Multimedia Tools and Applications
https://doi.org/10.1007/s11042-021-11876-y

1161: MULTIMEDIA ALTERNATE REALITIES

Pull-the-strings

Generic mapping model for digital puppetry

Luis Leite[1]

Received: 7 June 2020 / Revised: 26 July 2021 / Accepted: 23 December 2021

Abstract
Pull-the-Strings presents a mapping model for digital puppetry based on a transparent framework to support generic device controllers and generic tools. Digital puppetry requires a creative interaction design, in particular in the way designers map the puppet to the puppeteer using specific devices. This process depends on a constantly changing interface technology, which limits the reuse of devices and mappings. This paper proposes a methodology and a set of tools that facilitate the mapping process, and promote the recycling of technologies. A flexible and generic environment independent from device specifications. By abstracting the hardware layer, the artist is motivated to think in terms of signal flow, establishing relations through meaningful mappings instead of handling the diverse specifications of each device and application. *Pull-the-Strings* is a data-flow ecosystem that focus on the functional usage of control signals. It provides a scalable environment for building semantic blocks that connect, transform and generate signals for the manipulation of virtual objects. Its goal is to make technology as transparent as possible, facilitating connections and reducing the obstacles between the performer and the performing object. On the other hand, it proposes an interaction design space that takes into account the manipulation and perception distance, responding to the specifications of the digital puppetry medium. This model was evaluated comparing a set of tools and methods with experienced and non-experienced users.

Keywords Interaction design · Digital puppetry · Performance animation · Node-base interfaces · Real-time mapping

1 Introduction

Animation presents an expressive medium for storytellers. However, animating puppets using traditional keyframe techniques takes too much time and practice. Digital puppetry techniques can simplify this process by presenting a performance-driven animation making the puppet reactive to the motion of the performer in real-time. By turning the production into real-time, storytellers with no experience in animation can easily build animated plays,

✉ Luis Leite
luisleite@esmad.ipp.pt

[1] uniMAD, ESMAD/P.Porto, Vila do Conde, Porto, Portugal

exploring their imagination by just pulling the ‘strings’. By observing the feedback of their motion in real-time, they can practice the performance and adjust the manipulation. The interaction becomes fluid and intuitive because it is driven by the performer, making the story comes to life in a spontaneous way. When puppets are performed in a dramatic way, they produce the illusion of life. However, digital puppetry presents the challenge of mapping a set of input signals to a set of output joints that might differ in amount, scale and property. In particular, when the puppeteer drives a puppet with a large amount of Degrees of Freedom (DOFs) using a limited input device with few DOFs. Furthermore, when dealing with a known input device we can establish a direct mapping between a specific input value and the target DOF of the puppet, because the connection relies on underlying hardware specifications. However, if the specifications change through a shift in technology the previous mappings are lost, and if using an unknown device the mapping becomes a challenging process. Therefore, it would be desirable to develop a transparent interaction model for digital puppetry independent from the underlying technology as well from the target application. In this way, it would be possible to reuse the mappings and continue to animate the joints with different input devices and with any application. This flexibility presents the freedom for the designers to choose which applications and devices they want to employ in a specific context. Much work has been done on digital puppetry technologies, however, few researchers addressed the problem of interoperability between devices and applications that could facilitate the mapping between the puppeteer and the puppet. Existing frameworks are too specific and restricted, focusing on particular technologies, computer platforms, or target applications. Previous explorations based on interoperable systems applied to digital puppetry can be seen in projects such as “Digital Theatrograph: Cinematographic Puppetry” [13], a live cinematic-puppetry that employs a multi-application setting supported by an interoperable system, or “Solitária - Gestural Interface for Puppetry Performance” [10], a multi-dimensional and multi-modal puppetry performance developed with an orchestra of digital tools and data flow that controls the play. However, these interoperable approaches were developed to solve specific problems related to each puppet play. On the other hand, the “Common Spaces: Multi-Modal-Media Ecosystem for Live Performances” [12], aimed a wider application field, proposing a real-time media convergence interface based on an interoperable framework. This approach can be applied in different settings with a wide range of applications. However, it does not facilitate the mapping process and does not present a robust and generic framework. Pull-The-Strings was developed as an integral part of a digital puppetry PhD thesis [9] that addresses this problem. This paper starts by discussing in Section 2 the digital puppetry mechanics proposing a design space for performance animation. In Section 3 it is described a generic interaction model that supports the creative tasks of digital puppeteers in a collaborative environment, identifying its requirments. The set of tools that were developed to support the interaction model are presented in Section 4. Finnaly, the model is evaluated in Section 5 comparing a set of tools and methods with experienced and non-experienced users.

2 Design space for digital puppetry

2.1 Reality interaction space

Digital puppetry requires a specific interface design approach than those oriented for the classical computer animation tasks. An interaction approach can provide the appropriate interface design to handle the multitude of media that characterizes digital puppetry, supporting its expressive manipulation in real-time. The post-WIMP interface approaches seek interaction styles closer to the real world experiences. Jacob et al. [8] proposes a Reality-Based Interaction (RBI) framework based on real world knowledge and awareness with four themes: naive physics, the awareness and skills from the human body; from the environment; and from the social interaction.

- Naive Physics (NP) - the perception that people have of the basic physical world principles such as the understanding of gravity;
- Body Awareness and Skills (BAS) - the awareness of our physical bodies and the skills we have to control and coordinate them;
- Environment Awareness and Skills (EAS) - refers to our physical presence in the environment and the skills we have for manipulating it or for its navigation.
- Social Awareness and Skills (SAS) - the awareness of other people and the skills for interacting with them.

These reality-based interaction themes covers the most important manipulation aspects of puppetry. While the notion of physics (NP) is fundamental for operating a marionette taking advantage of its weight, the body skills (BAS) allows the puppeteer to use his hand dexterity for manipulating a glove puppet. The puppeteer uses the surrounding environment (EAS) to drive the puppet through a wooden floor, and interacts with the audience (SAS) inviting them to participate, or to colaborate with other puppeteers, when manipulating a multi-puppeteer puppet, such as the Bunraku. These themes are appropriated for generic interaction design and provide a good starting point for digital puppetry.

2.2 Digital puppetry mechanics

Digital puppetry presents similar interaction mechanics as the traditional counterpart. The puppeteer operates a controller that is mapped to a puppet. It involves an input device, a mapping scheme, and a puppet structure (Fig. 1).

- The input device determines the interaction method, the number of degrees of freedom (DOF), and which part of the puppeteer's body is used for manipulation. The interaction method can be classified according to the degree of abstraction between the puppeteer and the virtual puppet [3]. With direct manipulation, the body of the puppeteer is used directly as an input device. With indirect manipulation, the device controller (i.e gamepad) is the interface between the two agents, providing similar functions as the traditional marionette controller.
- The mapping scheme is used to transform and assign each DOF from the input device to the corresponding puppet's DOF. The mapping scheme is the process that establishes the necessary connections and semantics between the input signal and the desired output behavior. This is a central process in digital puppetry and is represented in Fig. 1 by pulleys at the center, and by red wires that establishes the connection between the components.
- Morphology is the rigging structure that supports the puppet's motion, the number and type of DOFs. While the input device determines how to capture the input physical morphology of the puppeteer, the virtual puppet armature defines its morphology and its behavior. It can replicate the input rigging for direct correspondence, such as a biped skeleton, or provide different amount of degrees of freedom with indirect correspondence that requires indirect mapping.

