JAMOMA MODULAR: A C++ LIBRARY FOR THE DEVELOPMENT OF MODULAR APPLICATIONS FOR CREATION

Théo De La Hogue  Julien Rabin  Laurent Garnier
GMEA / Jamoma  GMEA / Jamoma  Galamus
theod@gmea.net  julien.rabin@jamoma.org  lgarnier@galamus-software.fr

Disclaimer: this is an automatic translation of the paper submitted at the Journées d’Informatique Musicales conference. Please refers to the original french version available at the following url or contact the authors for further information.


ABSTRACT

Among the different libraries constituting the platform Jamoma, Jamoma Modular presented in this paper offers a set of solutions for the representation, observation and exploration of application functionality in the form of a tree, for exchange data locally or within a network or to the manipulation of presets, automation or mappings. Written in C++, Jamoma Modular is now used for making various applications whose implementation in the form of externals and patches for Max/MSP is a leading example.

1. INTRODUCTION

In the context of computer development for music creation, those aimed at the creation of tools for live performance can be distinguished in a singular way: every art project or musical is indeed a set of requirements leading to a specification of its own. It is then up to the person responsible for development to implement a single software environment meeting the specific needs of this project.

However, the observation of practices associated with these computing devices as well as the study of devices even allow, in terms of software developments, to qualify this assertion. As part of this article, we will identify two facts underlying the work presented here.

First, the design of a software dedicated to an artistic or musical project can still often rely on a set of general and recurring features: the backup and playback of memories (presets, cues) that describe the state of the computing device at a time \( t \), the mapping of values from a capture system parameters of a synthesis module for example, are such generic features that the developer can use.

Secondly, the computing environment used for the execution of the work may result from the communication of specialized applications: digital signal processing (sampler virtual synthesis engine in real time, etc.), motion capture tracking, (video gesture follower, etc.) video processing (3D computer graphics, etc.) for example. In addition, the networking of various applications get more complex when it is opened up and dynamically. Therefore, a set of generic tools and meet criteria of modularity can be very useful to the person in charge of developing the computing device.

The Jamoma project now offers to application developers a set of multimedia libraries C++ (frameworks) dedicated for example to signal generation and processing, to achieve synchronous and asynchronous graphs. One of them, Modular Jamoma whose complete overhaul performed GMEA - Centre National de Création Musicale d'Albi-Tarn marks the transition to version 0.6, offers a range of solutions for the representation of application functionality in the form of a tree and its manipulation locally or remotely, or for the implementation of generic services such as management of presets or mappings. If there are now several dedicated libraries to meet one of these aspects, however Jamoma Modular offers a development framework integrating these different features into a unified and low level architecture.

After a short presentation of the Jamoma project and its specific objectives, this article will present the library Jamoma Modular and the solutions it can offer to developers. Finally, two projects implementing some of its features will be given as examples.

2. A PLATFORM FOR RESEARCH AND CREATION

Since 2005, the project Jamoma is defined as « a platform for interactive art-based research and performance ». International project and open source (released under BSD license 1), it is being developed by Timothy Place (Cycling 74, United States), Trond Lossius (BEK, Norway), Nils Peters (CNMAT, USA), Alexander Refsum Jensenius (University of Oslo, Norway), Pascal Baltazar (Composer, France), Theo De La Hogue

1 http://opensource.org/licenses/bsd-license
(GMEA, France) and Julien Rabin (GMEA, France) as well as by many users to contribute. It is also supported by various organizations such as 74 Objects society, BEK, didascalie.net or the GMEA.

In particular through its implementation in Max/MSP, Jamoma is now used in many musical or artistic projects such as those conducted at GMEA as well as at other creative centers supporting their development and also sees a growing user community.

3. MODULAR ENVIRONNEMENT OVERVIEW

3.1. General Architecture of Jamoma

Originally envisioned as a set of recommendations for the development of standardized modules (Max/MSP patches) and their implementation, Jamoma today covers several cross-platform C++ libraries used especially for developing multimedia applications.

The whole project is organized around an architecture layered according to its different objectives:

- Foundation Layer is a library providing a set of classes dedicated to the dynamic construction of reflexive objects;
- DSP layer is a specialization of some of these classes, optimized for digital signal processing;
- The libraries Graph and Audio Graph are respectively devoted to the creation of topographies for asynchronous and synchronous communications between different objects;
- Finally, Graphics layer is dedicated to the achievement and rendering graphical user interfaces.

