SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2002; **32**:1411–1435 (DOI: 10.1002/spe.487)

Automated test case generation for the stress testing of multimedia systems

Jian Zhang[1] and S. C. Cheung[2,*,†]

[1]*Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing 100080, People's Republic of China*
[2]*Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, People's Republic of China*

SUMMARY

With the advancement in network bandwidth and computing power, multimedia systems have become a popular means for information delivery. However, general principles of system testing cannot be directly applied to testing of multimedia systems on account of their stringent temporal and synchronization requirements. In particular, few studies have been made on the stress testing of multimedia systems with respect to their temporal requirements under resource saturation. Stress testing is important because erroneous behavior is most likely to occur under resource saturation. This paper presents an automatable method of test case generation for the stress testing of multimedia systems. It adapts constraint solving techniques to generate test cases that lead to potential resource saturation in a multimedia system. Coverage of the test cases is defined upon the reachability graph of a multimedia system. The proposed stress testing technique is supported by tools and has been successfully applied to a real-life commercial multimedia system. Although our technique focuses on the stress testing of multimedia systems, the underlying issues and concepts are applicable to other types of real-time systems. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: multimedia systems; stress testing; static analysis; resource consumption; test case generation; constraint solving

INTRODUCTION

Background

The popularity of multimedia systems is growing rapidly, and they are the subject of much current research. Typical examples of multimedia systems include video conferencing, home shopping,

*Correspondence to: S. C. Cheung, Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, People's Republic of China.
†E-mail: scc@cs.ust.hk

Contract/grant sponsor: Hong Kong Research Grant Council; contract/grant number: HKUST6159/98E
Contract/grant sponsor: Croucher Foundation Fellowship and the National Science Fund for Distinguished Young Scholars; contract/grant number: 60125207

Received 17 September 1999
Revised 29 July 2002
Accepted 29 July 2002

Published online 17 October 2002

information-on-demand, remote education, and so on. This type of application makes use of various kinds of media objects, such as audios, videos, images, and texts. A multimedia system, in its most general form, consists of a number of servers and clients (not necessarily distinct), connected by a suitable network. Despite their popularity, the development of such systems is not easy, not only because of their inherent complexity, but also for the most part because of the very nature of their requirements, which include stringent temporal and synchronization constraints [1]. While media servers must deliver data in the form of continuously running streams, media clients must be able to receive, decode, and resynchronize the data streams. Moreover, users may interact with the multimedia systems and alter the control flow. All of these tasks have to be performed in real time.

In a multimedia system, the data may not be delivered timely due to network congestion, transmission errors, or delays. Horn and Girod [2] reported that when time-consuming applications are running simultaneously, buffer overflow and packet loss may occur due to limited CPU capacity. For these and other reasons, timing requirements of multimedia systems may not always be satisfied, especially under strenuous situations. The problem has so far received relatively little attention. Testing of multimedia systems is different from that of traditional software, which does not normally have temporal requirements. The major challenge of testing multimedia systems lies in the identification of strenuous situations because the problem state space explodes quickly with the inclusion of the temporal requirements. In this paper, we address the problem using constraint solving techniques to identify test cases that lead to potential resource saturation. The flow and concurrency control of a multimedia system is modeled using a Petri net while the temporal relations of media objects are described in temporal constraints. Test coverage can be defined upon reachable markings identified using reachability analysis.

A multimedia system is more likely to misbehave when system resources are heavily utilized. This misbehavior may be due to errors like mismanagement of buffers or threads. As such, testing of systems under strenuous conditions can lead to significant reduction in test efforts. This is referred to as the *stress testing approach* [3]. However, identification of possible strenuous conditions is a challenging problem since various timing constraints of multimedia systems must be satisfied both for intra-stream and inter-stream relationships. Although our testing methodology is based on multimedia systems, the issues and underlying concepts are applicable to other types of real-time systems. The test case generation in our methodology is automated by a support tool called *stem*, detailed in this paper.

Related work

Much work has been done on the modeling and testing of distributed systems and protocols [4]. However, few studies have explicitly considered the multimedia and the timing issues. Techniques proposed for multimedia modeling and timing tests are mainly based on three approaches: labeled transition systems [5–7], Petri nets [8–11], and constraints [12,13].

An interesting approach to testing distributed systems with time was proposed by Ates and Sarikaya [6]. It specifies a system as a timed automata from which a finite state machine (FSM) is derived. A timed trace of the FSM describes a sequence of transitions leading from the initial state to some destination state. Each timed trace forms a test case. Since the number of possible timed traces explodes quickly with time, exhaustive testing of a distributed system with temporal properties is generally infeasible even for a short period of time. However, the issue of test case selection was not addressed.

Santos *et al.* [14] proposed a design methodology for hypermedia systems. The methodology translates a high-level hypermedia authoring model into a RT-LOTOS specification which is then verified by a reachability analysis tool [15]. Analysis is conducted in terms of timed automata that are derived from the specification in RT-LOTOS. Two sets of properties can be checked. The first set is classified as intrinsic consistency properties, such as an assertion that a *start* event of a node must eventually lead to an *end* event. The second set is classified as extrinsic consistency properties, such as an assertion that two audio streams cannot be mixed on the same audio channel.

Various extensions of Petri nets have been proposed in the literature for the modeling of multimedia systems. A well-known formalism is the object composition Petri net (OCPN) [8]. It augments Petri nets with logic on time intervals. However, user interactions cannot be specified in this model. To deal with this problem, Prabhakaran and Raghavan [9] suggested the use of dynamic timed Petri nets. Wang and Wu [11] described a multimedia and hypermedia Petri net (MHPN) model and the implementation of a supporting system. Senac *et al.* [10] proposed hierarchical time stream Petri nets (HTSPNs). However, they did not address the testing of multimedia systems.

Work has also been proposed to apply constraint satisfaction techniques for protocol testing [12,13]. Higashino and Bochmann in [16] presented an algorithm to derive test cases from a subset of LOTOS called P-LOTOS, where data types are limited to integers and Booleans, and operations to addition, subtraction, and comparison. None of these works considered timing or resource consumption. Realizing the importance of testing temporal properties, Mandrioli *et al.* in [17] proposed a test case generation technique for functional testing from a given set of temporal logic formulas. The test case selection criteria are based on the structure of formulas in the specification.

The importance of considering resource utilization in software testing was raised by Avritzer and Weyuker in [18]. The work discussed load testing of software based on a specified 'operational profile'. It identified the most probable system states using Markov chains and argued that these states should be tested thoroughly. Yang and Pollock [19] described an analysis algorithm to identify the *load sensitive parts* in sequential programs, but the work did not address how to derive input data for the tests.

A testing architecture for multimedia systems was first proposed by the authors in [20] and later refined to test temporal relations that concern only two media objects in [21]. This paper differs from [21] in that it examines the testing of multimedia presentations with more than two media objects under strenuous conditions. It extends our previous work in [22] which specifies the criteria for test case selection based on integer constraint solving techniques. Our methodology allows test cases for strenuous conditions to be seamlessly derived from a model of multimedia systems. Since solving constraints of arbitrary forms in finite integer domains is known to be expensive, the constraints introduced in our methodology have been carefully transformed to a tractable form.

Paper outline

The paper is organized as follows. The next section discusses common characteristics of multimedia systems. The subsequent section explains the formulation of the behavior and requirements of multimedia systems using Petri nets and standard temporal operators. After that, we give a detailed account of the technique to generate test cases, which likely expose the anomalous behavior of a multimedia system. This is followed by a section reporting the experiment of applying the proposed stress testing technique to a real-life multimedia system. The final section concludes the paper.

