Software for Configuration of a Video Server

A Project submitted to the Oregon State University

in partial fulfillment of the Requirements for the Degree Master of Science

Saumya Venkataraman
venkatsa@cs.orst.edu

Department of Computer Science
Oregon State University
Corvallis, Oregon

Abstract

A video server is the multimedia equivalent of a data file server. As video goes digital video servers that support digital media are required for production and broadcasting in television companies. Grass Valley Group provides comprehensive digital video solutions through a range of video servers called Profiles.

The Profile Media Platform provides a multi-channel, high bandwidth platform for the storage and manipulation of video and audio in professional applications. It has a wide range of capabilities, from a stand-alone digital disk recorder to being part of a large network of video servers.

In this project, I have designed and implemented software to add new functionality to software used for the configuration of the Profile. A Profile system administrator uses this software to set up a Profile before it can be used for production and broadcasting.

CONTENTS

Acknowledgements

I would like to express my gratitude to some wonderful people who helped me complete this project and get through graduate school.

Thanks to Grass Valley Group, Beaverton for giving me the opportunity to do an internship. I would like to thank my manager Ed Vandehey and my mentor Marla Bennett whose patience and guidance helped me complete this project.

My earnest thanks are extended to my major professor Dr. Timothy Budd, who provided direction and valuable insights through all phases of this project. I would like to express my gratitude to Dr. Timothy Budd for his counsel and support. I would also like to thank Dr. Prasad Tadepalli and Dr. Walter Rudd for their guidance and encouragement.

Thanks to my parents who have been a constant source of love and inspiration throughout my education. A special thanks to Rudhra whose support helped shape this work. My deepest appreciation to my friends for their help and encouragement. My gratitude extends far beyond the acknowledgement.

1. Introduction

Life in a television production and broadcasting environment is fast, deadline driven and offers no margin for error. Instant access and management of all shots and spots needed for broadcast and post-production are critical. Grass Valley Group, located in Beaverton, Oregon provides digital video systems (hardware with required software) for television broadcasting companies. Their platforms support video production, storage and manipulation while supporting multiple video compression formats and resolution necessary to meet content creators' and distributors' needs.

A video server is the multimedia equivalent of a data file server. It is a full computer system, including storage, processing, and input output functions. In addition, it contains video encoders and decoders to convert analog content into digital and vice versa. The difference between ordinary data files and video data files places extraordinary demands on the architecture of a video server. Video data, particularly MPEG-2 data, differs from typical computer files in that the MPEG file is a large, continuous, and complex stream of information that must be managed as it travels through then entire system.

The Profile family of Networked Video File Servers manufactured by Grass Valley Group offers wide range of capabilities, from standalone digital disk recorders to a large network of video file servers. The Profile Media Platform provides a multi-channel, high bandwidth platform for the storage and manipulation of video and audio in professional applications. A digital disk recorder is the most cost efficient method available for the storage, playback, editing and caching of video data. The advantage of using a single, highly reliable digital disk recorder is that it replaces multiple, maintenance intensive Video Tape Recorders (VTRs'). Multiple Profiles can be connected to act as one huge "virtual recorde r" with media being transferred over Fibre Channel. The Profile can be used in a wide variety of applications from spot (advertisement) insertion, program delay, store and forward, to more complex applications such as live event coverage, news operations, and non-linear editing. Grass Valley Group products support multiple compression formats like Jpeg, Standard Definition Mpeg, High Definition Mpeg, and DVC.

1.1 The PVS1100 News & Sports Server

Few broadcast environments match the pressure of a newsroom because it is deadline driven. This requires a news system that gives total access to all the material, supports creative freedom and lets one quickly re-purpose the material. To satisfy production applications such as news and sports, Grass Valley Group (GVG) has added the PVS1100 to its Media platform to support news and sports production workflow. The PVS1100 features support for compression formats DVCPRO25, DVCPRO 50 and

MPEG II 4:2:2 I-Frame support up to 50Mb/s, and Mpeg D10 compatibility. Configurations include 2, 4, 6 and 8 bi-directional channels that can be reconfigured as needed. To keep pace with production workflows, PVS1100 features built-in, dual-channel SDTI support, enabling it to accept compressed VTR data streams up to four times real-time speed.

1.2 Problem Statement

In order to setup a Profile according to a broadcast company's needs, the system operator or administrator has to configure various parameters on resources (like encoders, decoders, inputs and outputs) and define channels in order to use those resources. Profile applications use channels to control disk recording and playback. A channel defines a grouping of Profile video, audio, and timecode resources and has a unique identifier.

The user gets to change settings on a Profile through the software application called the *Configuration Manager*. The Configuration manager is used to set parameters for Video Input, Output and Timing, Audio Input and Output, Channel Configuration etc. The Configuration Manager is essentially used to set certain parameters and send it down to the real time system, which allows media to go in and out of the Profile according to user settings. The *Channel Configuration*, which is invoked from Configuration Manager, is used to define channels on the Profile. A channel mainly specifies which input and output resources are to be used and which compression format the video would be in.

The application software mentioned above is present on existing Profiles. For the PVS1100, since more features have been included, changes needed to be made to the existing applications to support the new features. The key new features to be added to the Configuration Manager and Channel Configuration are

- Support for DVCPRO25, DVCPRO 50formats
- MPEG II 4:2:2 frame up to 50Mb/s and 4:2:0 up to 15Mb/s.
- Mpeg D10 compatibility
- SDTI support - bi-directional channel types with configurable input and output with 1X and 4X playout rates on DVCPRO 25 channels and 1X and 2X playout rates on DVCPRO 50 channels.

My work at Grass Valley Group involved incorporating these new features into the Configuration Manager and Channel Configuration software applications. GVG does not write separate application software for each of its Profile products. Software is written in such a way that appropriate features are enabled depending on the circuit boards present and messages given by the real time system on

the Profile. I was responsible for the design and implementation required to configure new features on the PVS1100. Software written had to be compatible with earlier versions of the software.

2. Overview of the Component Architecture

Before proposing a solution to the problem it will be helpful to overview the existing design model for the Configuration Manager and Channel Configuration. They are both dialog based user interface applications that run on Windows NT. They let the user configure resources on a Profile. Both these applications can run on a Profile or on a PC that connects remotely to a Profile.

2.1 Configuration Manager

The Configuration Manager is a Windows NT GUI application, which lets a user configure various settings on the Profile.

2.1.1 Need for Configuration Manager

Various settings need to be configured on the Profile by the system operator or administrator before it can be used. There are two operating systems on a Profile: Windows NT and VxWorks (for real time processing). All configuration changes made by a user in the Configuration Manager on Windows NT need to be communicated to the real time side; this is done through IPM (Inter Personal Message) messages passed between Windows NT server and the real time server.

The Configuration Manager is the tool used to set up the Profile for operation. It is an NT based GUI application which lets the user configure various hardware settings (on circuit boards) and other settings. Examples of settings that can be set using the Configuration Manager include information about the port through which media comes in, the expected kind of audio feed, creation and modification of channels. Settings configured using the Configuration Manager have to be communicated to the real time processor. When the configuration manager application is brought up, it creates a COM server called the *ConfigBroker* on the Profile. All interactions between the configuration manager and the layers of software that interact with the Real time system on the Profile are through the ConfigBroker object. The ConfigBroker retrieves information about various settings (including certain circuit board settings) to display in the Configuration Manager user interface. The ConfigBroker server interacts with an NT library called *tekcfg*, which has information about the various settings. The tekcfg in turn exchanges IPM messages with the real time system. Any changes made in the Configuration Manager are also communicated to tekcfg using the ConfigBroker.