Jamoma Modular layer detailed in the following sections is based on the Foundation layer set forth above. It aims to initially provide infrastructure for the development of software tools in programming environments like Max/MSP, Pure Data or wider stand-alone applications.

3.2. Specifications

Since its inception, the project aims to provide Jamoma beyond issues strictly computer solutions based first and foremost on the needs from the musical practices and, more widely, the practices of artistic creation in general. The user feedback from the practice of Jamoma through rich and varied projects have been able to establish the specification for version 0.6 in this article.

Moreover, the study of creative practices conducted by Alessio Santini, Anne Sedes and Benoît Simon in the ANR project Virage in which Jamoma was one of the experimentation supports has revealed many challenges for software development. In the article published at the 14th Journées d’Informatique Musicale [1], Virage project members wrote:

«The main finding of this uses study confirmed the need to work on the concepts of interoperability and modular, targeting the combination of our developments with the software environment and existing materials and corresponding habits of practitioners rather than the integration of this complex and multifaceted reality in a single software illusory and certainly too versatile to be viable.»

These observations and subsequent developments have greatly contributed to the development of specifications for Jamoma Modular 0.6. This included in particular the objectives of providing:

- A «Model – View – Controller» oriented architecture[9];
- The tree representation of the services of an application;
- Generic functions of presets, automation, mappings;
- Exposure to any type of network.

Different libraries currently offer dedicated or some of these partial mapping capabilities such as Libmapper developed by the laboratory IDML of McGill University or tree representation of features such as those proposed in the Project Integra. However, Jamoma Modular aims to integrate these solutions within a unified environment. The work carried out for

---

2 Une liste des soutiens passés et actuels est consultable en ligne, à l’adresse http://jamoma.org/support.html
3 http://74objects.com/
4 http://bek.no
5 http://didascalie.net
6 http://www.gmea.net
7 The full source code Jamoma is maintained with the Git version control and is publicly available online at https://github.com/jamoma
8 http://www.plateforme-virage.org
9 Architecture "Model - View - Controller" to separate the operational heart of an application for presentation to the user.
10 http://www.idmil.org/software/libmapper
11 http://www.integralive.org/
over a year and have led to successively implement these different points in the version presented today.

4. JAMOMA FOUNDATION

Most of the features developed in Jamoma Modular relies on the properties of the layer Foundation. The latter, which relies on the entire project Jamoma, including sets the foundation for object-oriented programming environment and a bookshop (NodeLib) for representation in tree form.

4.1. The programming environment

Jamoma Foundation adopts a particular style of object-oriented programming based on a set of classes (class-based programming) that define the basic elements (environment, class, object, attribute, message, value) necessary for the programmer to dynamically de-finish objects, instantiate and use a generic.

The environment manages objects using the class named TTObject. Reflexive in nature, instances of TTObject are able to describe themselves by listing their attributes and their methods. The class also provides a mechanism for observing attributes and methods, so to be notified of a change or appeal. The various objects Jamoma Modular inherit this class and generalize the discovery of properties and services of an application.

4.2. Tree representation

The namespace of an application is a way to organize and access its services via strings (and not with the memory addresses that manages the computer). Features of a modular application that can be prioritized according to the different modules that constitute the organization of namespaces is effected in the form of a tree structure. This extends from the root (the application itself) until the leaves (the objects that constitute it).

Between the root and leaves are one or more branches representing the organization of different features of the application. These may correspond to different sub-parts of this may decide. The addresses are strings representing the path from the root to the designated object. Each level is separated by a «/ » (except the root, which was named «/ »)\(^{12}\).

4.2.1. Construction

The library NodeLib included in the Foundation layer provides a class (TTNodeDirectory) allow both to build a parallel tree and table. The tree consists of nodes (TTNode) knowing their children, their parents and class TTNodeDirectory that reference. The table contains about her all the possible addresses in the tree, allowing direct and rapid access to nodes.

![Figure 3. TTNodeDirectory](image)

Each node contains two essential information: the name and instance. The notion of instance allows an application to manage duplicates of nodes having the same name. The instances are specified when creating the node, or generated automatically, if necessary. Nodes reference their children through a table of names where each entry references an instance table (each entry references a child). A node can have multiple children with the same name. Their differentiation is then carried through their body. This concept introduces the character of proceedings «. » as part of syntax in addresses to separate the node name of its instance.

