Autonomic Control for Quality Collaborative Video Viewing

Dan Phung
Computer Science Dept.
Columbia University
New York City, New York
phung@cs.columbia.edu

Giuseppe Valetto
Telecom Italia Lab
Turin, Italy
giuseppe.valetto@tilab.com

Gail Kaiser
Computer Science Dept.
Columbia University
New York City, New York
kaiser@cs.columbia.edu

ABSTRACT

We present an autonomic controller for quality collaborative video viewing, which allows groups of geographically dispersed users with different network and computer resources to view a video in synchrony while optimizing the video quality experienced. The autonomic controller is used within a tool for enhancing distance learning with synchronous group review of online multimedia material. The autonomic controller monitors video state at the clients' end, and adapts the quality of the video according to the resources of each client in (soft) real time. Experimental results show that the autonomic controller successfully synchronizes video for small groups of distributed clients and, at the same time, enhances the video quality experienced by users, in conditions of fluctuating bandwidth and variable frame rate.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Client/server, Distributed applications; C.4 [**Performance of Systems**]: Performance attributes; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems; H.5.3 [**Group and Organization Interfaces**]: Computer-supported cooperative work, Synchronous interaction; K.6.4 [**System Management**]: Quality Assurance

General Terms

MEASUREMENT, PERFORMANCE, EXPERIMENTATION

Keywords

Synchronized Collaborative Video, Autonomic Controller, Quality Adaptation

1. INTRODUCTION

In today's distance education programs such as the Columbia Video Network, lectures are frequently recorded, then post-processed and packaged for students to watch (and re-watch) at their convenience over the Internet as streaming videos. It seems that Internet-based video streaming should enable synchronous collaboration "situated" by collaborative lecture viewing, and introduce the possibility of forming "study groups" among off-campus students who view and discuss lecture videos together, thus approximating the pedagogically valuable discussions of on-campus students.

Unfortunately, collaborative video viewing has no built-in support in today's conventional Internet video technology. It is particularly challenging to enforce WISIWYS (What I See Is What You See) among geographically dispersed users, whose bandwidth and computer resources may be very diverse, and may also vary during the course of the video. Technically, collaborative video viewing poses a twofold problem: on the one hand, it is mandatory to keep all users synchronized with respect to the content they are supposed to see at any moment; on the other hand, it is important to provide each individual user with a level of quality that is optimized with respect to available resources.

In order to balance the group synchronization requirement with the optimization of the individual viewing experiences, we use an autonomic controller on top of a collaborative video viewing architecture we have developed for a project named AI^{2}TV, for Adaptive Internet Interactive Team Video. AI^{2}TV contributes to the area of autonomic computing as an interesting application domain, where the time scales make human "systems management" infeasible.

Our approach applies techniques and insights derived from our previous work on autonomic computing [23, 12, 13] to the novel domain of multi-user video synchronization, with its challenging soft real-time requirements. Our autonomic controller remains conceptually separate from the controlled video system, and employs a decentralized workflow engine geared towards the dynamic adaptation of distibuted software systems, named Workflakes [22], to enforce its adaptation scheme onto the video clients. A single controller is used for all clients in the same user group, so it can detect "skew" across multiple clients and coordinate its resolution.

This approach results from the evolution of an earlier version of AI^{2}TV, described in [10], in which group video viewing capabilities were embedded within a Collaborative Virtual Environment [7]. In that early system, each single client adjusted video playback on its own, on the basis of video synchronization packets exchanged in a peer-to-peer fashion and piggybacked on top of the UDP communication used primarily for updates of the CVE shared state. No explict control facility was hence included in that design, which did not perform satisfactorily with respect to synchroniza-

WOSS'04 Newport Beach, CA USA

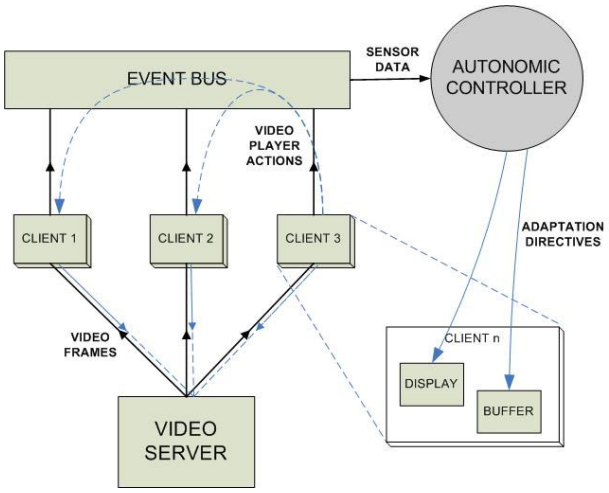

Figure 1: AI^{2}TV Architecture

tion, possibly because of the lack of coordination across the clients and the overhead related to processing many-to-many communication.