When the Configuration Manager displays a configuration setting, it first makes a call into one of the methods in the *ICfgBrokerObj* interface implemented on the ConfigBroker. The method called on the

ICfgBrokerObj makes a call into a function in the tekcfg library to retrieve the required information. The information is then returned to the Configuration Manager to be displayed by it.

2.1.2 Why COM?

The Configuration Manager (and Channel Configuration) can run on either a Profile or a remote PC connected to a Profile. This necessitates an efficient interface for communication between the machines. Before the Configuration Manager was designed earlier applications used standard C interfaces and custom RPC (Remote Procedure Call) implementation, which made development difficult and time consuming.

COM allows one to make function calls into other objects without concern for which processor machine they are in. It is built on top of RPC, which handles the interprocess/machine communication. This was one of the main reasons why the Configuration Manager was implemented using COM.

Immutability of COM interfaces: COM also better facilitates versioning of interfaces. Each COM interface has its own unique GUID (Globally Unique IDentifier). COM interfaces should never be modified after they have been published (released). If a new method is added to an already existing interface, it has to an entirely new interface with its own unique GUID. This is done in order to support *backward compatibility*. An object in COM can support multiple interfaces. Given a pointer to one of the interfaces we can get a pointer to any other interface in the object. This principle has been used widely in this project. A new interface was added to an object that has already been released in earlier versions of the software. It is possible to modify the implementation of methods in an already existing interface; however, changes to the prototype of the methods should not be made.

2.1.3 Sample User Interface

Stepping through one of the simple wizards on configuration manager should put things in perspective. Invoking the configuration manager on a Profile or a remote PC brings up the Configuration Manager dialog as shown in Fig. 2.1. Invoking the configuration manager immediately starts a session of the ConfigBroker server on the Profile.

Fig. 2.1 Configuration Manager Main Dialog

Selecting any of the buttons on the main dialog invokes a wizard associated with that button. The user can modify configuration settings in the wizard. Selecting the Video Input button invokes the Video Input wizard in which the user can modify various video input settings. Before the first page of the wizard comes up, function calls are made by Configuration Manager into the ConfigBroker server to retrieve information to be displayed in the wizard. ConfigBroker then makes function calls into the tekcfg library. As an example a page of the Video Input page is shown in Fig. 2.2.

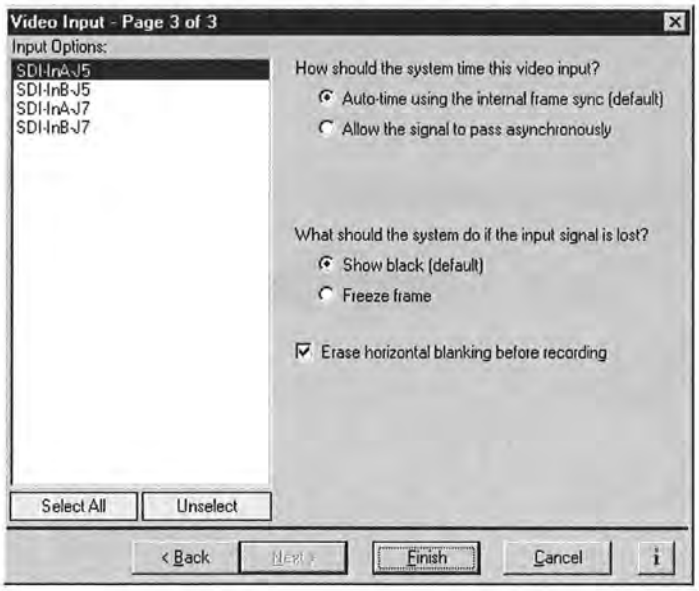

Fig. 2.2 Video Input Wizard Page

Once the user has made the required changes in the wizard page and hits Finish a Summary dialog is brought up. Making function calls into the ConfigBroker object saves all the settings changed by the user. An example of a summary dialog is shown in Fig.2.3.

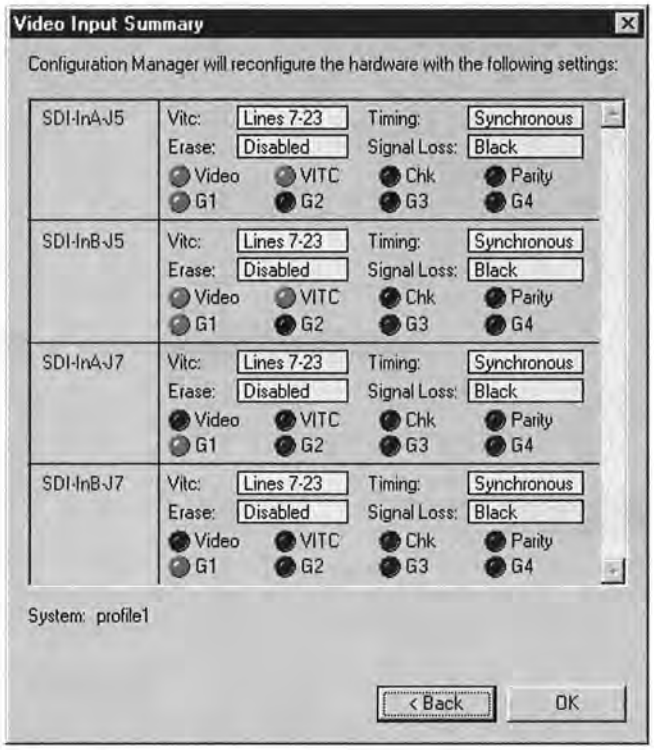

Fig. 2.3 Video Input Summary Dialog

2.1.4 Architecture

The Configuration Manager architecture is shown in Fig. 2.4.

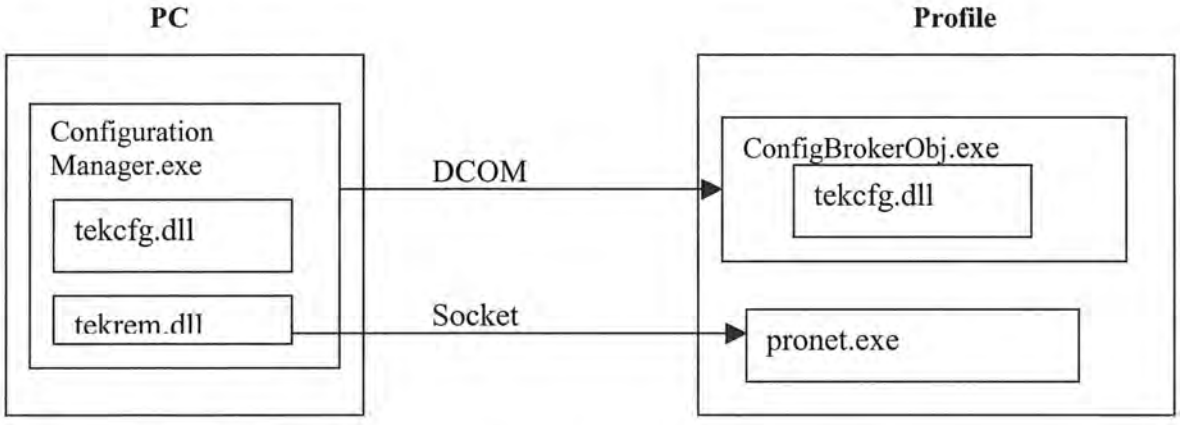

Fig. 2.4 Configuration Manager Architecture

All the components that have been shown to run on a PC will run on a Profile if the Profile is used for configuration. The tekrem.dll and pronet.exe are used to make a remote connection between the Profile and the PC. They are remnants of the custom-RPC interface that is still used for some remote communication.

