MarsyasX: Multimedia Dataflow Processing with Implicit Patching

Luis F. Teixeira, Luis G. Martins
INESC Porto
Campus da FEUP
Rua Dr. Roberto Frias, 378
4200 - 465 Porto, Portugal
{luis.f.teixeira,lmartins}@inescporto.pt

Mathieu Lagrange, George Tzanetakis
Dep. of Computer Science
University of Victoria
3800 Finnerty Road
Victoria, BC, Canada V8P 5C2
{lagrange,gtzan}@uvic.ca

ABSTRACT

The design and implementation of multimedia signal processing systems is challenging especially when efficiency and real-time performance is desired. In many modern applications, software systems must be able to handle multiple flows of various types of multimedia data such as audio and video. Researchers frequently have to rely on a combination of different software tools for each modality to assemble proof-of-concept systems that are inefficient, brittle and hard to maintain. Marsyas is a software framework originally developed to address these issues in the domain of audio processing. In this paper we describe MarsyasX, a new open-source cross-modal analysis framework that aims at a broader score of applications. It follows a dataflow architecture where complex networks of processing objects can be assembled to form systems that can handle multiple and different types of multimedia flows with expressiveness and efficiency.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*domain-specific architectures*; D.2.13 [**Software Engineering**]: Reusable Software—*reusable libraries*

Keywords

Multimedia processing framework, Dataflow processing, Open-source library

1. INTRODUCTION

Over the last decade we have witnessed a proliferation of multimedia content that is easily and widely accessible. One of the main challenges facing multimedia research is how to analyze and search such huge amounts of information. The multimedia signal processing community has been actively working on these problems.

This paper presents the most recent line of development of Marsyas[1], termed MarsyasX, which stands for Marsyas *"cross-modal"*. Marsyas is an open-source software framework which finds its roots in the Music Information Retrieval community. Its name stands for Music Analysis, Retrieval and SYnthesis for Audio Signals, and it started as a framework for the audio analysis, being especially suited for the development, testing and prototyping of analysis, processing and machine learning algorithms for audio signals [8]. Marsyas supports many ways of interacting with other programs and software environments including: run-time communication with MATLAB, Qt integration, bindings in various languages (Python, Ruby, Lua, Java) [9]. The new MarsyasX version implements a new payload architecture and extends the functionalities of Marsyas 0.2 (the current Marsyas version) by adding support for the processing of multiple flows with different modalities (e.g. audio, video). In addition to supporting real-time applications, MarsyasX has been designed with multimedia mining and retrieval applications in mind and has support for batch processing and machine learning tools (e.g. Weka).

Examples of other well known open source software frameworks for audio analysis and processing are CLAM [1], Chuck [10], Pd [6], and FAUST [5], among others. Commercial tools also exist in this area, as is the example of MAX/MSP. In what regards visual processing tools, there are many open libraries that deal exclusively with image or video processing, namely OpenCV, LTI-Lib, and The Recognition and Vision Library (RAVL), just to mention a few. However, these projects focus exclusively on computer vision and can be seen as utility libraries since they do not provide mechanisms to easily assemble algorithms based on building blocks. Even though the frameworks have been successfully used by users within their respective areas, the lack of intercommunication with other fields of research often leads to excessive overheads, reimplementation of similar techniques or even constant reinventions. Other frameworks and tools already exist that allow the integrated processing of audio and video streams. Although not originally created as multimodal platforms, Pd can be extended with visual processing modules from GEM, and Jitter adds video and image processing abilities to the MAX/MSP environment. EyesWeb, on the other hand, has been originally conceived for supporting research on multimodal expressive interfaces and multimedia interactive systems [3], but although being freely available, is not an open-source initiative.

MM'08, October 26–31, 2008, Vancouver, British Columbia, Canada.
Copyright 2008 ACM 978-1-60558-303-7/08/10 ...$5.00.

