet* **failure** violate ϕ. This ensures that the network of Timed Automata fully represents the property ϕ.

Definition 5. *Assume that ϕ is a Timeliness properties of degree K expressing a QoS statement on the set of actions $e_1, e_2, \ldots, e_K$. A* QoS Timed Automaton *corresponding to ϕ and $e_1, \ldots, e_K$ is a network of Timed Automaton $\mathscr{A} = QTA(\phi, e_1, e_2, \ldots, e_K)$ such that*

1. *$\mathscr{A}$ contains a distinct location* **failure***;*
2. *for each run σ of $\mathscr{A}$, if σ does not meet a* **failure** *state, i.e. a vector of locations with at least one co-ordinate* **failure***, σ satisfies ϕ.*
3. *for sequences, $t^1, t^2, \ldots . t^K \in \Gamma$ that satisfy ϕ, there is a run σ of $\mathscr{A}$ such that*
   (a) σ does not meet **failure***; and*
   (b) for each i $\text{Time}(\sigma, e_i) = t_i$;
4. *for finite sequences $t^1, t^2, \ldots, t^K \in \Gamma$, if ϕ does not satisfy $t^1, t^2, \ldots, t^K$, then there is a finite run σ of $\mathscr{A}$ such that σ ends in* **failure** *and $t^i = \text{Time}(\sigma, e_i)$.*

It poses as a question that which Timeliness properties correspond to QoS Timed Automata (QTA). Since Timeliness properties are boolean functions on Γ^K, the cardinality of the set of Timeliness properties is $\geq 2^{2^{\aleph_0}}$. Notice, the cardinality of Γ^K is the same as the cardinality of Γ, which is $\geq 2^{\aleph_0}$. The cardinality of the set of timed automata is $2^{\aleph_0}$ the majority of Timeliness properties are not in correspondence with any QTA. The question of characterisation of

all timeliness properties which can be translated to QTA is highly nontrivial. [2–4] adapts a Temporal logic approach to characterise all properties which are testable via timed automata. However, considering that the Timed Automata model of [2–4] does not include data variables, further research is required to characterise all Timeliness properties corresponding to QTA.

A Timeliness property ϕ deals with the time of occurrence of external events $e_1, \dots e_K$. Since actions are atomic, two consecutive external actions in a run σ have identical Timestamps. As a result, the order of occurrence of such events has no effect on "σ satisfies ϕ," when ϕ is of degree ≥ 2, i.e. the property ϕ can not differentiate between two runs which are identical except the order of consecutive actions with the same Timestamp. Consequently, it is important for a QTA to include all permutations of such actions.

Definition 6. *Suppose that $\mathscr{A}$ is a QTA corresponding to a Timeliness property ϕ and events $e_1, \dots e_K$. $\mathscr{A}$ is called a* Complete QTA *if for each run $\sigma := s_0 \stackrel{\lambda_1}{\Rightarrow} s_1 \stackrel{\lambda_2}{\Rightarrow} s_2 \cdots \stackrel{\lambda_n}{\Rightarrow} s_n$ of $\mathscr{A}$ with consecutive actions $\lambda_i, \lambda_{i+1} \in Act(\mathscr{A}) \cap \{e_1, \dots e_K\}$, the run $\sigma' := s_0 \stackrel{\lambda_1}{\Rightarrow} s_1 \stackrel{\lambda_2}{\Rightarrow} s_2 \cdots s_i \stackrel{\lambda_{i+1}}{\Rightarrow} s' \stackrel{\lambda_i}{\Rightarrow} s_{i+1} \cdots \stackrel{\lambda_n}{\Rightarrow} s_n$, which is created from changing the order of occurrence of λ_i and λ_{i+1}, is also a run of $\mathscr{A}$.*

The next theorem which is the main result of the paper, uses parallel composition of a QTA and the network of Timed Automata representing the functional behaviour of the system to verify Timeliness properties. In effect, the next theorem states that each QTA is a Test Automata [2–4].

Theorem 1. *Assume that $\mathscr{B}$ is a network of Timed Automaton such that $e_1, \dots, e_K \in Act(\mathscr{B})$. Suppose that ϕ is a Timeliness property of the degree K, for which a complete QTA $\mathscr{A}$ exists. $\mathscr{B}$ satisfies ϕ for $e_1, \dots, e_K$ if and only if no run of $\mathscr{A}' \parallel \mathscr{B}'$ meet the state* **failure**, *where $\mathscr{A}' = QTA(\phi, e_1, e_2, \dots, e_K)$ $[e_1?/e_1, \dots, e_K?/e_K]$ and $\mathscr{B}' = \mathscr{B}[e_1!/e_1, \dots e_K!/e_K]$, created from $\mathscr{A}$ and $\mathscr{B}$, respectively, by replacing $e_1, \dots, e_K$ with half actions.*

Proof: See section 5.

4 Verification of QoS for a Video Player System

In this section, we shall apply our results to verification of Timeliness QoS statements on a model of a Video Player system. Fig. 1 depicts a process oriented view of a Video Player system. The system consists of four components *Video Source, Buffer, Decoder* and *QoS Controller* that can be explained as follows.

Video Source: models the application that produces streams of video packets. The dispatch of each video packet is abstracted as the emission of a signal *packet!*. Fig. 2 depicts the behaviour of the *Video Source* as a Timed Automaton, which dispatches *packet!* signals with the periods T_0. The variable R_P, which models the rate of the creation of the packets, is used by the *QoS controller*.

