iGlue.v3: An Electronics Metaphor for Multimedia Technologies Integration

Tony Cabello-Miguel Oscar Fernández-Barracel Oscar García-Panyella

Audiovisual Technologies Department. Ramon Llull University.
Quatre Camins 2, 08022 – Barcelona (Spain)
Phone: +34 932 902 420

{ antonic, is05781, oscarg } @salleURL.edu

ABSTRACT

The iGlue Project is a set of tools meant for the integration of different multimedia technologies in creative applications like multimedia installations or live audiovisual shows.

Inspired by the way of working used in electronics, similar concepts are developed in a visual environment to interactively build applications out of components, wires and circuits. This approach enhances the entire R&D workflow, providing for application prototyping, empirical experimentation, collaborative work, software recycling and simplified maintenance.

Categories and Subject Descriptors

D.2.m [**Software Engineering**]: Miscellaneous – *Rapid Prototyping, Reusable Software.*
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces (GUI).*

General Terms

Design, Experimentation, Languages, Verification.

Keywords

Electronics UI Metaphor, Multimedia Technologies Integration, Interactive Development, Collaborative Work, Application Modeling, Component System.

1. INTRODUCTION

After more than 5 years of R&D, a new software tool, called iGlue, is formally presented. It is built on a novel metaphor coming from the electronics world, and has been implemented with techniques successfully used in other software packages. iGlue provides an easy & fun-to-use GUI, along with a set of tools and libraries. This combination provides new levels of collaborative work between artists and developers.

The main motivation for building the whole system was its application for entertainment and performance, in which the interaction with real time graphics is a key problem to solve. For this reason, while designing the underlying system to be media independent, the testing and experimentation phase was focused on real time graphics. Thankfully, current graphics acceleration APIs are mature enough for this task to be done in a general environment efficiently. Once the testing phase was finished, the resulting practices were successfully used to integrate other media technologies into the system, proving the media independence of the system core.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MM'04, October 10–16, 2004, New York, New York, USA.
Copyright 2004 ACM 1-58113-893-8/04/0010...$5.00.

2. SYSTEMS PHILOSOPHY

In a world with an increasing number of technologies and devices, iGlue has been designed to easily interconnect all of them in applications like multimedia installations or audiovisual shows. The main part of iGlue is the design tool based on an electronics metaphor. Each device and media file is represented as a component on screen. These can be easily linked together with wires to build complex circuits. Circuits define rules for connecting input devices like video cameras, midi instruments, media files like images, videos, 3D scenes, and output devices like video projectors. iGlue circuits can be easily controlled by custom made interfaces that can be created with programs like Macromedia Flash [1].

For instance, a circuit can mix a video camera input with some video files and show the result on a secondary monitor, while the parameters of the mixing are controlled by a video mixer-like interface built with Macromedia Flash.

3. STATE OF THE ART

Not only has electronics engineering inspired a solution for complex software development through component technologies (COM [2], CORBA [3], etc), it has also greatly influenced the graphics interface of all kinds of applications through the use of schematics to represent complex processes (Maya [4], Combustion [5], Reaktor [6], etc).

Pd [7] is another well known instance of a general purpose component-based graphical environment, initially developed for real time music generation, and possibly for this reason too focused on programmability, which makes it difficult for artists to understand. It also lacks advanced features for developers, which can not deal with an intermediate layer between the GUI and its components.

DirectShow [8] also uses a graphical representation greatly inspired by electronics schematics, but is only meant for software developers. Touch [9] is a great example of parametric controlling of graphics effects, but is a tool only aimed at artists. Max/MSP [10] and VVVV [11] are Pd-like environments, strong on image processing and hardware device interconnection respectively but with the same limitations as Pd.

4. CONCEPTUAL DESIGN

4.1 Visual Representation

As can be seen in Figure 1, the representation of the Hello World Circuit is very close to an electronics schematic. Below is a list of all the elements that can be visually represented on the interface:

- **Components:** Blocks that resolve a specific task by operating on signals (data). *Shown as big rectangles with the component name at the top.*
  - **Pins:** Elements for allowing the communication of data with other components. *Input pins are represented as small rectangles on the left side of the component body, and output pins on the right. The pin name is shown next to the pin inside the component body.*
    - **Immediate Value:** A fixed value passed to a pin. *Represented next to the pin outside the component body.*
  - **State:** This will determine the ability of a component to operate on the signals. *Represented as a colored square in the upper left corner of the component.*
    - **On:** Loaded and operating normally. *Green.*
    - **Disabled:** Loaded but not operating on signals. *Blue.*
    - **Not Ready:** Not ready to be loaded while lacking the required signals at its input pins. *Yellow.*
    - **Failed:** The component caused an operation error and has been switched off for security reasons. *Red.*
    - **Off:** The component is neither loaded nor operating. Its output pins act as if they were not connected anywhere. *Gray.*
- **Wires:** Links between output and input pins.

