CaML: Camera Markup Language for Network Interaction

Maxwell Sayles, Xiaojing Wu, and Jeffrey E. Boyd

Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4

ABSTRACT

As processor speeds increase and the cost of digital video technology falls, the use of video is expanding in a plethora of applications including video surveillance, human computer interaction, tele-instruction, and enhanced sports broadcasts. However, a major problem that now faces developers of video systems is the requirement to build the low-level video processing from the ground up for each application. This paper describes a camera system that acts not merely as a *provider of pixels*, but as a *video information server*. A video application interacts with the camera server using the Camera Markup Language (CaML, pronounced camel) proposed here. CaML is an XML-based (Extensible Markup Language) data format for exchanging video information with a server. It provides a layer of abstraction between the application and the pixels to simplify the development process and is well-suited to exchanging data over a network. Using a camera as a server on a network makes it a simple matter for a single application to use multiple cameras. Local- and wide-area networks (LANs and WANs) replace the need for conventional methods for routing video signals.

Keywords: XML, camera, surveillance, human-computer interaction, embedded cameras

1. INTRODUCTION

As processor speeds increase and the cost of digital video technology falls, the use of video is expanding in a plethora of applications. These applications include *video surveillance*[1,2] for security applications, *group awareness*[3,4] systems in which video systems facilitate group interactions over a network, *tele-instruction*[5,6] where video and sound are the essential media by which a lecturer communicates with students over a distance, and *enhanced broadcasts*[7] that use information extracted from a video stream to add information to, and thereby enhance, the broadcast content of a public events (most often sports).

A major problem that now faces developers of video systems is the requirement to build the low-level video processing from the ground up for each application. For example, these video systems usually perform the following basic tasks:

- **segmentation** of a video stream into foreground (moving objects) and background (static) pixels,
- **connecting pixels** that are in the foreground into objects,
- identification of salient **features** of the objects,
- **recognizing objects or activities** based on the extracted features, and
- some applications require **camera calibration**.

Furthermore, some applications require that a user of the system be able to control the camera. This includes controlling the pan, tilt, and zoom (PZT) of a camera, and image quality and format parameters. A developer must decide how to accomplish each of these tasks for every application on a case-by-case basis.

We see examples of algorithms that perform these tasks in a variety of video processing systems. Sudderth et al.[8] describe a method for segmenting moving object pixels from a video sequence. Horprasert et al.[9] describe a similar segmentation method that accounts for changes in illumination due to shadows. Grouping of pixels into connected components (see Horn[10] for example) and extracting descriptive features is a common paradigm in computer vision and pattern recognition. Wren et al. describe a system that tracks connected blobs that

Send correspondence to: boyd@cpsc.ucalgary.ca. CaML web page at http://www.cpsc.ucalgary.ca/~boyd/caml are part of humans. Haritaoglu et al.[11] identify and label parts of human body by analyzing the silhouettes corresponding to segmented objects. Hartley and Zisserman,[12] and Faugeras[13] describe methods for camera calibration and triangulation of object positions in three-dimensional space from calibration and image data.

Two conclusions emerge from a survey of video systems.

1. The systems share a common set of similar algorithms for performing the required tasks.
2. Video system development would be easier and faster if these basic tasks could be removed from the system design process so that each developer would not have to implement them.

This paper describes a camera system that acts not merely as a *provider of pixels*, but as a *video information server*. The system integrates the functions of a camera and the low-level tasks required by a video system, and then serves the resulting data to client applications. Thus a video application need not deal with low-level pixel data and algorithms, but can be built starting from the level of image object information, retrieving its input from the information server. An application interacts with the camera server using the Camera Markup Language (CaML, pronounced camel) proposed here. CaML is an XML-based (Extensible Markup Language) data format for exchanging video information with a server. We found inspiration in the work of Makatchev and Tso[14] who describe RoboML, an XML-base language for interaction with robots. Although CaML refers to the language of interaction, casually we also use the term to refer to the server.

