On the Applicability of the Multi Agent System Paradigm for Parsing Videos

Massimo De Santo, Member, IEEE, Mario Molinara, Gennaro Percannella
Università di Salerno - DIIIE
Via P.te don Melillo, 1, I-84084, Fisciano (SA), Italy
{desanto, molinara, pergen}@unisa.it

Abstract

In this paper we investigate the applicability of the MultiAgent paradigm to the realization of a distributed software system suitably designed for Mpeg video segmentation. This task is performed by combining at different levels video and audio information. We present a general discussion of the characteristics of Multi Agents showing how they satisfy the needs of the quoted video segmentation problem and we introduce the architecture of a prototypal implementation.

1 Introduction

More and more videos are generated every day, mostly produced and stored in analog form. However, the trend is toward the total digitization of movies and video products so that they potentially can be better organized and indexed. Improvement in the video production techniques, together with the availability of high speed networks and high performance, low cost computers, which allow effective multimedia processing and managing, has dramatically increased the amount of digital video in the last few years and, nowadays, video data bases of sequences stored in compressed form are widely available on the net, for education and/or entertainment. However, effective use of video in various domains is hindered by the difficulty of efficiently classifying and managing video data. Although browsing video sequences is a very important issue in many applications in which the user is required to choose few sequences among many, the tools available to accomplish such tasks remain quite primitive. A fundamental step toward an automated browsing and retrieving of digital video streams is to detect boundaries between consecutive camera takes in order to segment videos in elementary indexing units (shots). The simplest transition between two shots is represented by an abrupt transition, while more sophisticated shot changes are gradual transitions such as dissolve, wipe, fade-in, fade-out, etc. Much research work has been carried out to define ad-hoc algorithms able to automatically detect a particular type of the transitions described above. Such algorithms are generally designed utilizing some features extracted from the video track. Examples of used features are frame histograms, rate of change of frame histograms, subsequent frame differences, motion features, etc [1]. However, more effective segmentation can be obtained if semantically correlated shots are further aggregated into scenes [2]. Anyway, the latter is not a trivial task since it requires the comprehension and representation of the semantic content of the shots. Some algorithms try to solve the problem of automatic segmentation of video material at the scene level by simply clustering the shots characterized by a similar visual content generally represented through color histograms [3], [4]. More complex approaches making use of combined audio and video features are presented in [5], [6], [7].

In the approach proposed in [7], a scene break is associated with the simultaneous change of pictorial and audio features, while changes in the pictorial representation only denote a shot break. In [8], starting from these assumptions, more effective algorithms for Mpeg audio and video segmentation are presented together with the architecture of a system providing an integration framework for them. A scheme of the approach is shown in Fig. 1, where it is possible to notice the presence of different modules dedicated to different tasks. The modules constituting such architecture and their integration are motivated from the fact that complex perceptual tasks, such as analyzing either pictorial or audio information, can be optimally faced by combining in a flexible way different sub-modules involved in achieving the solution. Hence, each module is in turn constituted by different sub-modules working on the same stream of data: for instance, the VSM in Fig. 1(a) may be constituted by different units, each working on the video track, devoted to the detection of a different type of transition, as shown in Fig. 1(b). Here, in depth view of the VSM it is depicted showing the presence of the sub-modules for the detection of abrupt cuts (ACDM), of dissolves (DDM) and of fades (FDM) and the presence of the Video Information Fusion Module (VIFM) that collects and combines the results from the preceding sub-modules and works out the segmented video stream. Finally, it is
easy to figure out a deeper view (Fig. 1(c)), showing each sub-module constituted by further units devoted to the same task, but using different approaches. For example, it is possible to realize the DDM of Fig. 1(b) implementing different algorithms for dissolve detection and merging their results through a Dissolve Information Fusion Module (DIFM), in order to obtain a higher reliability than a single algorithm.

![Figure 1](image)

**Figure 1.** (a) Block diagram of the system architecture for Mpeg segmentation. (b) Detailed view of the VSM. (c) Detailed view of the DDM.

From previous considerations, it is evident that:
1. a system for video segmentation can be easily designed as composed by different units working independently and
2. for a given task several algorithms exist which can be used concurrently to obtain a more reliable result.