4.2 Execution Model

The circuit in Figure 1 works as follows: the first component (on the left) initializes a console (the window shown on the right). Then a component prints "Hello" and appends a carriage return. The last component finally prints "World!" on the console. Wires in iGlue are conceptually used to transmit data between pins, then circuits can be seen just as data-path representations. In order to work correctly, this approach requires some rules to be followed in the component implementation:

- Data objects are managed by their creator component and the rest of the components are only allowed to read or modify their value.
- Components must not save their internal state anywhere but in the objects, because otherwise this would lead to non-repeatability of results when altering the circuit topology.

The components are cyclically executed in a frame based cadence. The exact order depends on the data dependencies imposed by the connections, which is an effect also known as cascading.

5. SYSTEMS ARCHITECTURE

5.1 Overview

The iGlue system is clearly separated into 4 different layers, each aimed at a specific user role:

- Hardware and data files: Multimedia devices (Video cameras, Sensors, etc), and artistic data (Images, Sounds, etc). Artists are in charge of handling this material.
- Component System: This layer allows the integration of external technologies in the system. Advanced developers create those components, like: ConsoleInit, ConsolePrint, ImageLoad, DisplayInit, etc.
- Core: An API which allows circuits to be created programmatically through component interconnection. To be used by advanced developers in the creation of full featured applications.
- iGlue Circuit Designer (iDesigner): A tool provided by the project, built on the Core API. It contains a graphical environment for managing and interacting with circuits. Circuits can also be fed with external data coming from Macromedia Flash applications for instance. This layer is specially aimed at novice developers and artists.

5.2 Implementation

Although the Core is completely media independent and OS-independent, currently developed components and the iDesigner tool are designed for Win32. It becomes clear that it makes it difficult to port the whole system to other platforms.

Although the user base of the Mac environment (mostly artists) is well suited for this kind of project, iGlue will still rely on the Win32 platform because it is the most cost effective for this kind of development. This is due to the availability of better hardware and software, which also means increased support and resources. Anyway, it is still possible to feed an iGlue circuit from a custom Mac application.

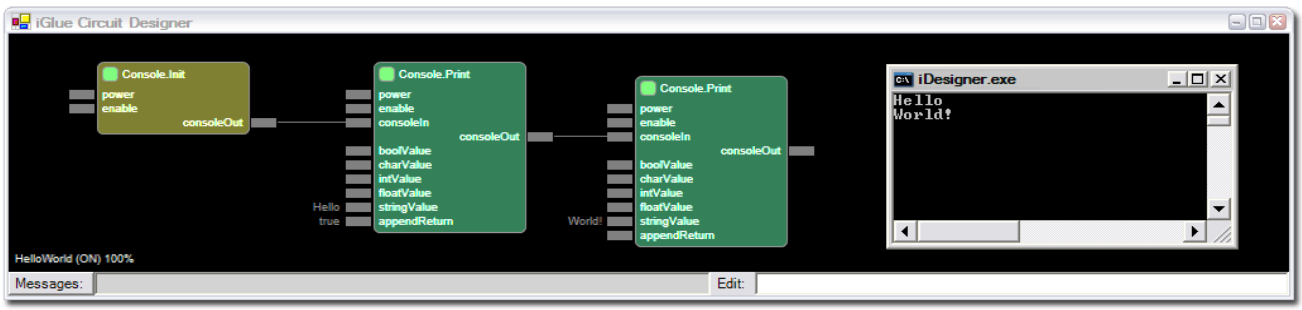

Figure 1: Hello World circuit

Both the core and the components are programmed in C++ for performance reasons. The iDesigner tool is built with Managed C++ [12] and OpenGL [13] graphics for faster GUI development.

As the global objectives of the project were very general, a Spiral lifecycle was chosen for approaching a good solution over several project versions (currently five).

The project was intended for multi-person development, so from the very beginning some conventions were chosen for programming: Java-like Subset of C++ and JavaSoft Coding Conventions [14].

5.3 Components

Components are the way for developers to extend iGlue with new functionality. They communicate with each other with typed data objects. There are five standard data-types: bool, char, int, float and string. External data types can be defined by extending the Object class and using the new type in the pin definition. It is a task of the component developer to match the object definition among all the components using each custom type.