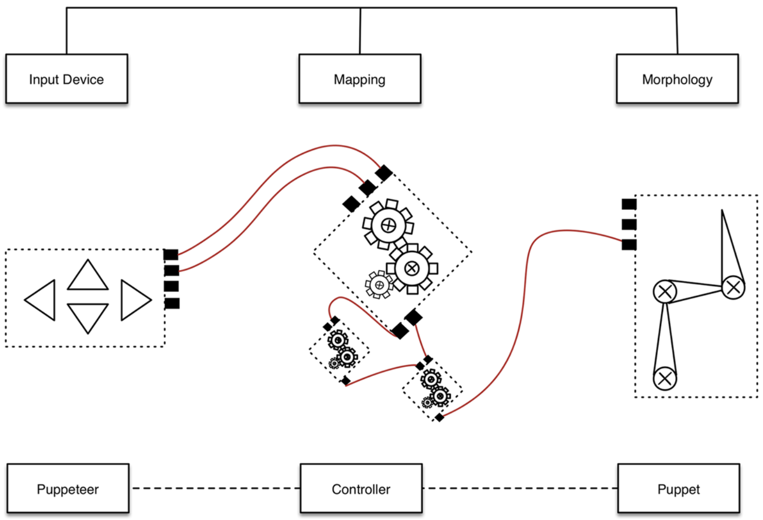


Fig. 1 Digital puppetry mechanics, the input device captures the puppeteer's motion, which is mapped to the puppet's morphology

These three components form the base aspects of Walther-Franks and Malaka [16] structural model for interactive animation. They define a triangular conceptual structure based on the relations between the input device, the mapping and the puppet, identified by: 1) integration (machine); 2) metaphor (artist); and 3) task (artifact).

1. The integration of control is defined by the input device or by the hardware layer;
2. The metaphor describes the performer engagement with the input device, the mapping between his intentions and the appropriate interaction with the hardware;
3. The task defines the intended action for the production of animation, which can be decomposed into a large amount of sub-tasks such as motion creation, motion editing, and motion viewing.

This conceptual space defines the relations between machine, artist and artifact relying on a spatiotemporal interaction, which is appropriated for traditional computer animation.

2.3 Digital puppetry interaction design

Digital puppetry relies mostly on real-time interaction, and therefore this space should be adjusted for performance animation specifications - a design space based on the puppeteer, controller and puppet relationship, as seen in (Fig. 2). These three elements can be used to define a triangular relationship for digital puppetry. The physical and perceptual distance between each vertex can be used to establish the manipulation type, as well as the puppet's behaviour.

The distance between each **vertex** of the triangle can increase or reduce depending on multiple aspects. This digital puppetry flexible triangular interaction is based on three main components: 1) puppeteer; 2) controller; 3) puppet.

1. The puppeteer uses his body for manipulation and can employ multiple senses at the same time. As in traditional puppetry, multiple puppeteers can manipulate one puppet, or one puppeteer can manipulate multiple puppets;

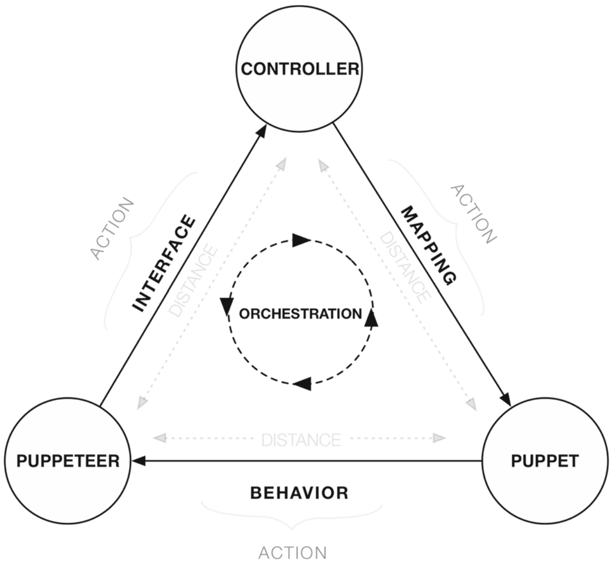


Fig. 2 Digital puppetry interaction scheme based on the relations between the puppeteer, the controller, and the puppet

2. The controller provides the interface for the puppeteer to manipulate the puppet, combining the physical device with the strings attached to the marionette;
3. The puppet is the performing object, developed to respond to specific tasks or actions supported by a morphology that will define its movement. It can assume many different forms. The puppet's appearance and behavior influences the puppeteer's manipulation.

Each **edge** of the triangle defines an action. The puppeteer uses an 1) interface for manipulation, capturing his signals that are 2) processed and mapped to the puppet, which 3) behaves according to its morphology.

1. The Interface, defines the puppeteer's manipulation method through input devices specifying the number and type of DOFs to be controlled simultaneously. It concerns the hardware and software dimensions allowing the puppeteer to operate the puppet;
2. Mapping, describes how the morphology of the puppeteer's physical input, captured by the input device, corresponds to the output morphology of the puppet. The mapping process transforms the input signals to fit into the desire output behavior;
3. The behavior of the puppet is determined by its morphology, defined by a structure that makes it move. A rigging armature with a set of controllers that deforms or articulates the shape of the puppet.

Similar to the Walther-Franks and Malaka's approach, the orchestration process occupies the center of the triangle. This is a critical process in a live digital puppetry show because it defines how to change the mapping schemes, how to switch between puppets, or how to select the input devices during a performance. For instance, the puppeteer may want to switch the input method, switching the manipulation from voice input to hand manipulation, as well as the target, changing from facial expression control to a full-body driven puppet. This switch requires a change in the mapping setup. Furthermore, designing a complete digital puppetry performance requires the configuration of lights, sound, cameras, and other media according to each cue of the play.

The distance between each vertex of the triangle influences the relationship between the puppeteer and the puppet.