[1] http://marsyas.sourceforge.net

2. ARCHITECTURE

MarsyasX borrows from Marsyas 0.2 most of the concepts, namely the *hierarchical composition* paradigm and the implicit patching of modules. However, it was designed to support multiple flows with different modalities instead of just audio. Similarly to Marsyas 0.2 and other module-based processing frameworks, such as SIMULINK and LabView, systems in MarsyasX are expressed as interconnected dataflow networks of processing modules. Each processing module performs a specific task that always consists of a matrix transformation. Audio and other types of data are represented by matrices with some semantics associated with them. Processing is performed on defined chunks of data and is executed whenever the *tick()* function of the module is called.

2.1 Implicit patching

To assemble multimedia processing systems, modules are implicitly connected using hierarchical composition [2]. Special "Composite" modules such as Series, Fanout, Parallel are used for this purpose. For example, modules added to a Series composite will be connected in series, following the order they were added - the first module's output is shared with the second module's input and so on. Moreover, the "tick" method is called sequentially following the same order. Figure 1 shows an example of how composite and non-composite modules can be used. This paradigm differs from typical processing tools based on explicit patching such as CLAM, MAX/MSP or PD, where the user would first create the modules and then connect them by explicit patching statements.

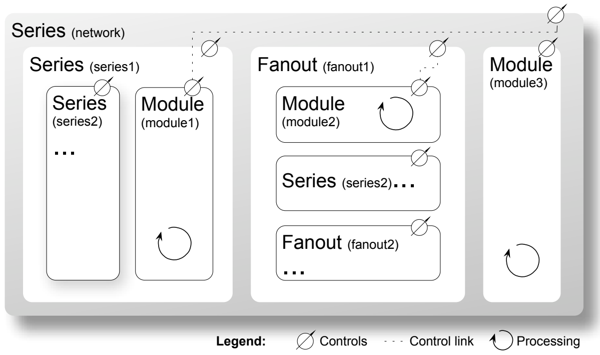


Figure 1: Building blocks in Marsyas.

2.2 Dynamic access to modules and controls

Each module can be accessed by querying the system with a path-like string. For example, to reach the processing module `module1` shown in Figure 1, the query path would be `/Series/network/Series/series1/Module/module1`. The first "/" indicates the outermost module and the rest of the path is always composed by the concatenation of `Type/Name` strings. This naming scheme was inspired from the way messages are exchanged in Open Sound Control (OSC) [11]. It is possible to access *controls* with a similar naming scheme. Controls represent internal parameters of the modules, and can be of different types (e.g. integers, floats, strings, vectors, or arbitrary user-defined types). Controls can be linked as shown in Figure 1, so that changes to the value of one control are automatically propagated to all the others.

2.3 Payload architecture

As in Marsyas 0.2, data is processed in defined chunks by calling a *tick()* function and each module also has a set of controls that are used to access their internal parameters. The main conceptual difference is in the way data is exchanged between processing modules. Instead of using shared matrices of real values, MarsyasX exchanges data through a *payload mechanism.* Whenever data is produced in a given module at each tick, a payload is created. This payload, "carrying" the data, is then sent to the output channel as depicted in 2. A channel is a connection between adjacent modules where payloads are stacked while waiting to be processed. It is important to note that channels are established implicitly, according to the type of composite being used.

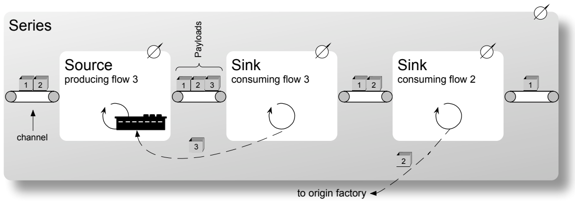


Figure 2: Payload architecture in MarsyasX.