5.3.1 Component Technology

The component technology is a proprietary system implemented the same way as most plug-in based systems (3ds max [15] for instance). Each Component has to implement a standard interface, in order to provide:

- Creating new instances of the current component.
- Getting the component type and pin-out info.
- Connecting its pins to other components.
- Switching ON/OFF, and running it.

Additionally, groups of components are packed into a single DLL (a ComponentPack) for better repository organization. The DLL exports four functions used by the container to obtain instances of each one of the components at run-time.

As this system has been specifically designed for this task, it fulfills the needs better than a general component technology like COM [2] or CORBA [3]. Advanced features like auto-description are supported through the retrieval of the pin-out definition, which allows for any kind of component to be transparently integrated into the system. Even individual pins can offer a specific range of acceptable values for user friendly parameter choosing. Additionally, component versioning is allowed through the use of versioned Component Packs. On the other hand pin-out versioning is not tracked, as it is an inherent problem of changing the pin-out definition during the development process.

5.3.2 Component Skeleton Generator

The process of generating the component skeleton from the definition (pin-out) is very repetitive and time-consuming, so it is prone to syntactic errors. Even worse, when a component definition has to be changed, there is a greater chance of different kinds of errors being introduced. Generating all of this code from the pin-out while preserving the specific code of each component from version to version is the task of the skeleton generator tool.

A Component Pack has all of its pin-out definitions in an XML [16] file, which is fed to the tool for generating the barebones implementation of each component interface. It is up to the programmer to fill out the result with the component functionality. The tool also allows pin-out definitions to be altered while keeping the old component implementations. Needless to say, this should be updated by the programmer to fit the new definition.

5.4 Circuits

In order to increase the versatility of component interconnection, pin definitions can include a combination of usage hints:

- **Optional:** It is not necessary for an optional pin to be connected for its owner component to run.
- **Isolated:** An isolated pin will not modify the signal at the output pin where it is connected, so it can share the connection with other isolated input pins.
- **Static:** A static pin resets its owner component each time the signal connected to the pin changes.

The Core takes care of the validity of the connections in a circuit, while following these rules:

- There is only one direction possible: output pins connected to input pins.
- Feedbacks are not allowed, neither directly nor indirectly.
- Pins must be equally typed in each connection.
- Only isolated outputs can be connected to more than one isolated input at a time.
- It is recommended that an immediate value is assigned to static pins rather than connecting them, because a constantly changing value will cause continuous component resets, and sometimes that can not be spotted easily.
- Connection to an output pin does not take effect until the source component is powered on. If the source component is disabled and the target pin is non-optional, the connection will still be correct. However, it is very likely that the target component will also get disabled or will even be unable to work in some cases.

As in electronics, all iGlue components automatically include “enable” and “power” pins. These allow the user to switch on/off and enable/disable any component at a given time without changing the circuit topology. Both pins are bool typed, so any component outputting bool values can be used to control them.

5.5 Core

The Core implementation is a 1:1 relationship with iGlue concepts. It defines all the required classes like Pin, Wire, Component, Board, etc. Boards are implemented as a list of Instances (both Components and Boards), sorted after each operation that changes the topology i.e. connecting a wire. The sorting process guarantees that each instance will be ordered before any other instance depending on it, with the output-only component instances at the beginning of the list. This way, the execution of static circuits is optimal, with no other overhead than the list iterator even with changing parameter values and enabling/disabling operations. Topology changing operations like connecting/disconnecting wires suffer from a small impact as they force the sorted state of the list to be validated. This approach avoids the implementation of a very complex scheduler, as each circuit can be seen as a directed acyclic graph (DAG) with an implicit execution order.

Execution of the circuits occurs in a frame-based cadence, with a maximum (configurable) speed of 100 times a second so as not to waste CPU cycles under low load conditions. This speed is enough for graphics applications, since most of the time the Vertical Sync is already limiting the speed to 50/60fps. Under this

execution model, audio components have to be programmed with a buffered approach, with parameter modifications on a frame-timed basis. Anyway, as long as iGlue is intended to be used for real time applications (> 25fps), it does not pose a big problem.

5.6 Boards

All the components, wires and immediates in a circuit are contained in a board, which is physically stored as an ASCII file in order to be readable by any text editor. The topmost board will be considered the main-board.

Sections of circuits can be selected within the graphics environment in order to create a new board. These boards will behave exactly like any other component, and since they store fully featured circuits, they are great for simplifying component pin-outs (i.e.: forcing some of their pins to have a default value) or for packing common component combinations into simpler blocks. Additionally, boards can be updated on the road by modifying their circuits and even their pin-outs.