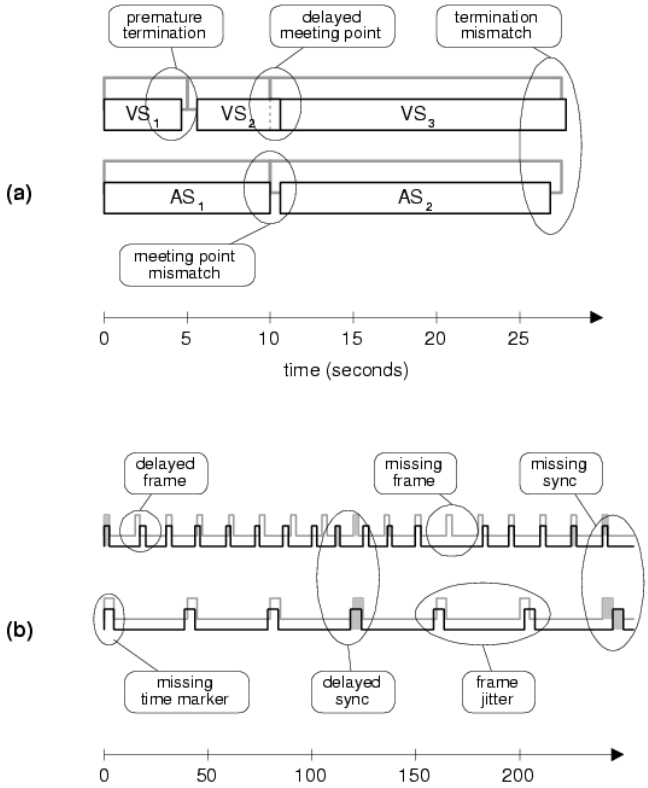


Figure 1. Example faults of temporal requirements at (a) object-level and (b) frame-level.

CHARACTERISTICS OF MULTIMEDIA SYSTEMS

Object- and frame-level temporal requirements

A multimedia system typically involves multiple types of media objects like text, image, audio, and video. Audio and video are considered as continuous media divided into frames. The granularity of temporal requirements can be classified into object-level and frame-level. The object-level requirements concern temporal constraints governing the *begin time* and *end time* across media objects. Frame-level requirements concern temporal constraints across the frames in media objects, such as jittering and lip synchronization. Figure 1 illustrates some faults due to violations of temporal requirements at object-level and frame-level. This paper is concerned mainly with the stress testing of multimedia systems with object-level temporal requirements.

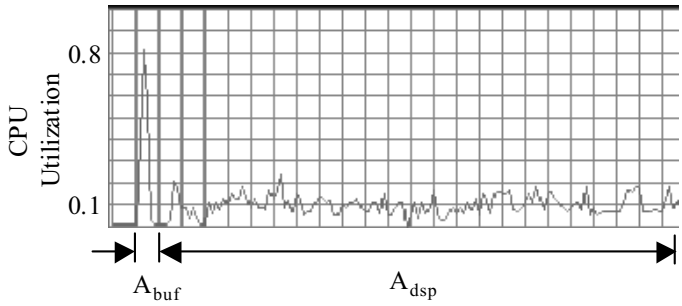


Figure 2. CPU history of an AVI video object.

Resource consumption

Another characteristic of multimedia systems is that they require manipulation of large amounts of data and tend to be resource intensive. For example, full-screen, true-color, uncompressed full-motion video requires an approximately 221 Mbps ($= 640 \times 480 \times 24 \times 30$) data rate. Thus, a 1 min video clip could require more than 1.6 GB of data. As such, video and audio objects are usually compressed before transmission in order to reduce the bandwidth utilization. The client decompresses the data before using them.

The three most important resources of a multimedia system are bandwidth, memory, and CPU cycles. Multimedia systems often require the transfer of large amounts of data. Heavy communication may cause buffer overflow and packet loss. This in turn could lead to insufficient CPU power as the majority of CPU is allocated to regulate the buffer and recover packets. From our experience, mismanagement of buffers is one of the common types of errors in developing multimedia systems. For the purpose of stress testing, we need to know the amount of resource that has been consumed by the media objects during a certain amount of time. This depends on factors such as the media types, quality of service, synchronization relationship, and resource type.

In general, the CPU utilization of an application process consists of two parts: execution of application code and operating system activities (e.g. context switches). The same media object may consume a different proportion of CPU cycles on different platforms. Wolf [23] discussed how to measure the CPU utilization of media stream handling components.

Media objects are often partially or completely buffered before display. For some media players, the buffering process may require more resources. For instance, the Windows media player utilizes more CPU power in the buffering stage when playing AVI video objects as shown in Figure 2. To allow for a more precise resource utilization, a video object A can be decomposed into two parts: A_{buf} and A_{dsp}, where A_{dsp} follows right after the completion of A_{buf}. The digital contents of the media object A are buffered within A_{buf} and displayed within A_{dsp}. The former one usually consumes a higher amount of resource than the latter. However, this is not necessarily the case for all resource types. Figure 2 gives the CPU history of an AVI video object being displayed by the Windows media player.

The pattern of resource utilization varies with media object types and the associated media players. For instance, audio objects generally consume fewer resources than video objects. For the sake of analysis, we use a *weight* to represent the utilization by a media object of a specific resource. The weight could represent either the peak or average value of resource utilization. In the situation where the resource utilization of $\mathsf{A}_{\mathsf{buf}}$ is remarkably different from that of $\mathsf{A}_{\mathsf{dsp}}$, we use a peak value for the former and an average value for the latter. Suppose A is the media object exhibiting the CPU history in Figure 2; the weights for CPU utilization of $\mathsf{A}_{\mathsf{buf}}$ and $\mathsf{A}_{\mathsf{dsp}}$ are 0.8 and 0.1, respectively. This implies the approximation of the CPU history by two regions: $\mathsf{A}_{\mathsf{buf}}$ that constantly consumes 80% CPU and $\mathsf{A}_{\mathsf{dsp}}$ that constantly consumes 10% of the CPU. For more precise modeling, a software testing personnel could further divide $\mathsf{A}_{\mathsf{buf}}$ or $\mathsf{A}_{\mathsf{dsp}}$ into subregions. From our experience, media objects of the same type exhibit similar weights under the same run-time environment. As such, it may not be necessary to identify the weight for each media object. We will explain later in the section on *Test Case Generation* the use of weights to derive potential resource saturation scenarios.

SPECIFICATION OF MULTIMEDIA SYSTEMS

To facilitate the analysis and testing of a multimedia system, we need to specify precisely its behavior and requirements. There has been much work on specifying various aspects of multimedia systems, for example Petri net based synchronization models [8–11], and the enhanced time-line model [24]. Allen's interval temporal logic [25] may also be extended to describe the temporal relationships among multimedia data [26]. In this paper, we use Petri nets to model the synchronization across multiple presentations and Allen's interval temporal logic to model the temporal relationship among multimedia data within a presentation. However, the testing framework we present is not restricted to a particular formal model. We envisage that the work can be adapted to test multimedia systems specified using other formal models with minimal efforts.

Temporal relations between media objects

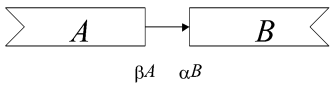
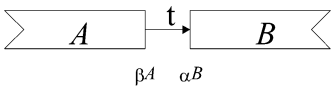
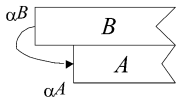
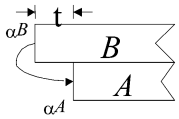
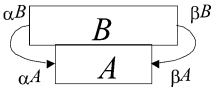
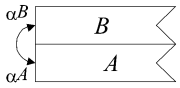
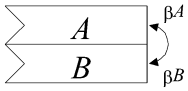
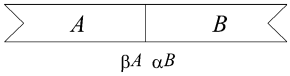
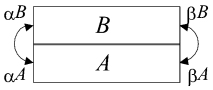
Several related media objects can be grouped into a multimedia presentation. The objects in a presentation have to satisfy some temporal constraints, which are often described by formulas in Allen's interval temporal logic [25]. In addition, we may quantify the temporal interval, such as A starts_after B by t. Table I shows the standard temporal operators and their meanings, where αA and βA denote the begin time and end time of a presentation of media object A. As explained in the previous section, a media object A may be represented in two parts, $\mathsf{A}_{\mathsf{buf}}$ and $\mathsf{A}_{\mathsf{dsp}}$, for more precise modeling of resource consumption. In this case αA is the begin time of playing $\mathsf{A}_{\mathsf{buf}}$ and βA the end time of playing $\mathsf{A}_{\mathsf{dsp}}$. Causal dependencies between begin time and end time are shown by arrows $\rightarrow$ in the timelines of Table I.

The duration of a media object A is denoted as $dur(A)$ whose value can be given by $(\beta A - \alpha A)$. As an example, suppose there are three media objects VideoA, VideoB, and TextC which form a presentation called Demo as shown in Figure 3. The durations of presenting these objects are as follows:

$$dur(VideoA) = 5, \quad dur(VideoB) = 6, \quad dur(TextC) = 15$$

In addition, we have the following temporal constraints:

Table I. Definitions of common temporal operators.