Buffer: (in Fig. 3) receives *packet?* signals and emits *o_packet*! signals in periods of T_0. The number of *packet*? in the buffer is denoted by c. If c is equal to L,

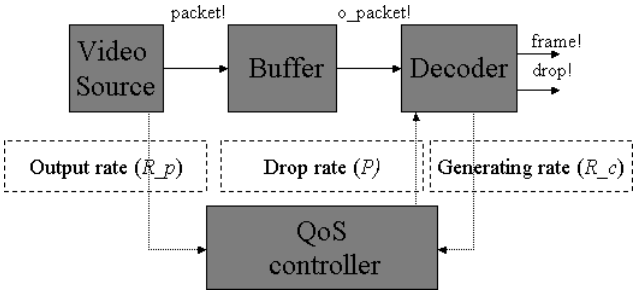

Fig. 1. A Process Oriented View of the Video Player System.

the length of the buffer, the buffer is full and the next signal causes an overflow of the buffer, which results an *Exception* being thrown. One of the objectives of the design of functional behaviour is to avoid an overflow of the buffer.

Decoder: (in Fig. 5) is used to convert arriving packets into video frames. For the purpose of simplicity, we assume that each frame consists of a single packet. On creation of a frame a half action *frame!* is emitted, which can be used to synchronise with the display driver. It takes at most T_1 unit of time and at least T_0 unit time to generate a frame from an arriving packet. The Decoder also generates *drop!* signals, which mark failure of generation of a frame. The emission of a *drop!* signal is controlled by two local variables r and p, and a global variable P. The value of variable P represents the drop rate ratio. For example for the drop rate of $\frac{1}{5}$, the value of P is equal to five, which denotes that one out of five frames are dropped. In this case the Timed Automaton creates one *drop!* in every five output signals.

The value of a global variable R_c, which shows the current rate of performance of Decoder, is incremented to mark the creation of a frame. The value is also periodically reset by QoS controller.

QoS Controller: (in Fig. 4) controls the drop rate P of the *Decoder*. To synthesise the controller, within each *unit* time, the current rate of the system performance R_p and R_c are compared. If $R_p - R_c > \theta_0$, the value of P is incremented. If $R_c - R_p > \theta_1$, the value of P is decremented. θ_0 and θ_1 are constant threshold values.

One of the outputs of the above Video Player example is a signal *frame!* representing the creation of a single frame. This signal is used to synchronise the Video Player with a display drive. For a display drive to present a high quality pictures, it is required that signals *frame!* are dispatched with a suitable throughput. In general, the QoS characteristic throughput of an event e is referred to as a lower bound or an upper bound time on the number of occurrences of the event e [10]. For the rest of the current section, we shall demonstrate our approach by an example of verification of the QoS Throughput. Formally, a Throughput of $k \in \mathbb{N}\backslash\{0\}$ within T_0 and T_1 unit of time ($T_0 < T$) is defined by

$$\forall n \; T_0 \leq \tau(e?, n + K) - \tau(e?, n) \leq T, \tag{2}$$

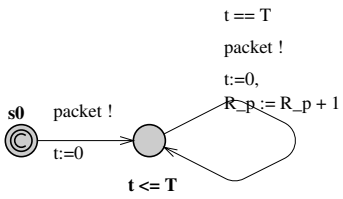

Fig. 2. Video Source represented in TA.

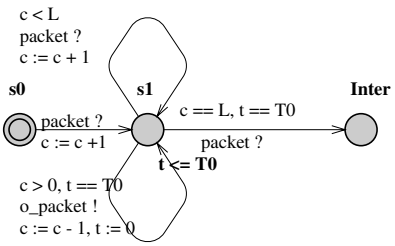

Fig. 3. RBuff represented in TA.

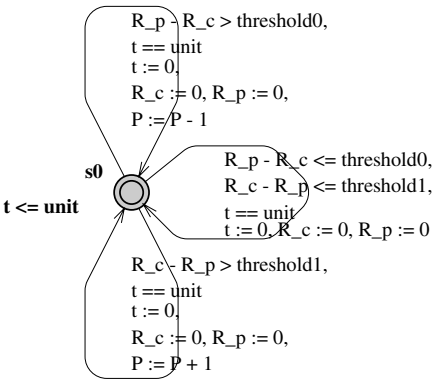

Fig. 4. QoS controller represented in TA.

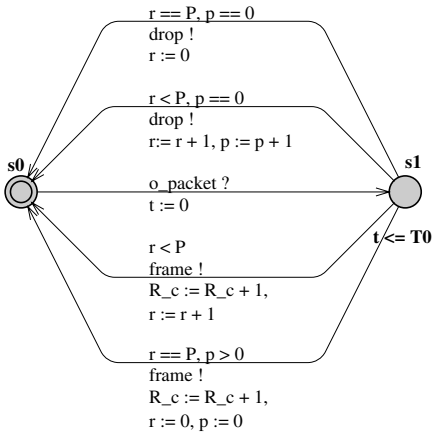

Fig. 5. Decoder represented in TA.

where $\tau(e?, n)$ denotes the time of the n-th occurrence of the event $e?$ in the system.

Example 1 expresses the throughput of 25 frames per sec. as a Timeliness property. Similarly, the general form of throughput, equation (2), can be written as a Timeliness property of degree 1. Our next aim is to present a QTA $\mathscr{A}$ corresponding to throughput that satisfies the definition 5. The first requirement is that all runs of $\mathscr{A}$ should be such that the time for the occurrence of e satisfies the equation 2. A solution is to create k clocks $t_0, t_1, \ldots, t_{k-1}$ and use each clock t_i to measure the time difference between the j-th and $i+k$-th occurrence of e, in a periodic form.

Fig. 6 represents a Timed Automata that checks if two consecutive occurrences of a signal $e?$ are within T_0 and T_1 units of each other. In order to check the Throughput, we require k parallel composition copies of the Timed Automata of Fig. 6. Each such copy of the Timed Automata of Fig. 6 has an index, denoted by i, which acts as an *identifier*. There is a global variable c, which determines which copy can fire an action $e?$. For example, in location **active**, if $c == i$ and $e?$ occurs within the period of $[T_0, T]$, a transition fires which sets the value

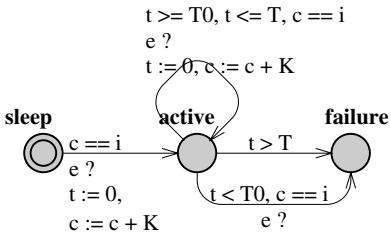

Fig. 6. QTA to measure time difference between corresponding events e.