5.7 Circuit Designer

The circuit designing tool of iGlue (iDesigner) is expected to be the primary means of interacting with the whole system, in most kinds of projects, except in custom application development. For this reason, iDesigner has been designed to offer the best clarity and usability, since it will be used most of the time.

As can be seen in Figure 1, in the lower left-hand corner of the interface there is the name of the current board and its current state. While the board is OFF, every component switched ON will show up as "not ready". This way one can stop the whole board at once, just as if it was a global power switch.

5.7.1 Graphics Design

Representation of iGlue circuits involves the same problems as representing electronics schematics. Some information is required to be on screen at design time, while other information needs to be at hand. After some testing, it was decided that some information was to be statically shown on the screen: component types, component states, pin names, wires and immediate values. On the other hand, pin types and usage hints are shown in a tool-tip when the mouse cursor is over the pin representation. This particular design requires a lot of screen space and therefore a high resolution display.

Older schematic software like Pd [7], represented similar circuits in much less space by hiding pin names. This allowed a component to be represented as a simple rectangle containing the component name and surrounded with small squares as pins. The connection wires went vertically from top to bottom, resulting in much more compact schematics. The limited amount of on-screen information of this representation noticeably affected the learning curve and the tool usability, and since modern displays are continuously growing in resolution, the alternative was adopted as an evolution for greater clarity.

Wire routing and automatic component placement was left pending for subsequent software revisions.

5.7.2 Usability

At the same time that a circuit has to be represented, it must be possible to apply a collection of commands to each of its elements.

There is a contextual menu that shows up when right-clicking on the working area. As it is contextual, it will show the available operations for the clicked object: board, component, pin. The main contextual menu, which pops up when clicking on the board background is shown in Figure 2.

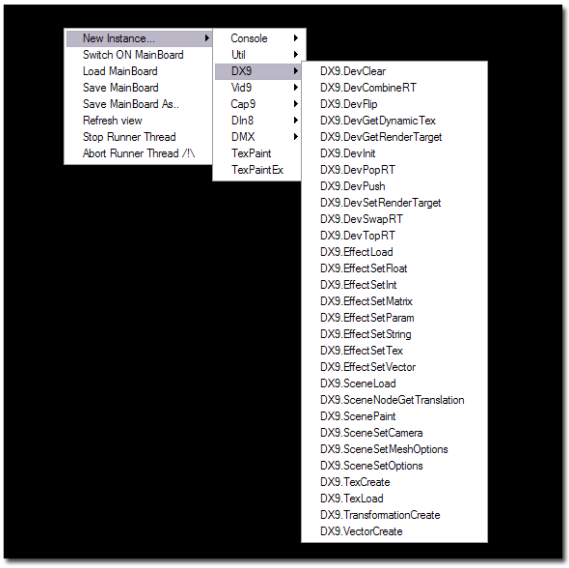

Figure 2: Main contextual menu

As a second example, Figure 3 shows the pin contextual menu. It contains just two commands: Choose Value and Reset Pin. Of specific interest is the former, as it allows for a very elegant file-choosing mechanism. As the component is asked for the list of values the clicked pin can accept, it can provide a filtered file list with all the valid strings, based on current disk contents.

Additionally, nearly all basic operations can also be done with simple mouse actions such as dragging (moving components, navigating through the circuit, creating/modifying/removing wires, altering immediate numerical values), double clicking (switching on/off components and the whole main-board, toggling immediate Boolean pins), dragging with the CONTROL key pressed (selecting/unselecting components), mouse-wheel (zooming in/out).

As seen in Figure 1, in the lower part of the interface, there are three standard controls: a button, a non-editable text-box, and an editable text-box. The non-editable text-box shows the error messages. If they do not fit in the text-box, the button can be pressed for the full message to show up in a pop-up window. The editable text-box is active when clicking on a non wired pin, allowing for immediate value creation and modification.

Another thing that greatly increased the usability of the tool was the component coloring. A color is determined for each component by applying a hash function on its type. As the number of colors is limited to be distinguishable between them, different typed components are often equally colored. But rather than giving each color a special meaning, the most important thing here is that color enables our visual memory for better orientation on a big circuit, and allows for faster navigation.