2.2 Channel Configuration

A channel is a group of resources on the Profile and the connections between these resources. A channel definition is the persistent state of a channel, including a description of the resources and resource settings. Basically, a channel has inputs and outputs; media comes into the Profile through an input resource and goes out through an output resource. Profile software supports three channel types: Recorder channel, Player channel and Player/Recorder channel. A channel used only for Input is a Recorder channel, one used only for output is a Player channel and one used for both is a Player/Recorder channel. Depending on the kind of channel, the user can select from a set of resources. The compression format of the Video determines the name of the channel (Mpeg Player, Dvcpro25 Player/Recorder, etc.).

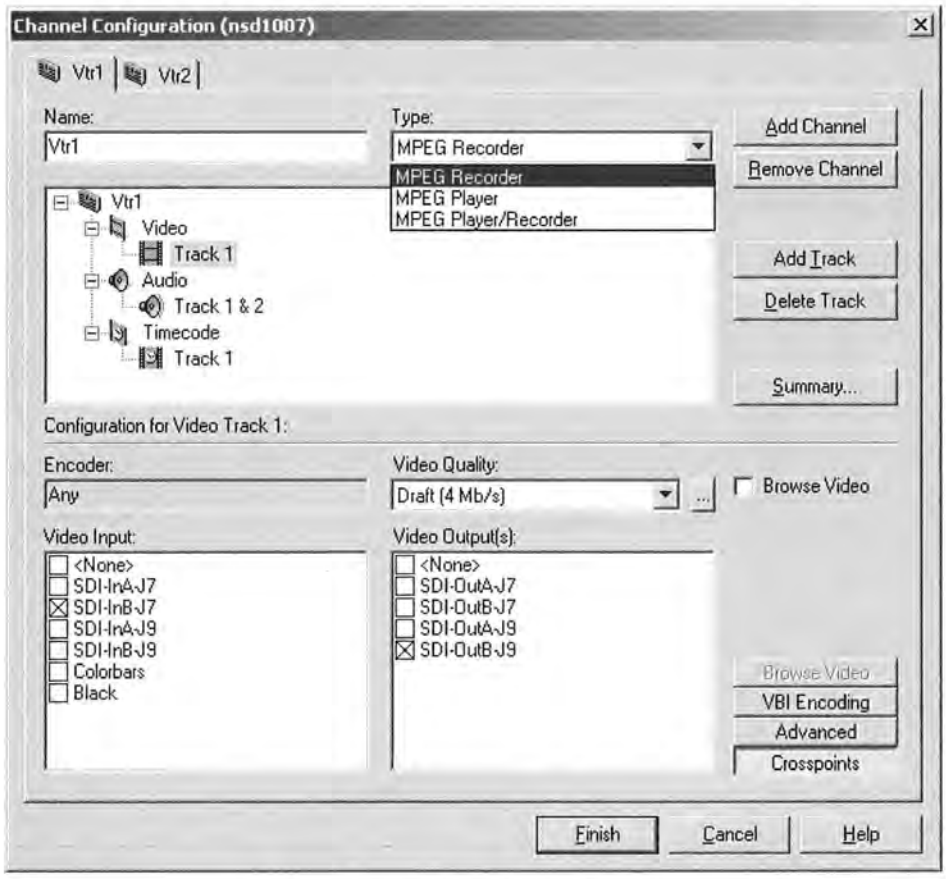

Fig.2.5 Channel Configuration Dialog

Various other parameters need to be selected for a channel depending on its type. The user sets these parameters in the Channel Configuration dialog. The user can then use these channels from a number of applications on the Profile, e.g. Time Delay, Toolbox editor, List Manager. A sample of a channel configuration dialog is shown in Fig. 2.5. The Channel Configuration is a part of the Central Resource Management system, which is discussed in the next section.

2.2.1 Need for Central Resource Management

Before the Central Resource Management model was introduced the user defined a different channel configuration for each application, forcing them to re-think which resources and configuration settings they need for every application. There was not one place where the user could define channels for all applications.

Central Resource Management (CRM) is the fundamental concept underlying the design of the Channel Configuration. It accomplishes three main tasks: (1) provides a tracking mechanism for the status of all resources on a machine, (2) abstracts away many of the details of resource management from the application developer, and (3) provides the user with a uniform view of channel configurations across applications.

2.2.2 Using Object-oriented API

The central resource management system has been implemented using *object-oriented technology*. The objects in the system provide a COM interface that can be accessed from either a local or a remote machine – no intermediary process need be defined for remoting. This interface can be used by applications that are written in Java or C++, which is one of the major advantages of using COM.

The object-oriented API provides an application developer with more channel and resource management functionalities. For example, the application can query the resource management objects for information about any aspect of a channel's definition. It also provides the capability to allocate a channel. An application allocates a channel by calling a single function which is defined to open a port, allocate resources, apply port and resource specific settings, and connect the resources. Such high-level functionality will make it easier to write applications that use Profile resources.

Many of the components in the existing system have been reused in the new system with few modifications. The most significant difference is that resource management is no longer an application-

specific task, but rather, a machine-specific task. Therefore, there are some additional issues that need to be addressed such as synchronizing access to data, and managing channel configurations for multiple processes.

Providing an object-oriented approach to central resource management is the first step in a larger plan to update and improve upon the current API. It is the first of a set of middleware components that will provide application developers with a richer system interface that encapsulates features common to a variety of client applications.

Thus, CRM makes use of the principles of **abstraction** and **encapsulation**. Data associated with an object is not accessible to the outside and only required functions can access it. Implementation details are hidden from developers who can now use functions without learning about how they work.

Central resource management also provides a better model to match the customer's workflow. With central resource management, an administrator can configure a set of channels for a machine that can be used by all applications. A channel labeled "MPEG Recorder1" is "MPEG Recorder1" to all applications on the network. Channel configuration becomes a part of the machine setup.

2.2.3 Architecture

This section will describe how the Channel Configuration user interface interacts with components of the central resource management system (interaction shown in Fig. 2.6).

The *Channel Configuration Dialog*, which is brought up from Configuration Manager, provides the user interface for defining channels. It interacts with three main components: the Channel Dictionary, Resource Monitor and TekVdr (which we will not discuss in detail). Following is a description of each of these components and how the Channel Configuration Dialog uses them.

The *Channel Dictionary* is the central repository for channel definitions. The Channel Configuration Dialog gets the channel definitions from the Channel Dictionary, and updates the Dictionary with new definitions if the user changes them.

The *Resource Monitor* keeps track of ownership information for all resources on a profile. It knows which resources are in use and which applications are using them. The Channel Configuration Dialog uses the Resource Monitor to get information about all system resources.

TekVdr is a library that provides basic functionality for using ports and resources. It uses the *Channel Dictionary* to look up a channel definition when an application wants to allocate a channel. It will notify the *Resource Monitor* whenever a resource is allocated or released.

In the diagram below the *Application* object represents any application that uses channels.