Temporal relation	Temporal constraint	Time line representation
A Before B	$\beta A < \alpha B$	
A Before B by t	$\alpha B = \beta A + \mathrm{t}$	
A Starts_after B	$\alpha A > \alpha B$	
A Starts_after B by t	$\alpha A = \alpha B + \mathrm{t}$	
A During B	$\alpha A > \alpha B$ $\beta A < \beta B$	
A Starts_with B	$\alpha A = \alpha B$	
A Ends_with B	$\beta A = \beta B$	
A Meets B	$\beta A = \alpha B$	
A Equals B	$\alpha A = \alpha B$ $\beta A = \beta B$	

- *VideoA* Before *VideoB* by 4 (i.e. $\alpha VideoB = \beta VideoA + 4$);
- *VideoA* Starts_with *TextC* (i.e. $\alpha VideoA = \beta TextC$); and
- *VideoB* Ends_with *TextC* (i.e. $\alpha VideoB = \beta TextC$).

The temporal relationship between the media objects in a presentation can also be described graphically as shown in Figure 3.

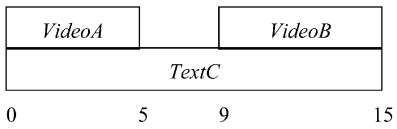


Figure 3. Temporal relationship between media objects in a Demo presentation.

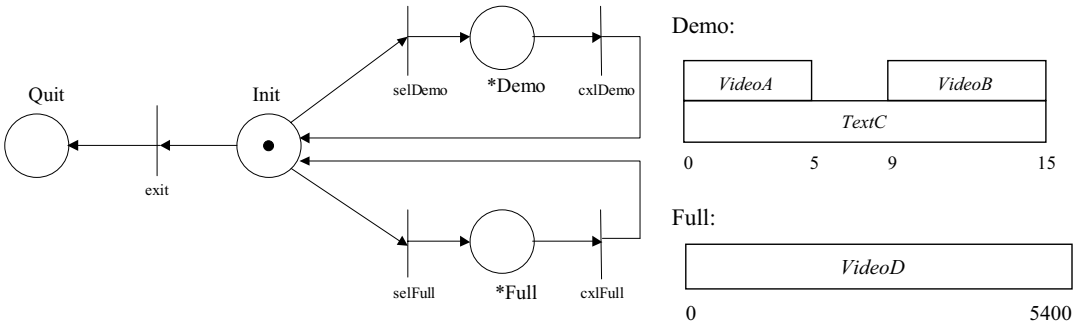


Figure 4. An example presentation modeled in Petri nets.

Control flow modeling in Petri nets

Multimedia presentations can be modeled using the formalism of Petri nets [27], which is a well-founded formal notation that has been widely used for the modeling of dynamic behavior. A Petri net is a triple $(\mathcal{P}, \mathcal{T}, \mathcal{F})$, where $\mathcal{P}$ is the set of places, $\mathcal{T}$ is the set of transitions, and $\mathcal{F}$ is the set of directed arcs. A Petri net starts with an initial assignment of tokens to its places. Places that contain tokens are said to be active; otherwise passive. A transition is enabled when all of its input places are active. A transition can fire only when it is enabled. When a transition fires, a token is removed from each of its input places and added to its output places. A Petri net is said to be 1-safe when each of its places may contain at most one token [28,29]. This safeness property of a Petri net can be verified by standard analysis techniques, such as reachability analysis [28,29].

The specification of a multimedia presentation can be described by a 1-safe Petri net with each place in the Petri net further defined by a tuple $(\mathcal{O}, \mathcal{Z})$, where O is a set of media objects and $\mathcal{Z}$ is a set of temporal constraints as discussed in the previous subsection. Figure 4 gives an example model of a multimedia system using a Petri net, where Demo and Full correspond to two places that involve multimedia presentations. The symbol '$*$' denotes a place that is associated with a non-null presentation. The Demo presents to users some marketing information about the VideoD before they decide to pay while the Full presentation plays the full version of VideoD. The transition selFull corresponds to an event signaling the completion of payment. Note that an enabled transition may wait

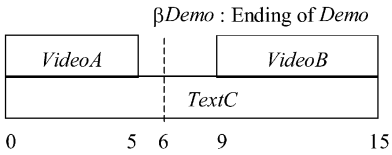


Figure 5. Premature termination of presentation instance demo.

for a non-deterministic amount of time before it is fired. This occurs when a transition represents a user interaction event or a delay due to network communication.

When a place is active, its media objects are activated according to a schedule that observes the associated temporal constraints. An ongoing schedule is prematurely aborted once the place is no longer active. In that case, all media objects that have not yet completed their lifecycles in the schedule are immediately terminated. Figure 5 presents a scenario where the transition cxlDemo takes place 6 s after it has been enabled, and the presentation Demo is immediately aborted. The media object TextC will be terminated and VideoB will not be started.

Since each place associates with a presentation, the terms 'presentation' and 'place' will be used interchangeably. Note that a presentation could be null and not contain any media objects. Since a presentation will be replayed when the associated place receives a token the next time, a presentation could be played several times during the lifecycle of a multimedia system. For a presentation $\mathsf{P} \in \mathcal{P}$, let us call a particular play of it a *presentation instance*, denoted as P. Similarly, *media object instance* A refers to a particular play of media object $\mathsf{A} \in \mathcal{O}$. We define three predicates on a presentation instance P and a media object instance A:

- $consistsOf(P, A)$ holds if and only if P consists of A;
- $active(P, t)$ holds if and only if P is playing at time t; and
- $live(A, t)$ holds if and only if A is playing at time t.

The predicates *consistsOf* and *live* can be related by Rule (1).

- Given a media object instance A and a presentation instance P,
  $live(A, t) \wedge (consistsOf(P, A) \Rightarrow active(P, t)$ Rule (1)

Note that a media object instance cannot appear in more than one presentation instance, i.e.

- Given a media object instance A and two presentation instances P and Q,
  $consistsOf(P, A) \wedge consistsOf(Q, A) \Leftrightarrow P = Q$ Rule (2)

Execution traces

An event may correspond to some user action (e.g. the occurrence of exit in Figure 4), or a signal initiated by some media object (e.g. start and termination of playing a media object). We define an execution trace to be a set of *timed events*, i.e. $\{(ev_i, t_i)\}$. Here, each t_i is a positive value denoting a time point, and ev_i is an event. The tuple (ev_i, t_i) therefore refers to a *timed event* where ev_i occurs

at t_i. The activation and termination of presentations are also treated as events. For presentation P, the corresponding events are denoted as P↑ and P↓, respectively. Thus, (Demo↑, 0) and (Demo↓, 6) refer to the timed events that Demo begins at the zeroth second and it ends at the sixth second, respectively. Let us further denote the begin time and end time of a presentation instance P as αP and βP, respectively. For example, if the presentation instance of Demo between (Demo↑, 0) and (Demo↓, 6) is labeled as *Demo* then $\alpha Demo$ and $\beta Demo$ have the values of 0 and 6, respectively[‡]. In contrast, the symbols αA and βA denote the *scheduled* begin and end times of a media object instance A, respectively. They are the times derived from the specified temporal constraints. For example in Figure 4, $\alpha TextC$ and $\beta TextC$ are 0 and 15 in the presentation instance *Demo*, respectively. A media object instance, such as *TextC* in Figure 4, may not be able to complete its lifecycle if the embedded presentation ends prematurely due to timeout or user interactions. As a result, we have the following rules (3), (4), and (5). Rule (4) assumes that a media object instance is live only between its scheduled begin time and end time. The rules map the dynamic behavior of the media systems into linear constraints:

- $active(P, t) \Leftrightarrow \alpha P \leq t \leq \beta P$; Rule (3)
- $live(A, t) \Rightarrow \alpha A \leq t \leq \beta A$; Rule (4)
- $consistsOf(P, A) \wedge live(A, t) \Rightarrow (\alpha P \leq \alpha A) \wedge (\alpha A \leq t \leq \beta P)$. Rule (5)

Let us consider the diagram in Figure 5. In this presentation, object instance *VideoA* terminates normally, but *TextC* ends at time $\beta Demo$, and *VideoB* is not activated at all. We have $\beta VideoB = \beta TextC = 15$, but neither $\beta VideoB$ nor $\beta TextC$ denotes the actual end time of the corresponding object instance. The following describes a possible execution trace of the multimedia system specified in Figure 4:

{(selDemo, 0), (Demo↑, 0), (VideoA↑, 0), (TextC↑, 0), (VideoA↓, 5),
(cxlDemo, 6), (TextC↓, 6), (Demo↓, 6), (exit, 8)} Trace (1)

Since places Init and Quit do not associate with any presentations, events Init↑, Init↓, Quit↑, and Quit↓, are omitted from the trace. Transitions selDemo, cxlDemo, and Exit fire at time points 0, 6, and 8, respectively. A test case corresponds to an execution trace. In general, the number of execution traces is infinite even for a simple multimedia system due to infiniteness of time and loops in control flows. As such, exhaustive exploration of test cases is generally infeasible. This necessitates the discussion of test coverage.