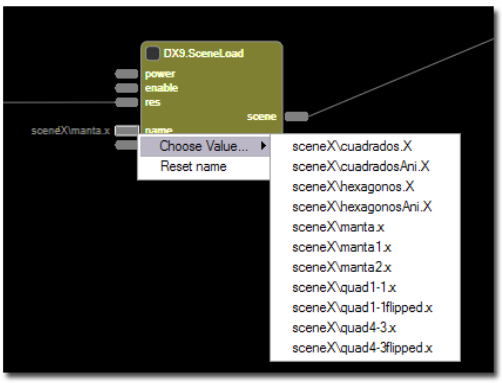

Figure 3: Pin contextual menu

Pin representation was also a critical point of the UI design. Lots of operations could be carried out at each pin (connecting, selecting, etc), so size was also important in order for them to be easily selected with the mouse. Additionally, the pin context menu has different options from the component context menu. For those reasons, pins should be represented as wide as possible, and have to be located outside the component body, just as in traditional electronics schematics software.

6. INTERACTING WITH IGLUE

Application interactivity can be done in four different ways. Components retrieving device information (like Video cameras, Sensors, etc) can directly drive other components. iDesigner also provides some degree of interactivity, although it is only intended for experimentation and testing. When more complex control-logic is needed for driving the circuit, there are two different possibilities: Static Circuit Interaction and Dynamic Circuit Interaction.

6.1 Static Circuit Interaction

In the "Util" Component Pack, there is a set of components meant for receiving data over the network (Util.ReceiveFloat, Util.ReceiveInt, etc). These components have an input pin for defining the receiving port, and an output pin for supplying the received data to other components. Then any external application will be able to send data to this port over the network. Additionally, the counterpart components are provided in order to send data from the circuit to an external IP:port.

It is worth noting that this feature is not a part of the Core nor the iDesigner, as it has been added through an iGlue Component. Additional features like encrypted communications can be integrated through other components the same way.

One interesting application built on this interaction type is a special Macromedia Flash player capable of sending variable values to specific IP:ports. This allows rich graphical interfaces to be created, which can feed parameters to all kind of iGlue circuits remotely. The downside is that circuit topology can only be altered in a limited way, by enabling/disabling components or even changing filename strings. If more control is required, it is possible to use the dynamic circuit interaction feature.

6.2 Dynamic Circuit Interaction

A feature not even contemplated in schematic software like Pd [7], is to offer a Circuit API in order to build custom applications directly over the Core and the Components. This way, applications can build completely different GUI metaphors using dynamic circuits under the hood, by connecting and disconnecting components depending on user actions.

This approach has two key benefits for application developers:

- An application can be built out of existing components, centering efforts on the UI development.
- As already happens with plug-in based applications, they can be updated and upgraded externally without recompilation. With iGlue, it is even possible to upgrade an application to handle new kinds of media through new components, thanks to the media independency the Core provides.

Windows applications can be built on the Circuit API by just linking the Core library statically. Additionally, any .Net application (such as iDesigner) can also do that thanks to the iCore wrapper class.

7. APPLICATION CASE STUDY

For better understanding of the whole iGlue architecture, a real application case will be examined step by step.

7.1 Designing the DX9 Component Pack

The most important part of a component pack is the definition of the component pin-outs, since this will effectively determine the potential of the whole component pack. A simple approach in each component definition will result in very complex circuits, while massively featured component pin-outs would not be flexible enough. It is like designing a "Lego" block game: the exact balance of simplicity and features per block is the key to building any kind of construction easily. Finding this tradeoff involves testing each design in real world applications. The final component definition is the result of a detailed study of common graphics techniques and an extensive process of trial and error.

There are two different factors involved in component pin-out design: operations and data-types. In the "Lego" example, the former could be seen as the block shapes while the data-types would be the connectors shape. For this reason, it is good to have as few connector shapes as possible in a block set, in order to maximize the number of blocks that can be connected together.

7.2 Implementation of the DX9 Components

DirectX [8] was finally chosen for the graphics engine because of its integrated full featured gaming library and its compatibility with a wide range of graphics acceleration cards. These two aspects reduce the development time in big application development, especially on small programming teams.

The core of the graphics engine has been designed as a common scene-tree with meshes, cameras, animations and materials stored in its leaves. Most of the components have been designed as an interface for controlling the scene graph, such as altering animations, changing parameters, etc. The material definition is done with HLSL (High Level Shader Language) [8] which allows control of the programmable graphics pipeline.

7.3 Using the DX9 Component Pack

Figure 4 shows a basic DX9 circuit, which simply loads and displays a Texture (Image). The component DX9.DevInit initializes the DirectX system while creating the window.

DX9.TexLoad loads the specified texture from the disk. DX9.DevClear clears the frame-buffer, then TexPaint draws the texture on the frame-buffer. Finally the DX9.DevFlip component is arranged to show the color buffer on the window.