1. Embodied distance: The distance between puppeteer and controller (**interface**) influences the performer's interaction experience determining the directness level. It is based on the interface quality and features. The number of DOFs provided by the input device and the interface ergonomics and sensibility constraint the performer's manipulation. On the other hand, the number of DOFs controlled simultaneously by the puppeteer, can increase the distance and the complexity of the manipulation. It determines the motor coordination as well as the engagement with the device (i.e. an embodiment interface);
2. Action distance: The distance between the action performed by the puppeteer through the interface, and the reflected action in the puppet (**mapping**) is central for establishing direct or indirect manipulation. It can be used to simplify the operations, or to increase the manipulation possibilities. The correspondence between the input and the output DOFs contributes to the distance between the puppeteer and the puppet. This distance contributes to the cognitive load of the puppeteer;
3. Feedback distance: The distance between the puppet's action and the puppeteers perception of the action (**behavior**) influences the puppeteer's manipulation. The motion response of the puppet to the controller, can be modified by external factors, such as gravity, and influence its movement. The rigging structure that supports the puppet's morphology is also determinant to constraint the motion of the puppet, that may respond with unexpected behavior forcing the puppeteer to adapt the manipulation. (feedback distance)

The distance between the components is determined by: 1) human factors; 2) manipulation factors; 3) design aspects of the puppet and interface as in Table 1.

These factors are interrelated, for instance the amount and placement of the input DOFs to be controlled simultaneously influences the motor coordination of the puppeteer. The morphology of the puppet determines its mechanics as well as the controller interface, and consequently influences the way the puppeteer synchronizes his body to perform each action. The mapping scheme influences the cognition load of the puppeteer, its design depends upon the selected interface and both influence the engagement level of the puppeteer.

3 Generic interaction model

Puppets present a variety of shapes, control methods and degrees of freedom making them an ideal interface for creative and collaborative work. No matter if the puppet is physical or

Table 1 Factors and aspects that influence the distance between the puppeteer and the puppet

Human factors	Manipulation factors	Design aspects
Motor coordination	Number of DOFs	Ergonomics
Motion synchronization	Morphology	Mechanics
Cognition load /engagement	Interface	Mapping

virtual, the relation between puppet and puppeteer is umbilical, as Francis states “The control may be through corporeal contact (hands-on, hands-in), or via strings, wires, wooden or metal rods. The figure animated electronically or even remotely is still a puppet if the performer is present at the other end of the cable or the machinery, controlling the movements, just as at the end of a simple string or rod.” [6, p.13]. On the other hand, the puppeteer is a multidisciplinary artist that combines creation with performance. The puppeteer can explore many skills and disciplines, from modeling to acting, from writing to producing. This multitude of skills, puppet types, control rigs, and manipulation methods require a generic interaction approach.

3.1 Digital ecosystem

In order to manipulate a puppet the puppeteer chooses the appropriate method based on multiple factores: a) the puppet type and morphology; b) the intended action; c) the artistic decisions; d) the puppeteer’s own expertise. On the other hand, a digital performance may not rely on a specific manipulation technique, but rather, on multiple interaction methods which are identified during the collaborative creation process. This creation process is characterized by a dialog between the members of the chain, through a constant coming and going, through trial and error, through multiple iterations. They begin by cutting the main problem into little blocks of problems, that simplify the process, and allow them to focus on small issues at a time. Each element of the team contributes to the process and they must work together. While working in a collaborative environment, each element of the team may require specific features. On a digital puppetry performance, the mapping designer and the animator-puppeteer that we can consider the “animateer” [9], require distinct control interfaces (hardware and software). The mapping designer may require a mouse device and a software environment to link the input DOF to the output DOF, and the “animateer” a control interface for manipulation and a visual feedback of the puppet’s motion. These distinct interfaces can be considered units of an interoperable collaborative system, that are integrated in a larger pipeline. By establishing a data flow framework and a communication layer that connects the different units (signal flow), each process can be done with variety of applications. By implementing a standard digital signal flow in the interface, we can map any control device (generic devices) to the puppet. A modular toolkit can be designed based on this flow of signals, to capture, process, map the input data and visualize the animation in real-time (Fig. 3). By working with independent tasks with a higher-level of control, the system becomes flexible. A generic interaction model can be conceptualized to support the creative tasks of digital puppeteers in a collaborative environment, based one the proposed design space. A methodology for performance animation to enhance the storytelling experience.

A critical aspect in a digital puppetry performance is to provide the appropriate feedback to the animateer. By producing an action, the performer expects the system to respond with the feedback from this action. This action-reaction is fundamental for the performer to be able to manipulate a puppet in an expressive way. This real-time inter-action is what makes digital puppetry possible, and the performer understands this relation as a direct manipulation. The motion energy from the puppeteer should be routed to the performing object through a chain of units and procedures. In each procedure the data suffers transformations, for instance the movement produced by the performer is captured into digital signals. These signals are processed through filters, then scaled and normalized. Finally the signals are mapped into the performing object establishing the puppet-to-puppeteer control relationship producing the motion (Fig. 4).

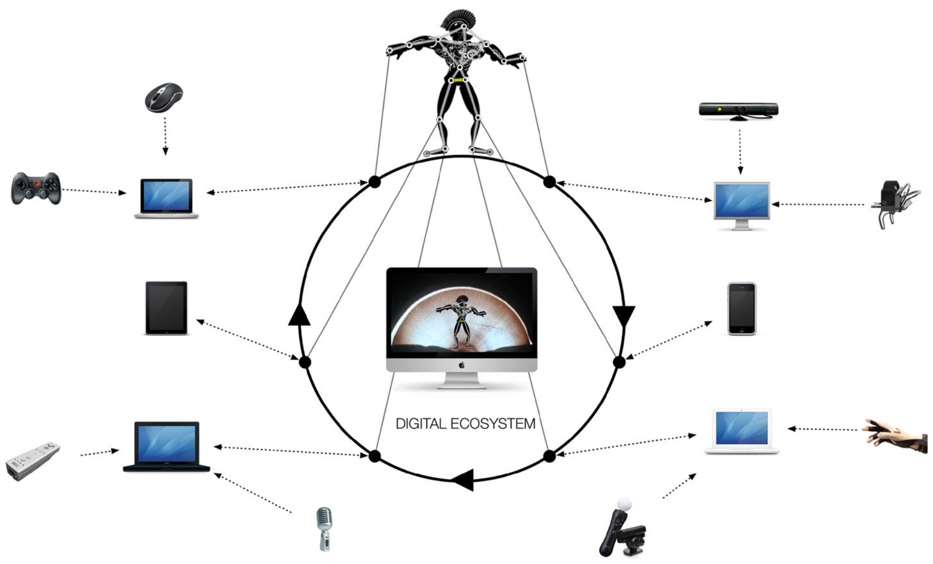


Fig. 3 Digital Ecosystem - An interoperable multimodal environment for digital puppetry

3.2 Signal flow

The interaction designer must choose the best strategies to extract meaningful data from the multitude of signals/devices available for capturing the motion of the performer. Today, all sort of devices provide some kind of sensing capability. Reading these sensors is now trivial with all sorts of Internet of Things (IOT) techniques. These sensors and devices can be combined in a network of signals providing powerful means of digital manipulation. In this way, it is important to develop a model capable of converging these signals providing methods to synchronize them, and extract meaningful data, as well, to process and route these signals to the desire actions and applications. Rather than thinking in specific interaction methods (i.e. press button, move the mouse), we could treat all data as abstract but normalised digital signals (1 to 0). These signals can be processed and mapped to specific tasks in a variety of tools. In this way, the model will be functional with today's technologies or

