An Open Framework for Video Content Analysis

Chia-Wei Liao, Kai-Hsuan Chan, Bin-Yi Cheng, Chi-Hung Tsai, Wen-Tsung Chang, and Yu-Ling Chuang*
*Advanced Research Institute, Institute for Information Industry, Taipei, Taiwan, R.O.C.
E-mail: cliao@iii.org.tw, kaihsuan@gmail.com, binyi@iii.org.tw,
brick@iii.org.tw, wtchang@iii.org.tw, ilmachuang@iii.org.tw

***Abstract*—In the past few years, the amount of the internet video has grown rapidly, and it has become a major market. Efficient video indexing and retrieval, therefore, is now an important research and system-design issue. Reliable extraction of metadata from video as indexes is one major step toward efficient video management. There are numerous video types, and theoretically, everybody can define his/her own video types. The nature of video can be so different that we may end up, for each video type, having a dedicated video analysis module, which is in itself nontrivial to implement. We believe an open video analysis framework should help when one needs to process various types of videos. In the paper, we propose an open video analysis framework where the video analysis modules are developed and deployed as plug-ins. In addition to plug-in management, it provides a runtime environment with standard libraries and proprietary rule-based automaton modules to facilitate the plug-in development. A prototype has been implemented and proved with some experimental plug-ins.**

I. Introduction and Related Work

With the development of video infrastructure such as faster network, cheaper and bigger storage, and popularity of digital cameras, the internet videos grow rapidly.

For example, 60 hours of video clips is, on average, uploaded to YouTube per minute, and over 4 billion video clips viewed per day [1]. In this tremendous growth of media data, videos can be searched more efficiently, if they are accurately tagged or annotated. Manual annotation is feasible when there are a small number of videos. We will need an automatic annotation/tagging mechanism when facing a sea of videos. Automatic annotation is a challenging problem [2] due to the difficulty in classification. Video analysis entails computer vision technologies and related domain knowledge (e.g. sports videos).

There is an abundance of the survey of integrated automatic annotation. In Reference [3], video abstracts are used for multimedia archives, and videos are segmented and analyzed to extract special features (e.g. faces, dialog, and text). These features help assemble video clips of interest into an abstract. References [4][5] provide a comprehensive overview of existing video abstract methods, problem formulation, result evaluation, and systematic classification of different approaches.

In addition to the integrated surveys, much research has been done on annotation of specific video types. Some researchers focus on the unified framework. By recognizing the common audio events [6][7][8], the events can be detected in sport videos (such as baseball, golf, soccer, swimming, and races). The use of low level visual features (such as color histogram, and oriented gradients) to detect highlights is proposed in [9][10][11]. References [12][13] suggest content-based retrieval to detect the court, players and the ball in a tennis game. Frameworks, which combine the visual and audio features to extract multimedia highlights, are introduced in [14][15][16].

The linguistic annotation tools, such as Transana, produce descriptive or analytic notations from raw data such as video [17], and are used by professional researchers to analyze digital video or audio data. Most linguistic annotation tools conduct analysis manually, and should be considerably benefited by automatic annotation if available.

In this paper, we propose an open automaton-based video analysis framework, where video analysis modules can be dynamically managed (e.g. added and deleted). Moreover, our framework lowers the threshold of the development of video analysis modules. We first give a high-level overview of our framework and depict major components afterwards. Then, an experiment based on a tennis plug-in (analyzer) validates our system design. At the end, the conclusion and future work are addressed.

II. Framework

Video metadata is a high-level representation of the original video. Indexed properly with right metadata, video can be efficiently retrieved and searched in a database. Extraction of reliable and useful metadata from a video involves video understanding and content analysis, and it is always a tall order in terms of both algorithms and system design (and implementation). With a framework providing run-time libraries and higher-level control commands, developers can focus more on algorithm development, instead of detailed and tedious system design and implementation (such as flow control and cache for optimization).

There are virtually an unlimited number of video types, and oftentimes the video type definition and classification are subjective. Each video type could need a dedicated content analyzer, and each analyzer, itself, is a challenge in both of the algorithm and system development. It is impractical to expect a single system to handle most of the video types.

Our proposed framework comes with a runtime environment for video analysis and a manager managing videos and their analyzers. In this framework, video analyzers can be added or removed freely as plug-ins. Video applications access plug-ins through the framework. The

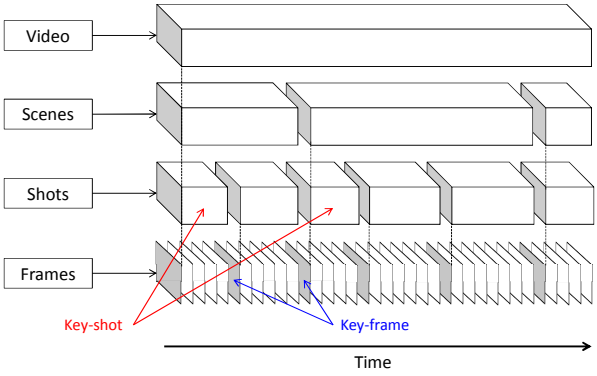

Fig. 1 Video structure model.

standard and proprietary libraries and an automaton-based rule engine, available in this framework, facilitate the development of the analyzers and their applications.