TexPaint is a board which encapsulates the five component circuit which performs the task of drawing a textured square on screen. It provides some optional pins for user supplied transformation matrix and color to be applied to the square. Finally, the effect pin allows us to select the shader that will be applied when drawing the texture.

Here the "texture" data-type serves to generalize very different graphics media: still images, videos, live video input and computer generated graphics (3D scenes). It lends excellent flexibility for interconnecting components and is a perfect mapping to the graphics acceleration hardware.

7.4 Interactive Prototyping

Instead of painting the texture onto a square, it is also quite easy to apply it to any kind of polygonal mesh, even animated. All these possibilities allow for the creation of all sorts of real time graphics effects. Some of them have already been implemented with circuits: spatial blurs, temporal blurs, fur, glow, deformations, etc. Needles to say every parameter of these effects is available on the circuit, so it can be changed or controlled in real time.

Therefore using this Component Pack and thanks to the graphics interface, all kinds of new graphics effects can be interactively prototyped, instead of writing code and recompiling the application to see the results. Additionally, these instant results can lead to new approaches during the creation process. Although artists can build some simple DX9 circuits on their own, developing most graphics effects out of components still requires a lot of technical knowledge, then it is a way of enhancing the development time of experienced programmers.

7.5 Interactive Experimentation

Despite the complexity of creating graphics effects with the DX9 Component Pack, artists can still use the Circuit Designer tool to combine circuits and tweak configurations. As the interface provides interactive results, the artist can experiment by plugging devices and media on his own. This process can bring new ideas, which programmers can use to develop new components or enhance the existing ones. This form of collaborative work is due to the new visual language that brings artists and developers together.

7.6 Parametrization

Once a circuit is built in iDesigner, it can be fed externally, as seen in section 6.1. In fact, a special application has been developed for artists using Macromedia Flash [1] to control circuits from Action Script. The application just sends variable values to the specified IP:port, allowing the artists to build their own graphics interfaces (out of Flash components) to parameterize a circuit as they like.

For instance, a media player-like interface can be easily designed in Flash, to control a simple video-playing circuit. Then adding