Experimental results of trials with the new system show quantitatively that the autonomic controller can ensure group synchronization and has a significant positive impact on the video quality experienced by each client.

2. ARCHITECTURE AND ADAPTATION MODEL

2.1 System Architecture

The AI^{2}TV system, shown in figure 1, involves several components (a video encoding system (not shown), a video distribution server, video clients, and a common communications infrastructure), upon which the autonomic controller operates.

The encoding system encodes videos prior to their distribution, according to a *semantic compression* technology developed by Liu and Kender [16] - also at Columbia - which is similar to cumulative layering [17], also known as scalable coding [14]. Cumulative layering produces a hierarchy of several encodings for the same video, with different quality levels. The semantic compression algorithm used in AI^{2}TV reduces a standard MPEG video to a sequence of JPEG still images, which represent semantically significant key frames within a sliding time window. By increasing the size of the window, a key frame will represent a larger time slice, hence producing less key frames as compared to a smaller window size. That way, it is possible to generate different sets of JPEG images for a range of different compression levels. The algorithm produces an effectively random distribution of key frames, hence the resulting video plays back at a *variable* frame rate. That adds substantial complexity to the bandwidth demands of the clients.

Through the semantic compression algorithm, we can provide *semantically equivalent content* to a group of clients with diverse bandwidth, by adjusting the compression level assigned to each client while watching the video. Thus, in AI^{2}TV, synchronization of video boils down to showing semantically equivalent frames at the same time to all clients (further details on the semantic compression tool used in AI^{2}TV are outside the scope of this short paper).

The video server provides the layered video to clients in a group. Each lecture video is stored according to the compression hierarchy produced via the semantic compression tool, together with indices of the key frames produced, which are annotated with playing time information. Once the layered video is codified, the task of the video server simply is to provide access to the index files and the frames.

The task of each video client is to acquire video frames, display them at the correct time, and provide a set of basic video functions, such as play, pause, goto and stop. Taking a functional design perspective, the client is composed of the following modules: a time controller, a video display, a video buffer that feeds the display, and a manager for fetching frames into the buffer.

The time controller relies on NTP [18] to ensure a common time base, through which each client can reference from the video indices the correct content to display at any moment, provided that a corresponding key frame at some quality level can be download in time.

The video display renders the JPEG frames at the correct time into a window and provides the user interface. The video display knows which frame to display by using the current video time and display quality level to select the representative frame into the frame index. The video display also includes a control hook that enables the autonomic controller to adapt the current display quality level.

The video manager constitutes a downloading daemon that continuously downloads frames at a certain level into the video buffer. It keeps a hash of the current reserve frames in the buffer for each quality level. The buffer manager also includes a control hook that enables the controller to adapt the current downloading quality level.

Video clients communicate with each other over a distributed publish-subscribe event bus. The bus propagates video actions taken by any user to all users in the group. Video actions are time-stamped, thus all clients respond to them in reference to the common time base.

The purpose of the autonomic controller is to enforce the synchronization constraint, and at the same time ensure that each client plays at its highest attainable quality level. The controller is itself a distributed system, whose design derives from a conceptual reference architecture for autonomic computing platforms proposed by Kaiser *et al.* [11], which is shown in figure 2.1. The architecture provides an end-to-end closed control loop, in which sensors attached to a generic (possibly legacy) target system continuously collect and send streams of data to gauges. The gauges analyze the incoming data streams and recognize conditions that require some adaptation, relaying that information to one or more "core" controllers. Those are coordination engines in charge of orchestrating the distributed actions needed to carry out the adaptation. At the end of the loop, actuators attached to the target system effect the needed adjustments under the supervision of the controller.