This content analysis framework is analogous to multimedia frameworks, such as DirectShow and GStreamer, in design, where complicated applications are more easily developed. Our plug-in (i.e. content analysis analyzers) is similar to the filter that can be dynamically added and deleted in DirectShow [18].

There are three stages in the pipeline of the framework– key frame detection, scene segmentation, and event detection. Scene segmentation is needed for structured videos (such as tennis), where "scene" is relatively clearly defined. Videos without clear structure (like drama) will go without scene segmentation.

Key frame detection comes in two flavors. The first passively detects the discontinuity created in post video editing (such as hard cuts, fades, dissolves, and wipe). The second actively detects frames with certain features in a continuous video (e.g. from a surveillance camera). Currently, our system detects hard cuts only, and others are yet to be implemented.

We have a framework prototype available, where a tennis plug-in has been developed. More plug-ins are on the drawing board to complete our framework.

A. Main System Components

Let's start with terminologies used throughout this paper. As in Fig. 1, a shot is an uninterrupted clip divided by key-frames. It is a physical entity. A scene is a collection of semantically related and temporally adjacent shots, depicting and conveying a high-level concept. A key-frame is defined by either some kinds of continuities or features in video. A key-shot is the beginning shot of a scene.

Our content analysis framework is scene-based. A plug-in detects video events (e.g. highlights) in each scene. For videos without clearly-defined structure, each shot is a scene. It first detects video key-frames to determine shots, group shots into scenes, and then detect events in scenes.

There are three main roles in the system: framework, plug-ins, and video applications. Video analysis is done by plug-ins. Framework manages plug-ins and provides infrastructure,

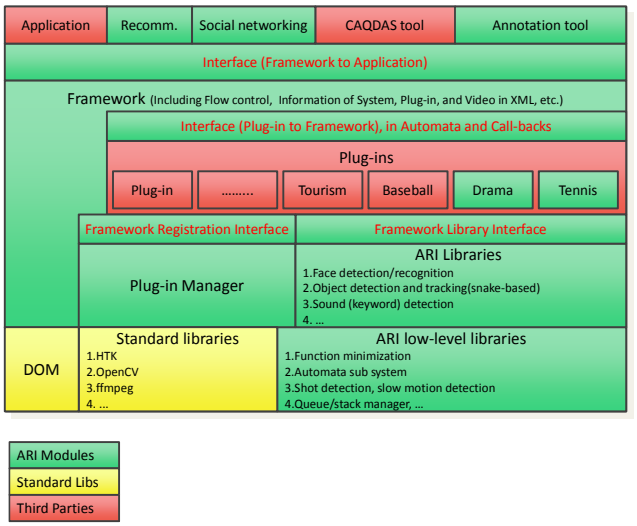

Fig. 2 The software stack of the framework.

such as run-time libraries, to plug-ins and applications. Applications (for example, automatic video annotation) are built on top of the framework, and access the video metadata and indirectly interact with plug-ins through the framework. Framework and plug-ins are our focus and will be elaborated on in the subsequent sections for scene segmentation and event detection.

B. Framework and Plug-in

Framework consists of standard and proprietary libraries as part of its run-time environment, and manages (e.g. register or de-register) content analysis plug-ins. Applications and plug-ins do not interact directly. Instead, framework serves as the liaison in between. For example, an application could send a query to framework about the capability of a plug-in.

Fig.2 shows the software stack of the framework. Framework manages plug-ins (e.g. registering, adding, deleting), and helps applications get detailed description and information of a plug-in. Each plug-in has its own attribute set, and attributes can be read or changed by the applications (through the framework) at runtime. Video metadata (i.e. tags and annotation) is generated in XML. To systematically optimize the system performance, dirty flags and cache are widely used throughout the framework.

C. Standard and Proprietary Libraries

Both plug-ins and applications have access to the framework libraries. Standard audio and video libraries such as OpenCV (for image processing), HTK, MITLM or SRILM (for audio signal processing) will be incorporated into our framework. DOM is helps generate metadata is in XML. Attributes of plug-ins are read and set through a Window-based (or Qt) graphics user interface system.

Rules, in the form of automata, are employed to segment scenes and detect events. Automata are heavily used through our system. An automaton module with a cache mechanism efficiently traverses automata, executes scene segmentation

and event detection, caches intermediate information, and returns the result.

Proprietary libraries (e.g. object detection and tracking, and the automaton subsystem) are under development, and will become part of framework.

D. Key-frame Detection for Shot Segmentation

Our framework first detects key-frames to segment shots (i.e. key-frame detection), group shots into scenes (i.e. scene segmentation), and then detect events in each scene (i.e. event detection).