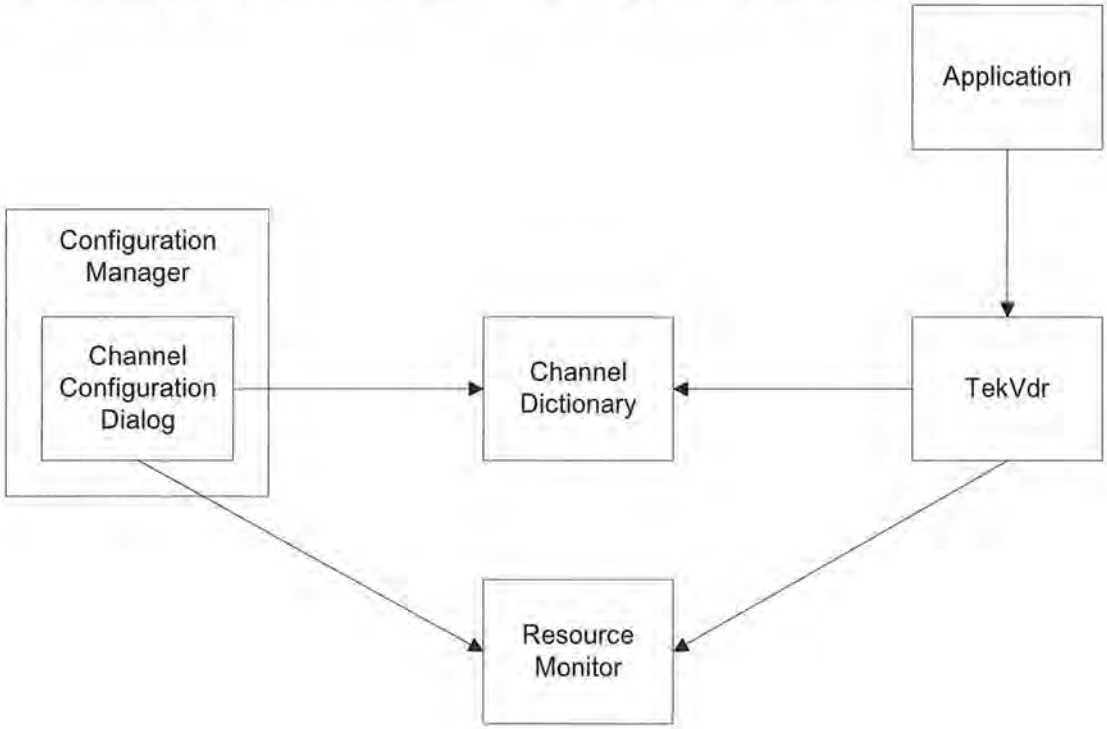

Fig. 2.6 Central Resource Management

The Channel Configuration Dialog has one class that manages the data (channel definitions and video qualities) and handles the interactions with the Channel Dictionary and Resource Monitor, the *ResourceMap* class. It encapsulates the data that will be displayed in the Channel Configuration dialog. This data includes the channel definitions, video qualities and system resources.

A list of ResourceInfo objects is used to track information about each resource in the system. Each ResourceInfo object has a list of ResourceLocation objects (which can be empty). Each ResourceLocation object in the list represents a track (audio, video or timecode) that has the resource specified on it. The list of ResourceLocations is used to display conflict information in the user interface i.e. if more than one channel tries to use the same resource. If the list of ResourceLocations is empty, the resource is not

specified in any channel definition; if there is one item in the list, the resource is in use by one channel definition; if two or more items are in the list, there is a resource conflict.

The Resource Map class has two attributes, the current channel and current track, which are used to optimize access to the channel and track that the user is currently editing. The Channel Configuration Dialog, along with a number of child dialog classes, display the data encapsulated in the ResourceMap.

3. Design and Implementation of New Features

In this chapter, we will highlight the new features that had to be added to the Configuration Manager and Channel Configuration, the issues involved and the design chosen and implemented.

3.1 Configuration Manager

In this section we will discuss the requirements, UI (User Interface) design changes that were made, and the new classes and COM interfaces that were implemented in the Configuration Manager to support a new board added to the Profile.

3.1.1 Requirements

A new board (C-cube Video Codec board) was added to the Profile, which consisted of two codecs. A codec is a term that refers to an encoder and a decoder. Configuration of the C-cube board was different in that; it could be configured to be an Mpeg Encoder, an Mpeg Decoder, a DVCPRO25 codec or a DVCPRO50 codec. Prior to the c-cube board, a codec could only encode/decode a single compression format. This necessitated a new wizard (*Video Codec* wizard) in the Configuration Manager in order for the system operator to configure the board depending on his requirements.

When the Configuration Manager UI detects that the Profile has C-cube codecs, it should enable the *Video Codec* wizard button. Pressing this button should open the Video Codec wizard page, which shows the codecs present on the Profile and what each one is configured to be at that instance of time. The operator should be able to reconfigure any of the codecs. The system has to be rebooted so that the appropriate microcode is loaded on to the real time system.

Software changes made should be compatible with earlier products in the Profile Media platform. The Configuration Manager should support systems configured with either IBM encoder and decoder based boards (for Mpeg and DVC compression formats) or the new C-cube video codec boards (which also have codecs for Mpeg and DVC encoding and decoding). Profiles with both types of boards need not be supported. They are licensing issues involved with the C-cube board; features like DVCPRO50 encoding and decoding which a user may not be licensed to use should be disabled.

Design Notes: As discussed in Chapter 2 the Configuration Manager does not make direct calls to the *tekcfg* library. It calls into the ConfigBroker server object, which interacts with the *tekcfg* library that in turn with the real time side using IPM messages. To implement the new wizard a new call has to be added

to the ConfigBroker to expose the presence of C-cube codecs, so that the UI can show the Video Codec wizard. The ConfigBroker has to set and get C-cube compression assignments. These values are stored in the non-volatile RAM on the codec board itself. The ConfigBroker object interacts with the tekcfg library to pass IPM messages to the real time side.

3.1.2 User Interface Design

A new *Video Codec* wizard was added to the Configuration Manager main dialog as shown in Fig. 3.1. It is enabled only when a C-cube board is detected on the Profile.

Fig. 3.1 Configuration Manager Main Dialog showing Video Codec button

Before the Configuration Manager dialog comes up, an instance of the Config Broker server is started on the Profile. When the Video Codec button is selected the wizard dialog shown in Fig. 3.2 is displayed. The Configuration Manager interacts with the ConfigBroker server to collect information that needs to be displayed on the dialog.

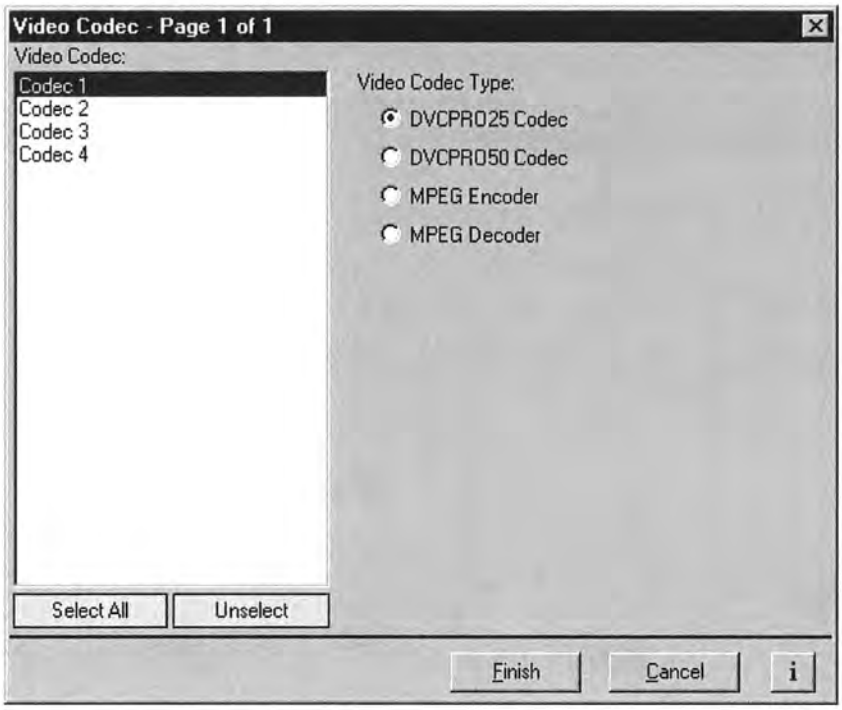

Fig. 3.2 Video Codec Wizard

This page allows the user to configure a Video Codec type for each C-cube codec detected on the Profile. The codecs detected on the Profile are displayed on the left side of the page. When the Finish button is pressed, the warning dialog (Fig. 3.3) appears if there have been any changes made in the page. Changes are passed on to the hardware only when the Profile is rebooted.