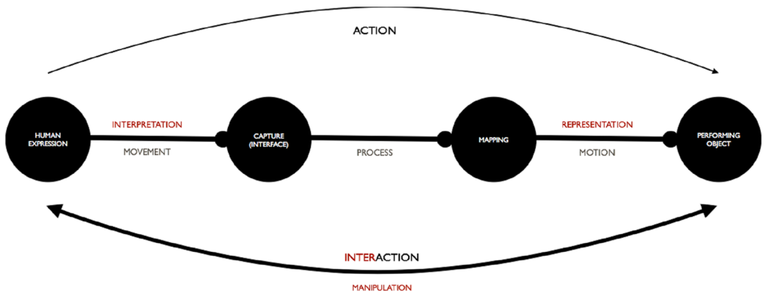


Fig. 4 Action > InterAction - real-time feedback is fundamental for the performer's acting

with the next generation devices. Humans can interact with computers with a multitude of senses, which requires certain cognitive skills. Technology is advancing towards a omni-sensing, to a multitude of input sensors providing the most rich and natural interaction. A multimodal approach can extend the interaction experience. Thus, rather than restricting users to interact with limited input devices available or with just one specific application, we can combine input devices from several applications/computers and share these resources among an interoperable network environment. This also applies to the outputs, and to the performing objects in particular, which can assume many different forms. Thus, using a similar interoperable approach it would be possible to take advantage of distinct animation tools to render a specific performing object and then, combining all performing objects together in another application. Hypothetically we could combine a 2d animation program with a word processing application and overlap them, creating a puppet above a text scenario.

3.3 Generic device

Signal abstraction presents an important paradigm for creating a flexible and scalable environment. This signal abstraction could be supported by a generic device. The concept of generic purposes devices is not new, and many authors have been studying devices and interaction models to support and enhance the user manipulation. Ed Anson [1] presented a device model for interaction based on objects that a user can handle directly to create the feeling that he is manipulating things. This concept is based on states, events, and actions. Ed Anson compares devices to programming variables in the sense that both report values or states that can be tracked and manipulated through user actions. In this way, it is possible to think in abstract devices for generic purposes, and although a specific device may be unknown in advance, its features can be connected and tuned. An interactive system can be designed by defining the features that are required for specific tasks and by defining their interconnections. This modular concept is simple, and well structured. On the other hand, virtual devices are software device abstractions implemented in the operating system. These software-based devices can be used to create composite devices combining software and hardware devices. These abstract devices act as an interface between other devices and applications, as an intermediary between the type of the device and the interaction method itself. They present some limitations extending the hardware input devices [7]. Card et al. [4] based on the Mackinlay work developed tools to describe the semantics of a device measuring its expressiveness. He employed the human performance theories to evaluate effectiveness of points in the design space. They have presented the notion of composite operators, where one input is connected to another input generating a different setting with three distinct composite operators: the merge composition, the layout, and the connect composition. The merge refers to the combination of devices through a cross product, the layout composition groups the two devices without exchanging their input domains, and the connect composition referes to the mapping of the output domain of one device into the input domain of another device. This architecture provides the basis for a simple interaction system. The term mapping is thus used as the computation of control values, and these values depend on the incoming data and state machines. The incoming data from devices is then mapped to specific features on the puppet. This mapping is defined by the designer that specifies how this signal is going to be interpreted and connected. However, the mapping process presents logical and artistic problems. The interaction designer cannot make arbitrary connections between a pair of input and output signals without any reasoning and expect a perfect match, that meets the objectives of the puppeteer. It is fundamental to establish logic connections between signals, where its nature depends on a particular device,

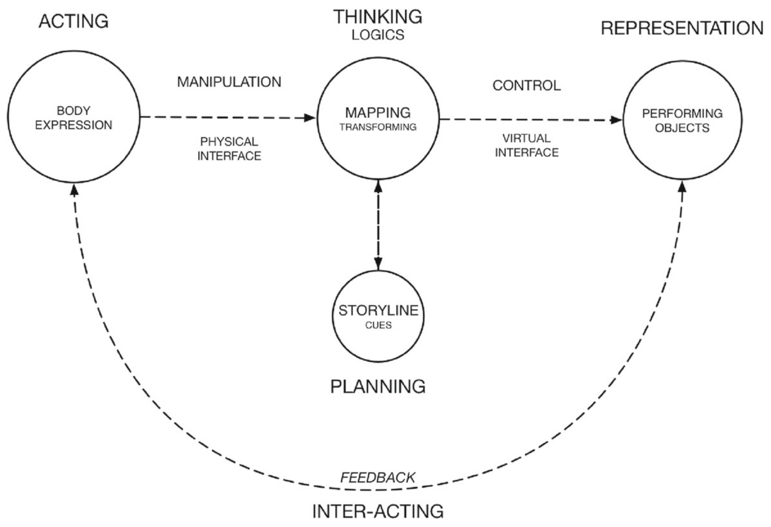


Fig. 5 Signal flow model for a generic interaction, user input signals are processed through the physical interface and mapped through the virtual interface to the performing object

and combine this decision with a broader artistic intentions from the puppeteer towards its specific context (Fig. 5).

3.4 Requirements

It is important to identify the major requirements for the design of a digital puppetry generic framework based on input and output abstraction. This framework should allow non-expert programmers to easily create, analyze and tune the mappings in real-time through a creative iterative process, providing the appropriate feedback to the user. The workflow should focus on the logic and semantic programming rather than on the device issues. It should provide interchangeable devices, and reuse the mappings in an ecological thinking. The ecological thinking concept is based on the way designers can recycle old device controllers and applications that were built, with other purposes than those from the digital puppetry, as well as the reuse of mappings as being a part of the ecosystem, as a creative process itself. This framework should not focus on specific devices neither specific puppet types, but rather support reusable modules, and provide a generic mapping design. The major requirements for this framework are: flexibility, usability, scalability, extensibility. The framework should also fulfil the following requirements: cross-platform, open source, readily-available device communication protocol, and with a visual-programming paradigm.

– **Usability**: can be seen as the technical skills that are required for the user to produce the necessary changes on the system, as well as the speed he establishes and tests a new mapping setup. The system should be highly interactive, avoiding scripting and providing instant feedback without depending on code recompilations every time the user makes a change. A graphical user interface presents the ideal iterative environment for exploring creative mapping in real-time, for changing the parameters interactively, and to establish their interdependencies.