A collection of active and passive key frame detectors will be implemented to process continuous videos (from surveillance cameras, for example) and edited videos with shot changes (such as hard cut, fade, and dissolve).

A simple-minded (but robust) hard-cut detector is available, which uses histograms to pinpoints significant discontinuity in distribution (of color or intensity) at the boundary of two neighboring shots. Detectors to handle more complex transitions in edited videos (e.g. fade and dissolve) are still under construction.

As to continuous video processing, automaton-based rules will be introduced for active shot detection. It is in the design phase. This automaton-based approach is, in design, similar to those taken in scene segmentation and event detection (described in the subsequent sections).

E. Automaton and Cache Mechanism

Automata are used to define the rules for scene segmentation and scene event detection. Before getting into the details of scene segmentation and event detection, we first explain the automaton mechanism.

Two types of final states, positive and negative, are defined in an automaton. When a positive final state is reached, the corresponding event of the automaton is detected. A low-level feature detector is attached to each automaton edge as a callback function. Starting with the initial state (as the current state), the embedded detector of each outbound edge of the current automaton state is invoked and executed. If the detector returns positively, the associated immediate neighbor (which is a state, too) will become a "next" current state. Otherwise, ignore this neighbor. This process continues till a final state is reached. In our framework, there are two types of automata for scene segmentation, and event detection, respectively. They are defined and provided by the content analysis plug-in, and executed by the framework.

A set of automata for scene-segmentation are applied to group consecutive video shots into scenes. Shots are sequentially injected into each automaton until any of them reach a positive final state, and thereafter a scene is segmented. Currently, if every automaton reaches a negative final state, the scene segmentation fails and the video shots are discarded. There is room for improvement, and we will discuss this later in the conclusion.

Then another set of event detection automata detects the events in scenes. More than one event can happen in a scene, and multiple detection automata may be associated with an event. Similar to scene segmentation, an event is detected if

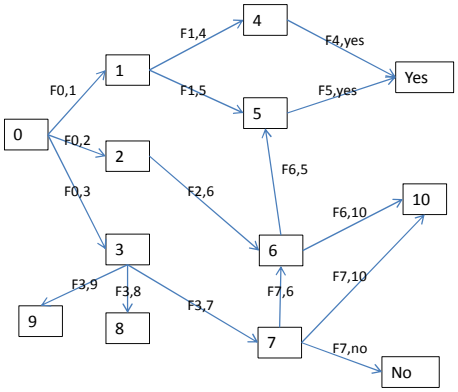

Fig. 3 A scene segmentation automaton, where state 0 is the initial state, detector $F_{0,1}$ is identical to detector $F_{0,3}$,and detector $F_{3,9}$ identical to detector $F_{1,4}$.

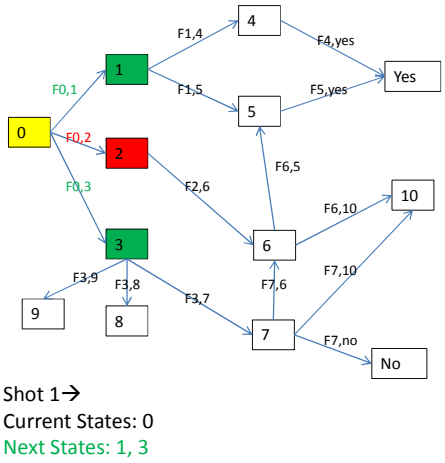

Fig. 4 State 0 is the current state (in yellow). Detector $F_{0,1}$ returns positively (in green), and state 1 becomes a next state. Since $F_{0,3}$ is the same as $F_{0,1}$ (cache), state 3 is also a next state (in green). $F_{0,2}$ returns negatively, so state 2 is rejected (in red). This process iterates till a final state is reached.

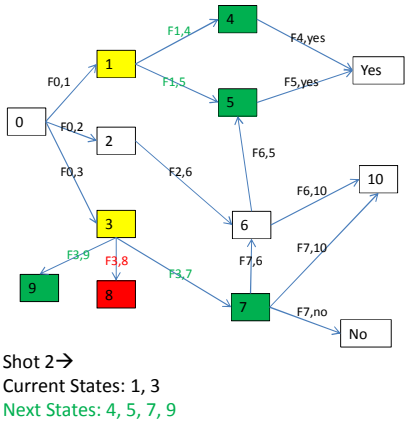

Fig. 5 Detector $F_{3,9}$ is the same as detector $F_{1,4}$. So they will be evaluated once only, with our cache mechanism.

