A Platform Independent Image and Video Engine

David Doermann[1], Arvind Karunanidhi[1], Niketu Parekh[1], Ville Rautio[2]

[1]Laboratory for Language and Media Processing, University of Maryland, College Park, USA
doermann@umiacs.umd.edu, arvind@umiacs.umd.edu, nkparekh@umiacs.umd.edu
http://www.umiacs.umd.edu/lamp

[2]Department of Electrical and Information Engineering, University of Oulu, Finland
wilse@ees2.oulu.fi
http://www.mediateam.oulu.fi

ABSTRACT

This paper describes progress on the development of a platform independent media engine for mobile devices. The effort is part of the CAPNET (Context Aware Pervasive NETworks) project and focuses on providing a multimedia presentation and analysis service as part of a larger mobile services architecture. We present a high level overview of design, provide a general description of the functionality of various components such as MediaAlerts, MediaCapture, MediaStorage and MediaProcessing and describe several applications in progress, which utilize media engine components.

1 Introduction

Current mobile applications make limited use of video as an information rich content that can be analyzed, manipulated, indexed and retrieved. Typically, content is in the form of individual images or video clips, possibly containing metadata created by hand. Often they are used as multimedia "messages", without any further consideration of how the content can be used or reused for other purposes. On desktop computers, users are becoming more accustomed to being able to edit and integrate image and video content into standard communications via email and WWW services. As the proliferation of camera and multimedia devices gain momentum, users will demand merging of these same capabilities in their mobile environments. We will expect to be able to easily add text, construct albums, exchange media clips and retrieval media from various remote repositories. Although somewhat cumbersome in their design, many of these applications are now beginning to appear.

At the same time, the research community is focusing extensively on the automated analysis of images and video from a wide variety of electronic sources including newscasts, surveillance, home video, web sources, and digital camera stills, as well as digitized or scanned archival material, newspapers and magazines. As these technologies mature, there will be a demand for integrating them into everyday use, and ultimately into mobile applications. Tasks such as automatic summarization and abstraction of sports video have been reported as a convenient way to deliver content efficiently to bandwidth- and display-limited mobile devices [3]. Similarly, document image analysis has been used to extensively analyze and reflow images of documents for display on devices with limited screen real-estate [9]. Providing such capabilities will ensure a path to integrating many types of multimedia not traditionally suited for mobile devices, in a way that users will ultimately demand.

Even more exciting is the possibility of integrating analysis of content captured with these devices. When face recognition technology matures, we could imagine taking a picture with our devices, passing it to a server, and asking questions such as "Who is that person? I have met them somewhere before". If our database is constrained to known acquaintances, for example, or to the people one would expect to see at a given University, the problem becomes feasible. New usage paradigms are emerging for mobile devices configured with cameras behaving as input tools for fax transmission, bar code recognition and optical character recognition. For example, various groups have successfully demonstrated the ability to integrate image capture, OCR and translation of Chinese signs on a PDA [2].

Over the past two years, the University of Maryland, in cooperation with the CAPNET (Context-Aware Pervasive Networks) Project at the University of Oulu, Finland, has been designing and implementing a set of components, which facilitate the rapid development of mobile multimedia analysis and management capabilities. The goal is to be able to provide an

integrated method to expand and rapidly prototype various pervasive applications. In MUM2002, we presented an overview and motivation for multimedia processing [7]. Recently, we have focused on implementing a number of key components including capture, storage/retrieval, media messaging and alerts and remote image and video processing. In this paper we will highlight the current architecture, and described the components fundamental to the Media Engine. The potential for customized multimedia information sources to be both "produced by" and "delivered to" users in a pervasive environment opens up numerous new and exciting applications. Our goal is to develop a framework and associated toolkit to facilitate seamless media processing.

2 Background

The CAPNET program creates a foundation for new information and communication technologies and new business models for mobile services. The research makes use of key ideas of ubiquitous computing and focuses on service profiling, software technology, and content rich environments. The program provides an innovative architecture that allows us to experiment with adaptive applications, and to prototype and test business scenarios.

2.1 CAPNET Architecture

The CAPNET architecture is based on the need for the user to interact with a mobile device, while making use of distributed network centric services.