![Figure 2. Tree of a modular application](image)

![Figure 4. Instances of a node in a tree](image)

\(^{12}\) The choice of the address syntax is inspired by the OSC syntax to facilitate network messages routing.
A node in the tree can reference any type of object (TTObject). Reflexivity of the latter makes it possible to access an attribute or calling a method with the character « : » in addressing. This syntax opens the possibility to easily implement an application-level query system [8]:

```
/filtre/gain:reset
```

Figure 5. Sample Query (reset to initial state)

```
/piste:*/filtre:*/gain
```

Figure 7. A sample query

The query above target and all nodes « gain » present in all instances of « filtre », which is located in all instances of « piste » (track).

4.2.2. Exploration

The development and use of a sophisticated application invoke a large number of services that can lead to large and complex tree. For this, the NodeLib provides mechanisms for exploration and research in the database that is such a tree. It is possible to search in a given address all nodes vali-ing test (lookfor) or whether at least one node validates (IsThere). Exploration can also return-ing the nodes corresponding to an object type or those whose objects have some Attributes of equal or different value. Furthermore, the use of character « * » in an address returns all the nodes designated by name, independently of their body13.

```
/*filtre*/
```

4.2.3. Observation

The declaration of a service application can be dynamic and therefore performed after the creation of customers. Therefore, the observation of the creation or destruction of nodes in the namespace is needed. An observation is from an address and is a mechanism of notification issued by the TTNodeDirectory after adding or deleting a node before under noted address. The notification transmits this node to the observer so that it can filter on the type or attribute values of object referenced. Again, the observation is generalizable to multiple addresses by using the character « * ».

5. JAMOMA MODULAR

An application offers the user to manipulate its functionality through data that can be commands, parameters or feedback[8]. The mechanisms for graphical display, presets, automation or mappings are many customers using or subscribing to services data and / or questioning their attributes. This design provides a simple way to implement an architecture «Model View Controller» within an application, thus separating the developments on the operational heart of the application (model) those relating to the graphical interface (the view), the interface between its two sides (the controller) will then be provided by Jamoma Modular.

5.1. Data declaration

Reporting of data by the application causes the instantiation of a specific class : TTData. His main attribute is its value in the first place, but many other attributes also specify the framework for use according to its role in the application. These attributes can be for example the type of data handled (integer, decimal, boolean, string, generic, none) or service which distinguishes three roles that can support a given application: parameter, message or return. Other attributes are also managed:

- default value;
- filtering répétitions;
- tag;
- priority;
- limits of possible values;
- behaviour around the limits;
- incrementation step size;
- description.

In addition, Jamoma Foundation provides for advanced use many services also accessible through attributes: temporal interpolation between two values (library RampLib [8]), definition of the type of unit value (Position, Gain, Height, Color, Temperature, etc...) and the unit in which it is expressed (eg size Position: Cartesian3D, Cartesian2D, Spherical, Polar, OpenGL, Cylindrical), conversions from one unit to another (library DataspaceLib [8]).

---

13 As defined in OSC 1.0 specifications, the symbol *** in an OSC address matches any occurrence of zero or more characters. (http://opensoundcontrol.org/spec-1.0)
5.2. Preset, automation et mapping

With his involvement in the practice of performing arts, Jamoma Modular design provides several elements for storage mechanisms (presets, cues) or linking (mapping) adapted to the specific application. TTReset class is dedicated to the memory of the state of part of the namespace. The memory management is broken down into four operations that the application must provide to the class: the filtering of objects involved, the choice of attributes stored, sorting and backup reminder.

The class includes several classes TTcue, TTReset to store the state of parts of the namespace and prioritize the recall. Classes and TTResetManager, TTcueManager manage for their respective handling of several classes and TTReset, TTcue for scheduling

TTMapper class provides a way to relate the value of a given input to the value of another output data. This mapping is effected by declaring the class TTMapper addresses of origin and destination. For example, a scaling by default is given by querying the values of each data terminals. It is also possible to specify and configure each of the transfer functions contained in the library FunctionLib (linear, power, cosine, etc.) accessible from the Foundation layer.

5.3. Discovery and exposure on the network

To connect with other environmental events (software or hardware) via various protocols (MIDI, OSC, DMX ...) or expose its application on the network to offer services, Jamoma Modular integrated a library originally developed in collaboration with LaBRI, the GMEA and Blue Yeti in the context of the Virus platform: the DeviceManager.