Fig. 3.3 Reset Warning Dialog

Pressing OK on this dialog brings up the Summary dialog (Fig. 3.4).

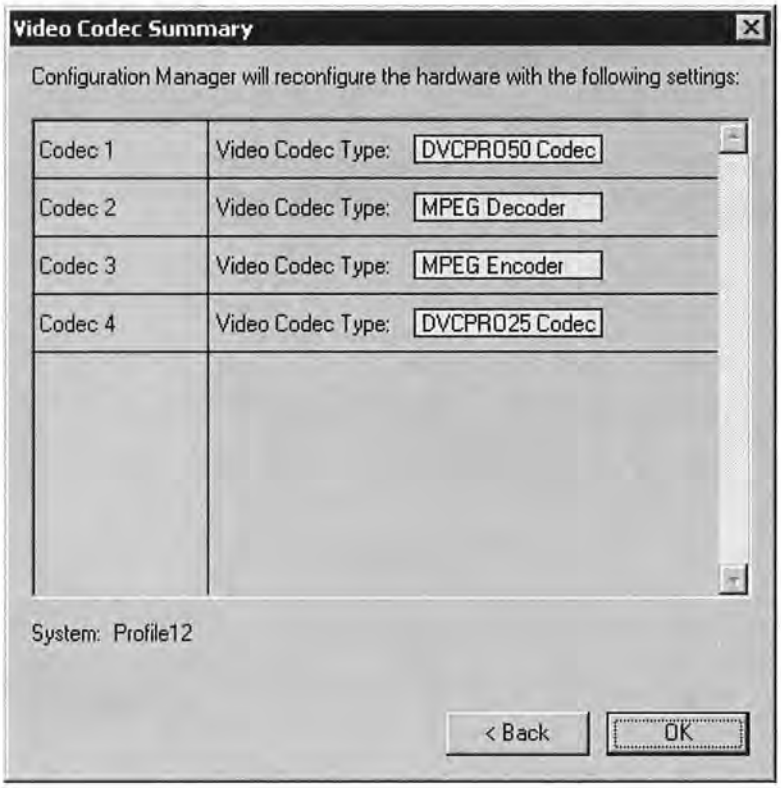

Fig. 3.4 Video Codec Summary Dialog

If the user hits OK on this dialog all the changes made by the user are to be passed on to the real time processor. Configuration manager passes on the data to the ConfigBroker object, which in turn passes it to the tekcfg library. Tekcfg gives the data to the real time processor using IPM messages.

3.1.3 Implementation Overview

Changes were necessitated the Configuration Manager and the ConfigBroker server. The important classes that were added in the Configuration Manager are as follows:

- *CVideoCodecPg* inherits from an already existing class CWizardPg. The purpose of this class is to maintain data that needs to be displayed on the Video Codec wizard dialog and also capture all user actions.
- *CVideoCodecState* inherits from a class called the CConfigState. It keeps track of the state of each object on the page; whether items need to be enabled or disabled, the number of items to be displayed, etc. This class dynamically changes the state of various objects in the wizard page.
- *CVideoCodecObj* inherits from a class called CConfigObj. This keeps track of the properties of each object in the wizard page. This class maintains a list of objects containing information about codec configuration i.e. the kind of resource it is at a particular instant.

- *CVideoCodecSummaryPg* inherits from CSummaryDlg. It is used to display the Summary Page. It interacts with objects of CVideoCodecState and CVideoCodecObj.

The header files for these classes has been included in Appendix A.

An OOP concept used here is **polymorphism**, which is the ability to treat objects (related by inheritance) from different classes identically because they all have one or more interfaces in common. It allows a child class to interpret a message to itself overriding a default implementation in the base class. An example of polymorphism being used is when a user selects a button in the configuration manager dialog; depending on which button is selected the appropriate wizard dialog is initialized (all wizard pages are inherited from the CWizardPg).

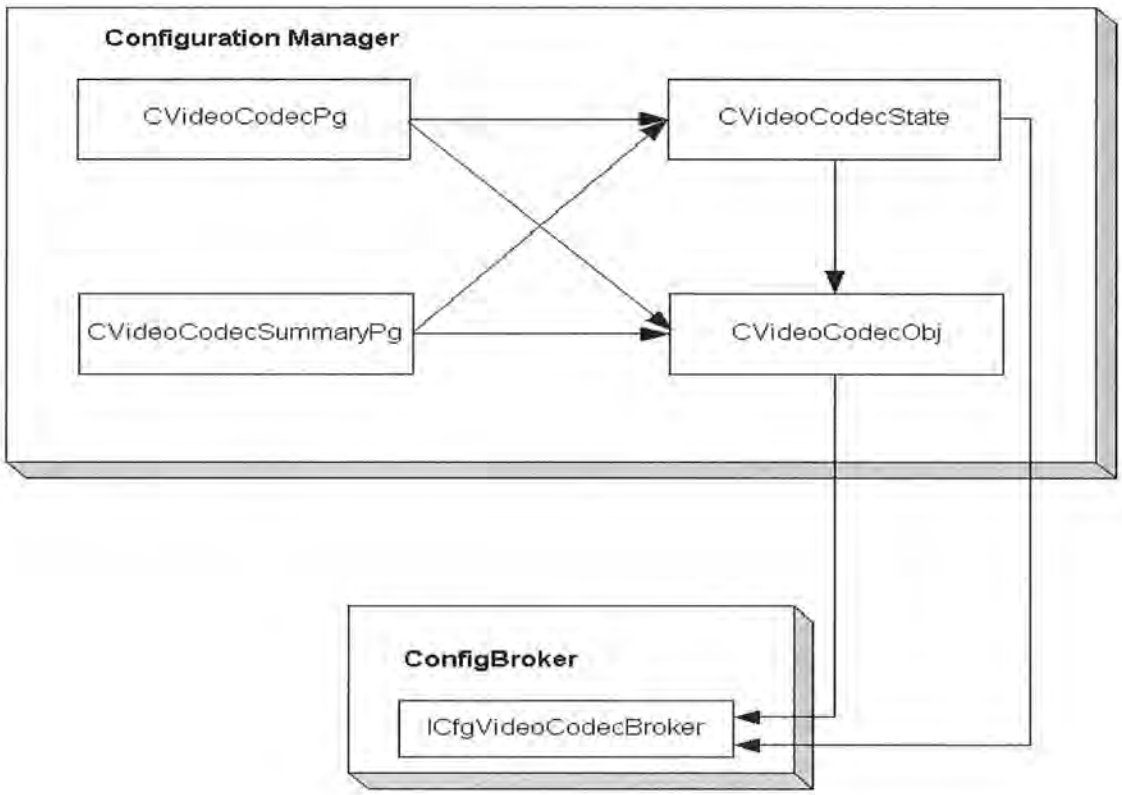

Fig. 3.5 Interaction between new objects implemented for the Video Codec wizard

A new interface was added to the ConfigBroker object called the *ICfgVideoCodecBroker*. A new interface is necessitated due to the **immutability of COM interfaces**. As discussed in section 2.1.2, COM interfaces should never be modified after they have been published (released). If a new method is added to an already existing interface, it has to be an entirely new interface. This is done in order to support

backward compatibility. Therefore, a new interface *ICfgVideoCodecBroker* had to be added to the ConfigBroker object. Various methods were implemented in the interface. These methods load (and save) settings on the c-cube video codec board through calls to the tekcfg library. The function prototypes are outlined in Appendix A The interaction between the various classes and COM server is shown in Fig. 3.5. The CVideoCodecPg object is invoked by the main configuration manager dialog.

