Research on Eclipse Based Media Art Authoring Tool for the Media Artist

Songlin Piao, Jae-Ho Kwak, Whoi-Yul Kim

Research on Eclipse based Media Art Authoring Tool for the Media Artist

Songlin Piao, Jae-Ho Kwak, and Whoi-Yul Kim

Image Engineering Lab
Division of Electrical and Computer Engineering
Hanyang University
Haengdang-Dong, Seongdong-Gu, Seoul, Republic of Korea, 133-791
{slpiao,jhkwak}@vision.hanyang.ac.kr
wykim@hanyang.ac.kr
http://vision.hanyang.ac.kr

Abstract. A media art contents authoring tool based on Eclipse called Exhibition Contents Authoring System (ECAS) is presented in this paper. Visual editor of ECAS is implemented using Graphical Modeling Framework which is composed of Eclipse Modeling Framework and Graphical Editing Framework. The rest of the system were implemented using Eclipse Rich Client Platform framework. With this tool, artists could present their works easily by drag-and-dropping icons without programming skills.

Key words: Media Art, Authoring, GMF, Eclipse, RCP

1 Introduction

It is becoming easier and faster for the artists to create their works thanks to the development of computer technology. But there are still some limitations. One of them is that they still have to write programming code to some degree. However, most of professional artists do not know how to program even a single line of code. Our software Exhibition Contents Authoring System (ECAS), a media art contents authoring tool, has been developed for those artists in mind so that they could create their media art contents very easily. All they need to do is just drop and drag components of their choice, and then just connect these components in particular sequence by drawing lines between them. ECAS was developed using graphical modeling framework [1].

1.1 Related Work

There are several media art contents authoring tools available on the market in the form of commercial software or open source software. Max/MSP [2] and *Processing* [3] are two of the most widely used programs all over the world. Max

was originally written by Miller Puckette as the patch editor for the Macintosh [4]. Then it was further developed by third parties to extend its functions gradually. There is an open source version of Max named Pure Data [5]. Pure Data, also developed by Miller, is an alternative tool for the students who learn digital music. *Processing* was firstly developed by the MIT Media lab in order to help out the artists who suffered from programming code. The development began in 2001 and the stable version 1.0 was firstly released by the year 2008. There are many media art contents developed and implemented using *Processing*.

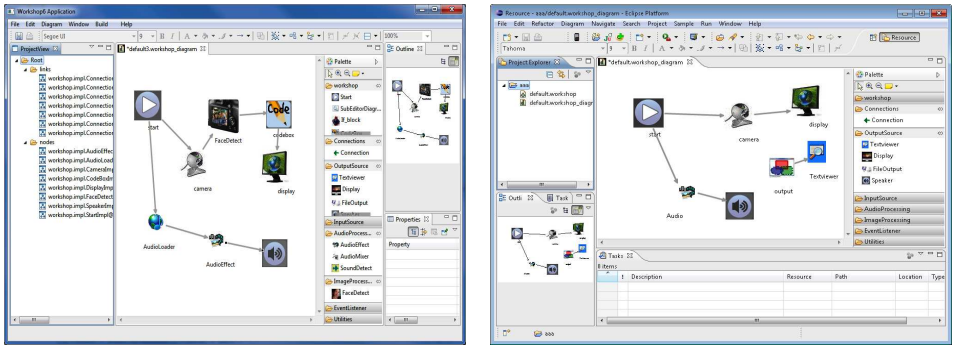
Besides these programs, there are also other media art contents authoring tools like VVVV [6], Quartz Composer [7] and Open Frameworks [8]. VVVV was designed to facilitate the handling of large media environments with physical interfaces, real-time motion graphics, audio and video that can interact with many users simultaneously [6]. Quartz Composer is a node-based visual programming language provided as part of Xcode in Mac OS X [4]. Open Frameworks is a C++ library designed to assist the creative process by providing a simple and intuitive framework for experimentation [8].