In the AI^{2}TV case, sensors at each client monitor the following information: the currently displayed frame, its quality level, the quality level currently being fetched by the buffer manager, the time range covered by buffered frames, and the current bandwidth. Gauges are embedded together with the coordination engine for expediency of design and to minimize communication latency in this real-time envi-

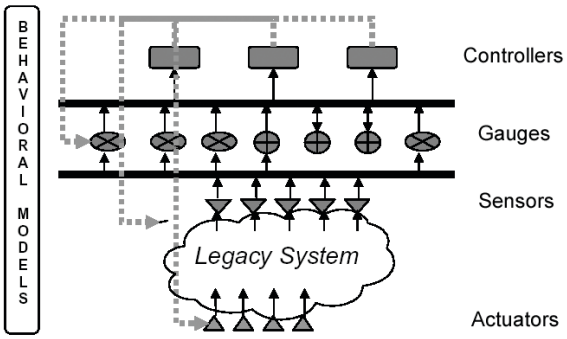

Figure 2: Controller Reference Architecture

ronment. They receive the sensor reports from individual clients, and collect them in buckets, similar to the approach in [8]. A set of helper functions tailored specifically for this application operate on the bucket data structure to compute the synchronization state of the client group and produce triggers for the coordination engine. When a trigger is raised, the coordination engine enacts an adaptation scheme, i.e., a workflow that takes effect upon the client modules through the control hooks they provide.

The adaptation scheme leverages the hierarchy of compression levels produced by the semantic compression tool, to adjust the client behavior regarding the next image to be displayed, and the next image to be fetched from the appropriate semantic compression level. That way, it can react to changes in the environment and the frame rate.

All communications internal to the autonomic controller, i.e, sensors reports and adaptation directives, also occur via the above mentioned publish-subscribe event bus.

2.2 Adaptation Model

The adaptation scheme consists of two levels: a higher level control flow, and a lower level adjustment heuristic. The higher level logic structures the adaptation according to a formal decision process; the diagram in Figure 3 shows the task decomposition hierarchy according to which that adaptation workflow unfolds, in the Little-JIL graphic formalism [5] employed. At the lower level, the adaptation scheme provides criteria as to when and how to adapt clients, in response to either low or high bandwidth situations.

Whenever a client has relatively low bandwidth, the client may not be able download the next frame at the current quality level in time. Then both the client and buffer quality levels are adjusted downwards one level. If the client is already at the lowest level in the hierarchy, the controller will estimate the next frame at that level that can be successfully retrieved before its own start time, while remaining synchronized with the rest of the group. The client will then be directed to jump ahead to that frame.

To take advantage of any available bandwidth surplus, the buffer manager will start to accumulate a reserve buffer. Once a threshold value of reserve frames is buffered, the controller will direct the manager to start fetching frames at a higher quality level. Once sufficient reserve is accumulated also at that higher level, the client is then ordered to start displaying frames at the higher level. If the bandwidth drops before the buffer manager can accumulate enough frames in the higher-level reserve, the buffer manager is dropped back

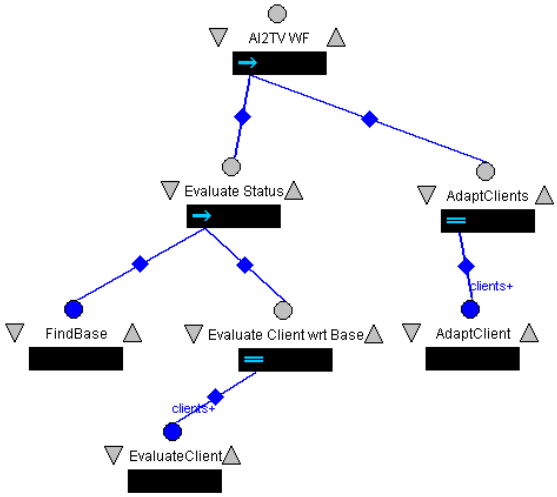

Figure 3: AI^{2}TV Workflow diagram

down one quality level.

A variation of this same adaptation scheme could also enforce a given upper limit to the amount of bandwidth employed by a client for video, in order to ensure that enough bandwidth remains available for other forms of communication, such as audio discussions among users.