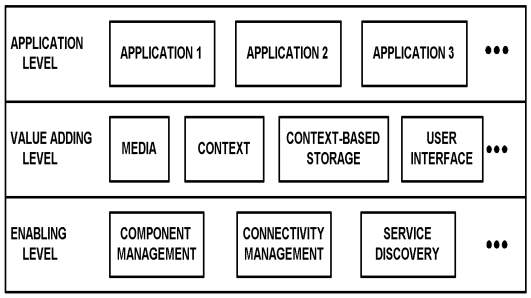

Figure 1: CAPNET Architecture Components

To facilitate operations and interoperability over a wide variety of configurations, the architecture must be adaptable. To accomplish this, it divides services into engines that implement a set of components that are necessary to support functionalities in each domain (Figure 1). These engines currently include *component management, connectivity, service discovery, context, storage, user-interface* and of course *media*, with the first three being required engines. The media engine framework is heavily dependent on the required engines. The component management engine handles direct configuration of the components for other engines while the connectivity engine handles messaging between components. Service discovery components are used for locating resources and services in the CAPNET space based on context information, service meta-data, user profiles, etc.

Each engine is dynamically reconfigurable and divided into a set of static components and dynamic components. The static components implement the basic functionality of the engine, with one of the static components being considered a core component and implementing essential capabilities. The dynamic components are downloadable and depend on the network, device configuration, and user preferences. Overall the components act independently, and rely on messages to communicate. More information about the general CAPNET program can be found at [8].

2.2 Mobile Media

As the next generation of networks make good on promises to provide seamless realization of mobile multimedia, service providers and application developers will look for new ways of providing multimedia content. In particular, frameworks such as Capnet will seek to abstract many of the lower level problems of delivery, device management, manipulation, and compression away for the application. As we have seen in PC environments, media support is being integrated at the operating system level; abstracting many of the challenges and making issues like device management transparent to application developers. Our media engine will attempt to abstract many of those same services for mobile multimedia.

2.3 Related Work

Although many groups have described approaches to the development of multimedia applications, it is essential that our work be integrated with the more general CAPNET component architecture and make use of its services. The approach allows us to build components that will be useful and extensible, but not necessarily interact smoothly with other standards.

Bates et al [5] describe a framework for the construction of mobile multimedia applications, which supports dynamic creation of media objects and movement of media objects from one terminal to another. Our media engine framework also supports storage and retrieval components and processing of media, which seems to be absent in their current framework. Media objects are quite similar in both the approaches except that the metadata is tightly integrated with the media. Low-level communication facilities are employed in their framework while we use xml-rpc communication standards and serialization to transfer media objects.

Another conceptual framework called the Situated Computing Framework [4] has been designed for mobile devices to access rich multimedia services and it provides a smart delivery service of multimedia content which calculates the optimal delivery sequence based on frequency of service request,

priority, type of media and type of output device. The remote invocation process running on the mobile devices is realized using Distributed Component Object model which makes it platform dependent. Our framework tries to solve this problem by using Java and XML - remote procedure calls.

A programming framework [6] has also been designed for quality aware mobile multimedia applications that can describe a high level application with quality requirements and controls of adaptation. The ubiquitous multimedia component model employed by this programming framework has a slight edge over our architecture since it uses QoS (Quality of Service) descriptors which will prove invaluable in case of applications like real-time video streaming.

3 Media Engine

The MediaEngine seeks to provide a seamless integration of multimedia services for end user applications, while maintaining its platform independent architecture. To do this, the system must be dynamically reconfigurable, so, for example, services that cannot run locally because of limitations of the device, constraints of the network or demand of the user, can run on the network or on a server. The MediaEngine, therefore, must provide basic capabilities, along with the logic necessary to dynamically reconfigure.

Like other CAPNET engines, the Media Engine is divided into a set of static components, containing the media core, and dynamic components, which can be adapted at runtime to provide the additional functionality. The Media Core is always present when there is any media engine related functionality present. Dynamic components are loaded on-demand, based on needs expressed by the client application.

All of the components are managed within the Capnet architecture, and hence it is not necessary to implement additional capabilities such as storage, or client server communications. The server side components are responsible for managing services that are used by individual clients, as well as shared services, that have performance requirements that cannot be satisfied by the client. The client components interact with the server by requesting services through a message passing interface.