Firing sequences

As mentioned, the timed events in an execution trace require quantification of the associated time values. These time values are to be determined analytically so as to exhibit a certain resource utilization pattern, such as the maximization of peak CPU usage.

To facilitate the determination of these time values, let us define a *firing sequence* to be $\langle (u_0, t_0), (u_1, t_1), \ldots \rangle$, where each u_i $(i \geq 0)$ is a transition and each t_i $(i \geq 0)$ is either a specific

[‡]The presentation instance could be labeled using any name, such as *Demo1*. For convenience, we label a presentation instance using its original name for general discussion of concepts. We use a similar convention in the labeling of media object instances.

number or a variable denoting u_i's firing time. Note that a transition may occur more than once in a firing sequence. The following is a possible firing sequence exhibited by the multimedia system in Figure 4:

$$\langle(\mathsf{selDemo}, t_1), (\mathsf{cxlDemo}, t_2), (\mathsf{exit}, t_3)\rangle, \quad \text{where } t_1 < t_2 < t_3$$

This firing sequence activates only one presentation instance, viz *Demo*. Its commencement and termination are triggered by the occurrences of the selDemo and cxlDemo at time $\alpha Demo$ $(= t_1)$ and $\beta Demo$ $(= t_2)$, respectively. Since the transition's firing times are represented by variables, a firing sequence corresponds to a set of execution traces.

The firing sequences of a multimedia system can be derived from its reachability graph. Since our specifications consist of two levels (i.e. the Petri net and the temporal relationships among media objects in each presentation), we may decompose the analysis into two stages. The first stage selects a set of firing sequences which cover all reachable markings. The second stage quantifies the time values in each sequence using constraint solving techniques.

The formulation of firing sequences allows to determine the start and end times of presentation instances using constraint solving techniques. The test case generation for stress testing is to determine for each firing sequence the values of the variables that maximize resource consumption. These values give the schedule of a test case.

TEST CASE GENERATION

The number of execution traces that can be exhibited by a multimedia system is often infinite. This is partly because of the infiniteness of the time domain. It is generally infeasible to conduct exhaustive testing for multimedia systems. As such, we present in this section the utilization of reachability analysis and constraint solving to generate test cases that are likely to lead to anomalous system behaviors

Test system architecture

Testing of distributed multimedia systems has been discussed in [20]. A typical architecture of the testing system is shown in Figure 6. It consists of the following components:

- the test case generator, which generates and selects test cases based on the specification of the multimedia system;
- the test setup, which includes the tester and the implementation-under-test (IUT); the tester executes the test cases and monitors the IUT responses;
- an oracle, which compares the output of the IUT with the expected responses, and provides the verdict (i.e. whether the IUT passed or failed the test).

In this paper, we focus on the first component, namely, the *Test Case Generator*, that is responsible for the selection and generation of test cases. We are mainly interested in stress testing [3], which focuses on system testing under strenuous conditions.

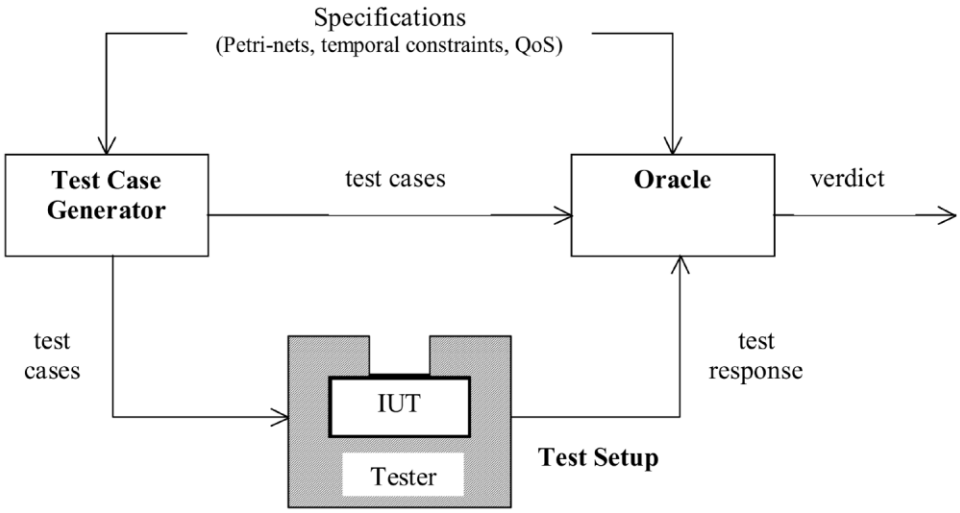


Figure 6. Test architecture for distributed multimedia systems.

Reachability analysis

The state space of a Petri net with k places is given by a set of *markings*, each of which represents the number of tokens being held in the places. A *marking* of a Petri net $(\mathcal{P}, \mathcal{T}, \mathcal{F})$, where $\mathcal{P} = \{P_1, \ldots, P_k\}$ can be represented by a tuple $(N_1, \ldots, N_k)$ where N_i $(1 \leq i \leq k)$ refers to the number of tokens being held by a place P_i. For instance, let $\mathcal{P} = \{\mathsf{Init}, \mathsf{Demo}, \mathsf{Full}, \mathsf{Quit}\}$ for the Petri net in Figure 4. The initial marking m_0 is therefore given by $(1, 0, 0, 0)$. A marking m is said to be reachable if it can be attained by some execution of the Petri net. As such, $(0, 1, 0, 0)$ is a *reachable marking* while $(0, 1, 1, 0)$ is not. A *reachability graph* of the Petri net $(\mathcal{P}, \mathcal{T}, \mathcal{F})$ is defined to be $(\mathcal{M}, \mathcal{E}, \mathcal{T})$, where:

- $\mathcal{M}$ is a set of reachable markings;
- $\mathcal{T}$ is the set of transition in the original Petri net;
- $\mathcal{E}$ $(= \mathcal{M} \times \mathcal{T} \times \mathcal{M})$ is a set of (m, t, m'), such that the state of the Petri net changes from m to m' with the firing of t.

Reachability graphs are very useful for analyzing the dynamic behavior of Petri nets. For example, the assertion that a Petri net is 1-safe can be verified using its reachability graph. Techniques to derive a reachability graph from a given Petri net are referred to as reachability analysis. Conventional reachability analysis techniques are available in [29]. However, reachability analysis is generally considered to be expensive for a non-trivial Petri net due to the state space explosion problem. Various techniques have been proposed to alleviate the problem, making reachability analysis more tractable in real-life applications [30–33]. Figure 7 presents the reachability graph of the Petri net in Figure 4.

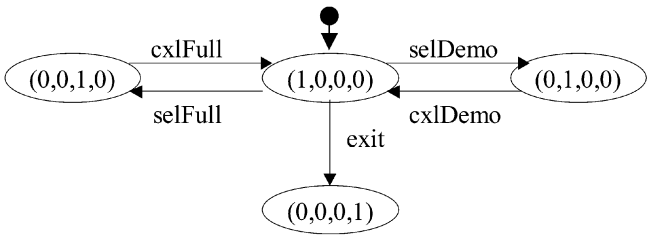


Figure 7. The reachability graph of the Petri net in Figure 4.

Reachability analysis is critical to stress testing. When a multimedia system reaches the same marking, it activates the same set of presentations with media objects confined to the same set of temporal constraints. As such, resource utilization can be analyzed by means of markings, each of which characterizes a specific resource utilization pattern. Thus, we can safely assume the amount of resources possibly consumed by a multimedia system can be characterized by a marking.

To ensure the coverage of all possible resource utilization patterns, all reachable markings should be considered in the process of test case generation. This implies that the test case selection strategy should be applied to a set of firing sequences that cover all reachable markings. This defines the *test coverage* adopted in our stress testing methodology. The coverage is analogous to the statement coverage in white box testing. The following are two possible firing sequence sets for the multimedia system in Figure 4. Both satisfy the test coverage criteria:

- $\{\langle(\mathsf{selDemo}, t_1), (\mathsf{cxlDemo}, t_2), (\mathsf{selFull}, t_3), (\mathsf{cxlFull}, t_4), (\mathsf{exit}, t_5)\rangle\}$, where $0 < t_1 < t_2 < t_3 < t_4 < t_5$
- $\{\langle(\mathsf{selDemo}, t_1), (\mathsf{cxlDemo}, t_2), (\mathsf{exit}, t_3)\rangle, \langle(\mathsf{selFull}, t_4), (\mathsf{cxlFull}, t_5), (\mathsf{exit}, t_6)\rangle\}$, where $0 < t_1 < t_2 < t_3 < t_4 < t_5 < t_6$.