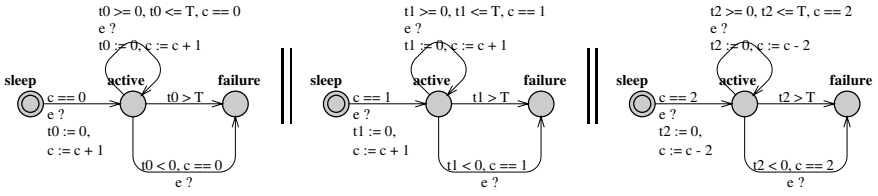

Fig. 7. QTA for the Throughput with $K = 3$.

of c to $c + K$. This means that, if the condition $T_0 \leq t \leq T$ is satisfied, *only* the copy of the Timed Automata of Fig. 6 with the index $i + K$ can fire. For example, the QTA for the Throughput of at least $K = 3$ within L unit of time, *i.e.*, $\forall n \quad 0 \leq \tau(e, n + 2) - \tau(e?, n) \leq T$ can be modelled via the network of Timed Automata depicted in Fig. 7.

The QTA of Fig. 7 works as follows. At first the value of the counter c is 0, therefore, if an action $e?$ occurs, then the Left Hand Side (LHS) Timed Automata changes its location to **active**, because condition $c == 0$ holds. Thus, LHS Timed Automata increments counter c by 1 resetting its own clock t_0. At this moment, if another action $e?$ occurs, then the Timed Automata in the middle changes its location due to *active*. It also increments the counter c by 1. Finally, the Right Hand Side Timed Automata changes its location on arriving the third action $e?$, because condition $c == 2$ holds. At this point, since $c := c - 2$, the value of c is set to 0. Now, if the fourth action e occurs within the period $[0, T]$, LHS Timed Automata again fires and c is again incremented from 0 to 1.

It can easily be seen that the network of Timed Automata of Fig. 7 satisfies the definition 5 and hence is the QTA for the throughput.

The rest of the current section demonstrates our method for the verification of Throughput of *frame?* signals in the Video Player system. In what follows, we have used UPPAAL (ver. 3.2.13) on SUN WS (Ultra SPARC Memory:4G) with the parameters specified in Table 1.

One of the requirements of the design is to ensure that the Buffer never overflows, i.e. the location **Inter** of the Buffer of Fig. 3 is not reachable. This has been verified checking the deadlock-freeness of the model.

Table 1. Video Player Parameters.

sub-system	parameter	value	details
Video Source	T_0	40	period of emission of packets
Decoder	T_0	30	Lower bound of time to generate a frame
Decoder	T_1	40	Upper bound of time to generate a frame
Decoder	P	5	The initial value of the drop-rate of the frames
Qos Controller	unit	1000	period of control
Qos Controller	θ_0, θ_1	5	Control Thresholds
Buffer	T_0	40	fixed period for the dispatch of packets
Buffer	L	5	length of the Buffer

Table 2. Result of the Verification of Throughput.

Number of *frame*? signals	Duration time	Result of verification	CPU time (sec.)
1	130	valid	15
1	129	not valid	
2	170	valid	30
2	169	not valid	
3	210	valid	120
3	209	not valid	
5	250	valid	60
5	249	not valid	
6	290	valid	100
6	289	not valid	
7	330	valid	600
7	329	not valid	

Checking the Throughput of K *frame?* signals per T msec. is straight forward. We only need to include K parallel composition copies of the Timed Automata of Fig. 6 and check for deadlock-freeness. Since, we have already verified that the buffer will not overflow, the only likely deadlock can occur from reaching a global state with a coordinate **failure**. But, how can we calculate K and T?

In general, estimating the Throughput of a given system is non-trivial. Here, we can see that the system produces at most K *frame?* signals every $T = 130 + (K - 1) \times 40$ msec. To see, this noticing that the system has a drop rate of one in five, we need to look for the worst possible delay between *frame?* signals. The worst scenario happens when two consecutive *o_packet*s are dropped. For example, consider the case that the 3-rd *frame?* signal is created in the possible time, i.e. 30 msec. after the arrival of the corresponding buffered packet. The 5-th and 6-th *frame*? signals are dropped and the 7-th is created at the latest possible time, i.e. 40 msec. after the arrival of the corresponding buffered packet. In this case, the time difference between the 4-th and 7-th *frame?* is equal to $130 = 3 \times 40 + 10$.

Table 2 depicts the result of the verification. It can be seen that Throughput of 1 frames in 130 msec., 2 frames in 170 msec., ... are verified, while the Throughput of 1 frames in 129 msec., 2 frames in 169 msec., ... are not valid. We have also included the CPU time for each experiment, which indicates an exponential increase in time. As a result, there is a clear scope in the research for finding faster method of verification of QoS Timeliness properties.

Of course UPPAAL itself is not a system development tool. However, in the early stages of the system design, it can be a strong tool for detecting time related design errors in the specification. For example, often choosing a wrong value for a constant or using $<$ instead of $\leq$ may creates a dead-lock. Such system errors can be easily detected using UPPAAL. When the designer developes an implementation as an executable code or a hardware logic design, it is hard to detect such errors.

5 Proof of Theorem 1

The aim of this section is to present a proof of the Theorem 1. Our first result establishes the relationship between runs of the parallel composition of two networks of Timed Automata with runs of each component. The idea is to *project* each run of the parallel composition to a run of the components. We shall start with the definition of a projection map.