Unlike other work, the media engine is being designed to provide not only multimedia download and display but also capture and delivery, storage, retrieval, local and remote processing. To realize these services, we have split the media engine into six components. The core components consist of MediaObject and Media Metadata while the dynamic components are comprised of MediaCapture/Playback, Media Storage/Retrieval, Media Processing and Media Alerts/Messaging. Figure 2 shows the overview of media engine architecture. In the remainder of this section, we provide an overview of the functionality of each component, and how they are used by end-user applications.

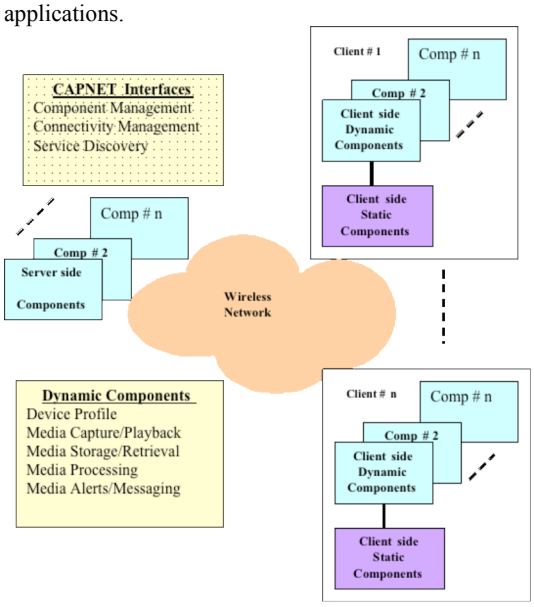

Figure 2: MediaEngine Overview

3.1 Core Media Component

The core component is the static part of the media domain area runtime structure. The core component defines the media objects and functions to create and manage them, including start, stop, reinitialize, etc. All dynamic components handle media as objects that are defined in the core of the domain area. Media core is always composed of the MediaObject component and a related MediaMetadata component.

3.1.1 MediaObject

A MediaObject defines the structure of media content and contains data along with the media metadata component. In case of composite media, the media object contains the data of all media elements, SMIL 2.0 (Synchronized Multimedia Integration Language) based presentation of the spatial, temporal and navigational relations of different media elements and metadata of each media element. Spatial relations define how media elements are spatially related to each other (Element A is on the left of element B). Temporal relations define how media elements are temporally related after each other (Element A is played after Element B). Navigational relations define how media elements are linked to each other (Element A could contain a hyperlink, which would take media playback into state where Element B is being played). Elements in media objects can be text, audio, image or video. Applications are needed to pass data between media components and they handle media objects in serialized form and can be deserialized at the receiver end. They do not have to understand any of the

internals of the media objects making it device independent.

3.1.2 MediaMetadata

The MediaMetadata component defines the metadata that is created for each media element and maintains information about the properties of the media object. Having the metadata of the media object is necessary for components that will manipulate the object, such as storage and retrieval components. A graphical description of the constituent parts of the metadata component is shown in Figure 3.

The MediaObject's metadata is split into four major elements: MediaObjectType, MediaObjectProperties MediaObjectCreation and ChangeHistory. Each element is further divided into sub-elements and contains more information about the media object. The MediaObjectType categorizes the media content while the MediaObjectProperties describe the characteristics of the media content. Media ObjectProperties include the color, orientation, duration, encoding rate, bits per pixel, key frame interval, type of codecs etc, necessary to render or process it. The MediaObjectCreation sub-elements include the device source, camera type, resolution, media source and date/time of creation. The

3.2 Dynamic Media Components

Dynamic Components are components which can be adapted at run-time to provide additional optional functionality. Dynamic components are loaded on-demand based on the needs of the application using the mechanisms provided by the service discovery and component management. Some of the dynamic components are constrained to run only on the server-side.

In the current implementation, we provide basic capabilities necessary to demonstrate the system. The DeviceProfile provides a platform and hardware specialization mechanism to allow easy reconfiguration on a new device. MediaCapture and Playback provide basic support cameras and rendering respectively. MediaStorage and Retrieval provide a way for end user applications to transparently access multimedia repositories. Finally, MediaAlerts and MediaMessages provide a way for pervasive applications to communicate with users and for users to share multimedia content either downloaded to or captured by the device.