The test case generation problem is therefore reduced to the problem of determining the schedule of each firing sequence that maximizes resource consumption.

Constraint solving

Let us now consider one firing sequence and present a constraint solving technique to determine the variable values that maximize resource utilization. The definition of a firing sequence involves the begin and end times of the presentation instances, but the timings for the objects will have to be calculated. The temporal relationship among the media objects can be described by formulas as mentioned. If the formulas contain only qualitative operators (such as Equals, Meets), they can be analyzed by polynomial-time algorithms [34]. Usually, however, the temporal specification also contains quantitative information (e.g. A Starts_after B by 3, $dur(A) = 5$). In this case, other techniques may be needed.

Linear programming

In a linear programming (LP) problem, the objective is to find values for a given set of variables such that an objective function is optimized while some conditions are satisfied. The objective function is linear with respect to the variables, and the conditions are either linear equations or inequalities. Techniques for solving LP problems are well developed and widely used in various applications. There are many tools available which are highly efficient. We use the package *lp_solve* (ftp://ftp.ics.ele.tue.nl/pub/lp_solve/) in our analysis.

For *lp_solve*, every variable is assumed to be non-negative. For the example in the previous section, suppose we want to know whether media object instances *VideoA* and *VideoB* can be active simultaneously. In this case, we solve the following set of linear equations and inequalities:

$$\alpha VideoB = \beta VideoA + 4, \quad \alpha VideoA = \alpha TextC, \quad \beta VideoB = \beta TextC,$$
$$\beta VideoA = \alpha VideoA + 5, \quad \beta VideoB = \alpha VideoB + 6, \quad \beta TextC = \alpha TextC + 15,$$
$$\alpha VideoA < t, \quad t < \beta VideoA, \quad \alpha VideoB < t, \quad t < \beta VideoB$$

For simplicity, we have omitted the objective function. The set of equations and inequalities does not have any solution. Thus, the media object instances *VideoA* and *VideoB* cannot be active at the same time.

Constraint solving

One restriction of the LP technique is that the conditions may only contain linear equations or inequalities. However, in some applications the conditions are Boolean combinations of arithmetic equations which need tools with more expressive languages.

Constraint Satisfaction Problems (CSPs) refer to a general class of problems where the values of some variables are to be determined such that a given set of constraints are satisfied. Historically, the CSP community did not pay much attention to the forms of constraints. In the mid-1980s, some researchers proposed the employment of the declarative features of logic programming in constraint solving. This is the so-called constraint logic programming (CLP) paradigm (see [35] for a survey in this area).

More recent tools (e.g. ILOG Solver [36] and NCL [37]) are not so closely related to logic programming. In some of our experiments, we use NCL, which is designed for solving constraints over finite integer domains. NCL accepts constraints such as the following:

$$(\alpha A > \beta B) \vee (\alpha B > \beta A) \quad \text{(i.e. } A \text{ and } B \text{ do not overlap)}$$

For such constraints, a LP package cannot be used directly. To solve a problem consisting of the above constraint and some other linear equations/inequalities, we need to solve two subproblems: one with the constraint $\alpha A > \beta B$, the other with the constraint $\alpha B > \beta A$. The original problem has a solution if either of the two subproblems has a solution.

Identifying scenarios of resource saturation

To maximize the peak usage of resources, we consider the resource types CPU, memory, network bandwidth, and display buffers. Suppose for some resource type the maximum is achieved at time t.

For simplicity of illustration, we assume that each object is active at most once. We introduce a set of binary variables which can only take the value of 0 or 1. If media object instance A is live at time t, $\lambda(A, t) = 1$; otherwise $\lambda(A, t) = 0$. To find the maximum resource usage, we study the following set of constraints:

- $t \leq$ TestPeriod;
- the temporal constraints involving A;
- the definition of the firing sequence (e.g. $\alpha P = t_1, \beta P = t_2, \ldots$);
- $\lambda(A, t) = 1 \Rightarrow (\alpha A \leq t \leq \beta A) \wedge (t \leq \beta P)$, where P is the presentation instance which consists of A. $(*)$

Let us denote the weight of a media object instance A to be ωA. The objective function to be maximized is therefore the summation of $(\lambda(A, t) * \omega A)$ over each object A and time t. The first constraint is used to restrict the time that a test case is allowed to run, and TestPeriod is a constant that can be set by the user. Usually there can be many test cases achieving the same amount of resource usage. Without the first constraint, we may generate a test case that runs too long. For the simple example in Figure 4, the following two test cases are the same in terms of peak resource usage:

$\langle$(selDemo, 2), (cxlDemo, 17)$\rangle$; $\langle$(selDemo, 2002), (cxlDemo, 2017)$\rangle$

In the first case, $t \leq 17$; in the second case, $t \geq 2000$. Obviously the former is better.

Alternative formulation

It is desirable to eliminate the logical operators in the constraints so that we can use a more efficient LP tool like *lp_solve*. For that purpose, we use the 0–1 integer variables $\mu(A, t)$ instead of $\lambda(A, t)$, where $\mu(A, t) = 1 - \lambda(A, t)$ (for every media object instance A). Then $(\lambda(A, t) = 1)$ is equivalent to $(\mu(A, t) = 0)$; and $(\lambda(A, t) = 0)$ is the same as $(\mu(A, t) = 1)$. The objective function to be optimized becomes the summation of $(1 - \mu(A, t)) * \omega A$ over each object instance A across time. Since the variable $\mu(A, t)$ is either 0 or 1, we may replace the constraint $(*)$ by the following inequalities:

$$t + \mathsf{K} * \mu(A, t) \geq \alpha A, \quad \beta A + \mathsf{K} * \mu(A, t) \geq t, \quad \beta P + \mathsf{K} * \mu(A, t) \geq t$$

Here K is a sufficiently large integer constant greater than βA (for any object instance A) and the TestPeriod. By looking at the two cases $\mu(A, t) = 0$ and $\mu(A, t) = 1$, we can easily know that the above constraints are equivalent to the constraint $(*)$. With the new formulation, we can use LP tools to solve the problem.

Tool support

We have built an automatic tool called *stem* that performs reachability analysis and generates inputs for the LP package *lp_solve*. The tool is available upon request from the authors. It is written in C, using YACC to parse the input specification. It runs on SUN Solaris and accepts line commands in the form of:

stem $-$ $\langle$*Time*$\rangle$ $<$ $\langle$*InputFile*$\rangle$

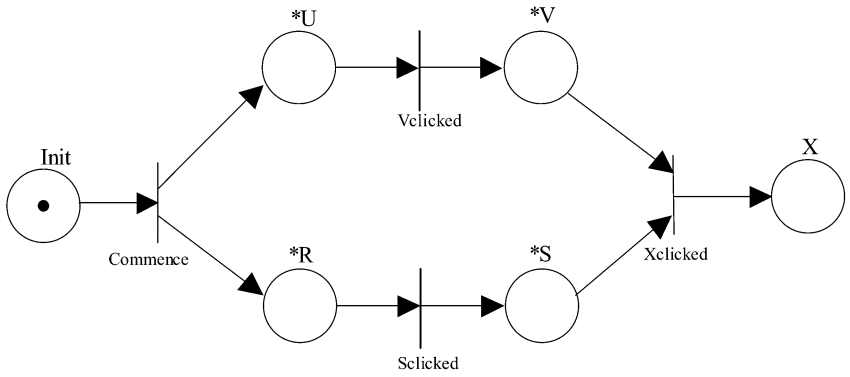


Figure 8. Another multimedia system.

The ⟨*Time*⟩ parameter gives the required TestPeriod in seconds while the ⟨*InputFile*⟩ specifies the Petri net, the media objects, and their temporal relationships. Please refer to Appendix A for an example of the input specification. The tool first generates a set of paths (starting from the initial state) such that every reachable state is visited. This is done by exploring the reachability graph. We may choose either the depth-first search (DFS) strategy or the breadth-first search (BFS) strategy. The BFS strategy may generate many short firing sequences. By 'short' we mean that the sequence has only a few transitions. Thus we implemented the DFS strategy in *stem*. After performing the reachability analysis, we obtained a set of firing sequences (or paths in the reachability graph). The tool *stem* formulates each sequence into a LP problem and saves it in an output file. For each problem, a test case is generated if a solution can be found using *lp_solve*.