Definition 7. *Suppose that $\mathscr{A}$ and $\mathscr{B}$ are two network of Timed Automata that share actions $e_1, e_2, \ldots, e_K$. Let $\pi_{\mathscr{A}} : Act(\mathscr{A} \parallel \mathscr{B}) \cup \mathbb{R}^{\geq 0} \longrightarrow Act(\mathscr{A}) \cup \mathbb{R}^{\geq 0}$*

$$\pi_{\mathscr{A}}(\lambda) = \begin{cases} \lambda & \lambda \in \mathbb{R}^{\geq 0} \\ 0 & \lambda \in Act(\mathscr{B}) \backslash Act(\mathscr{A}) \\ \lambda & \lambda \in Act(\mathscr{A}) \backslash Act(\mathscr{B}) \\ \downarrow \lambda & \lambda \in Act(\mathscr{A}) \cap Act(\mathscr{B}) \end{cases} .$$

It can be seen that the projection function $\pi_{\mathscr{A}}$ maps all actions $\lambda \notin Act(\mathscr{A})$ to 0. This can be interpreted by considering that the occurrence of such λ has no effect on the dynamics of $\mathscr{A}$ and takes zero-time. The projection map $\pi_{\mathscr{B}}$ can be defined similarly.

If $s = (\boldsymbol{l}, v)$ is a state of $\mathscr{A} \parallel \mathscr{B}$, then the vector of location $\boldsymbol{l}$ consists of coordinates representing locations in both $\mathscr{A}$ and $\mathscr{B}$. Also, the valuation is a map on the set of clock variables and data variables belonging to both $\mathscr{A}$ and $\mathscr{B}$.

Definition 8. *Suppose that $s = (\boldsymbol{l}, v)$ is a state of $\mathscr{A} \parallel \mathscr{B}$, define $pr_{\mathscr{A}}(s)$, called the* projection of the state s to $\mathscr{A}$*, as a pair $(\boldsymbol{l}_{\mathscr{A}}, v_{\mathscr{A}})$ such that $\boldsymbol{l}_{\mathscr{A}}$ is the restriction of $\boldsymbol{l}$ to the set of coordinates of locations in $\mathscr{A}$ and $v_{\mathscr{A}}$ is the restriction of v to the clocks and data variables in $\mathscr{A}$.*

The next lemma states that the projection of each run of $\mathscr{A} \parallel \mathscr{B}$ to $\mathscr{A}$ is a run of $\mathscr{A}$. Assume that $\mathscr{A}$ and $\mathscr{B}$ are networks of Timed Automata with shared actions $\{e_1, e_2, \ldots, e_K\}$.

Lemma 1. *If* $\sigma := s_0 \overset{\lambda_1}{\Rightarrow} s_1 \overset{\lambda_2}{\Rightarrow} \cdots \overset{\lambda_n}{\Rightarrow} s_n \cdots$ *is a finite/infinite of* $\mathscr{A} \,||\, \mathscr{B}$*, then* $pr_{\mathscr{A}}(s_0) \overset{\pi_{\mathscr{A}}(\lambda_1)}{\Rightarrow} pr_{\mathscr{A}}(s_1) \overset{\pi_{\mathscr{A}}(\lambda_2)}{\Rightarrow} \cdots \overset{\pi_{\mathscr{A}}(\lambda_n)}{\Rightarrow} pr_{\mathscr{A}}(s_n) \cdots$*, which we shall denote with* $proj(\sigma, \mathscr{A})$*, is a run of* $\mathscr{A}' = \mathscr{A}[\downarrow e_1/e_1, \ldots, \downarrow e_K/e_K]$ *created from* $\mathscr{A}$ *by replacing each half action with its complete form.*

Proof. The proof is by induction on n. We must show that if $s_n \overset{\lambda_{n+1}}{\Rightarrow} s_{n+1}$ then $pr_{\mathscr{A}}(s_n) \overset{\pi_{\mathscr{A}}(\lambda_{n+1})}{\Rightarrow} pr_{\mathscr{A}}(s_{n+1})$. Let $s_n = (\boldsymbol{l_n}, v_n)$ and $s_{n+1} = (\boldsymbol{l_{n+1}}, v_{n+1})$.

Case 1: $\lambda_{n+1} \in \mathbb{R}^{\geq 0}$, is trivial. As $s_n \overset{\lambda_{n+1}}{\Rightarrow} s_{n+1}$ implies that for each coordinate l_i of the vector of locations $\boldsymbol{l_n}$, $I(l_i)(v+d')$ for all $d' \leq \lambda_{n+1}$. This is true specially for the coordinates l_i of $\mathscr{A}$.

Case 2: $\lambda_{n+1} \in Act(\mathscr{B}) \backslash Act(\mathscr{A})$ i.e. λ_{n+1} is an internal action of $\mathscr{B}$ and occurrence of λ_{n+1} has no effect on $\mathscr{A}$. Consequently, none of the locations, valuation of clocks or data variable of $\mathscr{A}$ is changed i. e. $pr_{\mathscr{A}}(s_n) = pr_{\mathscr{A}}(s_{n+1})$ and we can write $pr_{\mathscr{A}}(s_n) \overset{0}{\Rightarrow} pr_{\mathscr{A}}(s_{n+1})$.

Case 3: $\lambda_{n+1} \in Act(\mathscr{A}) \backslash Act(\mathscr{B})$, then there is a location $l_i \in Location(\mathscr{A})$ such that $l_i \overset{\lambda_{n+1}, g_i, r_i}{\longrightarrow} l_i'$. As a result, $\boldsymbol{l}_{n+1} = \boldsymbol{l}[l_i'/l_i]$, $v_{n+1} = r(v_n)$, and $g_i(v) = g_i(v \mid_{Clock(\mathscr{A}_i) \cup Data(\mathscr{A}_i)})$, $pr_{\mathscr{A}}(s_n) \overset{\lambda_i}{\Rightarrow} pr_{\mathscr{A}}(s_{n+1})$.