- **Flexibility**: to provide a highly flexible environment such as those that we can find in traditional programming languages. On the other hand, these traditional programming environments do not present the most intuitive interface in terms of usability, in particular for non-programming users. Thus, it would be preferable to combine a visual-based authoring environment with the unlimited computational features of programming languages that could run in-between input devices and animation applications. There are several visual-based programming environments that support multiple input devices such as TouchDesigner, Pure Data or Node-RED. These tools share the same iterative paradigm based on graphs using a data-flow approach, where nodes or computational units process and generate output data from the input data. These nodes can be combined and establish complex mappings through a network of simple mappings.
- **Extensibility**: to support new features added through external plugins or scripts and extend the system's functionalities. This can be achieved by loading dynamic shared objects at runtime or implementing a script interpreter such as Lua or Python. This architecture allows adding new node functionalities without the need to recompile the code, acting as runtime extensions. These extensions should follow some specifications, such as the number of inputs and outputs, as well as the data type. With this approach, it would be possible to create new operator nodes as well as to define their data types.
- **Interoperability**: The framework should be able to establish a conversation between distinct applications and devices and act as an interface. A middleware agent that extends the functionalities of applications and devices by combining them in a digital ecosystem. One single environment might present limited functionalities in certain domains or even lack of the desired features. What if these required features could be borrowed from other applications? This framework could unite all the resources available in our computational environment, being hardware or software, located in our local computer or distributed among the network. To enable interoperable operations, the framework should be supported by a communication protocol. Open Sound Control (OSC) is one of the most popular high-level protocol that runs through UDP/IP and TCP/IP. This transport independent protocol is available in a wide range of multimedia applications, allowing the user to specify its custom messages through a name space similar to the Uniform Resource Locator (URL) structure. The advantage in customizing the messages provided by this open standard is clear, but this customisation can also present compatibility problems. The lack of a standard namespace prevents applications and devices from an immediate communication. In this way, it is not possible to establish a dialog between unknown applications or devices without a first contact, without establishing their syntax rules. Thus, it would be desirable to establish uniform namespaces or common guidelines for compatible purposes [18]. OSC provides communication and resource sharing across multimedia applications and devices.
- **Scalability**: To be able to grow when needed. By using the network environment we can connect multiple computers and mobile devices and expand the workflow. In this way, it is possible to create a collaborative play, where each computer acts as an instrument of a digital orchestra. It also contributes to decrease the computation power in each node, distributing the tasks through the available resources. The OSC protocol could be used to support the scalability. One OSC message could be sent to distinct applications within the local computer, or within a network of computers if available. The system grows by adding more devices to the network adjusting the system resources to the project requirements. This modular workflow based can be used in the hardware dimension, as well in the software dimension. By adopting the same logical workflow and communication protocol, such as the OSC, in the authoring environment as the internal

messaging system, the same message used to control an external device can be used to control and change states in the authoring environment itself. This transparent environment allows the same message to flow within the entire ecosystem. There are other data-flow methods capable of transporting generic information such as JSON, but its not supported by many animation packages. A scalable and flexible model contributes to an efficient system. Each computer becomes a node in this modular framework and expands the interaction capabilities, and the computational power.

3.5 Node-base programming (String-based)

Each digital puppetry performance requires specific mapping schemes that can be programmed through predefined operator nodes, or computation units. These nodes are building blocks designed to solve common mapping problems, such as value interpolation, scaling functions, or other mathematical operations. Typically in a node-base programming the user is free to connect nodes and define his own logic and data flow. This visual approach, presents a transparent and intuitive programming for establishing a semantic mapping. However, these environments presents limitations and constraints, in particular when handling with complex and long programs. This programs can be confusing, difficult to debug, or to manage sequences in time, such as triggering cues in a specific order. An important aspect to consider, when using these environments through the network, is to facilitate the device discovery and establish automatic connections. Connecting multiple devices can be time consuming. To speed up this process, we can use techniques such as Zero Configuration Networking (Zeroconf[1]). This set of techniques, which are based on the Internet Protocol (IP), provide automatic discovery of namespaces, and IP addresses on a network, without the need for user manual intervention. Zeroconf automatically manages the port allocation and distributes the data to the available clients on the network without requiring previous knowledge about the network settings. In this way, connecting all devices with virtual "strings" can be fast and easy.

3.6 Node-based environments

There are multiple node-base environments that fulfil partially the requirements presented previously. They demonstrate the relevance of the transparent, modular, and visual approach environments, in particular the use of the data-flow paradigm. There are many free and cross-platform environments such as NodeBox or Fugio. Node[2] box is a visual programming environment based on Python for creating visuals developed by Frederik De Bleser and Tom De Smedt [2]. It supports OSC as an experimental feature, requiring to enable the Device Support. Another flow-based programming environment for creativity that supports OSC is Fugio[3], a tool developed by Alex May oriented for projection mapping and video streaming. On the other hand the evolution of the Web technology transformed the Internet browsers from simple hypertext navigators into powerful tools. Browsers are now able to support features that were only possible with stand-alone applications years ago. It is now possible to edit video or work in 3D environments Online inside a browser and in a collaborative way. Browsers provide a natural cross-platform environment and can be accessed

[1]Zeroconf web site – http://www.zeroconf.org

[2]Node Box 3: https://www.nodebox.net/node/

[3]Fugio Web site: https://www.bigfug.com/software/fugio/

from any device. There are multiple advantages in using Web technology, even though there are still many limitations and constraints such as compatibility and performance issues. Many Flow-based programming (FBP) environments run in a browser, such as NoFlo, RPD, or Node-RED. NoFlo is a flow-based programming environment for JavaScript that works above the asynchronous programming library Node.Js. NoFlo is a library to develop flow-based programs in JavaScript and there is a component for working with the OSC protocol. Another visual programming environment for the Web is the RPD[4], which stands for Reactive Patch Development. It is a node-based user interface based on the Funcional Reactive Programming (FRP) library Kefir.js. It provides multiple rendering styles or interface skins that simulate the Quartz Composer or Pure Data visual environments. Node-Red[5] is another flow-based programming for the Internet of Things (IoT). Node-Red started to be developed in 2013 by Nick O'Leary and Dave Conway-Jones from IBM as a proof-of-concept for visualizing and manipulating mappings. It is a tool for wiring together hardware devices and applications, built on top of Node.Js. Node-RED is now open-source, and supports OSC through a third party extension.

These environments respond to most of the requirements such as being open-source, cross-platform, developed with a visual-programming paradigm. However, none of them presents a truly OSC-based environment, making use of the URL scheme as their internal data flow syntax to communicate among all types of nodes. Some environments support OSC messages which are converted to their specific data flow, and so, you can send and receive messages. However, if the system integrates OSC as its own messaging system, you could also send control messages. In this way, it would be possible to control the entire environment remotely, becoming a truly transparent ecosystem. This aspect is determinant to achieve a transparent and abstract framework compatible with multiple creative tools that already support this protocol, such as the IanniX [5]. A graphical open-source sequencer for digital art, that generates OSC signals sent as control events to manipulate graphics and produce sounds in all sorts of applications that support OSC. Even though there are no digital puppetry dedicated authoring environments, it is possible to produce performance animation with game engines (Unity3D, Unreal Engine), 3D animation packages (Motionbuilder, iClone), or with interactive visual tools (Touchdesigner). Some of these tools gather functionalities previously identified as requirements, however, none them presents an abstract and transparent framework directly. Thus, it is necessary to develop a new environment from scratch that acts as a middle layer, dialoguing with different interlocutors, capable of integrating distinct technologies, as well as functionalities.