CaML provides a layer of abstraction between the application and the pixels to simplify the process of developing video applications. It is well-suited to exchanging data over a network because of its XML roots. Using a camera as a server on a network makes it a simple matter for a single application to use multiple cameras. Local- and wide-area networks (LANs and WANs) replace the need for conventional methods for routing video signals.

In similar work Wolf et al. treat a camera as an embedded device.[5, 6] Their systems and CaML embed basic video processing algorithms into a camera. CaML, however, provides a format for the exchange of data and interaction with the camera, as well as exploiting existing networks for video applications.

2. SURVEILLANCE APPLICATIONS

Although there are many video applications that have common requirements for low-level processing, we focus on surveillance to motivate development and testing of CaML. Figure 1 shows a hypothetical surveillance system extrapolated from those described by Boyd et al.[1] for use with CaML. Two or more CaML camera servers observe the activity in a surveillance area. A CaML client requests and retrieves data from the servers over a network. The client assimilates the information from the servers into a model of the activity in the area. A user of the system observes the activity through interactions with the client.

Figure 2 describes steps in the flow of data through the system from image pixels to a scene model. In our CaML server we integrate the first five steps into the functionality of the camera. The client is responsible for the remaining steps, including triangulation, tracking, and user interface. This division of labor between client and server locates all the processing that is specific to a single camera within the camera. Therefore, it is a simple matter to add more CaML server/cameras to a system since all the camera-related processing is embedded.

The client need never process a pixel. It only requires the high-level object descriptions and the camera calibration data. From this information it can construct a model of area activity. The client displays the model to a user in the form of a map showing recent tracks of moving objects.

It is also possible for data to move from the user and client to the camera. For example, if a user is not satisfied that a camera has the best view of a scene, the client can request that the camera alter its pan, tilt, and zoom to get better images.

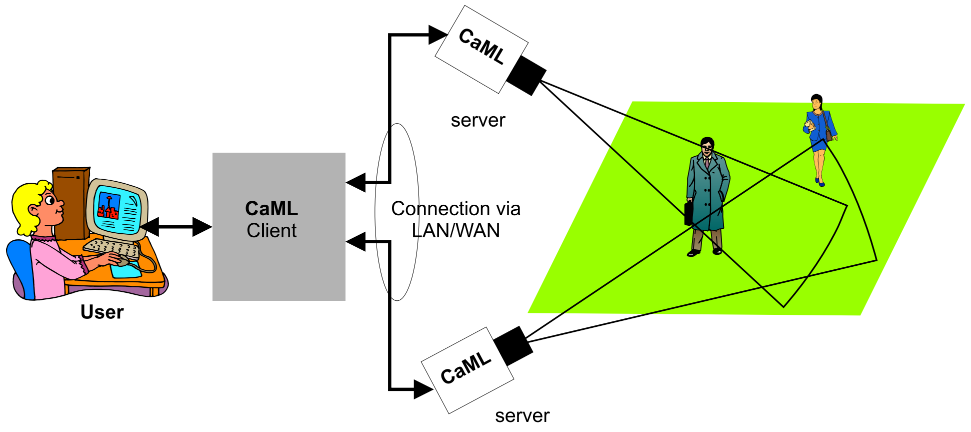

Figure 1. A typical video surveillance system using a CaML client and servers.

3. CAML DESIGN

A CaML server and client interact by exchanging CaML XML-style documents. Figure 3 shows a sample CaML document. In a typical interaction, a client application sends a document to the server to request data. The server responds by sending a document with the data to the client.

For simplicity, CaML uses the same format for both requests and responses. To initiate interaction, the client sends any valid CaML document to the server. The server responds with a document like that shown in Figure 3, containing the full set of data and URLs that the server can provide. When the interaction involves moving data from client to server, i.e., when the client wants to set camera parameters such as PTZ, image quality, or format, the client makes a request by sending a document with the desired parameters. In response, the server sets the parameters to the specified values, if it can, and returns a document containing full data set. The server may not actually set parameters for a variety of reasons,including:

- it may not be physically possible to alter the parameters, e.g., when the camera is static,
- the parameters may be out of range, or
- external control is not allowed, e.g., when a local operator is controlling the camera, or where external control would cause conflicts among multiple clients.