1.2 Motivation

Although the above tools are already convenient to use, there are still some obstacles that prevent media artists from easy creating their media contents using the tools. Max provides plenty of components, however, the structure is too complex and difficult for the artists to learn, not to mention the high price of commercial version. On the other hand, *Processing* is easier to learn and is free of charge, but instead of convenient graphical user interface, it still requires some programming code. VVVV has perfect rendering effect but it runs only on Windows platform.

There is a need to develop a cheap, easy to use, stable and efficient multi-platform media art authoring tool. Java and Eclipse were chosen to develop ECAS in order to get ability of multi-platform and higher extensibility. It is desirable that ECAS should not run only on the desktop PC but also run on the portable devices. It should also be easy to develop and maintain. As one of the world's most popular Integrated Development Environments, Eclipse has already provided these mechanisms. ECAS could be used as an independent desktop application or one of plug-ins [9] of Eclipse as in Fig. 1.

Although the code has been written in Java, there are several reasons that the speed of the program is as fast as in C/C++. Whatever Rich Client Platform (RCP) [10] or Graphical Modeling Framework (GMF), all the GUI frameworks from Eclipse are based on the SWT [11], which draws UI by calling system functions directly, therefore the speed grows up much faster than pure Java implementation. Just-in-time [12] compiling technology is used to compile the frequently used source code directly into the native code. While doing image processing and graphics rendering inside the software, JavaCV [13] and JOGL [14] are used. Both of these two libraries are implemented by calling native methods directly.



(a) ECAS as RCP (b) ECAS as Plugin

Fig. 1. Program appearance

The remainder of this paper is organized as follows. Section 2 discusses the whole architecture of proposed media art contents authoring tool. Subsection 2.1 introduces the procedure of designing model files using GMF. Subsection 2.2 shows the complementation procedure from two perspectives, one part is GUI and the other part is logic. The experimental result to evaluate the proposed media art authoring tool is described in Section 3. Section 4 gives the conclusion and future work.

2 Architecture

The system consists of two major parts: graphical user interface and program's logic. Usually, Graphical Editing Framework (GEF) [15] and Eclipse Modeling Framework (EMF) [16, 17] are used for designing GUI and program's logic, respectively. But Eclipse provides more convenient way to generate GUI and program's logic simultaneously. The framework used here is GMF which is the combination of GEF and EMF. That is, instead of designing each part separately, infrastructure of the whole program is firstly designed using GMF and then the generated source code would be further modified manually to be fully functional.

2.1 Designing using GMF

Domain model, graphical definition model, tooling definition model and diagram mapping model should be defined in order to develop GMF based infrastructure. The whole procedure involves following steps:

- Create a domain model, which defines the non-graphical information used and managed by ECAS
- Create a graphical definition model, which defines graphical elements to be displayed in the visual editor in ECAS