Case 4: $\downarrow \lambda_{n+1} \in \downarrow Act(\mathscr{A}) \cap \downarrow Act(\mathscr{B})$. In this case, λ_{n+1} is a shared action and there is an $e_n (1 \leq n \leq K)$, such that $\lambda_{n+1} = \downarrow e_n$. For example, $l_i \overset{e_n?, g_i, r_i}{\longrightarrow} l_i'$ and $l_i \overset{e_n!, g_j, r_j}{\longrightarrow} l_i'$ in $\mathscr{A}$ and $\mathscr{B}$, respectively. Moreover, $g_i(v_n), g_j(v_n)$ and $r_i \cup r_j \in \mathscr{R}$. As a result, replacing $e_n?$ with $\downarrow e_n (= \downarrow \lambda_n)$, we have $l_i \overset{e_n?, g_i, r_i}{\longrightarrow} l_i'$, $g_i(v_n), r_i \in \mathscr{R}$. Hence, $pr_{\mathscr{A}}(s_n) \overset{\lambda_i}{\Rightarrow} pr_{\mathscr{A}}(s_{n+1})$. □

The converse of the above lemma is not valid. In other word, it is not possible to start with *any* two runs $\sigma_1 \in \mathscr{A}$ and $\sigma_2 \in \mathscr{B}$ and *merge* them to create a run of the parallel composition. For σ_1 and σ_2 to synchronise, one of the requirements is that the order of the occurrence of the shared actions to be identical.

Definition 9. *Assume that* $\mathscr{A}$ *and* $\mathscr{B}$ *are two networks of Timed Automata with shared actions* $e_1, \ldots, e_K$*. Assume that* σ_1 *and* σ_2 *are finite runs of* $\mathscr{A}$ *and* $\mathscr{B}$*, respectively.* σ_1 *and* σ_2 *are called* Shared Action Compatible *if the order of the occurrence of shared actions in them are identical. i.e. if* $\{\mu_1^1, \mu_2^1, \ldots, \mu_n^1\}$ *and if* $\{\mu_1^2, \mu_2^2, \ldots, \mu_m^2\}$ *are ordered sequences of shared actions in* σ_1 *and* σ_2*, respectively, then* $n = m$ *and for each* i*,* μ_i^1 *and* μ_i^2 *are half actions of the same complete action, i.e.* $. \downarrow \mu_i^1 = \downarrow \mu_i^2$.

The following Lemma studies a special case under which it is possible to merge a run σ_1 of $\mathscr{A}$ and a run σ_2 of $\mathscr{B}$. For σ_1 and σ_2 to merge into a run of the the parallel composition $\mathscr{A} \,||\, \mathscr{B}$, they must have the same *time sequences*, for the corresponding shared actions *and* the order of the occurrence of shared actions with equal Timestamps must be identical.

Lemma 2. *Assume that $\mathscr{A}$ and $\mathscr{B}$ are network of Timed Automata with shared actions $e_1, e_2, \ldots e_K$. Assume that $\mathscr{A}$ and $\mathscr{B}$ have no shared clocks or data variables. Suppose that σ^1 and σ^2 are finite runs of $\mathscr{A}$ and $\mathscr{B}$ that*

1. *σ^1 and σ^2 are* Shared Action Compatible *and;*
2. *for each i, $Time(\sigma^1, e_i) = Time(\sigma^2, e_i)$,*

then there is $\sigma \in Run(\mathscr{A} \mid\mid \mathscr{B})$ such that $proj(\sigma, \mathscr{A}) = \sigma^1$ and $proj(\sigma, \mathscr{B}) = \sigma^2$.

Sketch of the Proof: suppose that $\sigma^1 = s_0^1 \overset{\alpha_1}{\Rightarrow} s_1^1 \overset{\alpha_2}{\Rightarrow} \cdots \overset{\alpha_N}{\Rightarrow} s_N^1$ and $\sigma^2 = s_0^2 \overset{\beta_1}{\Rightarrow} s_1^2 \overset{\beta_2}{\Rightarrow} \cdots \overset{\beta_M}{\Rightarrow} s_M^2$, where for each i, $\alpha_i \in Act(\mathscr{A}) \cup \mathbb{R}^{\geq 0}$ and $\beta_i \in Act(\mathscr{B}) \cup \mathbb{R}^{\geq 0}$.

Let us assume that Timestamp$(\sigma^1, S_N^1) \leq$ Timestamp (σ^2, S_M^2); the symmetric case of Timestamp$(\sigma^1, S_N^1) \geq$ Timestamp (σ^2, S_M^2) can be treated similarly. Also, without any loss of generality, we can assume that the set of all Timestamps of all states σ^1 and the Timestamps of the states of σ^2 that occurs before the Timestamp of the last state of σ^1 are identical, i.e.

$$\begin{array}{cc} \{Timestamp(\sigma^1, s_j^1) \mid 0 \leq j \leq N\} = \{Timestamp(\sigma^2, s_j^2) \mid \\ Timestamp(\sigma^2, s_j^2) \qquad\qquad \leq Timestamp(\sigma^1, s_N^1)\}. \end{array} \tag{3}$$

The above can be achieved by using Time Additive Axiom to modify a run and adding extra states. The proof of the lemma is by induction, we shall use the following notations in the rest of the proof:

- for $0 \leq p \leq N$, let $\sigma^{1,p} := s_0^1 \overset{\alpha_1}{\Rightarrow} s_1^1 \overset{\alpha_2}{\Rightarrow} \cdots \overset{\alpha_p}{\Rightarrow} s_p^1$,
- for $0 \leq q \leq M$, let $\sigma^{2,q} := s_0^2 \overset{\beta_1}{\Rightarrow} s_1^2 \overset{\beta_2}{\Rightarrow} \cdots \overset{\beta_q}{\Rightarrow} s_q^2$,
- also assume that for $0 \leq n$, $\sigma^n := s_0 \overset{\lambda_1}{\Rightarrow} s_1 \overset{\lambda_2}{\Rightarrow} \cdots \overset{\lambda_n}{\Rightarrow} s_n$ donates a run of $\mathscr{A} \mid\mid \mathscr{B}$.