This data exchange mechanism is highly generic and flexible, supporting any type of data (e.g. images, audio frames, MIDI, XML, lists of points, etc.). However, it does have its own specific issues such as timing and synchronization. Timing relations and constraints between data is assured by two time metadata fields in payloads – *Time of Capture (TOC)*, which stores at what time the data held by the payload was created or captured, and *Time to Schedule (TTS)*, which stores the time when the data should be processed by the modules that handle the payload. Whereas TOC is used to synchronize data of different flows (with the same or different types and names), TTS is used to schedule payloads for processing. The latter is especially important if, for optimisation reasons, we want to parallelize the work load while at the same time maintaining coherent time relations. If there are any payloads of the respective flow in the input channel, such that the current time $T_{cur} \geq TTS$, these are processed immediately and the resulting new payload is placed in the output. In case new data is created or if the processing consists of an in-place transformation, the same payload is forwarded. Any other payloads not satisfying this condition will remain in the input channel.

When a payload reaches a module that consumes the data without forwarding it or when it reaches the end of the network, the payload becomes no longer useful. From an efficiency point of view, creating and destroying payloads continuously would be computationally expensive. To avoid this overhead, a simple recycling mechanism is used. If a module is a source of payloads, it will have an associated *payload factory.* When a payload is required, it will be requested to the factory, which will either reuse an existing one or create a new payload based on a template. When the payload is no longer needed anywhere in the processing network, it will be returned to the corresponding factory where it will be stored in a recycle bin for future use. However, if the properties of the source have changed, a new template will be assigned and the old payloads are destroyed. This is a form of highly efficient type-specific garbage collection (or more accurately, recycling) that is enabled by the strict semantics of time and dataflow processing used in MarsyasX.

2.4 Data flows

Different payloads can be grouped together in abstract entities called *flows*. A flow consists of all payloads that have the same type and are tagged with the same name. A *flow type* field is used to distinguish different types of flows. Each payload of a given type must contain the same type of data. For example, a Visual flow payload should always contain an image, an Audio flow payload, an audio frame, and so on. On the other hand, a *flow name* field is used to identify different flows of the same type. Distinguishing flows of the same type can be useful for handling, for example, multiple video feeds with different image sizes. Moreover, it is often mandatory to distinguish different sources of data since many modules have stateful processing. If multiple sources are propagated in the same flow, unexpected behaviour on these modules will occur. Currently 5 types of flows are supported, namely `Audio`, `Visual`, `XML`, `Multidata` and `Legacy`. Each flow type is closely related to a data structure: matrix vector for multichannel audio frames, image supporting multiple colour spaces, XML tree structure, vector of independent matrices for generic data, and a matrix compatible with Marsyas 0.2 for legacy flows (see next subsection).

2.5 Legacy interface

An important feature of MarsyasX is the legacy interface with Marsyas 0.2. Undoubtedly it is very important to still be able to use the large collection of modules available in Marsyas 0.2 releases now and in the future. Since the framework base follows closely the previous, it is rather straightforward to support legacy modules. A MarsyasX module called MarSystemLegacy wraps a legacy module and synchronizes the controls of both. The most important difference is the way data is exchanged between modules. It is also the most costly operation, since it implies copying the data stored in the payload to the input slice of the legacy module and, after processing, from the output slice to the payload that will be sent to the output channel.

2.6 Modules

The similar architecture of MarsyasX and Marsyas 0.2, simplifies the porting of modules. Moreover, with the legacy interface it is possible to mix modules created with both versions. The plethora of Marsyas 0.2 modules for reading audio files, feature extraction, and audio analysis and synthesis can easily be made available for MarsyasX users. In addition to that, visual processing modules are being created, including modules for video and image IO, filtering, optical flow estimation, segmentation and feature extraction. Support for XML handling is also available. This is an ongoing process and more modules are expected to be developed.