Furthermore a distributed and scalable realization of such a system could allow to exploit the intrinsic parallelism just shown. Having in mind the above mentioned goals the authors started a research project for verifying the applicability of MultiAgent paradigm to the problem of Mpeg video segmentation. The present paper is aimed to describe the overall architecture of a MultiAgent realization of a system for parsing Mpeg videos and is organized as follows. In section II, we discuss the agent paradigm and the reasons for which it satisfy all the requirements of the given problem. In section III, we present the prototype implementation of our system for Mpeg video segmentation exploiting the agent oriented approach. Finally, we provide a discussion of the potentiality of the approach and future direction of the research.

## 2 The Multi-Agent Paradigm

In order to justify our choice of the MultiAgent paradigm, it is worth giving the basics of what an agent and a MultiAgent System (MAS) are.

### 2.1 The agent concept

Arguably, the most significant improvements in the field of software engineering have resulted from the introduction of powerful abstractions for managing the software inherent complexity. The key advances in program design and development over the past three decades – procedural abstraction, abstract data types, and most recently, object oriented programming (OOP) – all represent increasingly powerful examples of such abstraction. Probably the single most compelling argument in favor of agents for software engineering is that they represent yet another such abstraction.

Even if it does not exist a commonly accepted definition of the agent concept, field experts have agreed upon a position: the roots of agent oriented (AO) developing methodology attain from OOP methodology [9], [10], [11] and AI studies [12]; hence, an intelligent agent may be defined as a decision making system that acts on and reacts to the environment.

### 2.2 The MAS concept

The concept of MultiAgent System is an outgrowth of the Distributed Artificial Intelligence community. In [13], a MAS is defined as “a loosely-coupled network of problem solvers that work together to solve problems that are beyond their individual capabilities”. These problem solvers are called “agents” and are essentially autonomous, distributed and maybe heterogeneous in nature, characterized by various degrees of problem solving capabilities and usually have a single locus of control and/or intention. In order to obtain coherent system behavior, in-
Individual agents in a MAS are not only able to share knowledge about the problems and solutions, but also to reason about the processes of coordination among other agents.

2.3 MAS for Mpeg video segmentation

Starting from the definitions stated above, it is easy to describe the reasons motivating our choice in favor of MAS technology for the problem of Mpeg video segmentation.

They can be summarized as follows:

a. Given that the agent paradigm is inherently distributed in nature, the intrinsic parallelism of our task can take advantage of the presence of a network of computers;

b. MAS architecture allow to obtain optimum solution through cooperation/competition techniques of various computing methods: for example, it’s possible to think of an adaptive software architecture for Mpeg segmentation, capable to obtain the optimum solution as a compromise between computation times, used descriptors and reliability of the solution.

c. MAS architecture can be designed so to be highly scalable (for example, adding and/or removing an Agent from the architecture): consequently we can easily realize a MAS that can be used as a testbed environment allowing rapid prototyping and testing of new segmentation algorithms.

d. Some of the existing MAS frameworks allow us to readily integrate stand alone legacy software (e.g., C code of existing segmentation algorithms) in a distributed software capable to interwork.

2.4 MAS implementation issues

The choice of a proper tool can arm the developer with many advantages, while being careless about it can prove to be constricting in the long run. The inherent difficulties encountered in implementing coordinated behavior in any MAS are essentially the following:

 Communication: how to enable agents to communicate; what communication protocols to use;

 Interaction: what language the agents should use to interact with each other and combine their efforts;

 Coherence and Coordination: how to ensure that the agents coordinate with each other to bring about a coherent solution to the problem they are trying to solve.

Another important issue is the programming language. We chose Java because of the following advantages: architecture neutral and portable, multithreading, network savvy, secure, object oriented, database connectivity, native methods.

After a careful examination of the state of the art [14], we selected JAFMAS (Java-based Agent Framework for MAS) [15], because it pays special attention in satisfying all the discussed points. Furthermore, JAFMAS provides a well defined methodology to support all phases of MAS software development (from requirement specification to coding), according to most commonly accepted Agent Oriented Software Engineering (AOSE) approaches [9], [16].