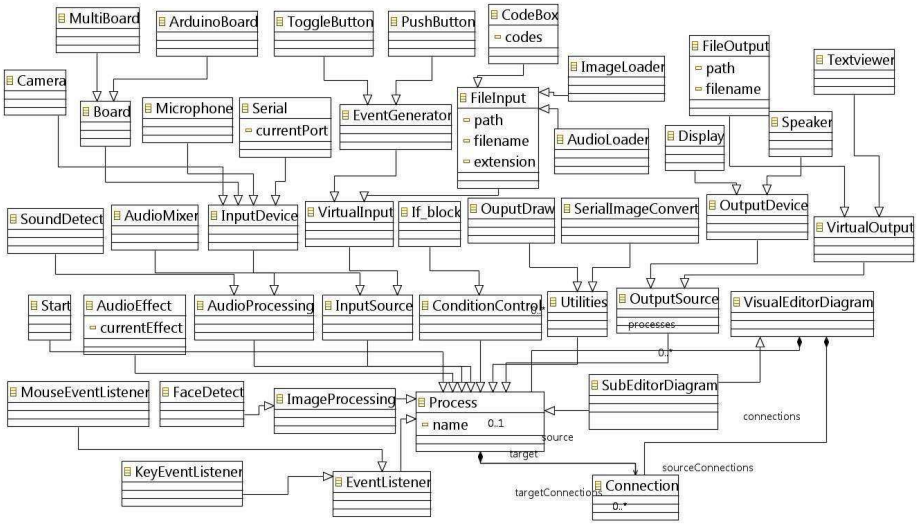


Fig. 2. ECAS function modeling

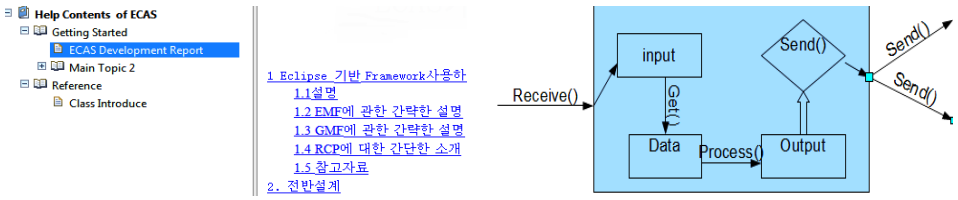
- Create a tooling definition model, which defines elements to be displayed in the palette in ECAS
- Create a diagram mapping model, which defines the mapping between domain model elements and graphical elements
- Generate the skeleton of graphical editor and enhance the generated code

First step is to create domain model, actually this is the part which is very closely related to program's logic. In the current version of ECAS, 42 classes are defined inside the domain model file (Fig. 2). The basic class here is named *Process*, because each block can be seen as a process when program is running. Seven methods are defined inside the *Process* class. They are listed below:

- void connect(*Process* target)
- void receiveMessage(Data data)
- void init() throws Exception
- void start()
- void process(Data param)
- void destroy()
- void stop() throws InterruptedException

After this domain model definition is done, a file which has the extension of *genmodel* is generated and then logic related source code is generated. The two main packages generated are workshop and workshop.impl. The definitions of the models' interfaces are located inside package workshop and the real implementation of the functionality is located inside the package workshop.impl.

Then the tooling definition model and graphical definition model should be defined. While designing tooling definition model, any number of groups could



(a) Help system (b) Process flow

Fig. 3. Help system and Process flow

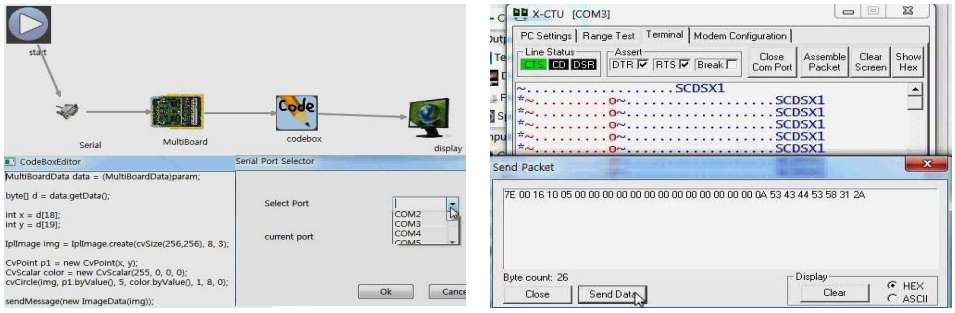
be created and each item is put inside the corresponding group. Icon image could also be changed by specifying the path of the image. In the case of graphical definition model, each node's figure should be specified inside the Figure Gallery. The default node shape is rectangle, but it could be modified to customized shape as in Fig. 1. Besides these, there are many functional options which could be implemented further. For example, whether put label inside or outside the node, whether fix the size of node or not and so on. The next step is to define diagram mapping model of these three files described above. After all these operations are finished, RCP based infrastructure was generated for the further development.

2.2 Complementation

The default generated source code is just the skeleton of the whole program. It already can run in this state, but the program cannot do real functions what artists want. Additional functions should be extended manually. As it was mentioned before, there are two main parts: one is GUI and the other is logic.

