Software Integration of Stream-based Software Applications

Guo-Ming Fang, Wen-Gung Cheng, and Jim-Min Lin
Department of Information Engineering and Computer Science
Feng Chia University, Taiwan
jimmy@fcu.edu.tw

ABSTRACT

Developing software systems by integrating the existing applications becomes mature and practical. Our previous works has successfully achieved the goal of intercepting/redirecting the data in the form of files and characters by adopting the wrapper technique. However, there are a huge number of multimedia applications today. In order to glue the stream based software applications, we adopt building a virtual device as the proposed approach in this paper. A Linux driver example that uses the ALSA library is also given to demonstrate our approach.

1. INTRODUCTION

Software projects consisting of existing software components received increasing interest in Component-Based Software Engineering (CBSE) [1-4]. One important intention of component-based development is that software developers can rapidly develop new software systems by reusing existing software components. Recently, studies [5-11] increasingly propose approaches to build software systems from Commercial Off-The-Shelf (COTS) [12] software applications because COTS products well tested by customers give higher reliability and correctness.

There have been a huge number of commercial software applications manufactured for MS-Windows operating systems. These software applications provide diverse and high-level functionality. However, there still exists several natural problems and pitfalls raised from COTS-based software development. Most of MS-Windows applications are commercial software products distributed over the markets. The sources of are seldom available from vendors. Thus, most MS-Windows applications are regarded as black-box applications. This characteristic makes the difficulty and complexity of reusing MS-Windows applications in a new software system arise.

For integrating MS-Windows COTS applications into software systems, our previous works [13-15] have presented a component wrapping approach. It uses I/O interception/redirection technique to adapt the GUIs of a MS-Windows application into programmable interfaces through message queue and clipboard space. However, the approach can only provide to handle the input/output data in the form of files and characters. Stream-based inputs/outputs such as video/audio streams are not be able to be dealt with. Therefore, the original wrapper approach can not support the

integration of multimedia COTS applications.

An approach proposed in this paper focuses on integrating stream-based COTS multimedia software applications. In order to integrate stream-based applications, some issues must be considered.

- How to intercept and redirect the input and output stream data of applications?
- How to handle the synchronization of stream data transmission?

The input and output data for a multimedia application is video or audio stream. In order to transmit stream data between two multimedia applications, building a virtual device is adopted as the method. Moreover, there exists a critical problem of synchronization in data transmission. Some handlings on synchronization issue are presented in this paper.

The rest of this paper is organized as following. We briefly describe our previous works in section 2. A stream-based software integration approach is explained in section 3. In section 4, a case study is given as an example. Finally, this study is concluded in section 5.

2. SOFTWARE INTEGRATION THROUGH WRAPPER

The wrapping approach was proposed in previous work to achieve software integration. To integrate a Windows application into a software system, the Windows applications are encapsulated with a wrapper program. The wrapper hides the operation details of Windows applications and provides a programmable interface. Therefore, a wrapper is responsible to convert the requests into corresponding operations on encapsulated Windows applications and get the output data from the encapsulated applications.

Because most Windows applications are driven by event messages and receive input from mouse and keyboard devices, the input of a Windows application can be redirected using an event message simulation for mouse and keyboard devices. Moreover, Windows applications can exchange data through Windows Clipboard mechanism. The Wrapper also uses the mechanism to trigger the wrapped application and save the output data in the clipboard space. Fig.1 shows the interception and redirection through Wrapper.

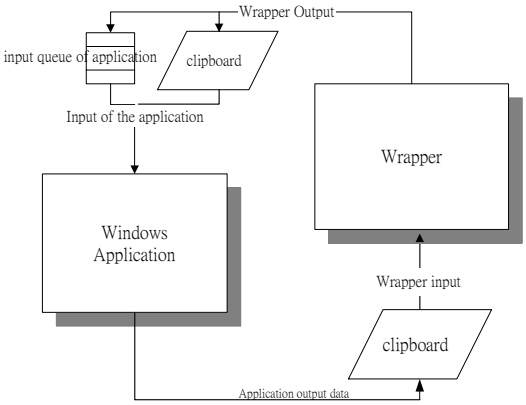

Fig.1 Interception and redirection through Wrapper

3. WRAPPER FOR STREAM -BASED APPLICATIONS

In order to intercept audio stream data from one application and redirect the data to another applications, it is necessary to understand how the driver of sound devices handle audio stream data. Under Linux 2.6, Advanced Linux Sound Architecture (ALSA) [16] is adopted to support sound driver. If a driver module is loaded, it registers the output and input function to kernel. Audio applications can use these functions to

input or output the audio stream data with standard ALSA sound driver. The handling of audio data by sound driver in Linux is shown in Fig.2.