3 Implementation of a Video Parser with AOSE

In this section we give some details of the preliminary implementation of a system for Mpeg video segmentation exploiting the agent oriented approach. This system constitutes also a laboratory with which we can verify real applicability and efficiency of algorithms already developed for audio and video segmentation and combine their results. Starting from the system described in [7], we start recalling its principal aspects and then we show how it can be naturally implemented through the AOSE approach.

3.1 Video Parser Architecture

The organization of the system, proposed in [7], can be summarized as follows: a De-Multiplexing and Synchronization Module (DMSM) separates video and audio streams. The analysis of visual data is accomplished by a set of algorithms, organized in a Video Segmentation Module (VSM). The VSM, by working on basic components of visual information, provides candidate video cuts. Meanwhile, the Audio Segmentation Module (ASM) extracts basic audio information such as speech, music, silence, and proposes candidate audio cuts.

The preliminary results obtained by VSM and ASM are subsequently exploited by the Information Fusion Module (IFM) whose focus indeed is video scene detection. The IFM incorporates strategies and prior knowledge to cope with information from heterogeneous sources, with the capability of “feeding back” to VSM and ASM components.

An overall picture of the system is presented in Fig. 1.

3.2 AOSE Development

The production of software according to the most common AOSE approach ([9], [16]) requires to perform the following steps:

i. Identifying the agents;

ii. Identifying the logic interaction among agents;

iii. MAS implementation.
Identifying the agents

In our system it is possible to identify two groups of agents: Agents managing the user interfaces and data flows and those ones devised for computation.

System Agent -- The system agent implements the DSMSM of the architecture depicted previously. This agent performs two main tasks: firstly, it splits the Mpeg stream into its audio and video components. The coded stream is organized in \textit{packets}, each consisting of a header and a payload; the latter contains audio and video data organized in \textit{packets}, which share the same structure of the packs. So the system agent simply parses the Mpeg stream separating the audio and video packets and stores them into local Mpeg/audio and Mpeg/video streams. Secondly, this agent acts as a video and audio packets server satisfying the requests of the computation agents for new data to elaborate. Furthermore, it also extracts the timestamps from the original packet headers allowing the Information Fusion agent to maintain a temporal reference for the audio and video proposed cuts.

Video Agent -- The video agent implements the VSM of the architecture depicted in Fig. 1(a). More generally this agent could be considered as a family of agents operating on the Mpeg/video stream. In fact, as already described in Section I, in a system for video segmentation we could think of having various modules each able to perform a particular type of analysis on the video track: for instance, abrupt shot change detection, dissolve detection, fade-in and fade-out detection, camera movement characterization, shapes recognition, object tracking, image segmentation, etc.

In particular, in our system we have integrated two

\begin{figure}[h]
\centering
\begin{tikzpicture}
\node [circular arrow] (s0) {	extbf{s0}};
\node [circular label] (s1) [right of=s0] {s1};
\node [circular label] (s2) [below of=s1] {end};
\node [circular label] (s3) [left of=s2] {end};
\node [circular arrow] (s4) [left of=s0] {s0};
\node [circular label] (s5) [right of=s4] {s1};
\node [circular label] (s6) [below of=s5] {end};
\draw [arrow] (s0) -- (s1); % s0 to s1
\draw [arrow] (s1) -- (s2); % s1 to end
\draw [arrow] (s2) -- (s3); % end to end
\draw [arrow] (s3) -- (s4); % end to s0
\draw [arrow] (s4) -- (s5); % s0 to s1
\draw [arrow] (s5) -- (s6); % s1 to end
\end{tikzpicture}
\caption{Audio/Video/System – Interface conversation: (a) Interface Agent Point of view; (b) Audio/Video/System point of view.}
\end{figure}

agents respectively for abrupt and gradual (dissolve and fades) shot change detection. Both these agents receive the raw Mpeg/video data from the system agent and perform some computation to detect abrupt and gradual transitions between shots respectively according to the algorithms described in [17] and [18]. Meanwhile the partial results are forwarded to the Information Fusion and to the Interface agents.