Since the induction base is trivial, we only need to prove the following claim, which implies the induction step.

CLAIM: for $p+q < M+N$, if $proj(\sigma^n, \mathscr{A}) = \sigma^{1,p}$ and $proj(\sigma^n, \mathscr{B}) = \sigma^{2,q}$, there is $\sigma^{n+1} \in Run(\mathscr{A} \mid\mid \mathscr{B})$ such that $proj(\sigma^{n+1}, \mathscr{A}) = \sigma^{1,p'}$ and $proj(\sigma^{n+1}, \mathscr{B}) = \sigma^{2,q'}$, where $p \leq p' \leq N$, $q \leq q' \leq M$ and $p + q < p' + q'$.

Proof of the CLAIM: The proof of above claim involves a number of cases. Let $s_p^1 = (\boldsymbol{l}^1, v^1)$, $s_q^2 = (\boldsymbol{l}^2, v^2)$ and $s_n = (\boldsymbol{l}, v)$, where $\boldsymbol{l}$ consists of coordinates of $\boldsymbol{l}^1$ and $\boldsymbol{l}^2$ and $v = v^1 \cup v^2$.

Case I: $\alpha_{p+1} = 0$ or $\beta_{q+1} = 0$ is trivial. For example if $\alpha_{p+1} = 0$ then $s_p^1 = s_{q+1}^1$. In this case, the sequence $\sigma^{n+1} = s_0 \overset{\lambda_1}{\Rightarrow} s_1 \overset{\lambda_2}{\Rightarrow} \cdots \overset{\lambda_n}{\Rightarrow} s_n \overset{\alpha_{p+1}}{\Rightarrow} s_{n+1}$ is such that $proj(\sigma^{n+1}, \mathscr{A}) = \sigma^{1,p+1}$ and $proj(\sigma^{n+1}, \mathscr{B}) = \sigma^{2,q}$. We see that $p' = p + 1$ and $q' = q + 1$.

Case II: α_{p+1} and β_{q+1} are both nonnegative real numbers. Using equation 3 we can show that $\alpha_{p+1} = \beta_{q+1} = d$. Now, if $\sigma^{n+1} := s_0 \overset{\lambda_1}{\Rightarrow} s_1 \overset{\lambda_2}{\Rightarrow} \cdots \overset{\lambda_n}{\Rightarrow} s_n \overset{d}{\Rightarrow} s_{n+1}$, we can see that $proj(\sigma^{n+1}, \mathscr{A}) = \sigma^{1,p+1}$ and $proj(\sigma^{n+1}, \mathscr{B}) = \sigma^{2,q+1}$. Consequently, $p' = p + 1$ and $q' = q + 1$.

Case III: One of α_{p+1} or β_{q+1} is a completed action. For example, if α_{p+1} is a completed action and enabled under s_n, α_{p+1} is enabled under s_p^1. Hence, if $\sigma^{n+1} := s_0 \overset{\lambda_1}{\Rightarrow} s_1 \overset{\lambda_2}{\Rightarrow} \cdots \overset{\lambda_n}{\Rightarrow} s_n \overset{\alpha_{p+1}}{\Rightarrow} s_{n+1}$ then $proj(\sigma^{n+1}, \mathscr{A}) = \sigma^{1,p+1}$ and $proj(\sigma^{n+1}, \mathscr{B}) = \sigma^{2,q}$. In this case, $p' = p+1$ and $q' = q$.

Case IV: Both α_{p+1} and β_{q+1} are half actions. By the Shared Action Compatibility, α_{p+1} and β_{q+1} are half actions of the same actions, *i.e.*, there is i such that $\downarrow \alpha_{p+1} = \downarrow \beta_{q+1} = e_i$ for $1 \leq i \leq K$. Then let $\sigma^{n+1} := s_0 \overset{\lambda_1}{\Rightarrow} s_1 \overset{\lambda_2}{\Rightarrow} \cdots \overset{\lambda_n}{\Rightarrow} s_n \overset{e_i}{\Rightarrow} s_{n+1}$. We can see that $proj(\sigma^{n+1}, \mathscr{A}) = \sigma^{1,p+1}$ and $proj(\sigma^{n+1}, \mathscr{B}) = \sigma^{2,q+1}$. Notice that $p' = p+1$ and $q' = q+1$.

Case V: One of α_{p+1} or β_{q+1} is a half action and other is in $\mathbb{R}^{\geq 0}$. Let $\alpha_{p+1} \in HA$ and $\beta_{q+1} \in \mathbb{R}^{\geq 0}$. By the Shared Action Compatibility, there is $r > q+1$, such that β_r and α_{p+1} are half actions of the same actions, *i.e.*, $\downarrow \alpha_{p+1} = \downarrow \beta_r = e_i$. This implies that $\beta_{q+1} = 0$, since $\beta_{q+1} \leq \text{Timestamp}(\sigma^2, s^2_{r-1}) - \text{Timestamp}(\sigma^1, s^1_p) = 0$. Using case I, there is nothing to prove. □

We shall end this section with the proof of **Theorem** 1.