First part of program is about extensions on GUI. It is known that Eclipse provides many extension points. There are three main extensions implemented inside ECAS. The first one is *Action Set*, the second one is *View* extension and the third one is *Help* extension. Menu *Build* is added to the program by extending the *Action Sets*. There are two other submenus named as *Preview* and *Validation*. *Preview* is for running the program before doing export and *Validation* is for checking the integrity of the source file. These action sets are implemented as the subclasses of IWorkbenchActionDelegate class. We override the *run*() method of parent class. And the other extension is to extend the *View* in order to implement the project explorer. Project explorer can be seen on the left side in Fig. 1(a). Although the method used to implement project explorer is a little simple and itself can only provide limited functions, it do really very much help. In the future version, this scheme would be re-implemented using Common Navigator Framework (CNF). Help system is very important for any software. Help system inside ECAS is implemented from two side perspectives: manuals for the users and class references for the developers as in Fig. 3(a).

Second part of program is about extensions on program's logic. As it was mentioned earlier, the basic class of this Media art contents authoring tool is *Process*. It is defined as a thread. As every class used by ECAS is implemented



(a) (b)

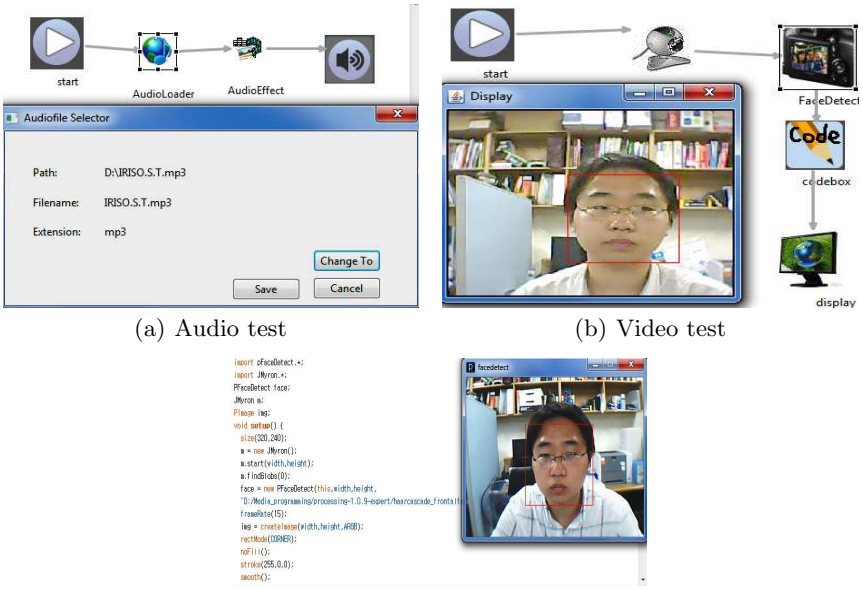
Fig. 4. Multi-sensor board function test

by inheriting thread class, it has its own run method. Fig. 3(b) shows the main concept of each *Process* class. The big box represents one node used inside the proposed tool. When the program is running, each node receives message from the previous node and stores the data to the *input* variable. Here *input* is a vector. In order to further process, the node first calls *get*() method to get the data from the *input* then uses *process*() method to do real logical processing, and then stores the result to the *output* variable. Finally, the node will call *send*() method to transfer the result to the next nodes. For example, if the node is Camera block, it receives the data from the local webcam, does some processing, and sends the result to the next node. When several nodes work together, we should provide the synchronization for these nodes.

3 Experiments

We did experiments on three aspects: external board, audio processing and video processing.

Fig. 4 shows the case for testing communication function block with external hardware, like Arduino Board [18] which is commonly used among the artists. The half above image in Fig. 4(a) shows the designed patch file and there are five blocks: start, serial, multi-board, code box, display. Start block means the start of the whole patch file; serial block is used for selecting the port which ECAS would use to communicate with multi-board; multi-board represents for the external device ECAS would communicate with; code box is designed for storing user defined code in order to deal with some problems more flexibly, in this case it will control how to display final result to the screen; display block is used for showing the result. The half nether image in Fig. 4(a) shows that artists could set or change some properties of each block through the prompt dialog box. After all the preliminary work is done, then we make the patch file running and Fig. 4(b) is the final result. Users can send and receive data through the interface presented in 4(b).