Audio Agent -- For the audio agent it is possible to repeat the same reasoning of the video agent in the sense that it implements the ASM of the architecture in Fig. 1(a) and it can be thought as a family of agents operating on the Mpeg/audio stream. This is due to the fact that we could use several kinds of algorithm to analyze the audio stream: silence, speech, music, noise classification, speaker and speech recognition, music genre discrimination, etc. In particular, in our system we have integrated an audio agent able to perform a classification of the sound

\begin{figure}[h]
\centering
\begin{tikzpicture}
\node [circular arrow] (s0) {	extbf{s0}};
\node [circular label] (s1) [right of=s0] {s1};
\node [circular label] (s2) [below of=s1] {end};
\node [circular label] (s3) [left of=s2] {end};
\node [circular arrow] (s4) [left of=s0] {s0};
\node [circular label] (s5) [right of=s4] {s1};
\node [circular label] (s6) [below of=s5] {end};
\node [circular label] (s7) [left of=s6] {end};
\draw [arrow] (s0) -- (s1); % s0 to s1
\draw [arrow] (s1) -- (s2); % s1 to end
\draw [arrow] (s2) -- (s3); % end to end
\draw [arrow] (s3) -- (s4); % end to s0
\draw [arrow] (s4) -- (s5); % s0 to s1
\draw [arrow] (s5) -- (s6); % s1 to end
\draw [arrow] (s6) -- (s7); % end to end
\end{tikzpicture}
\caption{Audio/Video Agent - System Agent conversation: (a) Audio/Video Agent point of view; (b) System Agent point of view.}
\end{figure}

track according to the silence, speech, music and noise classes [18]. This agent receive the raw data from the system agent and outputs the partial results to the Interface and Information Fusion agents.

Information Fusion Agent -- This agent implements the IFM of Fig. 1(a) and merges the information from audio and video agents to perform a partitioning of the video at scene level. In particular, this agent works according to the simple algorithm in [7]; each shot transition is classified as a scene transition, when it occurs at the same time of a transition between two different audio classes (silence, speech, etc).
It is worth noting that in this preliminary system implementation we pointed our attention mostly on one of the key features of Agent Oriented Programming, i.e. the simplification of the analysis and design phase of distributed systems, while partially neglecting the aspect of the agents intelligence. In fact, while considering the system outlined in Fig. 1, it is possible to figure out how the intelligence can be mainly located in the various Information Fusion modules spread in the different layers of the system.

However, in future implementation we planned to introduce more sophisticated techniques for information fusion in order to obtain better performances in terms of both shot/scene correct detection and their classification according to the semantic content. In particular, we will detect scenes by firstly classifying the shots as dialog, action, etc, and finally aggregating in scenes time-adjacent shots belonging to the same class. In [5] and [19] preliminary algorithms pursuing such approach are presented.

In particular, the classification of shots will be performed by firstly computing a vector of descriptors of low semantic level (luminance, chrominance, edge regularity, textures, presence of skin tones, object motion, audio energy, pitch, bandwidth, pause rate, etc.) extracted on the video and audio tracks. Secondly, on the basis of this vector it will be possible to infer some metadata (in-outdoor environment, close-up, crowd, shot length, editing effects, silence / speech / music / noise presence, etc.), which in turn will represent higher semantic level descriptors for the shot. Finally, a scene will be detected by joining together time-adjacent shots showing similar semantic content represented through the higher semantic level descriptors.

**Interface Agent** -- The Interface Agent is primarily assigned to search other agents on the net and to gather computation results. It provides the user with a control panel through which he can be informed about the available agents on the net and the resources provided by them, set up their parameters according to the type of segmentation he wants to perform on the Mpeg video.

**Identifying the logic interaction among agents**

In this second step, we model the interaction between the agents in the form of conversations. A conversation is an agent plan to achieve some goal, based on interactions with other agents. We identify every possible conversation that each agent in our system can engage in, and represent those conversations by developing a finite state machine (FSM) model for each of them.

In our MAS two conversation types can be identified:
- conversation among Audio/Video Agents and a System agent in order to exchange elaboration data (see Fig. 3).
- conversation among Audio/Video Agents and a System agent in order to communicate the end of elaboration (see Fig. 2)

Anyway, it should be stressed that the agent interaction may also happen without any conversation, for the availability of two peer-to-peer basic communication mechanism furnished by JAFMAS: multicast communication (via UDP sockets and class D IP addresses) and direct communication (via RMI of Java) (see Fig. 4, 5). This peculiarity is very important in a framework for MAS realization, because it allows to support both broadcast and directed modes of communication.