Proof of Theorem 1 $\Rightarrow$: We shall prove by contradiction. Suppose that $\mathscr{B}$ satisfies the property ϕ but there is a run σ of $\mathscr{A}' \parallel \mathscr{B}'$ ending in **failure**. Then by **Lemma** 1 $\sigma_{\mathscr{A}} = proj(\sigma, \mathscr{A})$ is a run of $\mathscr{A}$ ending in a **failure** state of $\mathscr{A}$, i.e. a state of $\mathscr{A}$ with a failure coordinate. As a result, $\phi(\text{Time}(\sigma_{\mathscr{A}}, e_1), \ldots, \text{Time}(\sigma_{\mathscr{A}}, e_K)) = \mathbf{F}$. Now, consider $\sigma_{\mathscr{B}} = proj(\sigma, \mathscr{B})$ which is a run of $\mathscr{B}$. Since, $\text{Time}(\sigma_{\mathscr{A}}, e_i) = \text{Time}(\sigma_{\mathscr{B}}, e_i)$, for each i,
$\phi(Time(\sigma_{\mathscr{B}'}, e_1), \ldots, Time(\sigma_{\mathscr{B}'}, e_K)) = \mathbf{F}$. Consequently, by definition 5, $\sigma_{\mathscr{B}}$ does not satisfy ϕ, which is a contradiction.

Conversely: The proof of this case is also by contradiction. Assume that no run of the parallel composition meets any **failure** state, but $\mathscr{B}$ does not satisfy ϕ. Then there is a run σ_2 of $\mathscr{B}$ such that $\phi(\text{Time}(\sigma_2, e_1), \ldots, \text{Time}(\sigma_2, e_K)) = \mathbf{F}$. Suppose that $t^1 = \text{Time}(\sigma_2, e_1)$, ..., $t^K = \text{Time}(\sigma_2, e_K)$. Since $\phi(t^1, \ldots, t^K) = \mathbf{F}$ by Definition 5. There is a finite run σ_1 of $\mathscr{A}$ such that σ_1 ends in **failure** and $t_1 = \text{Time}(\sigma_1, e_1)$,..., $t_K = \text{Time}(\sigma_1, e_K)$. Moreover, since $\mathscr{A}$ is a Complete QTA, σ_1 can be chosen such that σ_1 and σ_2 are Shared Action Compatible. Now, using σ_1, σ_2 and Lemma 2, we can conclude that there is a run of $\mathscr{A}' \parallel \mathscr{B}'$ ending in **failure**. This is a contradiction. □

6 Related Works

Formal specification of QoS in a distributed system via modelling languages such as Unified Modelling Language (UML) is an active area of research [22, 7, 15, 1, 11, 12]. In particular, the idea of specifying the QoS requirements as contracts [19] on the behaviour of the system is proposed [15] as a part of Model Driven Architecture, the new initiative by the Object Management Group (`www.omg.org`). However, the current research mainly deals with the issue of verifying of QoS property via Test Automata. The question that, which properties can be analysed by Test Automata is discussed in details in [2–4]. In particular, [3] presents a property language, called SBLL which is suitable for expressing safety and liveness properties of the real-time systems. SBLL is a testable language, in the

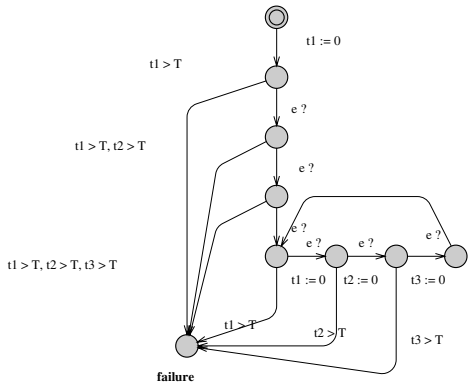

Fig. 8. QTA for checking Throughput of K occurrence of $e?$ in T unit of time.

sense that [4] presents an algorithm for the translation of SBLL formulae to Test Automata. SBLL has the following grammar:

$\phi ::= \mathbf{ff} \mid \phi_1 \wedge \phi_2 \mid g \vee \phi \mid \mathbb{W}\phi \mid [a]\phi \mid \langle a \rangle \mathbf{tt} (a \in \mathscr{U}) \mid x \ \underline{\text{in}} \ \phi \mid X \mid \max(X, \phi)$, where **ff** and **tt** stand for *false* and *true*, respectively. g is a guard expression on the clocks, $x \ \underline{\text{in}} \ \phi$ stands for resetting a clock x before evaluating ϕ, $\max(X, \phi)$ is the maximal fixed point solution on X in ϕ, $\mathscr{U}$ is a set of urgent actions and $\mathbb{W}\phi$ stands for ϕ holds *forever*.

The following formula represents the Throughput as an SBLL formula

$$\phi = t_1 \ \underline{\text{in}} \ [e?](t_1 \geq T \vee (t_2 \ \underline{\text{in}} \ [e?]t_1 \geq T \vee t_2 \geq T \vee (\cdots (t_1 \geq T \vee \cdots t_{K-1} \geq T$$

$$\vee t_{K-1} \ \underline{\text{in}} \ [e?](\phi'')) \cdots)))$$

$$\phi'(X) =$$

$$t_1 \geq T \vee t_1(\ \underline{\text{in}} \ [e?](t_2 \geq T \vee t_2 \ \underline{\text{in}} \ [e?](\cdots t_{K-1} \geq T \vee t_{K-1} \ \underline{\text{in}} \ [e?](X) \cdots)))$$

$$\phi'' = \max(X, \phi').$$

Fig. 8 depicts the Test Automaton for the Throughput for $K = 3$, created via the algorithm [4], in which all redundant transitions are omitted. Fig. 7 depicts the equivalent QTA created earlier. It can be seen that the QTA of Fig. 7 has the advantage of being *scalable*, i.e. the Test Automaton for the throughput of K signals $e?$ in T units of time can be created from the parallel composition of K copies of the QTA of Fig. 6. The reason behind scalability of our model is that, unlike SBLL, our model of Timed Automata includes data variables. There is a clear scope for research to extended the SBLL to include data variables. In particular, since $L_{\forall s}$, an extension of SBLL, completely characterises testable properties [2–4], an extension of SBLL to include data variables will enable to characterize the Timeliness properties which are testable.