4 Puppet tools - An interoperable environment

Based on the mapping model, an interoperable system was implemented with a set of tools developed in C++ and C# programming languages. These units constitute the modular marionette ecosystem known as the Puppet Tools. They are a combination of "middleware", "software", and "plugins" that work together in a sort of orchestra. These tools, "talk" to each other through a network of virtual wires creating an interplay of signals, as strings from

[4]RPD Web site: https://shamansir.github.io/rpd/
[5]Node–RED web site: https://nodered.org

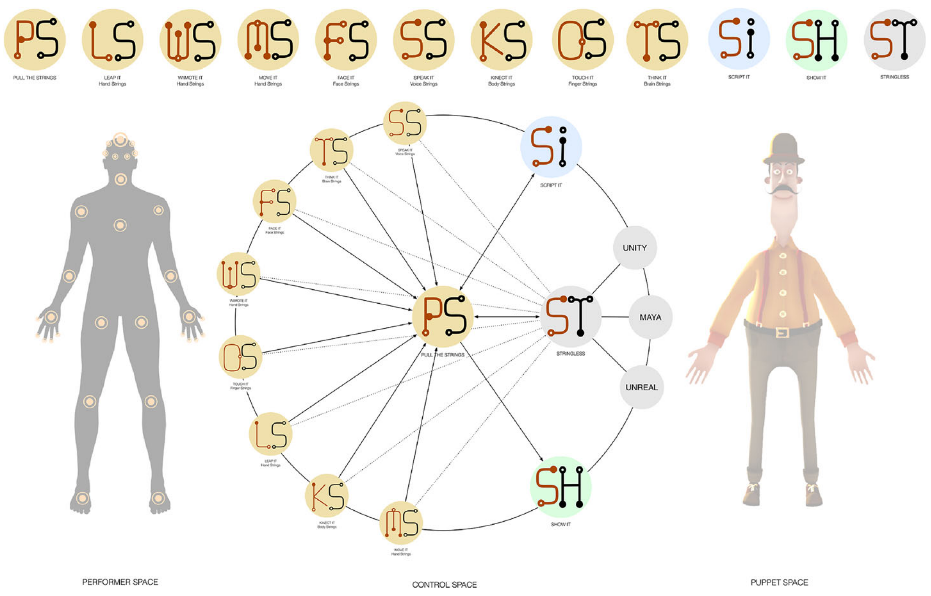


Fig. 6 Pull-The-Strings ecosystem - A control space based on small apps connected with virtual strings that link the performer to the puppet

a marionette controller. It is possible to run the same tool in several computers and distribute the computation power, or share resources from each system in a collaborative play.

These tools were designed to facilitate the sharing of resources among devices and applications, as well to route and process signals and to orchestrate the data flow (Fig. 6). The Puppet Tools communicate together through OSC using automatic feature discovery facilitating the connection and resource sharing. These tools are divided into three categories: 1) Input strings; 2) Output strings; 3) Orchestration strings.

1. The input strings establish the connection to input devices extracting their signals. They are middleware apps that capture the inputs of the performer, from voice to full body capture;
2. The orchestration strings handle the data-flow, allow the user to establish the mappings. These tools are responsible for tying up the virtual strings to external engines. They are applications that receive the input strings and manage the data flow. These apps generate, modulate, and route signals, as well as establish the connection with external engines such as Unity3D;
3. The output strings are attached to performing objects and make them move and feel alive. These apps produce the animation and the audiovisual result allowing the creation of a virtual puppet show. The output strings materialize the signals coming from the orchestra into moving images, sound, lights, cameras and all the motion data that constitutes a virtual puppet show. These strings, can be wrapped into a puppet show animation engine, or distributed through a set of modular apps that handle the audio and the visuals.

To demonstrate this interoperable environment, a set of Puppet Tools were initially designed supporting the proposed modular framework (Table 2). Some of these tools were

Table 2 Puppet tools: applications that support the interaction model

String type	Name	Description
Mapping and orchestration strings	Pull-The-Strings (PTS)	Marionette programming engine
	Stringless (STS)	Interface for different applications
	Script IT (SCP)	Script for show control
Input Strings	Leap IT (LS)	Hand String: Leap Motion
	Wiimote IT (WS)	Hand String: Wii remote
	Move IT (MS)	Hand String: Move controller
	Touch IT (OS)	Hand String: Touch surface
	Kinect IT (KS)	Body Strings: Depth sensor
	Face IT (FS)	Face Strings: Video camera
	Speak IT (SS)	Voice Strings: Microphone
	Think IT (IS)	Brain Strings: BCI device
Output strings	Show IT (SWO)	Puppet show animation engine

already released and are available for download at the Virtual Marionette[6] Web site or at the Github repository[7].

Digital signals flow through this ecosystem of tools, a network of applications that can run in parallel in one or several machines acting as one unique system. The data can be synchronized, mixed, or combined in a scalable and generic way. In this way, it is possible to connect new computers or devices acting as nodes of the same engine.

4.1 Pull-The-Strings (PTS) - The marionette programming engine

Visual Programming Languages (VPL) have been developed for a long time. Environments such as the GRaIL (GRaphical Input Language) system from 1968 provided the first intimate interface experience for users to interact with, which are appropriate for non-programmers. Visual programming languages are based on data-flow. Flow-Based Programming (FBP) is a component-oriented paradigm where applications are designed and developed by linking blocks of processes in a chain. It was invented or "discovered" by Paul Morrison in the early 1970's [14]. Rather than sequential processing, FBP is based on a network of asynchronous processes that communicate through streams of data chunks, or information packets (IPs). The way the data flows through these blocks or components defines the logic of the program. In FBP the data packets have a defined lifetime, named ports, and separate definition of connections. FBP is based on asynchronous processes that flow as chunks of data from node to node. A visual flow-based programming provides an environment where the programmer connects reusable components together inside a graph through "strings". Graphs provide an appropriate model for prototyping ideas. Humans respond faster and better to visual impulses and can recognize certain visual aspects very quickly. This aspect was verified by Rudraraju's [15] from observations of using a particular visual mapping software, the Vizmapper. After some time of experience, users do not need to pay attention to textual labels because of the visual arrangements. While graphs provide

[6]Virtual Marionette website: http://virtualmarionette.grifu.com
[7]Puppet Tools Github repository: https://github.com/grifu

a visual medium for solving logical problems, interactive graphs provide an ideal sandbox for experimentation and real-time prototyping. This seems to be the most appropriate environment for the digital puppeteer to design its mappings.

Pull-The-Strings (PTS) is inspired by Flow-based programming (FBP) which promotes code reuse, and combines it with a visual programming environment providing intuitive coding. It was developed as a modular system where the programming is made through a chain of nodes of processes, that act as building blocks. The designer connects these black boxes through strings to exchange data and establish the program logics. They can connect these nodes in many different ways to produce distinct behaviors.