Another example

There are four presentations in the multimedia system in Figure 8. The objects of each presentation and their temporal relationships are given as follows.

- U contains a presentation of objects A, B, and C. A Starts_with C and B Ends_with C.
- V contains a presentation of objects D and E. D Equals E.
- R contains only a presentation of one object, namely F.
- S contains a presentation of four objects: G, H, I, and J. H During G, I Starts_after G by 3, and J During H.

The durations and weights of the objects are given in Table II. To compare the different constraint solving methods, we normalized the weights to be integers. (Each of them is multiplied by 1000.)

We may use our tool *stem* to generate the test cases. The input is given in Appendix A. First, our tool finds the following two firing sequences:

⟨(Commence, t_{10}), (Sclicked, t_{11}), (Vclicked, t_{12}), (Xclicked, t_{13})⟩;
⟨(Commence, t_{20}), (Vclicked, t_{21})⟩

Table II. Durations and weights of the media objects in the example.

	A	B	C	D	E	F	G	H	I	J
Duration	5	6	15	8	8	10	20	10	5	6
Weight	100	1200	1800	56	1500	150	1440	25	200	230

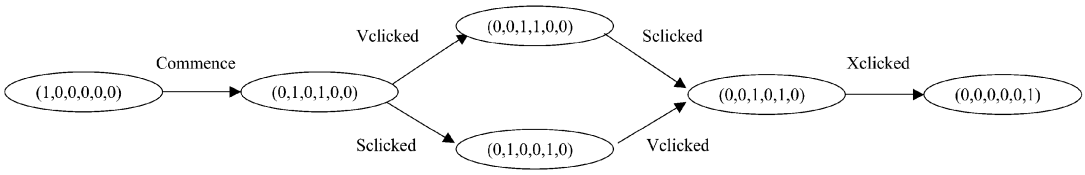


Figure 9. Reachability graph of the Petri net in Figure 8.

Here $t_{10} < t_{11} < t_{12} < t_{13}, t_{20} < t_{21}$. For simplicity, we assume that $t_{10} = t_{20} = 1$. With the above two firing sequences, all reachable markings are covered. Let us denote a marking by a vector of six integers (*Init*, U, V, R, S, X). Each field corresponds to the number of tokens in the places *Init*, U, V, R, S, and X. Figure 9 shows the reachability graph of the system.

Assume that TestPeriod equals 25 s. With the first firing sequence, NCL found an optimal solution in about 1 min on a SPARCstation 5: U and R are activated at time point 1, V and S are activated at time point 11 and 3, respectively. The starting times of the objects are as follows:

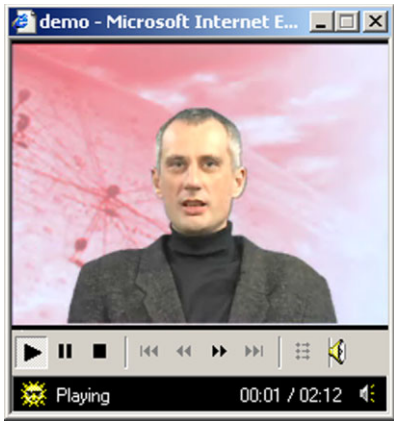
$$\alpha A = 1, \quad \alpha B = 10, \quad \alpha C = 1, \quad \alpha D = 11, \quad \alpha E = 11,$$
$$\alpha F = 1, \quad \alpha G = 3, \quad \alpha H = 4, \quad \alpha I = 6, \quad \alpha J = 5$$

When $t = 10$, the objects B, C, G, H, I, and J are active, and the objective function has the maximal value of 4895. Our tool is based on the new formulation of the problem, as described previously. With *lp_solve*, an optimal solution can be found within 1 s:

⟨(Commence,1), (R↑, 1), (A↑, 1), (C↑, 1), (U↑, 1), (F↑, 1), (A↓, 6), (Sclicked,7), (R↓, 7), (S↑, 7), (G↑, 7), (H↑, 8), (J↑, 9), (I↑, 10), (B↑, 10), (Vclicked,12), (U↓, 12), (V↑, 12), (D↑, 12), (E↑, 12), (I↓, 15), (J↓, 15), (H↓, 18), (Xclicked,25)⟩, (V↓, 25), (S↓, 25)

For this execution trace, the optimal time instant $t = 11$ and the maximal resource utilization is 4895. Note that in our method we do not need to solve a LP problem for every marking of the Petri net. One firing sequence may correspond to several markings. Consider the first sequence. Starting from the initial marking (1, 0, 0, 0, 0, 0), it visits the following markings: (0, 1, 0, 1, 0, 0), (0, 1, 0, 0, 1, 0), (0, 0, 1, 0, 1, 0), (0, 0, 0, 0, 0, 1). The critical time t_{cr} can occur at any point in the firing sequence. If it occurs before the firing of Sclicked, the system is in the marking (0, 1, 0, 1, 0, 0); if it occurs after the firing of Sclicked but before the firing of Vclicked, the system is in the marking (0, 1, 0, 0, 1, 0), and so on.

(a)



(b)

Figure 10. (a) A window opened for live-captured video of the instructor. (b) Pre-recorded presentation. (Reproduced by permission of eEd Vision Ltd.)

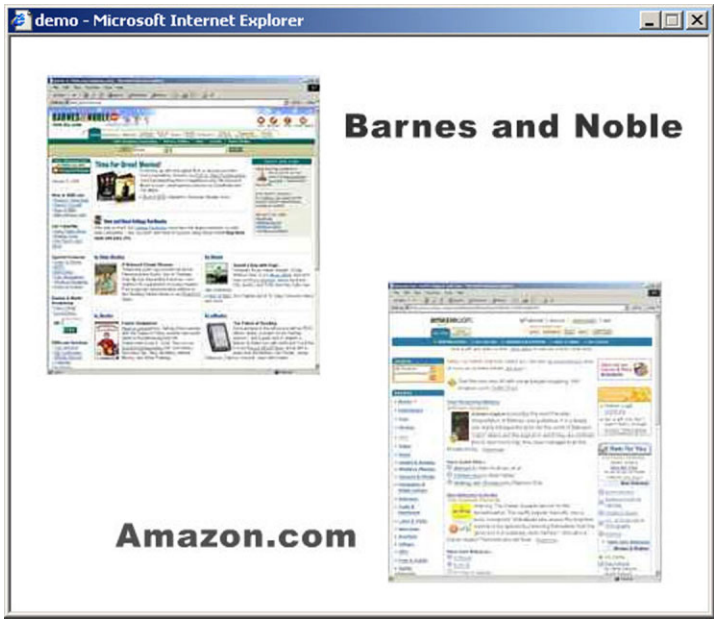


Figure 11. Two illustrations.

CASE STUDY

We applied the technique to a multimedia system for an online marketing course. Transmission of digital contents is based on streaming technologies over the Internet. Test cases were generated for the conduction of stress testing at the client end based on a Pentium III (933 MHz) computer running Windows 2000. The case study focuses on the CPU utilization.

Description

Initially, a window (see Figure 10(a)) was opened to broadcast the live-captured video of the course instructor. The live-captured video continues until the end of the online course. After introducing the topics to be covered, the instructor asked students to watch a presentation on eMarketing as shown in Figure 10(b). The presentation consists of two portions: *Intro* and *eMktShow*. This is modeled using two different places in the Petri net in Figure 12(a).