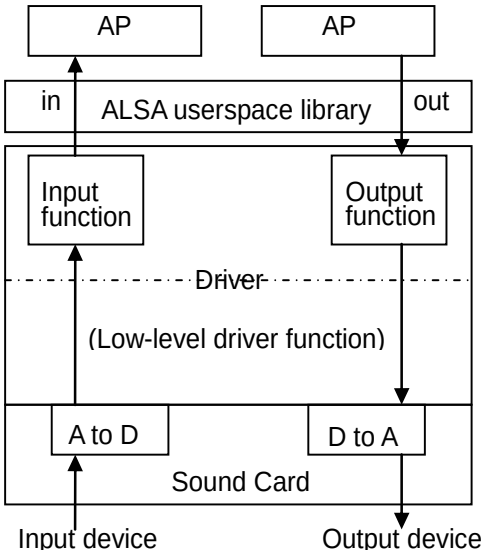

Fig.2 Working of ALSA sound driver in Linux

In the process of outputting audio stream to sound device, applications use the ALSA user-space library to outputs the audio stream data to the driver. The audio stream data is handled by the output function in the driver. Before outputting the audio data to physical sound device, the driver decodes the audio data to the analog audio signal by Digital-to-Analog conversion.

An application can get audio stream from sound device through a driver. The driver gets the audio signal from sound device and provides the audio data to applications. In order to get audio data, the application uses the ALSA user-space library to access the input function of driver. During receiving audio signal from sound device, the driver encodes the analog audio signal to the digital audio data by Analog-to-Digital conversion.

According to the concept of the working of sound driver in Linux, we design the wrapper program as a driver. The wrapper does not associate to any physical device and likes a virtual device in the system. With setting default input/output device to the wrapper, applications can output/input the audio stream data to/from other applications. Fig.3 shows the integration between two applications through the driver input and output function implemented in wrapper.

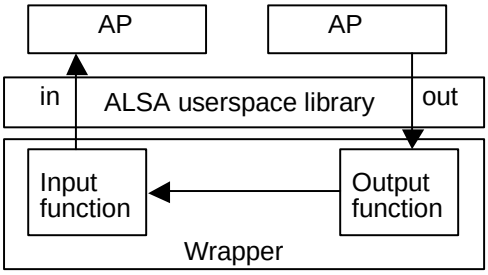

Fig.3 Integration between two audio applications through Wrapper

Different to the general sound driver, wrapper trigger the output function to transmits the audio data to the input function instead of a physical device when applications output the audio stream data. Therefore, an application can provide its output audio stream for other applications that input channel are bound to the wrapper.

In Fig.4, the structure of wrapper is presented. The input and output functions in wrapper provide the interface to input/output data to/from applications. During transmitting data between two applications, wrapper stores the audio data in a buffer space and takes care the data synchronization problem.

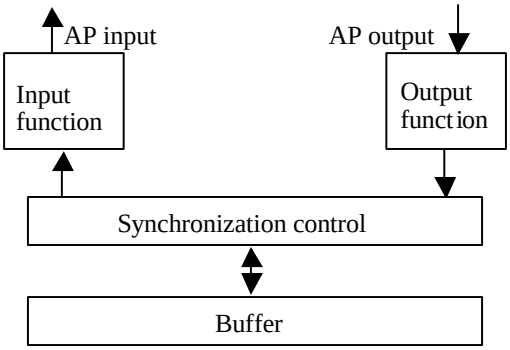

Fig.4 Structure of a wrapper

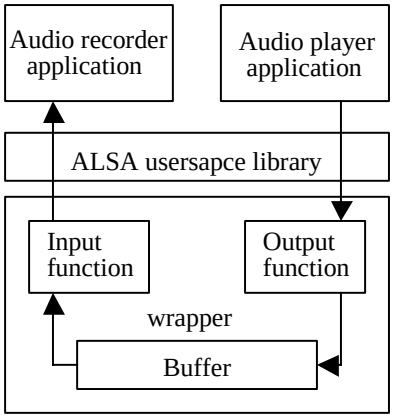

Fig.5 Integration of data from player to recorder

To handle the problem induced by the difference of data processing speed between two applications, we present some processing on the synchronization problem when buffer is full.