3.2.1 DeviceProfile

As a result of technological advances and new

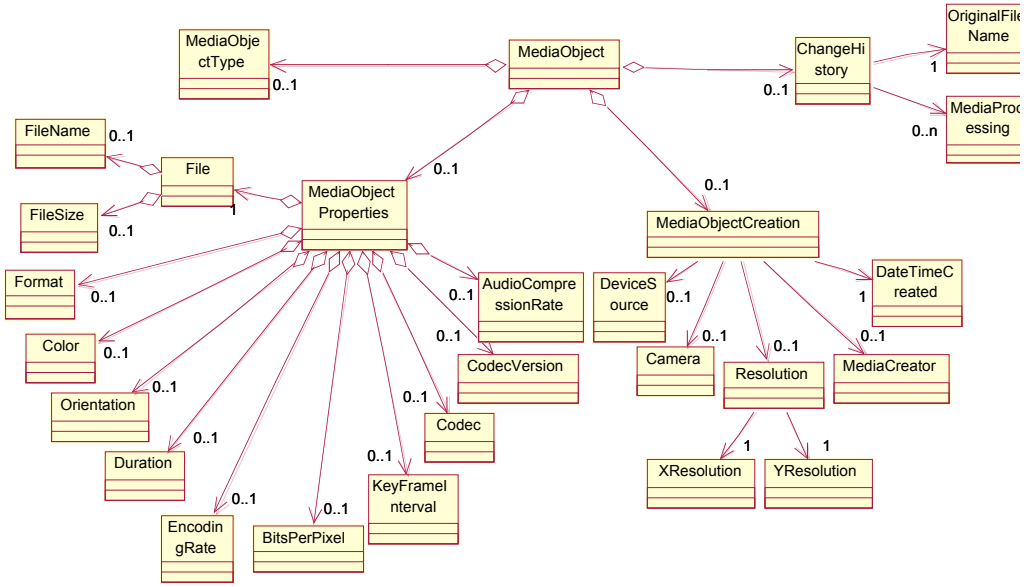

Figure 3: Media Object and Metadata Schema

ChangeHistory provides a container for information about the changes made on the media object (such as being processed remotely with some set of image processing algorithms, and has the actual file name along with the media processing type as its sub-elements.

manufacturers, the mobile device market has diversified, which makes the mobile devices more difficult to manage from the application service provider's perspective. In order to solve the problem, device information should be readily available so that the application knows the capabilities of the device and can tailor the content based on that information. For example, the application can automatically scale

the image based on the device display size and resolution. Hence, a device profile component is required which maintains a centralized repository of device specific information for various mobile devices. The device profile is a CC/PP (Composite Capabilities / Preference Profile) XML file, which contains the display resolution, display depth, supported media formats and characteristics of the mobile access network. This component provides APIs to parse the device profile information from the XML file using MinML and be used by the server to configure dynamic components for download.

3.2.2 MediaCapture and Playback

The media engine offers a media capture and playback component for applications to capture and play media content. The device must have the necessary native hardware and drivers to access it. In case of unsupported or missing capture devices, initialization of this component will fail and exit gracefully. The capture component queries the device profile component before initializing to see whether the device has the capability. The selection of an appropriate driver is completely transparent for the components and applications using it. Media Capture has a video preview component with which the application can utilize it before the image/video capture. Video capture requires the appropriate codecs to be downloaded dynamically to the device. For video capture on the PDA, we are currently using Microsoft Portrait. Portrait provides us with a compressed video stream broken up into frames which we can store locally, upload to a server, or upload to the server and transcode for delivery to other devices. The Media capture supports the Flycam Compact Flash camera, Pretec camera and the HP Pocket camera.

Video playback currently uses a native rendering component for standard encoded media such as MPEG or AVI, and we are developing a rendering component for either custom encodings or data we can easily decode such as MSPortrait.

3.2.3 MediaStorage and Retrieval

The Media Storage component offers media storage and retrieval capabilities for other components and applications. This function provides an extra layer of context-based storage, offering media specific functionality in an easy to use and transparent way.