2.3 Implementation

The system is fully implemented in Java. The video client uses `javax.swing` to render frames. The coordination core of the controller, Workflakes, is built on top of the open-source Cougaar multi-agent system [1], which it extends to allow the orchestration of distributed software for autonomic purposes (explained further in [23]). We used the Little-JIL graphic workflow specification language [5] to define the adaptation plan. We chose the freely available Siena [4] publish-subscribe event system as our communication bus.

3. EVALUATION

Our assessment considers two factors: group synchronization and optimal video quality delivery. Our results were computed from client configurations consisting of 1, 2, 3, and 5 clients together running a semantically summarized video for 5 minutes, with sensors probing clients state every 5 seconds. The compression hierarchy consists of 5 levels.

We define a baseline client against which the performance of our approach can be compared. The baseline client's quality level is set at the beginning of the video and not changed thereafter, which represents a client in AI^{2}TV without the adaptive elements. The baseline quality level is computing using a value we identify as the average bandwidth for the set level, i.e., the bandwidth needed, on average, for the buffer manager to fetch the next frame on time. We provide the baseline client with the corresponding bandwidth to the video server by using a throttling tool ([20]). Note that using the average bandwidth does not account for the inherent variability in video frame rate and the likely bandwidth fluctuations in real-world conditions, where adaptive control can make a difference.

For each trial, we record any differences resulting from

the controller's adaptation of the clients' behavior *versus* the behavior of the baseline clients.

3.1 Evaluating Synchronization

To measure the level of group synchronization, we probe the video clients at periodic time intervals and log the frame currently being displayed. This procedure effectively takes a series of system snapshots, which we check to see whether the frame being displayed at a certain time corresponds to one of the valid frames for that time, on *any* quality level. We allow any level here because the semantic compression algorithm ensures that all frames designated for a given time will contain semantically equivalent content. We obtain a score by summing the number of clients not showing an acceptable frame and normalizing over the total number of clients. A score of 0 indicates a fully synchronized system.

In a first set of trials, clients were assigned an initial compression level matching on average their available bandwidth: all of those trials showed a total score of 0, notwithstanding variations in the frame rate and/or occasional fluctuations in the actual bandwidth; also, full synchronization was achievd without missing any frames. This result demonstrates that the chosen baseline combinations of compression levels and throttled bandwidths did not push the clients beyond their bandwidth resource capacity.

Then we ran another set of experiments, in which the clients were assigned more casually selected starting bandwidth. Said casual selection is representative of real-world media streaming applications, in which users select stream quality based on their nominal connectivity, but may actually be receiving a significantly lower actual data rate. Our clients were assigned bandwidths one level lower than the preset quality level. We ran this set of experiments first without the aid of the autonomic controller and then with it. In the former, non-adaptive case, clients with insufficient bandwidth were stuck at the compression level originally selected, and - in order to remain synchronized - were forced to miss an average of 63% of the frames. In the latter case, the same clients only missed 35% of the needed frames. Although both situations show a significant amount of missed frames, these results provide evidence of the benefits of the adaptive scheme of the autonomic controller.

3.2 Evaluating Quality of Service

For this evaluation, we had to formulate a scoring system relative to the baseline client's quality level, since other proposals, e.g., [2, 25] are not constrained by the group synchronization requirement. We give a weighted score for each attained quality level above or below the baseline level: the weighted score is calculated as the ratio of the frame rate of the two levels. For example, if a client is able to play at one level higher then the baseline, and the baseline plays at an average `n` frames per second (fps) while the higher level plays at `2*n` fps, the earned score is 2. That way, scores are sensitive to the relative differences between quality levels.

Our experiments show that baseline clients scored a group score of 1 (as expected) while the controller-assisted clients scored a group score of 1.25. The one-tailed t-score of this difference is 3.01, which is significant for an α value of .005 (N=17). This result demonstrates that the autonomic controller enabled our system to achieve a significant positive difference in received frame rate, which translates to better video quality. Since the t-score does not measure the *degree* of benefit achieved, we also measure the proportion of additional frames that each client is able to display. We found that, overall, adapted clients received 20.4% ($\pm$ 9.7, N=17) more frames than clients operating at a baseline rate.