- *To drop data*. If applications need the real time audio stream data and do not care about its completeness, the audio data sent to the buffer is dropped.
- *To wait buffer*. If buffer is full, wrapper stop to send audio data to buffer until it is available. However, this method may delay the data transmission.
- *To enlarge buffer size*. If the buffer is set as larger as possible, the probability of full buffer may be low.

4. AN EXAMPLE

To demonstrate the feasibility of the proposed approach, we give an example of intercepting and redirecting audio stream data between two audio media application. In the example, an audio player application plays an audio file and an audio recorder application records the audio stream through the wrapper.

Fig.5 shows how a wrapper program integrates two audio applications. The audio player and recorder applications are bound to the wrapper by setting the default output and input device. The applications read/write audio data from/into the buffer in the wrapper through the input/output functions provided by the wrapper. The synchronization control in this sample wrapper is to drop audio data when buffer is full. Fig.6 to Fig.8 show the sample code in the wrapper.

```
static struct snd_rawmidi_ops
snd_wrapper_midi_output = {
    .open = snd_wrapper_midi_output_open,
    .close = snd_wrapper_midi_output_close,
    .trigger=snd_wrapper_midi_output_trigger,};

static struct snd_rawmidi_ops
snd_wrapper_midi_input = {
    .open = snd_wrapper_midi_input_open,
    .close = snd_wrapper_midi_input_close,
    .trigger= snd_wrapper_midi_input_trigger, };

static int __init alsa_card_wrapper_init(void){
snd_rawmidi_set_ops( rmidi,
    SNDRV_RAWMIDI_STREAM_OUTPUT,
    &snd_wrapper_midi_output);
    snd_rawmidi_set_ops( rmidi,
    SNDRV_RAWMIDI_STREAM_INPUT,
    &snd_mycard_wrapper_input);
}
module_init(alsa_card_wrapper_init);
```

Fig.6 Initialization of the wrapper driver

Fig.6 shows the initialization of the wrapper in the form of driver. The wrapper uses *snd_rawmidi_set_ops()* function to register *snd_wrappe_midi_output_trigger()* and *snd_ wrappe_midi_input_trigger()* as the handler of audio output and input.

```
static void snd_wrapper_midi_output_trigger
(struct snd_rawmidi_substream *substream,
int up){
   unsigned char data;
   while
      (snd_rawmidi_transmit_peek(substream
      , &data, 1) == 1) {
         wrapper_try_to_writebuffer(data);
         nd_rawmidi_transmit_ack(substream,
         1);
    }
}
```

Fig.7 Output function in the wrapper

Fig.7 shows how the *snd_wrapper_midi _output_trigger()* handles the audio data gotten from player application. In the sample codes, the substream is a pointer to application's output audio stream. The *snd_rawmidi_transmit_peek()* function is used to receive audio data from application stream. The buffer in sample code is an array of unsigned char. The *wrapper_try_ to_writebuffer()* will check the buffer and copy the audio data in the data array of unsigned char to the buffer if buffer isn't full, otherwise the data will be dropped. After the data is stored in buffer, the *snd_rawmidi_transmit_ack()* is used to remove the data from the audio stream.

```
static void snd_wrapper_midi_input_trigger
(struct snd_rawmidi_substream *substream,
int up){
   while (wrapper_buffer_empty()!=true) {
      unsigned char data;
      data = wrapper_buffer_read();
      snd_rawmidi_receive(substream, &data,
      1);     }
}
```

Fig.8 Input function in the wrapper

Fig.8 shows how the *snd_wrapper_midi _input_trigger()* provides audio data to the recorder application. In the sample codes, the *wrapper_buffer_read()* retrieves audio from buffer. The *snd_rawmidi_receive()* is used to send data to the audio application.