The MediaStorage component indexes media objects using their metadata, which allows more flexible queries. This component assumes that the context-based storage is available for use from the device that it is being executed on. The client component can store several media objects by providing a MediaObjectList in a vector format. Similarly, a query can be formulated by the client to retrieve media objects all at once which returns a vector of serialized media objects.

The component also provides temporary storage on the local device and brings up a graphical user interface to browse media files stored locally and/or remotely and the ability to retrieve media files matching specific metadata. The application makes calls to the storage functions, and media is transferred and stored without regard to network, transfer protocol, location of the server-side storage or the type of the repository.

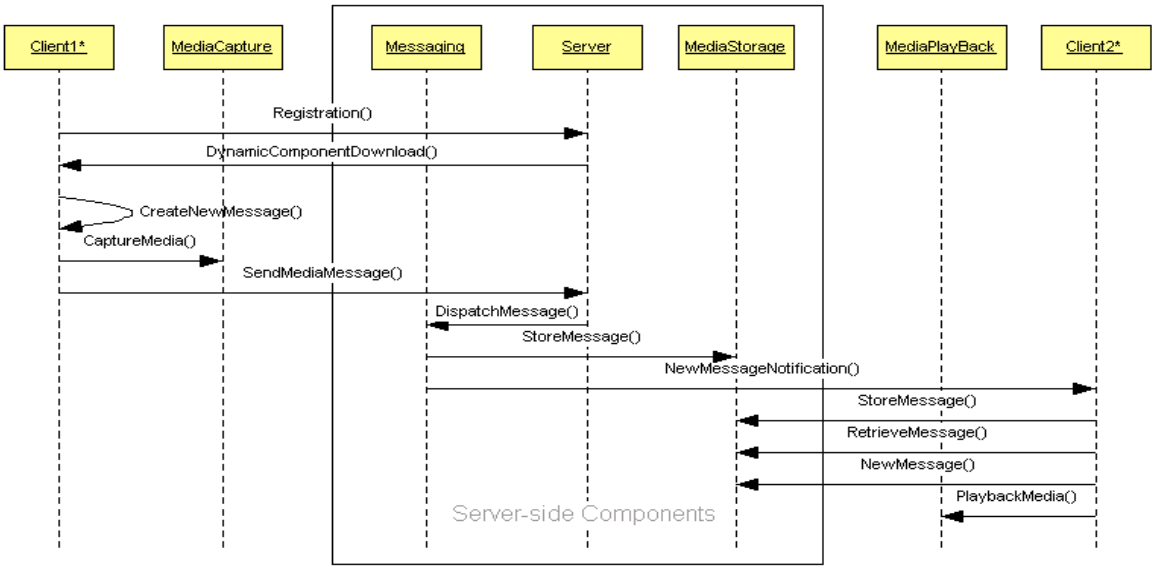

Figure 4: Example of Component Interaction for Media Engine

3.2.4 MediaAlerts and MediaMessages

The MediaAlert component provides a push-type media object delivery mechanism for the components and applications. This component borders on the level of an application and provides an essential service for pervasive applications. Components and applications that want to receive push-type content must subscribe to the MediaAlert component to receive it. A MediaAlert component can be used for two different purposes:

1. Pushing media alerts to components and applications.
2. Delivering media messages between components and applications.

The first is intended primarily for third party applications, such as advertising tools, in pervasive environments. The second should be though of as a way to deliver multimedia messages between clients.

Media alerts are generated when some event occurs and they are delivered to all components and applications that have subscribed to receive media alerts. Components and applications use a media alert to deliver messages to components that have subscribed by providing the media object and recipient's identification.

Components and applications can also invoke different events on the media alert component. This allows a flexible infrastructure where new alert generating components can be written easily. Clients subscribe for media alerts and they receive notification of an event, if they have subscribed to events with a unique identification. Clients can discover different types of event generating components with a service discovery component.

The functionality of the media alert component is asynchronous. If a client is not available, the media alert component will queue the messages and alerts that the client component would get if it were available.

In case of MediaMessaging, the components that wish to receive media messages subscribe to the sender of the message, which can forward messages to the receiver by using this component. Figure 4 illustrates an application using the media messaging component and its interaction with the other media engine components.