Running the client close to or at a level higher than the baseline puts the client at risk of missing more frames, because the autonomic controller is trying to push the client to a better-quality, but more resource-demanding, level. Therefore, we also count the number of missed frames during a video session. In our experiments, there was only one instance in which a controller-assisted client missed some frames - in particular, two consecutive frames. Upon closer inspection of the log of that trial, we found that the semantically compressed video demanded a higher frame rate at the same time that the network bandwidth available to that client became relatively low. The client was able to consistently maintain a high video quality level after that epoch.

4. RELATED WORK

Intra-stream synchronization, which is concerned with ensuring the temporal ordering of data packets transmitted across a network from a single streaming source to one or more delivery sinks, is a widely studied topic in multimedia research. Intra-stream synchronization schemes typically implement some form of trade-off related to some parameter impacting the quality of service offered to client; those schemes can be rigid or adaptive [6].

Many adaptive schemes trade off synchronization for play-out delay. Examples include the Adaptive Synchronization Protocol [19], the Lancaster Orchestration Service [3], the work of Gonzalez and Adbel-Wahab [9], or that of Liu and El Zarki [15]. They are based on data buffering at the sink(s) and the introduction of some delay before the play-out of buffered data units (i.e., frames), to accommodate slower clients. The delay is recomputed continuously while streaming is under way, to try to minimize it and still ensure synchronization. Those schemes must accept temporary synchronization inconsistencies and/or some data loss, in case the guessed delay is at times insufficient (due, e.g., to varying network conditions) and needs to re-estimated. Other adaptive schemes, like Concord [21], allow to adapt also other quality parameters, e.g, packet loss rate.

Our work differs from the majority of adaptive schemes since it is not based on play-out delay. Instead, we take advantage of layered semantic compression, coupled with buffering, to "buy more time" for clients that coul not otherwise remain in sync, by putting them on a less demanding level of the compression hierarchy. When resources are low, we sacrifice frame rate at the client end to enforce the synchronization requirement.

With respect to the software architecture, AI^{2}TV is a simplification of our KX infrastructure [12, 13], which in turn is a reification of the conceptual model in Figure 2.1. With respect to KX, AI^{2}TV trades off general applicability for faster, ad hoc sensor data processing, and distribution for lower communication latency through the feedback loop: those trade-offs are both a premium in a real-time scenario.

Among video synchronization architectures developed in the multimedia community, the most similar is perhaps the Lancaster Orchestration Service [3], which is also based on a high-level controller that coordinates, via appropriate directives, remote control units placed within the clients. The Lancaster approach employs an adaptive delay-based scheme,

hence it tends to adapt to the lowest bandwidth client and to degrade the playback experience of the other participants. Our approach seems preferable, since each client should receive video quality commensurate with its resources.

Walpole *et al.* also incorporate a software feedback loop within a distributed real-time MPEG player, to optimize frame rate [24]. Adaptation, however, remains local to each client, while the coordination aspect in our work enables also video synchronization across a small group of clients.

5. CONCLUSIONS

We have applied an autonomic approach to the problem of enabling geographically dispersed user groups to collaboratively view videos in synchrony, while dynamically adapting the video quality according to clients' resources. Our system distributes appropriate quality levels of the video to clients, in conditions of fluctuating bandwidth and variable video frame rate. Experimental results demonstrate the advantages of this autonomic approach in terms of video synchronization and optimized video quality.

This work offers proof that techniques developed within application domains typically associated to autonomic computing, such as self-management of data centers and information systems, can be valid also in (soft) real time situations in general, and for multimedia systems specifically.

6. ACKNOWLEDGMENTS

We would like to thank John Kender, Tiecheng Liu, and other members of the High-Level Vision Lab for their assistance in using their semantic compression software. We would also like to thank the other members of the Programming Systems Lab, particularly Suhit Gupta for early work on AI^{2}TV, and Matias Pelenur who implemented PSL's Little-JIL interpreter on top of Workflakes/Cougaar. Little-JIL was developed by Lee Osterweil's LASER lab at the University of Massachusetts, Amherst. Cougaar was developed by a DARPA-funded consortium; our main Cougaar contact was Nathan Combs of BBN. Siena was developed by the University of Colorado, Boulder, in Alex Wolf's SERL lab. PSL is funded in part by National Science Foundation grants CCR-0203876, EIA-0202063 and EIA-0071954, and by Microsoft Research.

7. REFERENCES

[1] Cognitive Agent Architecture (Cougaar) Open Source Project. `http://www.cougaar.org/`.