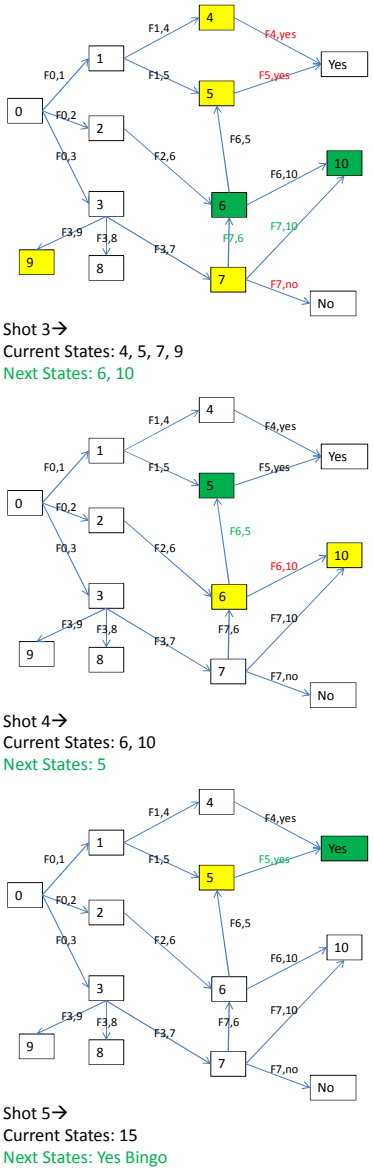

Fig. 6 A positive final state is reached, a key-shot is detected, and a scene has been segmented.

any of its automata reaches a positive final state, given a full scene. If all automata reach negative final states and fail, there are no events whatsoever in this very scene.

Cache is a key factor to the system performance. Detectors usually do signal or image processing. They are expensive in nature and might be invoked multiple times for a given data set. For example, a sound detector to detect the hit sound in tennis might be used in various automata or edges in the same automaton. To alleviate this problem, a cache mechanism is attached to each detector (and therefore each automaton edge). Oftentimes, a detector is used (or shared) by multiple edges or automata. The result of a detector is automatically cached in our framework, the first time it is called. When a detector is called on a video shot, we first check its dirty flag and see if the associated cache is valid. If yes, return the cached result immediately. Otherwise, execute the detector, cache and return the result. This cache mechanism is transparent to plug-in developers, and significantly improves the framework performance without their knowledge.

F. Scene Segmentation Automata

A set of automata is defined to segment scenes in a video. A video shot is injected, one by one, into each and every automaton. A scene is segmented when any of the automata hits a positive final state.

BFS (breadth first search) is applied to traverse a scene segmentation automaton. A set of current states and a set of next states are maintained during automaton traversal. The next states become the current states when a new video shot is fed into the automata (as illustrated in the figure below). The implementation is relatively complicated, because the behavior of an automaton also depends on subsequent "unknown" shots and some intermediate information needs to be kept.

An illustration of traversing a scene segmentation automaton is in Fig.3-Fig.6. When a video shot comes in, breadth-first search for all the next states (immediate neighbors of current states), and evaluate the detector (embedded in each edge). If the detector returns positively, the corresponding next (neighboring) state becomes a current state for the next video shot. Search stops when there are no current states or a final state has been reached.

An automaton is discarded if it hits a negative final state or all the detectors attached to the edges of the current states return negatively (e.g. no more next states). If all automata have been discarded, we abort the current scene segmentation and trash the shots processed so far (in the current scene segmentation). We will start anew at the next video shot for the next scene segmentation.

G. Scene Event Detection Automata

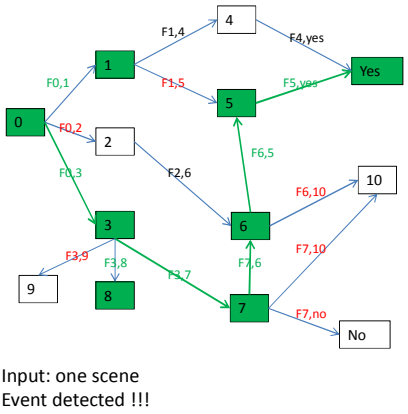

Fig. 7 Given a scene, a scene event detection automaton is depth-first traversed. Each detector is executed at most once, thanks to the cache mechanism. In this illustration, the thick green path shows how an event is detected in a scene.

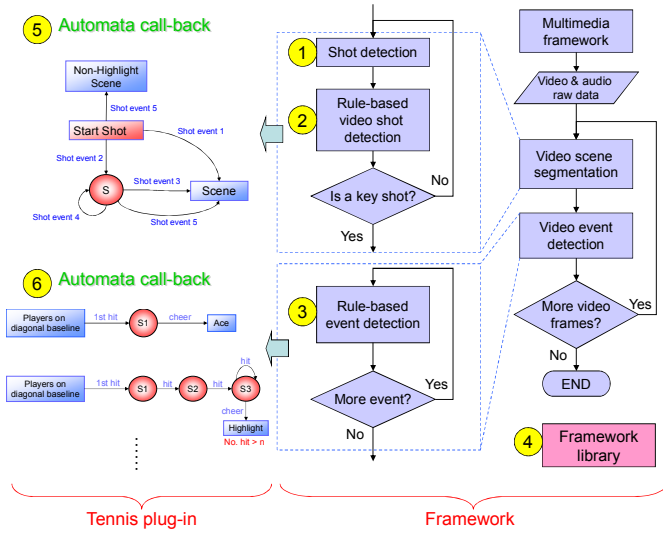

Fig. 8 Framework control and data flow.