In order to design a MAS able to find a coherent solution to the entire system problem, it is important to check the logical consistency of all agent conversations. Automata models like concurrent finite state machines and Petri Nets can provide useful tools for checking system coherency by analyzing the conversation models.

**MAS implementation**

The implementation of our system required the adaptation to Java of the existing legacy software to allow its integration into the distributed environment. The legacy
software was represented by the audio classifier and video cut and dissolve detectors, each written as a C or C++ stand alone executable.

All these software programs shared the same structure with two modules, as showed in Fig. 6: the splitter module for separating the Mpeg file under analysis in its audio and video components and the elaboration module, the latter specifying the task of the program. Each of these modules has been therefore transformed in a Win32 DLL encapsulated in a Java Agent (Fig. 7).

In particular, the splitter module, present in each application, was embedded in a unique System Agent; while each elaboration module was embedded in an agent for Audio or Video elaboration. The link layer between C world and Java world was implemented by employing Java Native Interface (JNI) library [20].

The second step required the generation of the Java infrastructure for MAS, starting from extension points introduced by the JAFMAS framework. In particular, we were asked to extend some JAFMAS classes and implement some other new Java classes to provide the system with the structures needed to work properly.

3.3 User Interface

Let us now give some details about our system from the user point of view. Three principal steps can be identified while using such system:
1. Computation set-up;
2. Computation processing;
3. Analysis of segmentation results.

It is worth noting that once the user started all the agents, he manages all the preceding computation steps through the GUI provided by the Interface Agent.

Computation set-up -- At startup phase, Interface Agent sends multicast messages on the net, in order to identify other agents that are running and not engaged in other elaboration (Fig. 9, 10). The available agents respond according to their typology, providing some information about themselves. For instance, System Agents will respond with the lists of the Mpeg files available on the machines on which they are running (Fig. 11); Audio and Video Agents with information about the type of computation they are able to perform, etc (Fig. 12).
The Interface Agent shows the information received from the available agents on the net, so that the user can choose the agents needed for video segmentation and initialize their parameters (Fig. 11, 12).

**Computation --** Once the user has done his choices, computation starts. The Interface Agent transmits the initialization parameters to each agent involved in the segmentation process. Furthermore each agent is informed about the other agents that will cooperate with it, while the Interface Agent displays partial and final results from the computation agents (audio, video, information fusion) (Fig. 13).

**Analysis of segmentation results --** When all the computation agents complete their analysis, the Interface Agent shows a panel through which the user can browse and analyze the segmentation results (Fig. 14).
4 Conclusion and future work

In this paper the use of MultiAgent paradigm for realizing a distributed Mpeg Segmentation Environment has been discussed. The characteristics of the paradigm have been matched with the intrinsically parallel nature of the application and the way for defining suitable Agents and their interactions has been presented.

A prototype version of the environment utilizing legacy software developed by some of the authors in previous works has been realized and is under evaluation.

Two main further research activities are in progress:

a. The definition of experiments and tests for achieving optimal results from the performance point of view;

b. The improvement of Agents intelligence in dynamically combining different semantic description levels of the scenes.

As far as concerning the first activity, we are searching for optimal ratio between the number of machines utilized and the number of Agents defined through the analysis of Agents allocation on the quoted machines. We are testing the system in two different conditions: on a farm composed by four Pentium II bi-processors with WinNT 4.0 operating system and on a farm composed by twenty-five Pentium II single processor with a mix of Win 98 and Win NT 4.0 operating system.

As far as concerning the second research line, we are developing Information Fusion Agents making use of Bayesian Networks and of Multiexpert Neural Networks at the different phases of the segmentation process. In particular, Multiexpert neural Networks are under investigation for the Audio Information Fusion while Bayesian Networks are under investigation for the Video and for the final Information Fusion Phases.

5 References


Figure 13. The User Interface for evaluating final segmentation results.