Pull-The-Strings (PTS) (Fig. 7) is a central application in the proposed ecosystem, a virtual marionette programming approach. It is an authoring environment for processing, routing and mapping signals from the puppeteer to the puppet. A framework for artists without programming experience, inspired by the strings from the marionette controller and the patch cords from the video synthesizers. By pulling the strings from node to node, the artist establishes semantic connections, thus constituting a real-time programming paradigm that supports easy prototyping.

Pull-The-Strings environment is based on well-defined tasks presented as black box components. These components can be connected with each other in several ways creating different semantics. In this way, the components are being reused. This authoring environment can be consider as a digital mediator between devices and applications providing the logics for the digital puppeteer to establish the manipulation metrics. Developed in C++ using Openframeworks, it was designed with a minimalistic, but intuitive interface, and designed as a OSC-based system to achieve a transparent environment. The internal dataflow is based on OSC, and all nodes communicate through this message format internally. The nodes respond in a similar manner if the message is received from an internal node, as well from an external device. In this way it is possible to control nodes from external applications creating a transparent and scalable environment. It supports Bonjour protocol

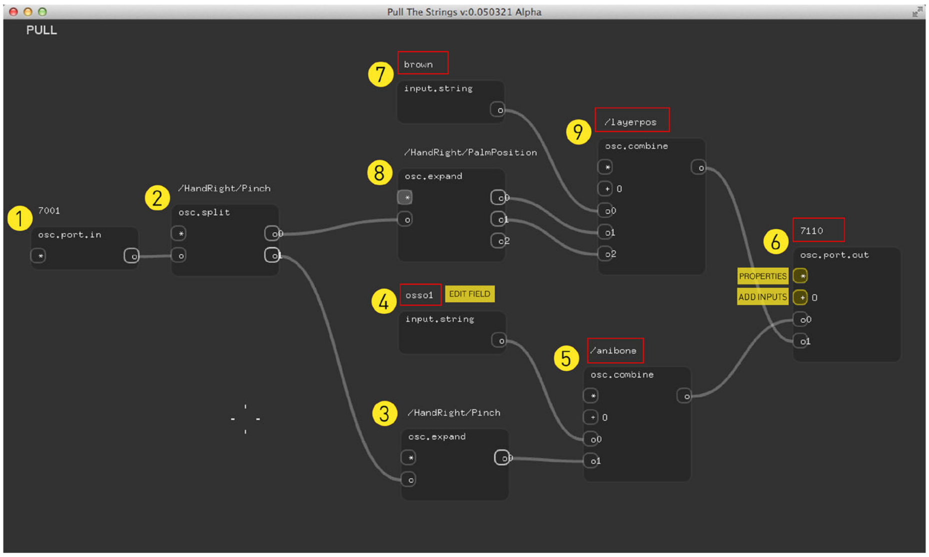


Fig. 7 Pull-The-Strings is a virtual marionette programming environment

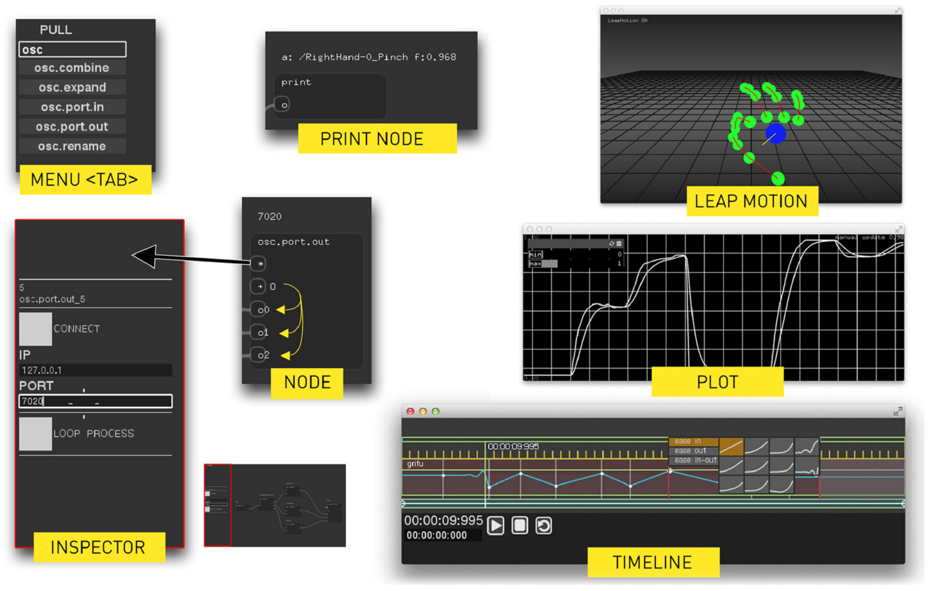


Fig. 8 PTS presents operators for capturing, processing and routing signals using the strings of a marionette as metaphor

to facilitate device and App discovery (i.e. TouchOSC). It is possible to access and control the program features through remote applications, for instance using the RemoteUI. PTS presents many operators (nodes) that help the artist to trace the signals (show.plot) to generate animation (timeline), to process the signals (math operators), or to connect directly input devices (leap motion) (Fig. 8).

Pull-The-Strings was built from multiple Openframeworks add-ons. The main application data-flow environment was based on the ofxDuct. Other components were produced with the help of add-ons such as ofxRemoteUI.

A complex programming environment should provide a *transparent* and direct way of manipulation. The graphical user interface hides all the complexity giving the sensation to the user, that he is handling directly the task (problem). The design of the graphical user interface of this environment provides good signifiers for the application affordances. It allows the user to create, save and load projects, as well as to create, delete and change operators, or to navigate through the canvas. The major operator nodes of PTS can be found in Table 3.

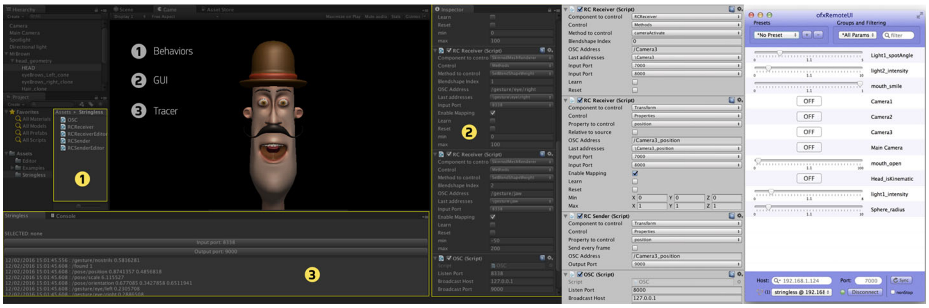
4.2 Stringless for unity - Flexible mapping framework

A Puppet Tool that plays an important role as a bridge between input devices, Pull-The-Strings, and game engines is Stringless for Unity 3D. Stringless for Unity 3D (Fig. 9 - Left) is a remote-control interface that works over the network - an open source flexible mapping framework. This plugin was built for controlling and exposing generic properties of objects using remote devices and applications that support OSC. It provides the GUI to facilitate the mapping procedure, and hopefully facilitate the mapping procedure for non-expert programmers that can easily establish control semantics and link a specific device feature to an object property. For instance, to map a virtual camera orientation to a gyroscope in a