The system now renamed ApplicationManager is designed to remain compatible with new technologies in managing these protocols in the form of plugins. Thus each new communication protocol can be made compatible by implementing a plugin interface. It considers that any protocol can be reduced to no more than four types of operation: listen, ask, send and explore. To date, three plugins have been developed during the project: the protocols OSC, Minuit and CopperLan.

TTApplicationManager class manages the declaration and communication with your applications-

5.4. Storage and documentation generation

Reflectivity of the objects presented above gives the possibility to automatically generate backup files or the standard documentation in various formats (HTML, LaTeX). The mechanism is based on a system of classes that support writing or reading files (handler).

In writing, after creating the file on disk, class system takes care of calling the WriteAs method of the object it describes (and format it supports it). The subject then wrote his description with the values of its attributes.

Reading, after opening the file since said that this same mechanism calls the ReadFrom method of the object it describes (and the format it supports it). The object then proceeded to read its description to change the values of its attributes.

An object can recursively forward the mechanism of reading / writing of any objects that it manages, allowing for example the class TTResetManager to describe these operations by sending the set of classes that constitutes TTReset.

If at that date, only the TTXmlHandler is used, it will soon be supplemented by a class TTTextHandler or other formats.

6. AN EXAMPLE OF IMPLEMENTATION IN MAX / MSP

The environment Jamoma within Max/MSP aims at providing to beginners and experienced developers a set of modules dedicated to the generation or processing of audio or video content, modules implementing a variety of spatial techniques, tracking, video etc... It also provides the infrastructure needed to build personal standard modules. Unlike other personal or collective libraries to improve or facilitate the development (TapeMovie, ou plus récemment Vizzie) designed from the standard functions of Max/MSP, the implementation of Jamoma in Max / MSP is based on a library of specific externals. These exploit the features of different framework to standardize the exchange of data and applications, systematic separation of engine and its graphic representation in architecture "Model View Controller" in Max / MSP.

20 http://tapemovie.org/
21 http://cycling74.com/2010/11/19/introducing-vizzie/
6.1. Max/MSP Externals inherited from Jamoma classes

Max/MSP externals generation from TTOBJECT of Jamoma libraries is done automatically by a wrapping mechanism. This consists to ask the class for his attributes and messages to create attributes and messages as defined by the Max API. The generated external benefits the same features as the Jamoma class and allows Max/MSP programmers to use the tools described above.

For example, jcom.parameter, jcom.message and jcom.return objects inherit from TTData, jcom.namespace wrapping the class TTEXplorer while jcom.send and jcom.receive objects directly expose the functionality of TTSender and TTRceiver classes.

```c
int TTCLASSWRAPPERMAX_EXPORT main(void)
{
    return wrapTTModularClassAsMaxClass(
        TT("Mapper"), "jcom.map", NULL);
}
```

Figure 8. jcom.map external generation

However this mechanism agrees to integrate the specificities related to Max/MSP. Thus, many externals are not direct wrapping of classes, but increased with a set of features unique to this environment. In addition, some classes are not subject to wrapping but are used in code to provide facilities (eg. TTXmlHandler).

All the features discussed with the description of Jamoma Modular is available through various externals.

6.2. Construction d’un patch « modèle »

A model patch is the operational heart of the program. It can house a technology to generate, process or distribute media that a Max / MSP programmer would make it accessible by providing a logical use.

This is done by inserting specific externals into the patch for the declaration of parameters (characterizing the state of the patch), orders or feedback. jcom.parameter, jcom.message and jcom.return objects are three class-specific wraps of TTData (with attribute ‘service’ respectively equals to parameter, message or return).

```c
Figure 9. Video player model
```

When a Max/MSP program instantiates a model patch, the namespace is automatically increased by the parameter names, messages and returns that contains it. These are recorded under the root to form a “flat” namespace, that is to say without branch.

The tree structure of a Max/MSP program (including several levels of sub-patches) is obtained by inserting jcom.hub and specifying its attribute @context model. His presence embodies the existence of a branch in the tree. The addresses of jcom.parameter, jcom.message, jcom.return objects are therefore related to the hub and are registered under a node whose name is that of the patch.

```c
Figure 10. Namespace of a model
```

The tree then becomes representative of the organization of the program making it possible to create such models including themselves other path models by inserting earnings models, filtering or equalization.