Later, students were asked by the instructor to *stop* the pre-recorded presentation and activate a flash presentation showing successful stories (as shown in Figure 11) based on Amazon.com and Barnes & Noble. After the flash show, students were asked to continue the second part of the video clip, which is eMktShow until the end. This online e-marketing course can be specified by the Petri net in Figure 12(a). The live-captured video LV and Flash Image GF do not have a fixed duration except the

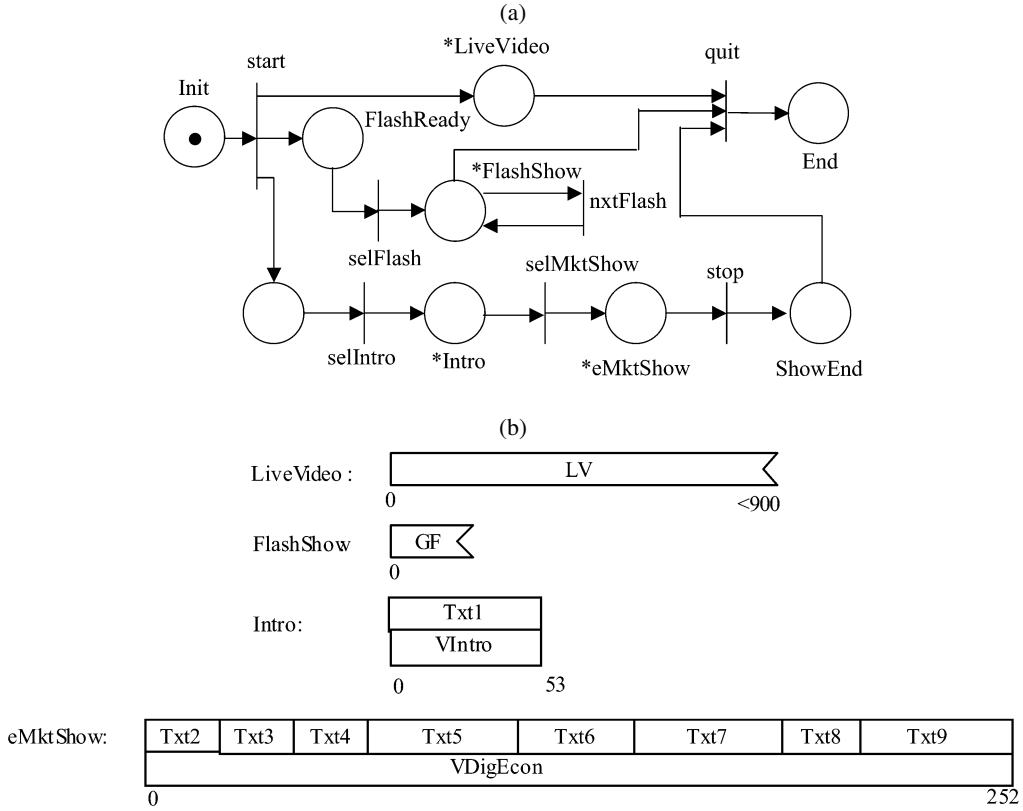


Figure 12. (a) The Petri net that models the control flow. (b) Temporal relations of media objects in various presentations.

online interaction is expected to finish in 15 min. There are in total five GIF images to be shown in turn by clicking a button corresponding to the firing of nxtFlash transition in the Petri net.

Test case generation

For the above Petri net, our tool found the following two firing sequences through reachability analysis:

$\langle$(start, t_{10}), (selFlash, t_{11}), (selIntro, t_{12}), (selMktShow, t_{13}), (stop, t_{14}), (quit, t_{15})$\rangle$;
$\langle$(start, t_{20}), (selIntro, t_{21}), (selMktShow, t_{22}), (stop, t_{23})$\rangle$

Here $t_{10} < t_{11} < t_{12} < t_{13} < t_{14} < t_{15}, t_{20} < t_{21} < t_{22} < t_{23}$. Let us consider the test cases that maximize the peak resource usage. The weights used in the constraint formulation are given in Table III. Here, we decompose a media object A into two parts, $\mathsf{A}_{\mathsf{buf}}$ and $\mathsf{A}_{\mathsf{dsp}}$, as explained. The former part $\mathsf{A}_{\mathsf{buf}}$ captures the buffering of digital contents before actual playing in the latter part $\mathsf{A}_{\mathsf{dsp}}$.

Table III. Weights and durations of various media objects in the case study.

	Contents buffering part (A_{buf})		Contents display part (A_{dsp})	
Media type	Weight w	Duration (s)	Weight w	Duration (s)
Txt1	–	–	0.01	53
Txt2	–	–	0.01	37
Txt3	–	–	0.01	20
Txt4	–	–	0.01	35
Txt5	–	–	0.01	15
Txt6	–	–	0.01	35
Txt7	–	–	0.01	20
Txt8	–	–	0.01	17
Txt9	–	–	0.01	73
GF	0.25	4	0.01	–
LV	0.8	6	0.10	<900
VIntro	0.8	6	0.10	53
VDigEcon	0.8	6	0.10	92

The same weight is assumed for the same media object type. Since 'text' objects can be loaded almost instantly, they do not contain the buffering part.

Assuming a TestPeriod of 100 s, we found the following two firing sequences and the associated time values.

- FR_1 : ⟨(start, 1), (selFlash, 2), (selIntro, 3), (selMktShow, 4), (stop, 99), (quit, 100)⟩, where the maximum CPU utilization should occur at the fifth second.
- FR_2 : ⟨(start, 1), (selIntro, 3), (selMktShow, 98), (stop, 100)⟩, where the maximum CPU utilization should occur at the fourth second.

These two firing sequences correspond to two test cases that altogether cover the reachable markings in the reachability graph. If we change TestPeriod from 100 to 1000, the generated test cases are similar. The software developers can then drive the system based on the two test cases and observe the resultant behavior.

Experimental results

Figures 13(a) and 13(b) give the actual CPU history of applying firing sequence FR_1 and FR_2, respectively. In Figure 13(a), the peak with a value close to 100% occurs at around 5 s after firing the *start* transition. The result is close to that derived from the constraints. We also observe an elongation of the buffering period (i.e. A_{buf}). Some jittering effects also appear in the eMktShow presentation. In Figure 13(b), the peak first occurs at 4 s after the firing of the *start* transition.

We encountered two problems in conducting stress testing experiments for multimedia systems. The first problem is finding real-life examples and identification of companies to sponsor the experimentation. The second problem is that few multimedia players support test control points or

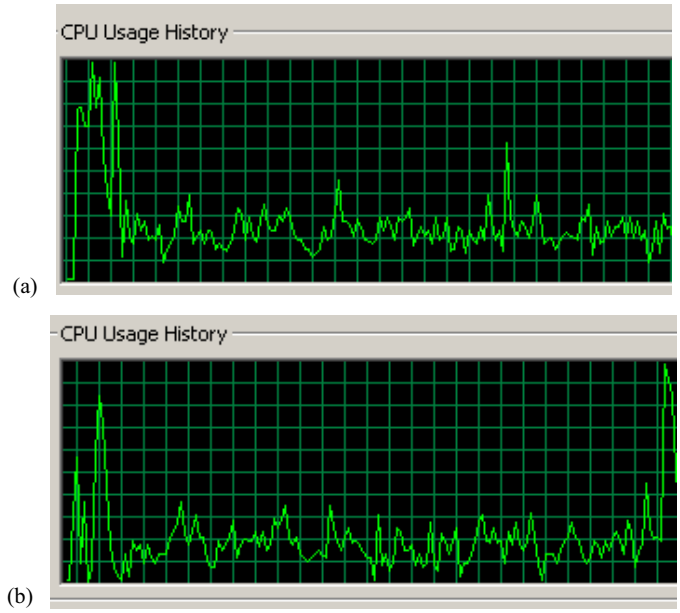


Figure 13. CPU history of firing sequence (a) FR_1 and (b) FR_2.

come with a software development kit allowing insertion of test control points. As such, we plan to develop a media player that supports test control points and the Synchronized Multimedia Integration Language (SMIL) [38] specification. SMIL is a recommendation of the World Wide Web Consortium for the authoring of interactive audiovisual presentations.

CONCLUSION

Due to the complexity of distributed multimedia systems, their execution may fail to meet some user requirements if testing is not done thoroughly before deployment. In this paper, the testing of this type of system is studied. We have focused on stress testing methodology, and defined several criteria for selecting test cases.

Moreover, we have presented methods for generating quality test cases from the specifications of multimedia software systems. Our techniques are applicable to the general case where the specifications consist of a temporal event Petri net and some temporal formulas for describing the relationships among media objects. To generate test cases, we first extract firing sequences from the Petri net. Then, for each sequence, we compute the event timings which maximize resource usage during a fixed finite period. We have given the necessary constraints and discussed how to formulate

the problem so that it can be solved more efficiently. The method is assisted by automated tools. Our experiments show that the test case generation can be carried out efficiently.

It should be noted that the automatic test case generation methods are based on some assumptions. Most importantly, we assume that there are linear resource consumption functions associated with the media objects (i.e. ωA is a constant). When the assumption does not hold, we may divide an object into several smaller objects each having a linear resource consumption function. We have shown with examples that the proposed techniques are quite efficient.