7 Conclusion

This paper presents a formal approach to the verification of Timeliness QoS properties, such as Throughput, Jitter and Latency, in object-based models of Distributed Multimedia Systems. For each Timeliness property ϕ, we define a QoS Timed Automata (QTA) such that all its runs that do not meet a failure location, satisfy ϕ. Moreover, all runs of the QTA that meet a failure location violate ϕ. The main result of the paper proves that a QTA is a Test Automata, i.e. it can be used to verify the property ϕ over a network of Timed Automata via parallel composition. We have demonstrated our approach by the verification of Throughput in a Video Player system.

References

1. D. H. Akehurst, B. Bordbar, J. Derrick, and a. g. waters: *design support for distributed systems: dse4ds* in J. Finney, M. Haahr, and A. Montressor, editors, proceedings of the 7th Cabernet Radicals Workshop, October 2002.
2. L. Ageto, P. Bouyer, A. Burgueño and K. G. Larsen: *The Power of Reachability Testing for Timed Automata*, In Proceedings of 18th Conference of Fundamental of Software Technology and Theoretical Computer Science (FST and TCS '98), LNCS **1530** pp.245–256, 1998.
3. L. Ageto, P. Bouyer, A. Burgueño and K. G. Larsen: *Model-Checking via Reachability Testing for Timed Automata*, In Proceedings 4th Conference of Tools and Algorithms for Construction and Analysis of Systems (TACAS '98), LNCS **1384** pp.263–280, 1998.
4. L. Ageto, P. Bouyer, A. Burgueño and K. G. Larsen: *The Power of Reachability Testing for Timed Automata*, available from `http://www.lsv.ens-cachan.fr/Publis/publis-y3-2003.php`, to appear in Theoretical Computer Science.
5. R. Alur and D.L. Dill: *A Theory for Timed Automata*, In Theoretical Computer Science **125** pp.183–235, 1994.
6. T. Amnell, G. Behmann, J. Bengtsson, P. R. D'Argenio, A. David, A. Fehnker, T. Hune, B. Jeannet, K. G. Larsen, O. Möller, P. Pettersson, C. Weise and W. Yi: *UPPAAL—Now, Next and Future* In proceedings of Modelling and Verification of Parallel Processes (MOVEP2k), LNCS **2067** pp.100–125, 2001.
7. J. Øyvind Aagedal and E. F. Ecklund Jr. *Modelling QoS: Towards a UML Profile* UML 2002, pp. 275–289, 2003.
8. J. Bengtsson, W. O. D. Griffioen, K.J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson and W. Yi: *Verification of an Audio Protocol with Bus Collision Using UPPAAL*, In Proceedings of the 8th International Conference on Computer-Aided Verification, LNCS **1102** pp.244–256, 1996.
9. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson and W. Yi: *UPPAAL, a Tool suite for automatic verification of real-time systems* In Proceedings of Workshop on Hybrid Systems III: Verification and Control, LNCS **1066** pp.232–243, 1995.
10. G. Blair, J.-B. Stefani: *Open Distributed Processing and Multimedia* Addison-Wesley, Boston, MA, 1997.
11. B. Bordbar, J. Derrick, and A. G. Waters: *A UML approach to the design of open distributed systems* In Chris George and Huaikou Miao, editors, Formal Methods and Software Engineering, LNCS **2495** pp.561–572, 2002.

12. B. Bordbar, J. Derrick, and A. G. Waters: *Using UML to specify QoS constraints in ODP* Computer network and ISDN systems **40** pp.279–304 2002.
13. H. Bowman, G. Faconti, and M. Massink: *Specification and verification of media constraints using UPPAAL*, In Proceedings of Design, Specification and Verification of Interactive Systems '98, Markopoulos and P. Johnos, editors, pp. 261–277 Springer, 1998.
14. K. Havelund, A. Skou, K. G. Larsen and K. Lund: *Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL* In Proceedings of the 18th IEEE Real-Time Systems Symposium, pp.2–13, 1997.
15. J.-M. Jézéquel *Model-driven engineering with contracts, patterns, and aspects* In Tutorial Program of AOSD 2003: 2nd International Conference on Aspect-Oriented Software Development, ACM-IEEE, March 2003.
16. K. G. Larsen, Paul Pettersson and W. Yi: *UPPAAL in a Nutshell*, In Springer International Journal of Software Tools for Technology Transfer **1(1+2)** 1997.
17. M. Lindahl, Paul Pettersson and W. Yi: *Formal Design and Analysis of a Gear Controller*, In Springer International Journal of Software Tools for Technology Transfer, volume 3, issue 3, pp. 353-368, 2001.
18. H. Lönn and P. Pettersson: *Formal Verification of a TDMA Protocol Start-Up Mechanism*, In Proceedings of 1997 IEEE Pacific Rim International Symposium on Fault-Tolerant Systems, pp.235–242, 1997.
19. Stephane Lorcy, Noel Plouzeau, and Jean-Marc Jézéquel *Reifying quality of service contracts for distributed software*, In 26th Conference on Technology of Object-Oriented Systems (TOOLS USA'98), August 1998
20. R. Milner, *Communication and concurrency*, Prentice Hall, Upper Saddle River, NJ, 1989.
21. *ITU Recommendation X.901-904 ISO/IEC 10746 1-4.* Open Distributed Processing Reference Model - Parts **1-4**, July, 1995.
22. R. Staehli, F. Eliassen, J. Øyvind Aagedal and G. S. Blair *Quality of Service Semantics for Component-Based Systems* The 1st International Workshop on Middleware for Grid Computing, pp. 153-157, 2003.
23. W. Yi, *Real-Time Behaviour of Asynchronous Agents*, In Proceeding of 1st Int. Conf. Theory of Concurrency (CONCUR90), LNCS **458**, pp. 502-520, 1990.