3.2 Channel Configuration

In this section we will discuss the requirements, UI (User Interface) design changes that were made, and the new classes and COM interfaces that were implemented for Channel Configuration to support a new board added to the Profile.

3.2.1 Requirements

New features were necessitated in the Channel Configuration dialog due to the addition of a new circuit board called the *SDTI* (Serial Data Transport Interface) board. The SDTI I/O (Input/Output) board is responsible for SDTI-DVCPRO25, and SDTI-DVCPRO50 I/O. Each board is capable of two channels of SDTI I/O. Each channel can be independently configured as either of the DVC formats.

When the Channel Configuration dialog opens it should check if there are any SDTI boards (resources) on the Profile. If SDTI resources are detected two channels SDTIDVCPRO25 Player/Recorder and SDTIDVCPRO50 Player/Recorder should be made available to the user. Each of the compression formats (SDTI-DVCPRO25 & SDTI-DVCPRO50) is associated with certain playout rates, which are the speeds at which playout is possible. Playout rates possible for SDTI-DVCPRO25 are 1X and 4X while those for SDTI-DVCPRO50 are 1X and 2X. The system operator should be able to choose the required playout rate.

When the user selects either of the SDTI channels a certain number of audio, video and timecode tracks should be added to the channel. 1 video track, 2 audio tracks, and 1 timecode track in the case of a SDTI DVCPRO25 Player/Recorder channel or 1 video track, 4 audio tracks, and 2 timecode tracks in the case of a SDTI DVCPRO50 Player/Recorder channel. A list of the available SDTI codec resources has to be displayed for the user to choose from. SDTI resources used by another channel should be grayed out and if the user still chooses from one of them a conflict should be displayed prompting the user to change the selected resource on one of the channels.

A new user interface had to be designed to support the two new channel types: *SDTI Dvcpro25 Player/Recorder* and *Dvcpro50 Player/Recorder.* All changes made should be saved on to the Channel Dictionary for other applications on the Profile to use. The Channel Dictionary (COM server) has to be able to recognize SDTI channels. This requires new COM interfaces and modifications to existing interfaces on the Channel Dictionary. Software changes made should be compatible with earlier products in the Profile Media platform.

3.2.2 User Interface Design

Two new channel types had to be added to the Channel Configuration dialog. Selection of any one of them should bring up the corresponding child dialog.

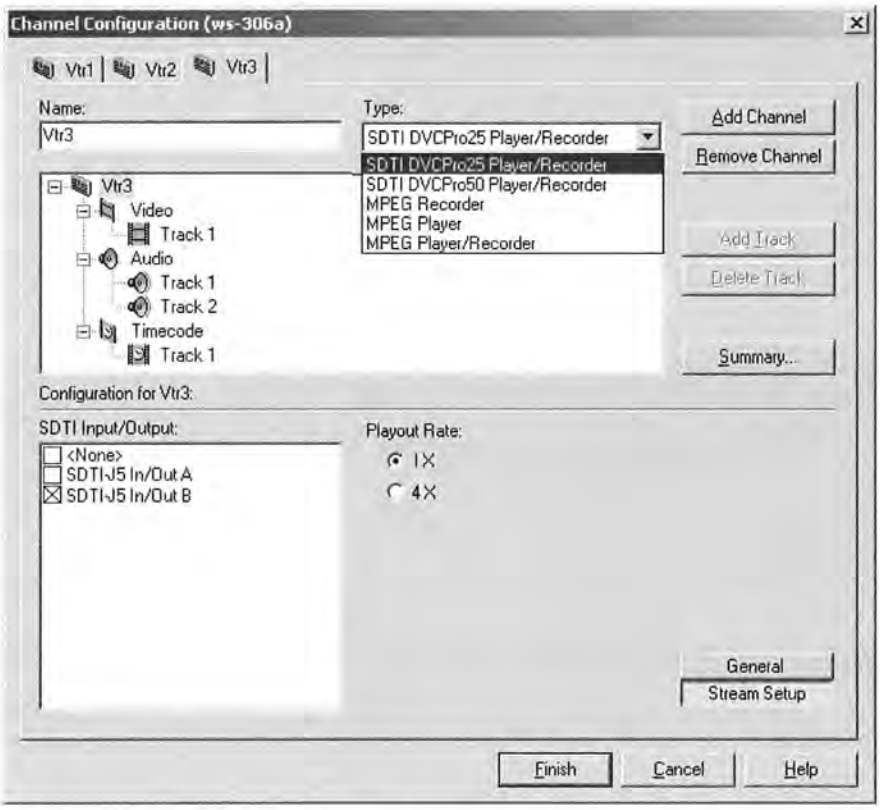

Fig. 3.6 Channel Configuration Dialog showing SDTI channels (Stream Setup dialog)

In the channel configuration dialog all controls and components change dynamically depending on the channel type that is chosen in the *Type* combo box. Each channel type has a few child dialogs associated with it. The number of child dialogs can be determined by the number of buttons on the lower right hand side of the dialog (two in this case: General and StreamSetup). In Fig. 2.5, we can see that there are four child dialogs associated with the Mpeg Recorder channel type. The number of Audio, Video and Timecode tracks present are also dependent on the channel type.

In Fig. 3.6, the SDTI Input/Output combo box lists all the SDTI codecs available on the Profile. If one of them is grayed out it means that the resource has already been used by another channel. If the user selects a grayed out resource it will show a conflict indicated by a red check.
Selecting the General button in the dialog brings up the dialog shown in Fig. 3.7.