[2] S. Baqai, M. F. Khan, M. Woo, S. Shinkai, A. A. Khokhar, and A. Ghafoor. Quality-based evaluation of multimedia synchronization protocols for distributed multimedia information systems. *IEEE Journal of Selected Areas in Communications*, 14(7):1388–1403, 1996.

[3] A. Campell, G. Coulson, F. Garcia, and D. Hutchison. A continuous media transport and orchestration service. In *SIGCOMM92: Communications Architectures and Protocols*, 1992.

[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, Aug. 2001.

[5] A. G. Cass, Barbara Staudt Lerner, E. K. McCall, L. J. Osterweil, Stanley M. Sutton, Jr., and A. Wise. Little-JIL/Juliette: A Process Definition Language and Interpreter. In *22nd International Conference on Software Engineering*, pages 754–757, June 2000.

[6] D. D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *ACM SIGCOMM: Communications architectures and protocols*, pages 14–26, 1992.

[7] S. E. Dossick and G. E. Kaiser. CHIME: A Metadata-Based Distributed Software Development Environment. In *Joint 7th European Software Engineering Conference and 7th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, 1999.

[8] L. Gautier and C. Diot. Design and evaluation of mimaze, a multi-player game on the internet. In *International Conference on Multimedia Computing and Systems*, 1998.

[9] A. J. Gonzalez and H. Adbel-Wahab. Lightweight stream synchronization framework for multimedia collaborative applications. In *5th IEEE Symposium on Computers and Communications*, July 2000.

[10] S. Gupta and G. Kaiser. A Virtual Environment for Collaborative Distance Learning With Video Synchronization. In *7th IASTED International Conference on Computers and Advanced Technology in Education*, August 2004.

[11] G. Kaiser, P. Gross, G. Kc, J. Parekh, and G. Valetto. An Approach to Autonomizing Legacy Systems. In *Workshop on Self-Healing, Adaptive and Self-Managed Systems*, June 2002.

[12] G. Kaiser, J. Parekh, P. Gross, and G. Valetto. Kinesthetics eXtreme: An External Infrastructure for Monitoring Distributed Legacy Systems. In *5th Annual International Active Middleware Workshop*, June 2003.

[13] G. Kaiser, J. Parekh, P. Gross, and G. Valetto. Retrofitting Autonomic Capabilities onto Legacy Systems. Technical Report CUCS-026-03, Columbia University Department of Computer Science, October 2003.

[14] W. Li. Overview of the fine granularity scalability in mpeg-4 video standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):301-317, 2001.

[15] H. Liu and M. E. Zarki. A synchronization control scheme for real-time streaming multimedia applications. In *Packet Video 2003*, April 2003.

[16] T. Liu and J. R. Kender. Time-constrained dynamic semantic compression for video indexing and interactive searching. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 531-538, 2001.

[17] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *ACM SIGCOMM*, 26(4):117-130, 1996.

[18] D. L. Mills. Network time protocol. RFC 958, 1985.

[19] K. Rothermel and T. Helbig. An Adaptive Protocol for Synchronizing Media Streams. *Multimedia Systems*, 5:324–336, 1997.

[20] L. Santi. A user-mode traffic shaper for tcp-ip networks. `http://freshmeat.net/projects/shaperd/`.

[21] N. Shivakumar, C. J. Sreenan, B. Narendran, and P. Agrawal. The concord algorithm for synchronization of networked multimedia streams. In *International Conference on Multimedia Computing and Systems*, 1995.

[22] G. Valetto. *Orchestrating the Dynamic Adaptation of Distributed Software with Process Technology*. PhD thesis, Columbia University, April 2004.

[23] G. Valetto and G. Kaiser. Using Process Technology to Control and Coordinate Software Adaptation. In *25th International Conference on Software Engineering*, May 2003.

[24] J. Walpole, R. Koster, S. Cen, C. Cowan, D. Maier, D. McNamee, C. Pu, D. Steere, and L. Yu. A Player for Adaptive MPEG Video Streaming Over The Internet. In *26th Applied Imagery Pattern Recognition Workshop*. SPIE, October 1997.

[25] Y. Wang, J. Ostermann, and Y.-Q. Zhang. *Video Processing and Communications*. Prentice Hall, 2002.