3.2.5 MediaProcessing

This component extends image-processing and manipulation functionalities, downloaded dynamically to the device. It currently includes basic image processing techniques (sharpening, blurring, edge detection, image transforms) as well as advanced image analysis and applications such as text detection and recognition, bar code recognition etc. Some of the media processing components involve the use of native code and achieving platform independence for these is an arduous task. Currently, we have those media processing components compiled for a specific platform and the appropriate components are downloaded based on the information specified in the device profile. If the device does not have the required resources to do advanced media processing, the client is provided with an alternative to do the media processing on the server side and return processed media. The goal is to set up Media Processing as a service oriented component, and make use of service discovery and the network centric processing to provide a wide range of processing capabilities.

4 Implementation

The media engine is designed to thrive in a heterogeneous wireless environment, which may have different configurations of processor architectures and platforms. We have implemented all components using Java. As mentioned previously, the engine has both static and dynamic components, which interact with each other using a generic message format. The server has a dedicated request handler for each component (e.g. MediaAlertHandler, MediaProcessingHandler, MediaStorageHandler), which processes each request coming from its clients and sends the response back from the server. The messages (requests and responses) between client and server components are asynchronous. If the client has to send a new request, it does not need the previous request. This asynchronous message exchange over wireless between client and server is achieved using a generic message format where each message is identified using a unique target component identifier. Our current demonstration environment for the media engine uses an iPAQ on the Windows CE platform, fitted with 802.11b wireless LAN card and a pocket camera. Following paragraphs discuss the implementation details of each component.

The MediaStorage component implements a simple database application where a user can store a media object from a mobile device to the media server (in BLOB format on MySQL database server) or retrieve a media object from the server and view it using the remote file browser facility. The remote file browser allows user to retrieve and/or view remotely stored media objects based on different attributes such as owner, resolution, last modified date/time etc.

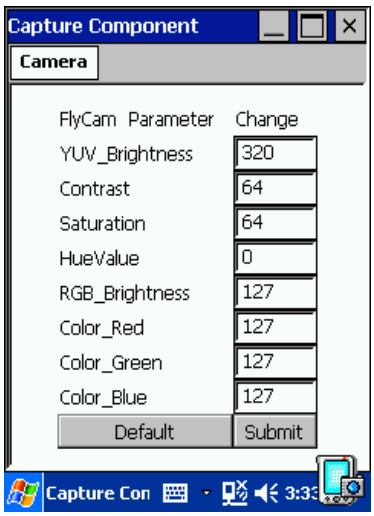

Figure 5: UI for the camera parameters

The MediaCapture component provides a user interface, which allows the user to invoke the various camera functionalities such as initializing the camera, capturing image/video, setting parameters etc. This component is capable of controlling a native API based camera as well as Java API based cameras. To control the native API based cameras, it uses Java Native Interface (JNI). This component has a viewfinder with which we can see the object and capture the image in a bitmap format.

As shown in Figure 5, the MediaCapture component allows the user to modify various camera settings such as brightness, contrast, saturation and hue. Considering the fact that most of the imaging devices for PDAs support native interface rather than java interface at the driver level, MediaCapture component is designed to provide a common interface to invoke native camera application using JNI, as well as camera applications supporting the java interface. The captured images are in Device Independent Bitmap form, which can be dynamically adjusted to adapt properly on any display device. In case of video capture, the video is encoded with low bit rate encoder and stored locally on the device.

The MediaAlerts and MediaMessaging components provide a framework which can be used by third party applications (such as video surveillance) to distribute alerts, facilitate advertising, or to share media between users. On the server side, it keeps the latest information about the subscribers for a particular type of alert. When the application generates an alert message, this component retrieves subscriber information from the database and distributes the alert message to all the subscribers. To exchange media content between users directly, it refers to the registered users information maintained in the database and forwards the message to the recipient of the message. The advantage of this mechanism is that even if the two users exchanging media are part of isolated networks, the MediaAlerts and Messaging component, can transfer information end to end.

5 Applications

We are currently using the media engine to implement a number of use case and application scenarios both inside and outside of the Capnet Architecture.

5.1 MediaToolkit

MediaToolkit is a PDA based demonstration environment for the Media Engine. It is set up as a downloadable client module, with server side video processing for storage and retrieval, media analysis, and remote messaging. The system will provide a WWW interface for registration. The user selects the desired capabilities through the WWW interface and the server builds in the capabilities simulating a fee for configuration based approach and allows the user to download only the components (including drivers, etc) they need.