Despite the complex architecture underlying the MarsyasX modules, the process of creating modules has been simplified. Typically, a user only needs to define the input and output flows and create a process function accepting the data according to the defined flows. This does not require any knowledge about the payload mechanism. Additionally, modules are managed using a simple plugin system. Plugins are dynamically loaded whenever necessary. Each plugin includes modules that are somewhat related. This has multiple advantages, namely: avoid excessive startup times due to module initialization, allow the deployment of smaller packages containing only the strictly necessary, and open the possibility for third-party modules (possibly with different licenses).

3. EXAMPLE APPLICATIONS

3.1 Music Information Retrieval

Marsyas was recently used for the submission of several algorithms to the MIREX2007 evaluation exchange[2], showing comparable results to other state-of-the-art algorithms (e.g. it was ranked first in the *Audio Mood Classification* task and second in the *Audio Artist Identification* task). Interesting to note are the computational runtimes achieved by the Marsyas algorithms when compared to the other contestants, being systematically lower in several orders of magnitude (e.g. in the *Audio Mood Classification* task, the Marsyas based algorithm, ranked first, took 122 seconds per fold against the 521 seconds per fold taken by the second fastest, though ranked last, algorithm).

3.2 Visual object segmentation and tracking

Another application implemented in MarsyasX was a visual object matching algorithm, described in more detail in [7]. Only visual processing modules are used. The algorithm consists of three steps: (1) segment each relevant visual object, (2) extract a representation for each object, (3) compare this representation with a database of objects, (4) if a given object is known, label it accordingly, and (5) update the database with the new information, if it is found relevant. In MarsyasX each of these steps corresponds to one or more modules performing a specific task. The main modules for this application include: background modelling and subtraction for object segmentation, extraction of local descriptors and vocabulary-based representation, and SVM classification.

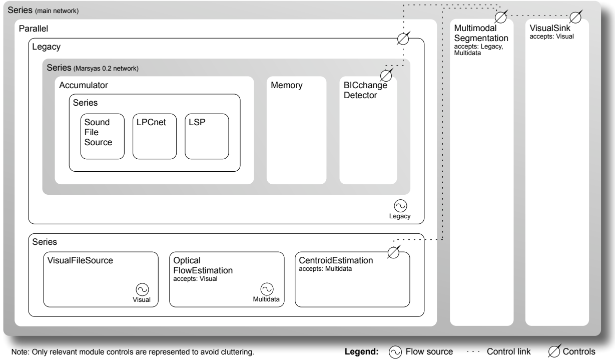
3.3 Multimodal speaker identification

An example of a multimodal application implemented in MarsyasX is a speaker segmentation system [4]. The algorithm and the corresponding network can be broadly separated in three parts: the audio speaker segmentation algorithm, the visual motion estimation and centroid calculation, and finally a multimodal speaker segmentation module that combines the visual and audio results.

The audio algorithm used for the speaker segmentation assumes no prior knowledge about the number of speakers or their identities and presumes that the audio input contains only speech. The method follows a metric-based approach for coarse speaker segmentation using Line Spectral Pairs (LSP), which is subsequently validated by means of the Bayesian Information Criterion (BIC).

The visual algorithm part of the network considers scenarios with only two speakers facing the camera, such as in-

[2] `http://www.music-ir.org/mirex/2007/index.php/Main_Page`



Note: Only relevant module controls are represented to avoid cluttering.

Figure 3: Network used for the speaker segmentation scenario.

terviews or lectures. It is assumed that the speaker will be located in the region containing the most amount of motion. The separation of these regions is defined by a boundary that for simplicity is kept as a vertical straight line splitting the image in two halves. A centroid of the motion is calculated and is used to detect the potential speaker.

The multimodal speaker segmentation algorithm takes into account two constraints: people tend to move their bodies, arms and lips before producing any sounds and the first sounds produced are usually non speech vocalizations such as breath, etc, hence the visual change detector is then more likely to be fired before the audio one; also, this last audio detector is more likely to detect the correct boundary but with a higher false alarm rate due to the presence of non speech sounds and background noise. When a speaker segmentation is detected by the multimodal module it is signaled in a control. This control is linked to the VisualSink that will display which speaker is speaking.