Server	Acquire images of the scene.	**pixels**
	Acquire a camera calibration matrix.	$\Downarrow$
	Segment the images into foreground and background regions.	$\Downarrow$
	Group pixels to form connected component objects.	$\Downarrow$
	Compute statistical descriptions of the objects.	$\Downarrow$
Client	Use camera calibration and object statistics to triangulate object positions in three dimensions.	$\Downarrow$
	Track the position and speed of objects as they move to build a model of activity in the area.	$\Downarrow$
	Allow a user to view and query the model.	**model**

Figure 2. Flow of data through a video surveillance system.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CaML version="1.0" id="morse" received_at="2557.036" generated_at="2557.037">

        <!-- video server information -->
        <video>
                <URL>136.159.10.66:47772</URL>
                <width>320</width>
                <height>240</height>
                <format>JPEG</format>
                <quality>10</quality>
        </video>

        <!-- object description stream -->
        <objects>
                <URL>136.159.10.66:47773</URL>
        </objects>

        <!-- camera calibration matrix -->
        <calibration_matrix>
                <rows>3</rows>
                <columns>4</columns>
                <elements>
                        12.0111142336 -21.5643221086 4.4931787861 0.2897473789
                        -0.8053618985 -1.6692540724 22.8213908659 -53.2487828207
                        -0.0358006705 -0.0754323871 0.0275177356 0.0124607047
                </elements>
        </calibration_matrix>

        <!-- show range and value of camera pan-tilt-zoom -->
        <camera_control>
                <min_pan>-880</min_pan>
                <max_pan>880</max_pan>
                <pan>0</pan>
                <min_tilt>-300</min_tilt>
                <max_tilt>300</max_tilt>
                <tilt>0</tilt>
                <min_zoom>0</min_zoom>
                <max_zoom>1023</max_zoom>
                <zoom>0</zoom>
        </camera_control>
</CaML>
```

Figure 3. A sample CaML document.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CaML version="1.0">

        <!-- video server information -->
        <video>
                <width>320</width>
                <height>240</height>
                <format>JPEG</format>
                <quality>10</quality>
        </video>


        <!-- show range and value of camera pan-tilt-zoom -->
        <camera_control>
                <pan>10</pan>
                <tilt>-30</tilt>
                <zoom>60</zoom>
        </camera_control>
</CaML>
```

Figure 4. A sample CaML request to set pan, tile, zoom, image quality and format.

Figure 4 shows a sample request for setting PTZ, image quality, and format.

In the case of streaming data from the server, CaML specifies a URL for the data stream to avoid exchanging documents for every frame of video and reduce communication overhead. Although the statistical descriptions of objects require few bits, it is convenient to provide that data through a separate stream too. The CaML document in Figure 3 shows a URL for video, single image, and object data streams.

Note that CaML specifies a format for data exchange. It does not care where the object descriptions come from. It is left to the developer of a CaML server to select the best algorithms for the low-level processing tasks. Thus there is room for competing or even complementary server implementations.

4. SERVER IMPLEMENTATION

To implement our CaML server prototype we use the following algorithms. We compute optical flow from the video sequence using the method of Lucas and Kanade.[15] A threshold applied to the magnitude of the flow distinguishes between moving and stationary pixels. The connected component algorithm described by Horn[10] groups the moving pixels into objects.

This process results in many spurious objects due to noise and variations in illumination from fluorescent lights. To eliminate spurious objects we do two things. First, we discard any object below a minimum size threshold. Second, we use a tracker built on Kalman filters[16] and nearest-neighbor data association[17] to track objects in the two-dimensional image space. The server only reports objects whose tracks persist for ten frames or more.