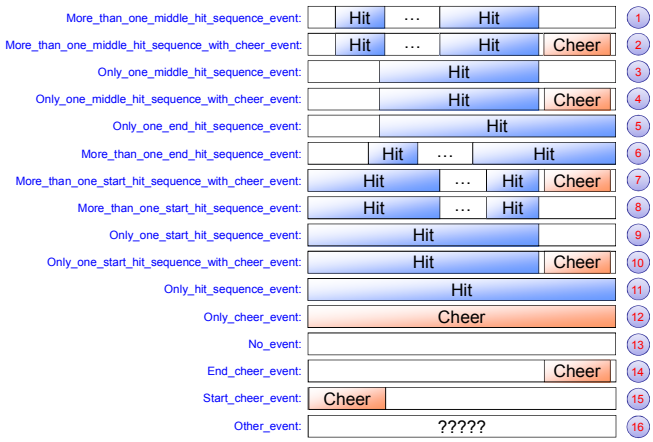

Fig. 9 The probable video shots.

A scene might have multiple events. A set of automata (multiple automata) is attached to an event. An event is detected in a scene if, at least, one of its automata reaches a positive final state.

Unlike scene segmentation, DFS (depth first search) is applied to traverse each automaton, given a video scene. An example is shown in Fig.7. All the scene data are available in hand when we examine video; the implementation of the automaton system is easier than that of the scene segmentation.

H. Design Strategy of Plug-in Automata

A plug-in, instead of the framework, defines the scene segmentation and event detection automata. Multiple rules can be resented by either one big complicated automaton or multiple simple automata. We recommend multiple simple automata. They are easier to maintain, and each automaton can be changed (or debugged) locally. In a big complicated automaton, any change to this big automaton would impact globally the evaluation of each and every rule. Debugging would be more difficult in this case.

Multiple small automata are more likely to share detectors and same detectors could be invoked multiple times on the same data. Thanks to the cache mechanism, a detector is executed once at most when a shot (or scene) is processed.

Automata represent high-level logics (e.g. rules and behaviors) of a plug-in, while detectors deal with low-level logics (such as image and signal processing). Plug-in developers need to focus only on high-level design and low-level processing, and are free from the tedious details of the system implementation. With built-in libraries of the framework, plug-ins would be lighter and easier to implement.

I. Framework Control and Data Flow

A Multimedia framework (ffmpeg, currently) decodes and pushes the audio/video (raw) data to an internal buffer of the framework. Fig.8 briefly displays of the data flow. Shot (key-

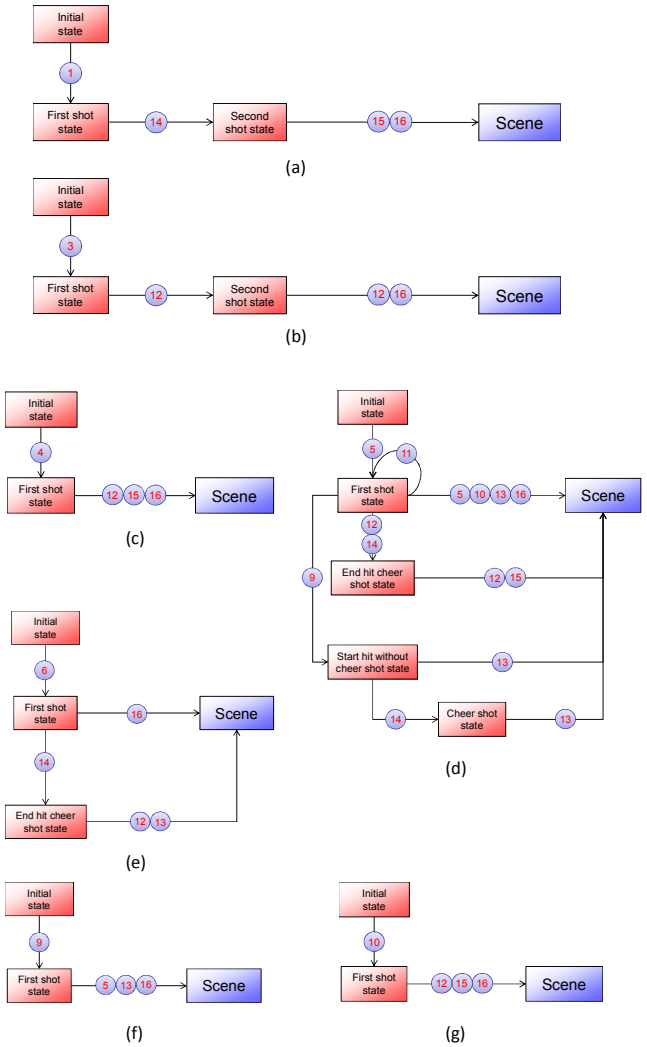

Fig. 10 The scene segmentation automata part I.