This work implements a late fusion scheme where one classifier is attached to each modality and the decisions of the classifiers are finally combined. Future work will concentrate on early fusion, namely the use of only one classifier that considers all the modalities at once, which is usually considered more reliable but harder to implement. MarsyasX can be powerful in such scenarios, since audio and video data can be conveniently aligned and combined within the same network of data.

4. DISCUSSION AND CONCLUSIONS

Cross-modal processing is an important and growing field of research among the scientific community. However, creating applications that rely on audio or visual processing can often be a cumbersome task. Although, there is a wide offer of specific tools and libraries, both commercial and free (open source or not), problems arise if one wants to combine some of these. The first problem is that data structure and semantics are almost always different and the user ends up creating custom wrappers or sometimes reimplementing functionalities. This is even more evident with a combination of audio and visual processing libraries. In fact, multimodal processing is an important and growing field of research among the scientific community. Having the ability to, under the same framework, use or develop new tools and algorithms is undoubtedly important. We proposed in this paper MarsyasX, a broad framework that attempts to solve these problems. By abstracting both data structures as well as its flow, and by using uniform procedures to define and set parameters, a considerable effort of integration can be removed from the user.

Marsyas is a software multimedia processing framework, with a special emphasis on audio processing. MarsyasX extends the functionalities of Marsyas 0.2 to visual support alongside audio. It is however not limited to audio and visual processing but can in fact be seamlessly used for generic data processing. Data is exchanged between modules using timed payloads, which in turn are implicitly grouped in flows, enabling efficient processing and memory recycling.

5. REFERENCES

[1] X. Amatriain. CLAM, a framework for audio and music application development. *IEEE Software*, 24(1):82–85, Jan./Feb. 2007.

[2] S. Bray and G. Tzanetakis. Implicit patching for dataflow-based audio analysis and synthesis. In *Proceedings of International Computer Music Conference (ICMC)*, 2005.

[3] A. Camurri, P. Coletta, A. Massari, B. Mazzarino, M. Peri, M. Ricchetti, A. Ricci, and G. Volpe. Toward real-time multimodal processing: Eyesweb 4. In *Proceedings of AISB 2004 Convention: Motion, Emotion and Cognition*, Leeds, UK, 2004.

[4] M. Lagrange, L. G. Martins, L. F. Teixeira, and G. Tzanetakis. Speaker segmentation of interviews using integrated video and audio change detections. In *Proceedings of International Workshop on Content-Based Multimedia Indexing (CBMI)*, Bordeaux, France, 2006.

[5] Y. Orlarey, D. Fober, and S. Letz. Syntactical and semantical aspects of Faust. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 8(9):623–632, September 2004.

[6] M. Puckette. Pure data. In *Proceedings of International Computer Music Conference (ICMC)*, pages 269–272, 1997.

[7] L. F. Teixeira and L. Corte-Real. Video object matching across multiple independent views using local descriptors and adaptive learning. *Pattern Recognition Letters*, 2008. (in press).

[8] G. Tzanetakis and P. Cook. Marsyas: a framework for audio analysis. *Organized Sound*, 3(4), 2000.

[9] G. Tzanetakis, L. G. Martins, L. F. Teixeira, C. Castillo, M. Lagrange, and R. Jones. Interoperability and the Marsyas 0.2 runtime. In *Proceedings of International Computer Music Conference (ICMC)*, 2008.

[10] G. Wang and P. Cook. Chuck: A programming language for on-the-fly, real-time audio synthesis and multimedia. In *Proceedings of ACM International Conference on Multimedia*, New York, USA, 2004.

[11] M. Wright, A. Freed, and A. Momeni. Opensound control: State of the art 2003. In *Proceedings of International Conference on New Interfaces for Musical Expression (NIME'03)*, Montreal, Canada, 2003.