APPENDIX A

For the multimedia system in Figure 8, we may give the following as input to our tool:

```
PL Init (1) {
}
PL X (0) {
}
PL U (0) {
     MOBJ A WEIGHT 100 DURATION 5;
     MOBJ B WEIGHT 1200 DURATION 6;
     MOBJ C WEIGHT 1800 DURATION 15;
     A STARTSWITH C;
     B ENDSWITH C;
}
PL V (0) {
     MOBJ D WEIGHT 56 DURATION 8;
     MOBJ E WEIGHT 1500 DURATION 8;
     D EQUALS E;
}
PL R (0) {
     MOBJ F WEIGHT 150 DURATION 10;
}
PL S (0) {
     MOBJ G WEIGHT 1440 DURATION 20;
     MOBJ H WEIGHT 25 DURATION 10;
     MOBJ I WEIGHT 200 DURATION 5;
     MOBJ J WEIGHT 230 DURATION 6;
     H DURING G;
     I STARTSAFTER G BY 3;
     J DURING H;
}

TR Commence {
     Init -> U, R
```



```
}
TR Sclicked {
     R -> S
}
TR Vclicked {
     U -> V
}
TR Xclicked {
     V, S -> X
}
```

Here, PL is a keyword denoting a place. It is followed by the name of the place and the number of tokens initially. If the place is associated with a presentation, the media objects in the place and their temporal relationships will be specified. The keyword TR denotes a transition.

ACKNOWLEDGEMENTS

We would like to thank eEd Vision Limited (www.eedvision.com) for the provision of their production contents and technical assistance in conducting the experiments. We would also like to acknowledge Samuel Chanson and the anonymous reviewers for their constructive suggestions on previous drafts of this paper. The work was partly supported by the Hong Kong Research Grant Council under grant HKUST6159/98E, the Croucher Foundation Fellowship and the National Science Fund for Distinguished Young Scholars under grant 60125207.

REFERENCES

1. Blakowski G, Steinmetz R. A media synchronization survey: Reference model, specification and case studies. *IEEE Journal on Selected Areas in Communications* 1996; **14**(1):5–35.
2. Horn U, Girod B. Scalable video transmission for the Internet. *Computer Networks and ISDN Systems* 1997; **29**:1833–1842.
3. Beizer B. *Software System Testing and Quality Assurance*. Van Nostrand Reinhold: New York, NY, 1984.
4. Sarikaya B. *Principles of Protocol Engineering and Conformance Testing*. Ellis Horwood: New York, NY, 1993.
5. Ates AF *et al.* Using timed CSP for specification, verification and simulation of multimedia synchronization. *IEEE Journal on Selected Areas in Communications* 1996; **14**(1):126–137.
6. Ates AF, Sarikaya B. Test sequence generation and timed testing. *Computer Networks and ISDN Systems* 1996; **29**(1):107–131.
7. Blair L *et al.* Formal specification and verification of multimedia systems in open distributed processing. *Computer Standards and Interfaces* 1995; **17**:413–436.
8. Little TDC, Ghafoor A. Synchronization and storage models for multimedia objects. *IEEE Journal on Selected Areas in Communications* 1990; **8**(3):413–427.
9. Prabhakaran B, Raghavan SV. Synchronization models for multimedia presentation with user participation. *Proceedings of ACM Multimedia'93*, Anaheim, CA, August 1993. ACM, 1993; 157–166.
10. Senac P, de Saqui-Sannes P, Willrich R. Hierarchical time stream Petri net: A model for hypermedia systems. *Proceedings of the 16th International Conference on Application and Theory of Petri Nets*, Torino, Italy, June 1995. Springer, 1995; 451–470.
11. Wang HK, Wu J-LC. Interactive hypermedia applications: A model and its implementation. *Software—Practice and Experience* 1995; **25**(9):1045–1063.
12. Chun W, Amer PD. Test case generation for protocols specified in Estelle. *Proceedings of Formal Description Techniques III, 3rd International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE'90)*, Madrid, Spain, November 1990. North-Holland, 1990; 191–206.

13. Chanson ST, Zhu J. A unified approach to protocol test sequence generation. *Proceedings IEEE INFOCOM'93, 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, San Francisco, CA, March 1993. IEEE Computer Society, 1993; 106–114.
14. Santos CAS, Soares LFG, de Souza GL. Courtiat J-P. Design methodology and formal validation of hypermedia documents. *Proceedings of 6th ACM International Multimedia Conference (ACM Multimedia'98)*, Bristol, U.K., September 1998. ACM, 1998; 39–48.
15. Courtiat J-P, de Oliveira RC. A reachability analysis of RT-LOTOS specifications. *Proceedings of the IFIP TC6 8th International Conference on Formal Description Techniques*, Montreal, Quebec, October 1995. Chapman and Hall, 1995; 117–124.
16. Higashino T, Bochmann Gv. Automatic analysis and test case derivation for a restricted class of LOTOS expressions with data parameters. *IEEE Transactions on Software Engineering* 1994; **20**(1):29–42.
17. Mandrioli D *et al.* Generating test cases for real-time systems from logic specifications. *ACM Transactions on Computer Systems* 1995; **13**(4):365–398.
18. Avritzer A, Weyuker EJ. Generating test suites for software load testing. *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, Seattle, Washington, August 1994. ACM, 1994; 44–57.
19. Yang CSD, Pollock LL. Towards a structural load testing tool. *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, 1996; 201–208.
20. Misic VB, Chanson ST, Cheung SC. Towards a framework for testing distributed multimedia software systems. *Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems*, Kyoto, Japan, April 1998. IEEE Computer Society, 1998; 72–81.
21. Cheung SC, Chanson ST, Xu Z. Towards generic timing test of distributed multimedia software systems. *Proceedings 12th International Symposium on Software Reliability Engineering (ISSRE 2001)*, Hong Kong, November 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001; 210–220.
22. Zhang J, Cheung SC, Chanson ST. Stress testing of distributed multimedia software systems. *Proceedings of Formal Methods for Protocol Engineering and Distributed Systems (FORTE XII/PSTV XIX'99)*, Beijing, October 1999. Kluwer Academic Publishers, 1999; 119–133.
23. Wolf LC. *Resource Management for Distributed Multimedia Systems*. Kluwer Academic Publishers: Boston, MA, 1996.
24. Hirzalla N, Falchuk B, Karmouch A. A temporal model for interactive multimedia scenarios. *IEEE Multimedia* 1995; **2**(3):24–31.
25. Allen JF. Maintaining knowledge about temporal intervals. *Communications of the ACM* 1983; **26**(11):832–843.
26. Little TDC, Ghafoor A. Interval-based conceptual models for time-dependent multimedia data. *IEEE Transactions on Knowledge and Data Engineering* 1993; **5**(4):551–563.
27. Brauer W, Reisig W, Rozenberg G (eds.) Petri nets, central models and their properties. *Advances in Petri Nets 1986, Part I. Proceedings of an Advanced Course*, Bad Honnef, September 8–19, 1986 (*Lecture Notes in Computer Science*, vol. 254). Springer: Berlin, 1987.
28. Murata T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* 1989; **77**(4):541–580.
29. Peterson JL. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall: Englewood Cliffs, NJ, 1981.
30. Cheung SC, Kramer J. Context constraints for compositional reachability analysis. *ACM Transactions on Software Engineering and Methodology* 1996; **5**(4):334–377.
31. Cheung SC, Kramer J. Checking safety properties using compositional reachability analysis. *ACM Transaction on Software Engineering and Methodology* 1999; **8**(1):49–78.
32. Godefroid P, Peled D, Staskauskas M. Using partial-order methods in the formal validation of industrial concurrent programs. *IEEE Transactions on Software Engineering* 1996; **22**(7):496–507.
33. Valmari A. The State Explosion Problem. *Lectures on Petri Nets I: Basic Models. Advances in Petri Nets*, Dagstuhl, Germany, September 1996. Springer, 1996; 429–528.
34. Gerevini A, Schubert L. Efficient algorithms for qualitative reasoning about time. *Artificial Intelligence* 1995; **74**(2):207–248.
35. Jaffar J, Maher MJ. Constraint logic programming: A survey. *Journal of Logic Programming* 1994; **19/20**:503–581.
36. Ilog, ILOG optimization suite. http://www.ilog.fr/products/optimization/.
37. Zhou J. Introduction to the constraint language NCL. *Journal of Logic Programming* 2000; **45**:71–103.
38. World Wide Web Consortium. http://www.w3.org/TR/smil20/.