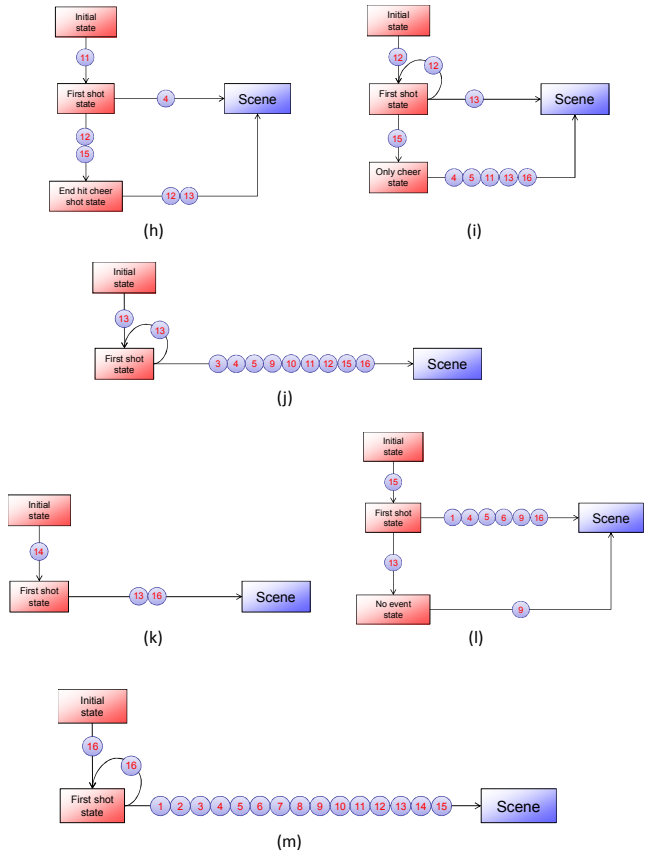

Fig. 11 The scene segmentation automata part II.

frame) is detected with a key-frame detector. Automata are employed to segment scenes and detect events. The result is exported in XML.

J. *Applications*

Various types of applications can be developed on our framework. They probably will follow a similar pattern in design. First, an application creates a context for the video and the video analysis plug-in, loop through the video to segment scenes and detect events, process the events (depending on the application), and finally delete the context.

III. Experiments for Tennis Video Event Detection

To validate our framework, we have implemented a tennis plug-in. Two sets of automata define the high-level logics of scene segmentation and event detection. The detectors attached to automaton edges are responsible for low-level feature detection and extraction. Some detectors compute the positions (and possibly orientations) of the tennis court and the players, and others spot sounds such as hit and cheer. Please refer to [19][20] for the detailed algorithm. Most events occur during the rallies between players. Rallies usually end with cheers [19], if there are great shots. Audio plays an important role in event detection, and audio features are cheaper to detect (then the video ones). Audio features help significantly reduce the search space in video processing (e.g. court and player tracking). The scene segmentation and event detection automata for the tennis analysis are listed as follows.

Since this plug-in is mainly to validate our framework, the lists of automata for scene segmentation and event detection are not complete and do not cover all possibilities.

A. *Automata Design of Scene Segmentation*

The scene segmentation is primarily based on audio, in this experiment. In Fig.9 is a list of the possible types of tennis shots. Fig.10 and Fig.11 show thirteen scene segmentation automata used in our experiment. These automata are designed based on the assumption that a non-trivial scene usually contains continuous hits and sometimes a cheer at the end.

We take automaton (d) in Fig.10 as an example to explain the basic idea of the scene segmentation. The first shot of a scene contains a sequence of hits toward its end, the middle ones is completely filled with hits, and the final shot starts with hits and ends with either a cheer or silence.

B. *Automata Design of Event Detection*

The event detection is launched after a scene is segmented. Here, we define four event types, and each event has one associated detection automaton (as shown in Fig.12). The