Some of the functions, such as the *snd_rawmidi_set_ops(), snd_rawmidi_transmit _peek(), snd_rawmidi_transmit_ack(),* and *snd_rawmidi_receive()* are supported by the ALSA library.

5. CONCLUSIONS

The gluing of stream based software applications is a critical technique in the software integration of multimedia software applications. In this paper, we have successfully achieved the goal of intercepting/redirecting the audio stream data by adopting the wrapper technique. A wrapper can be a virtual device driver and is used to simulate general sound device driver to receive audio data from applications and to transmit audio data to other applications. A simple example connecting audio data between two audio media applications demonstrates the feasibility of our proposed method. However, the synchronization issue still needs further development in the future. Using our approach, it may not be suitable to integrate real time applications (like *Skype* and so on) and an application that takes much time on data processing, such as sound encoding, because of data lag. It should be noted, video stream applications may not be integrated using the same method yet. It will be also an important future research to develop other feasible solutions to glue video-stream based

applications.

REFERENCES

[1] I. Crnkovic, ”Component-based software engineering – new challenges in software development,” *Proceedings of the 25th International Conference on Information Technology Interfaces*, 2003, pp.9-18.

[2] G.T. Heineman and W.T. Council, *Component-based software engineering: Putting the pieces together*, 2001, Addison-Wesley.

[3] P. Brereton and D. Budgen, “Component-based systems: A classification of issues,” *IEEE Computer*, 2000, pp.54-62.

[4] A.W. Brown and K.C. Wallnau, “The current state of CBSE,” *IEEE Software*, 1998, pp.37-46.

[5] S.B. Sassi, L.L. Jiani, and H.H.B. Ghezala, “Towards a COTS-based development environment,” *Proceedings of the 2006 International Conference on COTS-Based Software Systems (ICCBSS 2006)*, pp.10, 2006.

[6] G. M. Fang,, *Z. W. Hong*, J. M. Lin, ”An architecture for multi-agent COTS software integration systems,” *Proceedings of the 11th International Conference on Parallel and Distributed Systems*, Vol.2, pp.580-584, 2005.

[7] J. D. Smith and D. Hybertson, “Implementing large-scale COTS reengineering within the United States Department of Defense,” *Proceedings of the 1st International Conference on COTS-Based Software Systems (ICCBSS 2002)*, pp.245-256, 2002.

[8] V. R. Basili and B. Boehm, “COTS-based systems top 10 list,” *Digital Object Identifier*, IEEE, Vol.1, pp.91-95, 2001.

[9] D. Carney, S. A. Hissam, and D. Plakosh, “Complex COTS-based software systems: Practical steps for their maintenance,” *Journal of Software Maintenance: Research and Practice*, Vol. 12, pp.357-376, 2000.

[10] M.R. Vigder and J. Dean, “Building maintainable COTS-based systems,” *Proceedings of the 1998 International Conference on Software Maintenance*, pp.132-138, 1998.

[11] M. R. Vigder and J. Dean, “An architectural approach to building systems from COTS software components,” *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, pp.20, 1997.

[12] P. Oberndorf, “COTS and open systems,” *SEI Monographs of the Use of Commercial Software in Government Systems*, 1998.

[13] J.M. Lin, *Z.W. Hong*, G.M. Fang, H.J. Jiau, and W. C. Chu, “Reengineering Windows software applications into reusable CORBA objects,” *Journal of Information and Software Technology*, Vol. 46, 2004, pp.403-413.

[14] Z.-W. Hong, Jim-Min Lin, H. C. Jiau, G.-M. Fang, and C. W. Chiou, “Reengineering Windows-Based Software Applications into Reusable Components using Pattern Language,” *Journal of Information and Software Technology*, Vol.48, 2006, pp.619-629.

[15] J.M. Lin, *Z.W. Hong*, G.M. Fang, C.-T. Lee, “A style for integrating MS-Windows software applications to client-server systems using Java technology,” *Software practice and experience (in press).*

[16] Advanced Linux Sound Architecture, http://www.alsa-project.org/