When executed, the interface will first provide basic capture capabilities from a menu. The drop down interface provides video preview followed by either image or video clip capture. The video clip can be stored locally or remotely, sent as a media message or set of a variety of remote processing applications on the server. Figure 6 shows the types of media processing that can be done with the media toolkit.

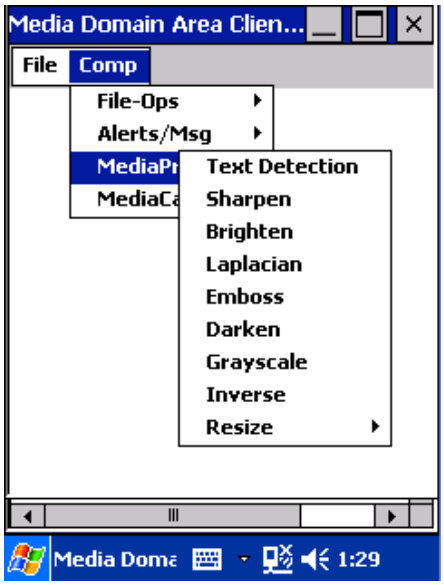

Figure 6: Media Toolkit UI

5.2 Remote Surveillance

We are also integrating our system with existing video surveillance applications. One such application is a remote video activity analysis system. Currently, a "driver" application provides a background subtraction module and a primitive classification of content [1]. The surveillance module is set up so that it records video and keyframes from periods where there is significant activity and uses simple profile measures to classify content as being a person, vehicle or unknown. The content is stored on the server.

The clients can subscribe to various surveillance alerts and then retrieve video or keyframes from the event. The frequency of the alerts and the detail provided can also be configured.

6 Extensions and future work

The focus of this system has been to provide a framework for media processing. The goal is to provide the basic capabilities so that users can focus on building applications, and not on the underlying implementation. We are in the process of integrating a number of other capabilities that will be essential for general pervasive applications. Video streaming will be added and integrated with media messaging, and alerts. In the immediate future, the service will be implemented by calling a third party server and client. Although the current system supports media objects, streaming is seen as an important, yet realizable feature which has received a great deal of attention.

The second area of future work is enhanced media processing. A great deal of work at Maryland has focused on image and video processing, including face detection and recognition, recognition of gestures, mosaicing, image stabilization, etc. The ability to provide those capabilities for media obtained from mobile devices is clearly a significant challenge. Yet as technology matures, demand for such capabilities will increase.

7 References

[1] T. Horprasert, D. Harwood, and L.S Davis. A Statistical Approach for Real-time Robust Background Subtraction and Shadow Detection . ICCV Frame Rate Vision Workshop, 1999.

[2] J. Zhang, X. Chen, J. Yang and A. Waibel. A PDA-based Sign Translator Proceedings of the 2002 International Conference on Multimodal Interfaces (ICMI), October, 2002

[3] A. Ekin and A. M. Tekalp A generic event model and sports video processing for summarization and model-based search Handbook of Video Databases (ed. Borko Furht and Oge Marques). CRC Press 2003.

[4] T. Palm, G. Schneider and S.Goose A situated computing framework for mobile and ubiquitous multimedia access using small screen and composite devices. Proceedings of the eighth ACM international conference on Multimedia, October 2000.

[5] J.Bates, D.Halls, J.Bacon A framework to support mobile users of multimedia applications Special issue on mobile computing and system services, December 1996.

[6] D.Wichadakul, X.Gu, K.Nahrstedt A programming framework for quality-aware ubiquitous multimedia applications Proceedings of the tenth ACM international conference on Multimedia, December 2002**.**

[7] D. Doermann, A. Karunanidhi, N. Parekh and V.Rautio Video Analysis Applications for Pervasive Environments Mobile Ubiquitous Multimedia, December 2002.

[8] CAPNET Context Aware Pervasive Networking (http://www.mediateam.oulu.fi/projects/capnet)

[9] Breuel, T. M. Janssen, W. C. Popat, K. Baird, H. S. Paper to PDA. International Conference on Pattern Recognition, August 2002