So far we have limited our statistical description of the objects to the first moments of the pixel coordinates, i.e., the object centroids. There is no reason CaML cannot support more elaborate descriptions, but centroids were sufficient for our example surveillance application. Figure 5 shows an example CaML object-statistics document with object centroids reported. The client receives these documents in a stream from a URL specified in a CaML response document.

We transfer image data in JPEG format. This allows us to adjust the image quality and the bit rate over a wide range. However, barring limitations in network bandwidth, it would be possible to implement the server to support any image or video stream format.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CaML version="1.0" generated_at="2445.629">
        <objects count="2">
                <object id="3067">
                        <center>
                                <u>29</u>
                                <v>65</v>
                        </center>
                </object>
                <object id="3068">
                        <center>
                                <u>41</u>
                                <v>17</v>
                        </center>
                </object>
        </objects>
</CaML>
```

Figure 5. CaML object centroid data.

Our CaML server prototypes are implemented in the Python programming language for portability and ease of development. To obtain the speed necessary to do video processing from Python we use the Intel Integrated Performance Primitives Library with wrapper functions to allow use from Python. The prototypes run on 850MHz Pentium III machines running Linux. Currently we have two CaML servers operating continuously in our laboratory without failure.

5. CLIENT IMPLEMENTATION

We created a CaML client application to track and display the positions of people as they move throughout our laboratory. To accomplish this, the client does the following.

- Acquires camera calibration data from the servers.
- Acquires streams of object centroid data from the servers.
- Uses a Kalman filter[16] with nearest neighbor data association[17] to track the objects in three dimensions.
- Plots object positions on a map for visualization in real time.
- Logs the data for future incorporation into a database.

Figure 6 shows a screen snapshot from the client display. It shows a schematic map of our laboratory in the upper left corner, and video streams from two cameras along the bottom. The video streams are not necessary for the client to operate, but we include them for debugging purposes. In normal operation, the client would not display the video stream, nor even acquired it.

The display shows two people in the laboratory. The position of one, near the center of the room, is correct. The second person is shown as being against one of the walls and is in error by approximately $0.5m$. This occurs because only one camera sees this location in the laboratory, and the client compensates by assuming that the center of the person is $1m$ off the ground. Consequently, an error in location occurs if this assumption is not correct.