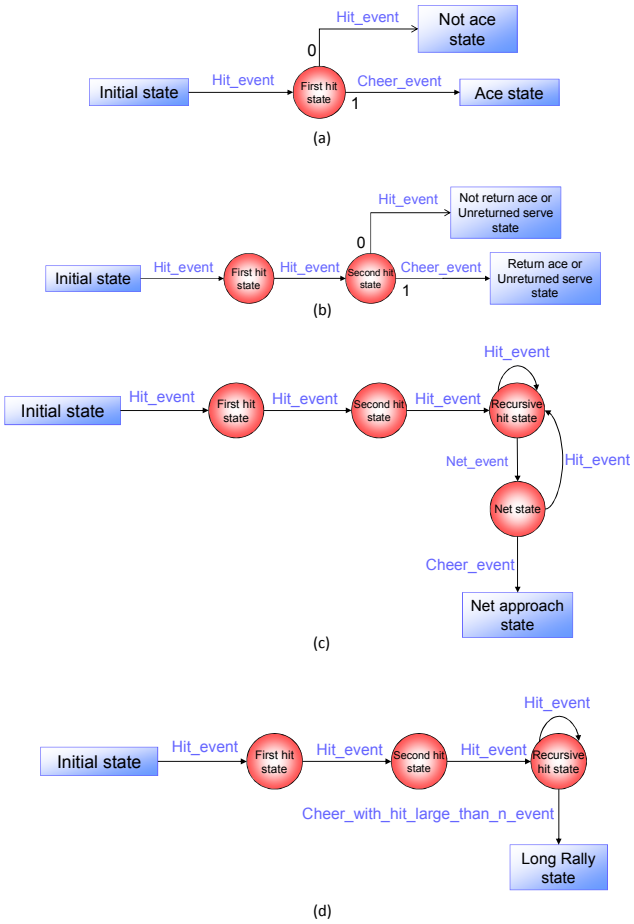

Fig. 12 The tennis event detection automata.

Fig. 13 Snapshots captured from the test tennis clips.

event types are ace, return ace or unreturned serve, net approach, and long rally, respectively. An event is detected if any of its automata hit a positive final state, given a scene.

For example, if a scene has a hit followed by a cheer, Automaton ace (a) will end up in a positive final state, and an ace event is therefore detected. If a player moves closer to the net before a cheer and after a sequence of hits, Automaton (d) will pick up a net-approach event.

C. *Experiments results*

In the experiment, we use two short tennis video clips. One is two-minute long and the other three-minute. The video format is MPEG-4, the frame rate 25 frames per second, and the resolution 512×384. This experiment is conducted on a PC with an Intel Core (TM) i7-870 CPU @ 2.93 GHZ 3.5GB RAM. Snapshots of the video clips are shown in Fig.13. A regular PC nowadays should suffice to process a video using our framework.

Totally, thirty-three scenes are segmented in the two test video clips. Most events are detected and the overall precision and recall rate are both about 97% (shown in Table I). Due to indistinct audio in part of the video clips, this plug-in is, somewhat, error-prone when detecting 'return ace or unreturned serve'. This experiment is to validate the design of the framework, though the result looks good.

The framework does mostly data management, and little computation. The system performance (i.e. accuracy, speed, and memory) depends mainly on the plug-in. The statistics of this experiment relate only to the tennis plug-in, and, in a sense, irrelevant to the design of the framework.

The systematic integration of a plug-in (providing only automata and low-level detectors) serves as a proof of the framework design.

IV. CONCLUSIONS

In the paper, we show an open video analysis framework, where a video analyzer can be dynamically inserted as a plug-in. It manages plug-ins, provides libraries, and offers a rule-based automaton module to facilitate the plug-in development. We have implemented a prototype, and developed a tennis plug-in on it. It is still under development.

TABLE I
PERFORMANCE OF EVENT DETECTION.

	Precision	Recall
Ace	100% (1/1)	100% (1/1)
Return ace or unreturned serve	50% (1/2)	100% (1/1)
Long rally	100% (4/4)	100% (4/4)
Net approach	100% (1/1)	100% (1/1)
Not highlight	100% (24/24)	96% (24/25)
Total	97% (32/33)	97% (32/33)

This framework does mostly data management, and little computation. The internal cache mechanism makes the overall system-wide optimization possible and easy. The bottleneck of the system performance (e.g. computation time and memory footprint) lies primarily in the feature detectors of the content analysis plug-ins.

Below is the list of our future work. We plan to have our framework production-ready next year.

1. Our key-frame detector handles hard cuts (abrupt transitions) only. Graduation transitions need to be taken care of soon. Active automaton-based key-frame detectors will be developed to deal with continuous videos (e.g. surveillance videos).
2. Video is analyzed and processed in raw data. This requires humongous amount of memory. A better resource management strategy is needed to handle multiple videos simultaneously. This is a scalability problem.
3. Keep error in check and localize errors of automata and callbacks. Scene segmentation automata and detectors are complicated, and the implementation might be flawed or error-prone. Due to data dependency, a mistake in scene segmentation might ripple through the rest of the video, and cause the segmentation failure in the following scenes. For example, when we discard video shots as discussed in the previous paragraph (when all segmentation automata fail), the following scene segmentation could fail, too, due to bad starting key-shots. We are trying a new strategy which, we believe, will alleviate this problem, and localize (and contain) the influence of a mistake in scene segmentation. One possible solution is to launch a new set of scene segmentation automata for each shot. This would help localize a scene segmentation mistake at the cost of more computation time.
4. Develop more plug-ins and applications to validate our framework, integrate more standard libraries (such as multimedia framework), and make it more complete and robust. Now, we have OpenCV in our framework. Proprietary libraries for object (face) detection and tracking modules and sound detectors to the framework will be included. In the long run, we hope to have a rich set of libraries to facilitate the plug-in and application development, and lower the entrance barrier. Amateurs

can create their own simple-minded video analyzers simply with GUI-based automaton authoring tools and the libraries coming with framework.