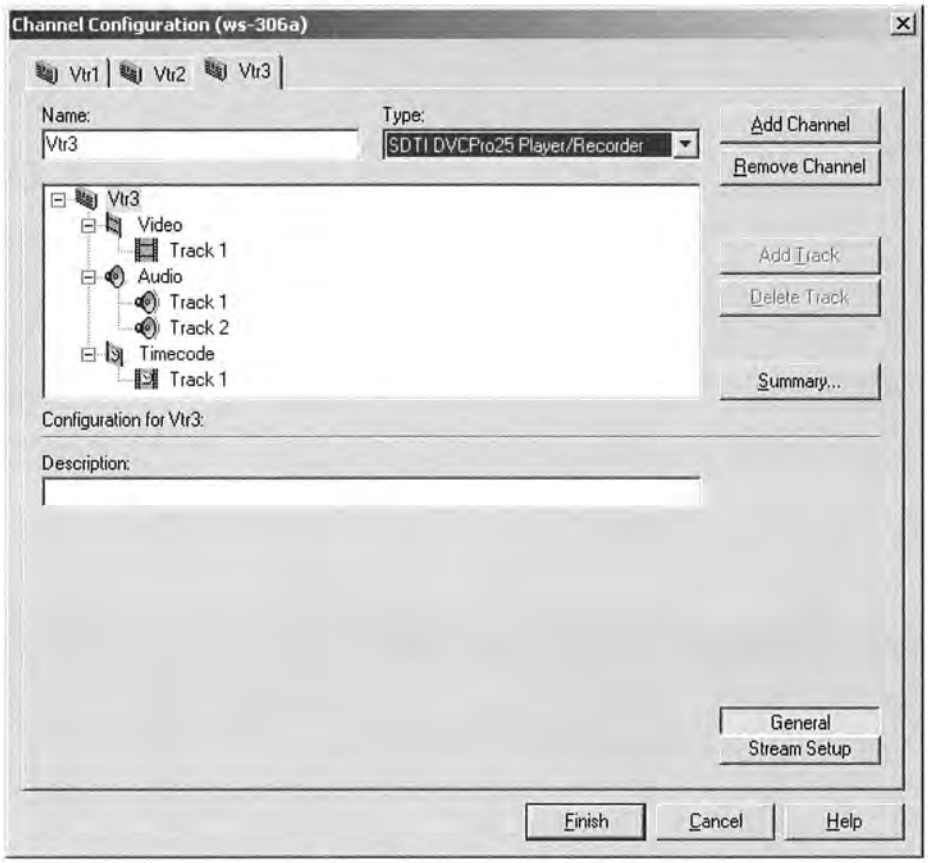

Fig. 3.7 Channel Configuration Dialog showing SDTI channels (General dialog)

3.2.3 Implementation Overview

Issues

The implementation of the two new SDTI channel types had to be different compared to other existing channel types. All earlier channel types associated each resource on the Profile to individual tracks of media. The new channel types had to associate resources to the Channel itself rather than each track in the channel. This is because unlike other boards a SDTI board comes with its own input and output port. Other boards like the Mpeg Encoder or Mpeg Decoder interact with Serial Digital Interface boards for their input or output.

Therefore, new functionality had to be added and existing software had to be modified to support SDTI resources. Backward compatibility was also an issue since modified software had to work even when a SDTI board was not present on a Profile.

Solution

Channel Configuration changes

The attributes of a SDTI channel were different from existing channels; hence, the existing child dialogs could not be reused. A new class, CChannelStreamSetupDlg was implemented in which the user could select the SDTI resource type to be used and the playout rate for the channel. This dialog is displayed when the Stream Setup button is selected in the Channel Configuration dialog. An existing class CChannelDescriptionDlg has been reused; it is displayed if the General button is selected.

Since, the Channel Configuration did not know about SDTI channels, changes were made in many places to support SDTI channels. Code was modified so that SDTI channels associated their resources with the Channel while the rest of the channels associated their resources with their tracks.

Channel Dictionary changes

Changes were made so that the Channel Dictionary would be able to recognize and store SDTI channel information.

A new interface IChannelPlayoutRate was added to save and load channel playout rates. This interface was written such that if any channel in the future had a playout rate associated with it, it could use this interface instead of defining one specific to itself. Existing interfaces were modified to support SDTI resources. COM interfaces allow new functionality to be added to existing methods on an interface as long as the function prototype remains unchanged. This property was made use of in that new functionality was added to existing methods to load, save, retrieve and remove SDTI resources. The header files for the main classes used and the new interface implemented in the Channel Dictionary has been outlined in Appendix B.

3.3 Validation & Testing

Tests were used to validate and verify the working of Configuration Manager (including Channel Configuration) on the PVS1100. The requirements were validated and tested for the C-cube board and

SDTI board. The software was tested at different levels of abstraction starting with Unit testing, moving onto Integration testing and finally System testing.

Unit Testing

Conditions were simulated such that the new user interfaces would behave like it would if C-cube boards or SDTI boards were detected on the Profile. The GUI was tested under these conditions to essentially verify if it captured and responded appropriately to user events and displayed correct information.

Integration Testing

Integration testing required the creation of many test cases since the working of this software involved the interaction of many modules. A few of the important ones used have been outlined below.

- *C-cube Video Codec board*

When a C-cube video codec board was added to a Profile the Configuration Manager detected the number of boards present. The Video Codec wizard button was enabled and the number of codecs available was displayed correctly in the wizard dialog page. Changes made in the wizard dialog were reflected in the real time side after the system was rebooted. This was verified by changes in the number of resources (Mpeg and DVCPRO) in the Channel Configuration dialog. User licenses were verified by the appropriate features being enabled or disabled depending on licenses. These testes verified the Configuration Manager's interaction with the ConfigBroker server object and the tekcfg library.

- *SDTI board*

Addition of a SDTI board was reflected by the addition of two channel types: SDTI DVCPRO25 Player/Recorder and SDTI DVCPRO50 Player/Recorder to the Channel Configuration dialog. Appropriate playout rates were displayed depending on the type of SDTI channel chosen. Each SDTI board consists of two SDTI codecs. The number of SDTI codecs available along with their input/output slot numbers was displayed. These tests verified the Channel Configuration's interaction with the tekcfg library.

Changes made in the Channel Configuration dialog were saved on to the Channel Dictionary correctly when the dialog was closed. The saved changes also got loaded correctly when the Channel Configuration dialog was reopen. Changes made in the Channel Dictionary were accessible to other applications on the Profile. This was verified by Recording and Playing using SDTI channels in other Profile applications.

System Testing

After the Unit testing and Integration testing was complete the software was tested on various Profiles with different configurations.

The Configuration Manager should support systems configured with either IBM encoder and decoder based boards (for Mpeg compression formats) or the new C-cube video codec boards. New software was loaded on Profiles configured with either type of board; appropriate features were enabled in the Configuration Manager dialog. Configurations on a Profile were saved as a configuration file and loaded on other Profiles. The Configuration Manager reflected the new configuration and behaved appropriately depending on the boards present on the Profile. Configuration Manager and Channel Configuration allow remote configuration of Profiles. This was verified by making remote connections between Profiles and between remote PC's and Profiles.

Regression testing

Regression testing was used to make sure that the older software still worked with the new changes. The new software had to be backward compatible and work even on an earlier product of the Profile media platform.

Issues

There were various errors that were found during the testing phases. Errors found during the Unit testing phase were relatively simple to locate and solve since they involved only the new software written.

During the Integration testing phase it was found that the addition of a new wizard in the Configuration Manager dialog did not invoke the appropriate wizard and the existing wizards also behaved incorrectly. This problem was traced to the design of the original software wherein addition of a new wizard involved a specific numbering scheme. The design was modified to allow easy addition and removal of wizards. This also involved changes in the existing software for remote connectivity between two Profiles.

Some of the requirements were slightly modified since a lot of the old software was supported by typedef C structures in which addition of new fields is cumbersome. For instance, a single field was added to an already existing structure to display an extra field in the summary dialog; this had repercussions in so many places that it was decided that it was not worth the effort.

Thus, testing helped correct errors in the software and validate the requirements. A few requirements were modified to support earlier released versions of software. A few design problems with the existing software were identified and modified.

4. Closing Remarks

The primary goal of this project was to enhance existing software for a new product on the Profile media platform. User interfaces had to be designed and implemented to enable a system operator to configure new settings and parameters on the Profile.

The user interface was developed in the form of wizards using Visual C++. It was implemented as a client that used COM servers on the Profile. The COM interfaces on the server and client were developed using the ATL (ActiveX Template Library) COM. The modules developed were successfully integrated and tested with the other Windows NT libraries used on a Profile. A system operator could now configure settings on the Profile using the new wizards.

One of most challenging aspect of this work was to ensure that the software would be backward compatible (compatible on a Profile even if certain boards were not present). During the course of my work I learned that decisions made about the software design years earlier have an effect on enhancements made later and some of these have to be stuck to even if better design models are available today.