As with the server, we implemented the client application in Python. We currently use a 1.4 GHz Pentium IV, but the client does very little computation compared to the servers. Almost any moderately equipped workstation can do the tracking and display. We have run the client for extended periods of time without failure. Currently, the only limitation is the disk space required for logging the track data.

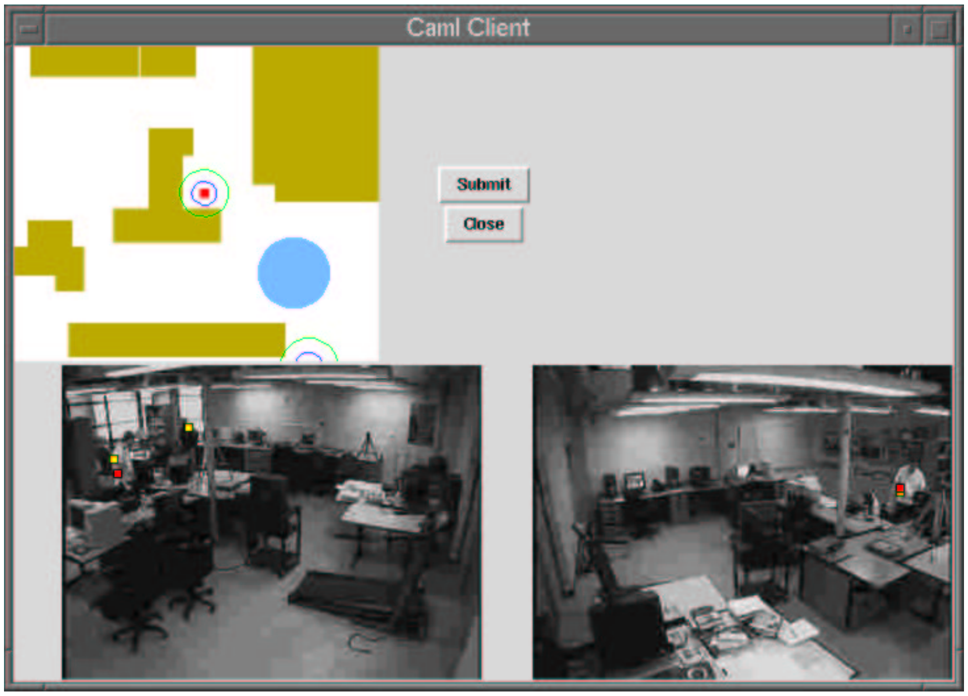

Figure 6. Screen shot of CaML client in operation.

6. FUTURE DIRECTIONS

CaML is a language for exchange of data with cameras. It does not specify what algorithms should be used to extract high-level information from the camera's video stream. However, it is unlikely that any single algorithm will be the best performer under all circumstances. Thus it is desirable for CaML to allow a server to switch algorithms as needed. A default algorithm would operate under most circumstances, but when a client decides that an alternate algorithm may perform better, the client can request that the server switch to a specified algorithm.

Currently, CaML documents are tailored to fit our sample surveillance application. Other applications are likely to use other types of segmentation and object descriptions. One option for supporting more applications is to expand the set of data that a CaML server can provide. A client then selects from a menu of available data. This approach adds complexity for the application writer who will need to be knowledgeable on the plethora of data that can be extracted from a video stream. A *neater* solution is to provide suites of data that are tailored to specific applications. For example, there could be a tracking suite, a gesture recognition suite, and a face recognition suite. A client then selects the data suite that best suits the application.

A CaML client is significantly easier to write than an entire video processing application. However, it is still not trivial and requires knowledge that is specific to video applications, e.g., camera calibration, Kalman filtering, and multiple target tracking. To address this problem we suggest building a set of intermediate clients. For example, Figure 7 shows a system in which intermediate tracking, gesture recognition and face recognition clients act as an interface between the client and the CaML servers. This approach provides further abstraction so that the client programmer needs less specialized knowledge about video applications.

7. CONCLUSIONS

We introduce CaML, the Camera Markup Language, an XML-based language for the exchange of information with cameras. CaML provides a layer of abstraction between the builder of video applications and the low-level processing of pixel data. We demonstrate CaML in a sample application for video surveillance. Future

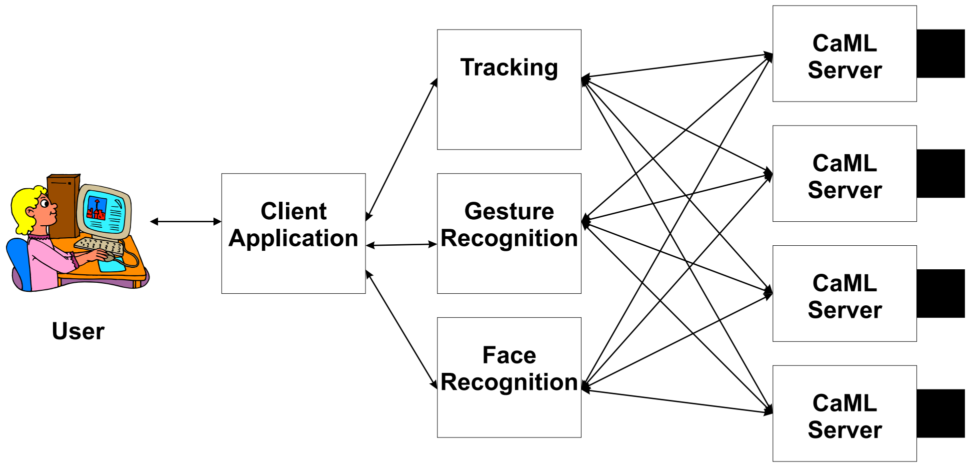