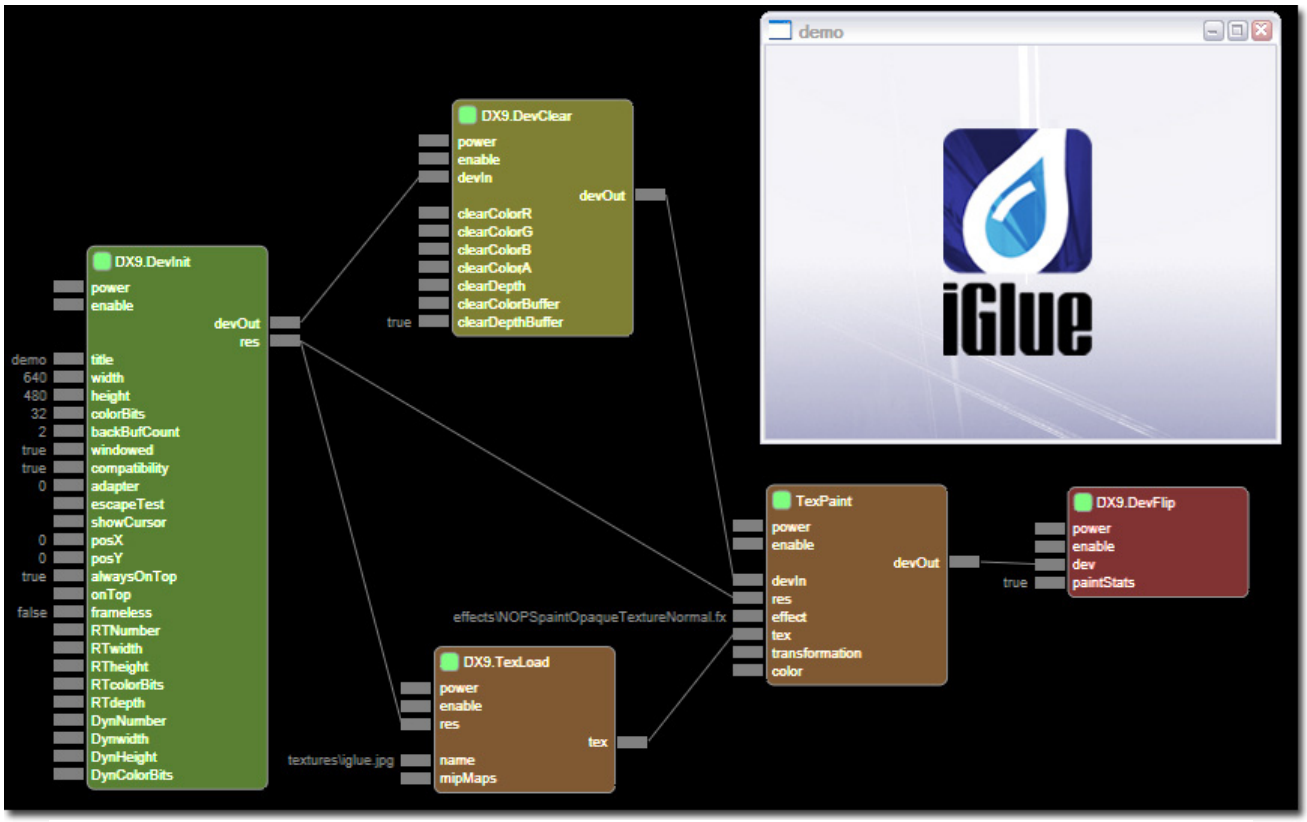

Figure 4: Texture Paint Circuit

new components to the circuit will allow the interface to control additional parameters as colour, brightness, etc. Another possibility is inserting another TexPaint component in the circuit in order to have an extra layer. The interface will be able to control the mixing through the opacity of each layer, as in the interface shown in Figure 5.

Once built, it is possible to upgrade this application in order to deal with new video formats or new video effects just by upgrading the components. Another possibility that does not involve changing the interface is dealing with new technologies. This is possible just by modifying the circuit and applying the parameter values to other components, such as audio or lighting machinery.

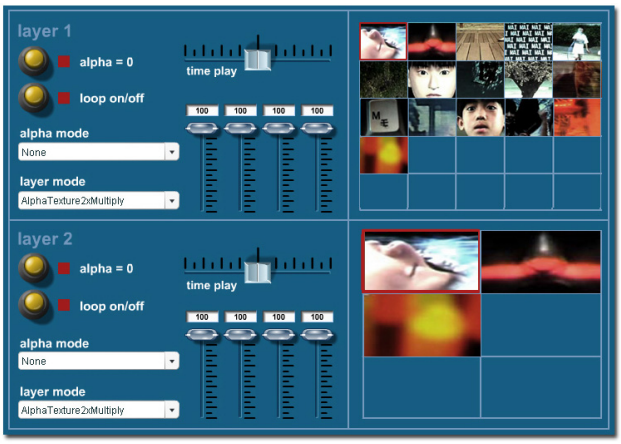

Figure 5: A Video Mixing interface built with Macromedia Flash

Input devices like joysticks, MIDI [17] controllers and the like, will be easily integrated into the Flash interface, as our Macromedia Flash player can also receive values from external circuits.

7.7 Advanced applications

A step beyond the interface built with Macromedia Flash is a custom application using the technique seen in section 6.2. This will be needed in case of a variable number of video layers or effects applied to each one. On such an interface, the circuit will be dynamically built depending on user actions in order to achieve the expected results.

This application will still benefit from all the features seen in the previous section, as Components and Circuits can be updated without recompiling the application. It is even possible, in a video sequencing-like application built this way, to change its internal graphics engine from DirectX to OpenGL without recompiling a single line of the application.

8. RESULTS

"The Hit VR" [18], was developed with the iGlue.v2 system (which still did not include a graphics interface), as well as old OpenGL components. However, it is still a great example of the iGlue concept in action, as it saved an important amount of development time due to the reutilization of previously developed components. Those components implemented a 3D engine which was used to fly around the 3D scene modeled by the artist. "The Hit VR" is the virtual reality version of "The Hit" [19], a short film by Jordi Moragues, an award-winner at Imagina'94. Through the use of a VR headset with an integrated tracker, the camera orientation changes accordingly to immerse the user into the short film. Various brands of VR headsets have been seamlessly integrated into the application, just by programming new components. Figure 6 shows the whole system running, along with some screenshots.

iGlue.v3 has been successfully used in various real applications, most of them based on the DX9 Component Pack. As a first test, a circuit simulating a Real Time Volumetric Light Effect was created out of standard DX9 components. Then a wireless joystick was used to control the light intensity and direction as can be seen in Figure 7.

Figure 6: The Hit VR through the VR headset along with 2 screenshots of the application

Figure 7: Top: Volumetric Light Effect, VideoWall application. Bottom: Videojockeying application on stage.

Next, a Video Wall application was built out of a simple video playing circuit. Four computers running iGlue were used to form an image controlled from a single Flash application. The entire application was developed by a Flash programmer who was only provided with the basic video playing circuit. The complete installation is shown in Figure 7.