For a distributed system like the Profile, COM is the best solution since it provides an easy way to write interfaces and makes communication between objects efficient. The same implementation using standard C-interfaces remoted using a custom RPC implementation which was done before COM was used, was difficult and bug prone.

In conclusion, adding new features to existing software requires an in depth understanding of the existing design and backward compatibility should be given a lot of thought at every level of software engineering.

References

1. Don Box. *Essential COM* (The DevelopMentor Series). Addison-Wesley Pub Co. January 1998.
2. Timothy Budd. *An Introduction to Object-Oriented Programming,* 2nd Ed. Addison-Wesley Pub Co. September 1997.
3. *System Guide software release 4.0.* Profile XP PVS Series Media Platform. January 2001.
4. *www.grassvalleygroup.com*
5. *Various concept and design documents* internal to Grass Valley Group
6. Bruce Eckel. *Thinking in C++: Introduction to Standard C++, Volume One* 2nd Ed, Prentice Hall. April 2000
7. Bjarne Stroustrup. *The C++ Programming Language,* 3rd Ed. Addison-Wesley Pub Co. July 1997
8. Andrew S. Tanenbaum. *Computer Networks,*3rd Ed. Prentice Hall. January 1996

Appendix A

Source Code Listing for changes in the Configuration Manager

The class declarations for the new classes implemented for the Video Codec Wizard have been included in this section. The new COM interface implemented with the prototypes of its functions has also been outlined. In the following pages only the function prototypes have been shown since the complete listing of code will run into many pages.

Listing 1

```
// CVideoCodecPg dialog
class CVideoCodecPg : public CWizardPage
{
// Construction
public:
      CVideoCodecPg(CWnd* pParent = NULL);    // standard constructor
      virtual CString GetPageName();
      virtual BOOL Initialize( void );
      virtual BOOL VerifyInput( void );

// Dialog Data
      enum { IDD = IDD_VIDEO_CODEC_PG };
      CListBox    m_listBox;
      int         m_codecType;

// Overrides
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);

// Implementation
protected:
      virtual BOOL OnInitDialog();
      afx_msg void OnSelectAll();
      afx_msg void OnUnselectAll();
      afx_msg void OnSelchangeListBox();
      afx_msg void OnDvcpro25CodecRadio();
      afx_msg void OnDvcpro50CodecRadio();
      afx_msg void OnMpegDecoderRadio();
      afx_msg void OnMpegEncoderRadio();
      DECLARE_MESSAGE_MAP()

protected:
      void UpdateUI();

protected:
      CFlatButton m_selectAllBtn;
      CFlatButton m_unselectAllBtn;
      BOOL        m_bCodecTypeChanged;
};
```

Listing 2

```
// VideoCodecObj.h: interface for the CVideoCodecObj class.
class CVideoCodecObj : public CConfigObj
{
public:
      CVideoCodecObj(int iRsrcType, int iRsrcNum, CString strName);
      BOOL  LoadParameters ( void );
      BOOL  SaveParameters ( void );
      int   GetCodecType();
      void  SetCodecType(int codecType);
      int   GetCodecTypeName();

protected:
      int m_codecType;  // The radio button to be selected
      int m_codecTypeName; // The Resource type for each of the codecs
};
```

Listing 3

```
// VideoCodecState.h: interface for the CVideoCodecState class.
class CVideoCodecState : public CConfigState
{
public:
      CVideoCodecState();
      virtual BOOL Initialize( void );
      virtual BOOL ConstructConfigObjects ( void );
      virtual BOOL SaveConfigObjects ( void );

public:
      BOOL GetDVCPRO50Supported();
protected:
      int m_numVideoCodec ;
      BOOL m_bDVCPRO50Supported;
};
```

Listing 4

```
// CVideoCodecSummaryDlg dialog
class CVideoCodecSummaryDlg : public CSummaryDlg
{
// Construction
public:
      CVideoCodecSummaryDlg(CWnd* pParent = NULL);

// Dialog Data
      enum { IDD = IDD_SUMMARYDLG_VIDEO_CODEC };
      CScrollBar  m_scrollBar;
      CEdit       m_SummaryEditCtrl;
      CString     m_machineName;
```

```
// Overrides
protected:
      virtual void DoDataExchange(CDataExchange* pDX
protected:
      int         m_numCodecs; // To store number of actual codecs
      CString     m_strCodecName[ MAX_VIDEO_CODEC_SUMMARY_LINES ];
      CString     m_strCodecType[ MAX_VIDEO_CODEC_SUMMARY_LINES ];

// Implementation
protected:
      virtual BOOL OnInitDialog();
      afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor);
      afx_msg void OnPaint();
      afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar*
                              pScrollBar);
      DECLARE_MESSAGE_MAP()

private:
      void UpdateChannelSummary ( int startChannel, int endChannel );
};
```

Listing 5: New COM interface for the ConfigBroker Server

```
interface ICfgVideoCodecBroker : IUnknown
{
[helpstring("method GetNumVideoCodec")] HRESULT GetNumVideoCodec([out]int*
nVideoCodec);

[helpstring("method GetVideoCodecResourceType")] HRESULT
GetVideoCodecResourceType([in] int codecNum, [out] int *pResourceType);

[helpstring("method IsDVCPRO50Supported")] HRESULT IsDVCPRO50Supported([out]
BOOL *bDVCPRO50Supported);

[helpstring("method SetVideoCodecResourceType")] HRESULT
SetVideoCodecResourceType([in] int codecNum, [in] int resourceType);
};
```

Appendix B

Source Code Listing for SDTI Channel Configuration

The class declarations for the classes implemented for SDTI Channel Configuration have been included in this section. The new COM interface implemented with the prototypes of its functions has also been included. In the following pages only the function prototypes have been shown since the complete listing of code will run into many pages.

Listing 1

```
// CChannelStreamSetupDlg dialog
class CChannelStreamSetupDlg : public CConfigurationDlg
{
// Construction
public:
      CChannelStreamSetupDlg(CWnd* pParent = NULL);
      virtual void Initialize();
      virtual void UpdateResourceData();
      void SetTypeAndFormat(ChannelType type, CompressionFormat
                            format);
      void UpdatePlayoutRateRadioText(CompressionFormat format);

// Dialog Data
      enum { IDD = IDD_CHANNEL_STREAM_SETUP };
      int         m_playoutRateRadio;

// Overrides
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);

// Implementation
private:
      CChanCfgListBox         m_lbLeft;
      ChannelType             m_channelType;
      CompressionFormat       m_compressionFormat;
      double                  m_playoutRate;


protected:
      virtual BOOL OnInitDialog();
      afx_msg void OnDefaultPlayoutRateRadio();
      afx_msg void OnNonDefaultPlayoutRateRadio();
      DECLARE_MESSAGE_MAP()
};
```

Listing 2

```
// CChannelDescriptionDlg dialog

class CChannelDescriptionDlg : public CConfigurationDlg
{
public:
      CChannelDescriptionDlg(CWnd* pParent = NULL);
      virtual void Initialize();
      virtual void UpdateResourceData();
      void  SaveDescription() { OnKillfocusDescription(); }

// Dialog Data
      enum { IDD = IDD_CHANNEL_DESCRIPTION };
      CString     m_csDescription;

// Overrides
      // ClassWizard virtual function overrides
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);

// Implementation
protected:
      virtual BOOL OnInitDialog();
      afx_msg void OnKillfocusDescription();
      DECLARE_MESSAGE_MAP()
};
```

Listing 3

```
interface IChannelPlayoutRate : IUnknown
{
      HRESULT GetPlayoutRate([out] double* pRate);
      HRESULT SetPlayoutRate([in] double rate);
};
```