6.3. Construction of a « view » patch

A view patch allows to create an interface to the total or partial model patch, allowing the realization of a logic of use and tailored to specific context. The cons-

---

22 These features are specific to Max/MSP and inspired the development of the classes TTContainer and TTSubscriber. Theses are available to any C++ project but it remains to assess how their specification is generic enough to appeal to other environments.
struction is to declare objects \texttt{jcom.view}\textsuperscript{23} access and/or observation of parameters, messages or returns.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure11.png}
\caption{Movie player interface}
\end{figure}

It is possible to select data from one or more model patches that a user would see together. Moreover, it is possible to dynamically change the choice of parameters, messages and returns observed. Thus a single view patch can display the status of multiple instances of a model.

Instantiating a view patch also registers the names of \texttt{jcom.view} objects in the tree by the same logic described for the model patches.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure12.png}
\caption{Namespace of the interface}
\end{figure}

7. **UN AUTRE EXEMPLE D’UTILISATION PAR GALAMUS**

Beyond the strict confines of musical creation, the company Galamus-Software\textsuperscript{24} illustrates another case of exploiting certain features work Jamoma Modular. New company research and development, design and editing of specialist software, Galamus-Software develops\textsuperscript{25} innovative and creative interactive solutions including digital signage, digital media, interactive communication. The company shares as such the need for communicating to different software environments, able to communicate and be easily customizable.

7.1. **Galalib**

As part of its activities, Galamus is currently conducting a multiplatform C++ API and Open source developed and tested in parallel with Mac OSX as Windows7, allowing to easily extend functionality of Modular Jamoma to other software environments.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure13.png}
\caption{General organization of Jamoma and Galamus libraries}
\end{figure}

Galalib provides an interface "Namespace.h, which provides a set of methods to create and manage the simplest possible tree parameters, to create and delete settings, make update and retrieve attributes, create plays on the parameters. These operations may take place within the application but also between remote applications.

Galalib uses features preset and mapping Jamoma Modular. It also has methods for loading and saving Namespaces in files in XML format: for example it is possible to create a hierarchy of parameters by writing into a dedicated XML file. It will then automatically generated to load the file.

This library aims to facilitate the handling of these environments Galamus developers and anyone with similar needs.

\textsuperscript{23} \texttt{jcom.view} inherits from TTVreader comprising a TTSender and TTRceiver.

\textsuperscript{24} http://galamus-software.fr

\textsuperscript{25} The research and development activities are coordinated by Jean-Michel Couturier (co-founder of Blue Yeti, partner in the project structure Virage)
7.2. Examples of features

In addition to its integration into the software mentioned in the previous illustration (Adobe Flash, Ventuz, Unity, Virage sequencer) the use of Galalib is also provided in:

- Gallib drivers for devices;
- Drivers Galalib pour des applications de rendu et de diffusion de contenus;
- Galamus Management Tools.

The following figure gives an example of the use of three applications by communicating Galalib:

- an application of capture (eg returning the position of the hand of the person in front of the screen);
- a broadcast application (eg posting content 3D display);
- a sequencer (like the project turns) to modify the mapping and the value of certain parameters over time.

Figure 14. an example of applications communicating through Galalib

Each application has access to the namespace of the other applications through the network, via the class TTApplicationManager. An application can also load the file (in xml format) previously stored in the tree of a second application. This allows including work outside the network.

Thus the various software components may communicate to change settings, create mappings and load presets.

8. CONCLUSION

Jamoma Modular 0.6 results of reflections and developments undertaken in more than a year at the GMEA and more largely within the project Jamoma. Following these efforts, the first feedbacks of third developers seem to confirm the interest of the library to develop software related to creation. Furthermore, especially through the implementation of its functionality in the form of Max / MSP externals with the alpha version was released in February, Modular Jamoma seems to satisfy the needs initially established. Oriented Architecture “Model - View - Controller” allows the separation of different centers of attention and the deportation of the control interface. Using a tree structure for representing the structure of software seems an effective and flexible way for the developer or end user to explore the functionality of an application. Moreover, the addition of generic functions are meeting the needs of applied software for the creation or, more widely, software media convening of a different nature. Finally, since developments initiated under the project and turns through the combined efforts of Jamoma and Galalus a generic gateway for communication within a network is now available. In the coming months, several aspects should receive special attention. First, there should be greater phases of experimentation and testing the soundness and efficiency of Jamoma Modular. Performance measures in the context of complex creative uses can then guide the possible optimizations. Then, beyond the project Jamoma and goals that are his, its degree of genericity will continue to be carefully assessed. Why it is important to consider projects that use features Jamoma Modular and those to come as ways to support and further exchanges between the involved community members and computer music.

9. REFERENCES