(a) Audio test (b) Video test

(c) Face detection in *Processing*

Fig. 5. Audio and Video processing function test

Fig. 5(a) shows the case for testing audio processing function block. There are four blocks in this case; start block, audio file block, audio effects block and speaker block. Start block is the same as in the previous case; audio file block is used for selecting the audio file for ECAS; audio effects block provides several kinds of effects, in this example only two are in use: *mixing* and *changing sampling rate*; speaker block is used for playing sound.

Fig. 5(b) shows the case for testing video processing function block. The test functionality in this case is face detection skill which is very commonly used in the image processing and computer vision area. The face detect function block is used for detecting faces inside one image frame. First, sequential images are grabbed from camera block then the camera block sends the data to the face detect block. The face detect block would detect human faces at each frame then send the result to the code box. Test code is set inside the code box block in order to retrieve the face information and draw it to the image buffer. Fig. 5(c) shows the same function implemented in *Processing*. Users need to write down about 50 lines code, but ECAS only uses five blocks.

4 Conclusion and future work

A media art authoring tool based on Eclipse is presented in this paper. The proposed scheme solves several problems of the existing tools. It can provide media artists more efficient and easier way of creating media so that they could

create art contents very quickly and easily. Furthermore, it also makes the procedure easier for the developers so that they can improve this tool or maintain the existing code more efficiently.

Every block used inside ECAS is implemented as a thread. Although the thread can go to sleep when there is nothing to do, it still remains inefficient compared to those using static compiling method. Currently our users can only edit and run the program. But if there is an error inside, one cannot monitor exact situation. The function that users can also debug inside ECAS should be provided in future. The program is currently developed using GMF based on the Draw2D package. Although 2D is already enough for the artists to work, the mechanism that they work on 3D based graphical editor [19] would be added in the future, too. It is much more intuitive and user-friendly.

Acknowledgement This research is supported by Ministry of Culture, Sports and Tourism (MICST) and Korea Creative Content Agency (KOCCA) in the Culture Technology (CT) Research & Development Program 2009(2010).

References

1. Graphical modeling framework. `http://www.eclipse.org/modeling/gmf`.
2. Cycling74 - tools for media. `http://www.cycling74.com`.
3. Processing - open source programming language for media art. `http://www.processing.org`.
4. Wikipedia. `http://en.wikipedia.org`.
5. Puredata community site. `http://puredata.info`.
6. A multipurpose toolkit. `http://www.vvvv.org`.
7. working with quartz composer. `http://developer.apple.com`.
8. open source c++ toolkit for creative coding. `http://www.openframeworks.cc`.
9. Eric Clayberg and Dan Rubel. *Eclipse Plug-ins*. Addison-Wesley Professional, 2008.
10. Jeff McAffer and Jean-Michel Lemieux. *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java(TM) Applications*. Addison-Wesley Professional, 2005.
11. Matthew Scarpino, Stephen Holder, Stanford Ng, and Laurent Mihalkovic. *SWT/JFace in Action: GUI Design with Eclipse 3.0 (In Action series)*. Manning Publications Co., Greenwich, CT, USA, 2004.
12. T. Suganuma, T. Ogasawara, M. Takeuchi, T. Yasue, M. Kawahito, K. Ishizaki, H. Komatsu, and T. Nakatani. Overview of the ibm java just-in-time compiler. *IBM Syst. J.*, 39(1):175–193, 2000.
13. Java interface to opencv. `http://code.google.com/p/javacv`.
14. Java binding for the opengl. `http://kenai.com/projects/jogl/pages/Home`.
15. Eclipse graphical editing framework. `http://www.eclipse.org/gef`.
16. Eclipse modeling framework. `http://www.eclipse.org/modeling/emf`.
17. David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2009.
18. Arduio home page page. `http://www.arduino.cc`.
19. Kristian Duske and Jens von Pilgrim. GEF goes 3D. Teil 3: 3D-Fizierung von GMF-Editoren. *Eclipse Magazin*, 2.10:79–82, February 2010.