Finally, a videojockeying application using the dynamic circuit interaction approach was built. A graphics interface was designed to meet specific artistic requirements: lots of video layers, music synchronized sequencing and tangible controllers. The GUI was used to arrange the clips and effects on a timeline while the tangible controllers were used to change the effects parameters interactively. This allowed the performer to play video compositions on the rhythm of the music. The MIDI controller and the application on stage is shown in Figure 7.

iGlue.v3 is available for download at [20], as it has been released as a Freeware tool.

9. CONCLUSIONS AND FUTURE WORK

Through empirical testing, the current version of iGlue has been proven to offer the following benefits:

- The component architecture allows better software development and maintenance, and provides for a natural effort reutilization.
- The true media independence of the core allows the integration of new technologies transparently into iGlue applications, as long as they only rely on components and circuits. Independence from the visual environment allows any kind of custom software to be developed, through the use of dynamic circuits.
- The visual tool offers a clear and productive interface to work and interact with circuits. It results in an enhancement of the workflow between artists and developers as it serves for rapid prototyping, effects modeling and experimentation. Circuits in the visual environment can be parametrically controlled or sequenced externally for simplified development.
- There are a large number of technologies successfully integrated into iGlue: State-of-the-art 3D graphics through DirectX9, Keyboard, Mouse and Joystick through DirectInput8, TCP/IP intercommunication, MIDI music control, Video Capturing through DirectShow, and Macromedia Flash. Apart from the discontinued OpenGL and VR headset components which were available in previous versions.

There is still a long road ahead as there are many areas which can be further polished:

- The core needs to be optimized for dealing with really huge circuits, as working with boards increases the circuit complexity exponentially. Additionally, a feature worth being directly supported at this level is remote circuit management.
- Elements should be even better organized and displayed on screen. More control elements should be provided on the GUI for better circuit interaction.
- Some key multimedia technologies remain to be fully added to the iGlue repository, such as DirectPlay (for enhanced networking features) and audio processing.

10. ACKNOWLEDGMENTS

The following people have contributed in different ways to this project, from discussing techniques to testing or giving invaluable advice: Alvaro Uña, Joan Coll, Jorge Cabezas, Jordi Carlos, Jordi Cabeza, Mikko E. Mononen, David Notario, Alex Evans, Rosa Ros, Francisco González, David Domingo, Ricardo Cabello, Alberto Garcia, Victor Jurado, Josep Garrido, Javier Campos and all the people from the Audiovisual Technologies Department at La Salle.

This project could not have been carried out without unconditional support from our family and close friends. We extend our sincere gratitude to them.

11. REFERENCES

[1] Macromedia, “Flash”, http://www.macromedia.com/software/flash

[2] Microsoft Corporation, “COM: Component Object Model”, http://www.microsoft.com/com

[3] Object Management Group, “CORBA: Common Object Request Broker Architecture”, http://www.omg.org

[4] Alias Systems Corp., “Maya”, http://www.alias.com/eng/products-services/maya/

[5] Discreet, “Combustion”, http://www4.discreet.com/combustion/

[6] Native Instruments, “Reaktor”, http://www.native-instruments.com/index.php?reaktor4_us

[7] Miller Puckette, “Pd real-time music and multimedia environment”, http://www-crca.ucsd.edu/~msp/software.html

[8] Microsoft Corporation, “DirectX”, http://www.microsoft.com/directx

[9] Derivative Inc., “Touch”, http://www.derivativeinc.com

[10] Cycling '74, “Max/MSP”, http://www.cycling74.com

[11] Meso, “VVVV”, http://vvvv.meso.net

[12] Microsoft Corporation, “.Net Platform”, http://www.microsoft.com/net

[13] Silicon Graphics Inc, “OpenGL”, http://www.opengl.org

[14] Sun Microsystems, “JavaSoft code conventions”, http://java.sun.com/docs/codeconv

[15] Autodesk Inc, “3D Studio Max”, http://www.discreet.com/3dsmax

[16] W3C, “XML: Extensible Markup Language”, http://www.xml.com

[17] Midi Manufacturers Association Inc, “MIDI: Musical Instrument Digital Interface”, http://www.midi.org

[18] Oscar Fernandez & Tony Cabello, “THE HIT VR”, http://www.salleurl.edu/~is05678/iglue/thehitvr

[19] Jordi Moragues, “THE HIT”, http://www.iua.upf.es/~jordi/prods/hit.html

[20] Tony Cabello, “iGlue Website”, http://www.iglue.org