Figure 7. A future CaML system using intermediate clients to further abstract video application creation.

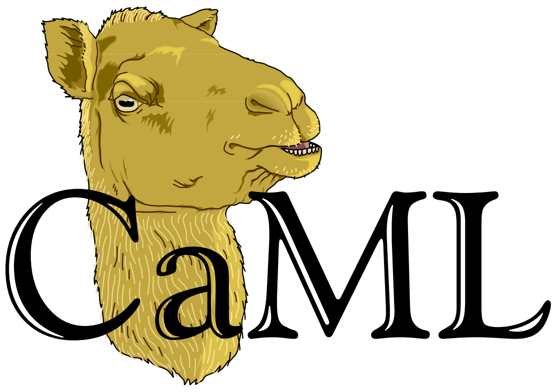

Figure 8. The CaML project logo.

developments will include data suites that are tailored to specific classes of applications, and additional layers of abstraction to further simplify the development of video applications. Figure 8 shows the CaML project logo.

REFERENCES

1. J. E. Boyd, E. Hunter, P. Kelly, L. Tai, C. Phillips, and R. Jain, "Mpi-video infrastructure for dynamic environments," in *IEEE Conference on Multimedia Systems 98*, (Austin, TX), June 1998.
2. I. Haritaoglu, D. Harwood, and L. S. Davis, "W4s: a real time system for detection and tracking people in 2.5d," in *Fifth European Conference on Computer Vision*, (Freiburg), June 1998.
3. M. Boyle, C. Edwards, and S. Greenberg, "The effects of filtered video on awareness and privacy," in *CSCW'00 Conference on Computer Supported Cooperative Work*, *CHI Letters* **2**(3), pp. 1–10, ACM Press, 2000.
4. S. Whittaker, "Rethinking video as a technology for interpersonal communications: theory and design implications," *International Journal of Human-Computer Studies* **42**(5), pp. 501–530, 1995.
5. W. Wolf, B. Ozer, and T. Lv, "Smart cameras as embedded systems," *Computer* **35**, pp. 48–53, September 2002.
6. T. Lv, B. Ozer, and W. Wolf, "Smart camera system design," in *International Packet Video Workshop 2002*, (Pittsburgh, PA), April 2002.

7. A. Gueziec, “Tracking pitches for broadcast television,” *Computer* **35**, pp. 38–43, March 2002.
8. E. Sudderth, E. Hunter, K. Kreutz-Delgado, P. H. Kelly, and R. Jain, “Adaptive video segmentation: theory and real-time implementation,” in *1998 Image Understanding Workshop*, **1**, pp. 177–181, November 1998.
9. T. Horprasert, D. Harwood, and L. S. Davis, “A statistical approach for real-time robust background subtraction and shadow detection,” in *IEEE Frame-Rate Workshop (at ICCV 99)*, (Kerkyra, Greece), September 1999.
10. B. K. P. Horn, *Robot Vision*, MIT Press, Cambridge, Massachusetts, 1986.
11. I. Haritaoglu, D. Harwood, and L. S. Davis, “Ghost: a human body part labelling system using silhouettes,” in *DARPA Image Understanding Workshop*, pp. 229–235, (Monterey, California), November 1998.
12. R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*, Cambridge University Press, Cambridge, UK, 2000.
13. O. G. Faugeras, *Three-dimensional computer vision: a geometric viewpoint*, MIT Press, Cambridge, MA, 1993.
14. M. Makatchev and S. K. Tso, “Human-robot interface using agents communicating in an xml-based markup language,” in *2000 IEEE International Workshop on Robot and Human Interactive Communication*, pp. 270–275, (Osaka, Japan), September 2000.
15. B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” in *International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.
16. A. Gelb, ed., *Applied Optimal Estimation*, MIT Press, Cambridge, Massachusetts, 1974.
17. Y. Bar-Shalom and T. E. Fortmann, *Tracking and Data Association*, Academic Press, Boston, Massachusetts, 1988.