5. Integrate with the linguistic annotation tools (such as Transana), define a good UI operation flow, and create a system where manual and automatic solutions blend well.

REFERENCES

[1] YouTube website. Online URL <http://www.youtube.com>.

[2] C.G.M. Snoek, M. Worring, J.C. van Gemert, J.-M. Geusebroek, and A.W.M. Smeulders, "The Challenge Problem for Automated Detection of 1.01 Semantic Concepts in Multimedia," *In Proceedings of ACM Multimedia*, pp. 421-430, October 2006.

[3] R. Lienhart, S. Pfeiffer, and W. Effelsberg, "Video Abstracting," *Communications of the ACM*, vol. 40, no. 12, pp. 55-62, December 1997.

[4] Y. Li, T. Zhang, and D. Tretter, "An Overview of Video Abstraction Techniques," HP Labs Technical Reports, 2001.

[5] B.T. Truong and S. Venkatesh, "Video Abstraction: A Systematic Review and Classification," *ACM Transactions on Multimedia Computing, Communication and Application*, vol. 3, pp. 3, 2007.

[6] Z. Xiong, R. Radhakrishnan, A. Divakaran, and T. S. Huang, "Audio Events Detection Based Highlights Extraction from Baseball, Golf and Soccer Games in a Unified Framework," *In Proceedings of ACM International Conference on Multimedia*, vol. 3, pp. 401-404, 2003.

[7] D. Tjondronegoro and Y.P.P. Chen, "Integrating Highlights for more Complete Sports Video Summarization," *IEEE Transactions on Multimedia*, vol. 11, pp.22-37, October-December 2004.

[8] B. Zhang, W. Dou, and L. Chen, "Audio Content-based Highlight Detection Using Adaptive Hidden Markov Model," *In Proceedings of 6th International Conference on Intelligent System Design and Application*, pp. 798-803, China, 2006.

[9] Y.Q. Yang, Y.D. Lu, W Chen, "A Framework for Automatic Detection of Soccer Goal Event Based on Cinematic Template," *In Proceedings of the International Conference on Machine Learning and Cybernetics*, vol. 6, pp. 3759-3764, 2004.

[10] B. Han, Y. Yan, Z. Chen, C. Liu, and W. Wu, "A General Framework for Automatic On-line Replay Detection in Sports Video," *In Proceedings of the 17th ACM International Conference on Multimedia*, pp. 501–504, October 2009.

[11] H. Tang, V. Kwatra, M.E. Sargin, U. Gargi, "Detecting Highlights in Sports Videos: Cricket as a Test Case," *In Proceedings of 2011 IEEE International Conference on Multimedia & Expo – Workshop on Visual Content Identification and Search*, Barcelona, Spain, 2011 July.

[12] G. Sudhir, J.C.M. Lee, and A.K. Jain, "Automatic Classification of Tennis Video for High-Level Content-based Retrieval," *In Proceedings of the Workshop on Content-based Access of Image and Video Databases*, pp. 81-99, 1998.

[13] H. Miyamori and S. I. Iisaku, "Video Annotation for Content-based Retrieval using Human Behavior Analysis and Domain Knowledge," *In Proceedings of Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, pp. 320–325, 2000.

[14] Z. Xiong, R. Radhakrishnan, A. Divakaran and T. S. Huang, "Highlights Extraction from Sports Video Based on an Audio-Visual Marker Detection Framework," *In Proceedings of IEEE International Conference on Multimedia and Expo*, 2005.

[15] L. Agnihotri, J. Kender, N. Dimitrova, and J. Zimmerman, "Framework for Personalized Multimedia Summarization," *In Proceedings of the Workshop on Multimedia Information Retrieval*, pp. 81-88, 2005.

[16] M.H. Kolekar and K. Palaniappan, "A Hierarchical Framework for Semantic Scene Classification in Soccer Sports Video," *In Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology*, India, 2008.

[17] Linguistic Annotation Tools. Online URL < http://annotation.exmaralda.org/index.php/Linguistic_Annotation>.

[18] DirectShow Filters. Online URL <http://msdn.microsoft.com/en-us/library/windows/desktop/dd375464(v=vs.85).aspx>.

[19] W.T. Chu, M.C. Tien, Y.T. Wang, C.W. Chou, K.Y. Hsieh, and J.L. Wu, "Event Detection in Tennis Matches Based on Real-World Audiovisual Cues," *In Proceedings of* the 20th IPPR conference on Computer Vision, Graphics, and Image Processing, pp. 541-548, August 2007.

[20] M.C. Tien, Y.T. Wang, C.W. Chou, K.Y. Hsieh, W.T. Chu and J.L. Wu, "Event Detection in Tennis Matches Based on Video Data Mining," *In Proceedings of IEEE International Conference on Multimedia & Expo*, pp.1477-1480, June 2008.