Table 3 Pull-The-Strings node operators

Operator node	Description
input.number	Creates a Float or Int value
input.string	Creates a string value
input.mouse	Captures and sends the mouse X,Y coordinates
Input.XY	Sends a XY coordinates that can be manipulated through a UI panel
input.wii	Captures and sends the WIImote features (accelerometer, IR, buttons)
math.scale	A scalar function with a learning input range option
osc.combine	Combines several OSC messages into one, allowing message syncing
osc.expand	Expands an OSC message into separated nodes
osc.port.in	Opens port for incoming OSC data
osc.port.out	Opens port for sending OSC data
math.dot.vec	Produces the dot product from two vectors
osc.rename	Renames the address of the message
osc.split	Splits messages
osc.rec	Records the OSC messages into a file
osc.play	Loads and reproduces the OSC messages from a file
filter.data	Applies signal transformations, noise reduction
timeline.number	A timeline for animation that returns a float value
print	For debugging, prints in the interface the incoming messages
show.plot	Visualize a plot from the incoming values
leapmotion	Captures and sends the hand and finger motion from LeapMotion

smartphone device. Stringless provides OSC learn functionality that simplifies the mapping process avoiding the need to write the addresses manually. It is similar to the MIDI listening feature employed in music applications. Furthermore, an interactive scale function was introduced that registers the range limits of objects while the user manipulates them in the scene view of the Unity editor. Stringless was already implemented in a Hand-based digital puppetry project to connect the Leap Motion device controller and digital glove puppets [11].



SELECTED: none
Input port: 8338
Output port: 9000

```
12/02/2016 15:01:45.556 : /gesture/nostrils 0.5816281
12/02/2016 15:01:45.608 : /found 1
12/02/2016 15:01:45.608 : /pose/position 0.8741357 0.4856818
12/02/2016 15:01:45.608 : /pose/scale 6.115527
12/02/2016 15:01:45.608 : /pose/orientation 0.677085 0.3427858 0.6511941
12/02/2016 15:01:45.608 : /gesture/eye/left 0.2305708
12/02/2016 15:01:45.608 : /gesture/eye/right 0.2886508
```

Learn ☐
Reset ☐
min 0
max 100

RC Receiver (Script)
Component to control SkinnedMeshRenderer
Control Methods
Method to control SetBlendShapeWeight
Blendshape Index 1
OSC Address /gesture/eye/right
Last addresses \gesture\eye\right
Input Port 8338
Enable Mapping ✓
Learn ☐
Reset ☐
min 0
max 100

RC Receiver (Script)
Component to control SkinnedMeshRenderer
Control Methods
Method to control SetBlendShapeWeight
Blendshape Index 2
OSC Address /gesture/jaw
Last addresses \gesture\jaw
Input Port 8338
Enable Mapping ✓
Learn ☐
Reset ☐
min -50
max 200

OSC (Script)
Script OSC
Listen Port 8338
Broadcast Host 127.0.0.1
Broadcast Port 9000

RC Receiver (Script)
Component to control RCReceiver
Control Methods
Method to control cameraActivate
Blendshape Index 0
OSC Address /Camera3
Last addresses \Camera3
Input Port 7000
Input Port 8000
Learn ☐
Reset ☐

RC Receiver (Script)
Component to control Transform
Control Properties
Property to control position
Relative to source ☐
OSC Address /Camera3_position
Last addresses \Camera3_position
Input Port 7000
Input Port 8000
Enable Mapping ✓
Learn ☐
Reset ☐
Min X 0 Y 0 Z 0
Max X 1 Y 1 Z 1

RC Sender (Script)
Component to control Transform
Control Properties
Property to control position
Send every frame ☐
OSC Address /Camera3_position
Output Port 9000

OSC (Script)
Script OSC
Listen Port 8000
Broadcast Host 127.0.0.1

Light1_spotAngle	0 – 1.1 – 5
light2_intensity	0 – 1.1 – 10
mouth_smile	0 – 1.1 – 1
Camera1	OFF
Camera2	OFF
Camera3	OFF
Main Camera	OFF
mouth_open	0 – 1.1 – 100
Head_isKinematic	OFF
light1_intensity	0 – 1.1 – 8
Sphere_radius	0 – 1.1 – 10

Host: 192.168.1.124 Port: 7000
(1) stringless @ 192.168

Fig. 9 (Left) Stringless for Unity - a flexible mapping framework. (Right) Stringless plugin and RemoteUI application working together

This plugin uses unityOSC developed by Thomas Fredericks to communicate with other applications and devices that support the OSC protocol. A triggering method was implemented to allow remote activation and deactivation of the component instances. In this way, it is possible to orchestrate the play and switch mapping with few instructions. Stringless supports zero-configuration networking that enables automatic discovery of devices, computers and applications through Bonjour, facilitating the connection by avoiding manual specification of IP addresses. Furthermore, a remote control protocol was implemented to facilitate the connection as well as feature discovery. This protocol uses the "ofxRemoteUI" add-on for Openframworks, developed by Oriol Ferrer Mesià, for controlling variables from a remote User Interface (UI). Stringless broadcasts its presence through multicast, sending a list of available parameters for remote manipulation identifying their type, value and range. All the parameters are automatically setup in the client UI application (i.e. (Fig. 9 - Right)). In this way, it is possible to control all the available parameters even without knowing its addresses.

Stringless works above the standard OSC messaging and requires simple setup:

1. Initialize Stringless by dragging OSC to any GameObject in Unity and by setting up the ports with valid numbers (i.e. 7010)
2. Establish the mapping between the action/object and the message, by dragging the RCReceiver or the RCSender components to the GameObject. The RCReceiver allows the control of available features of a GameObject with a OSC message such as from a TouchOSC fader from an Ipad.
3. Setup mapping between the driver (input signal) and the driven object enabling the mapping. You can manually setup, or use the learn button for interactively setup. When learning the values, changes are automatically recorded through interactive modification on the inspector or viewport. For instance, moving the object in the viewport to the extreme positions.

Stringless is limited to the OSC features that are included in Thomas Fredericks UnityOSC. However, it is easy to improve UnityOSC. It is also possible to integrate a different OSC communicator such as the Rug.OSC. The Rug.OSC supports IPv4, IPv6, multicast and broadcast communication, as well as many message argument types including RGBA, OSC Time-tag, OSC-Midi, Blobs.

4.3 Leap IT and face IT strings

Leap IT String (LS) and Face IT String (FS) were developed as Input Puppet Tools. While LS focus on hand motion capture, FS refers to face capture. Leap IT is a multi-platform application that routes the hand motion through OSC. The user is able to choose which joint or gesture to send through the network.

Leap It is an application for recording and routing hand motion (Fig. 10 - Left) using the Leap Motion device. It provides the functionality to load motion data files, which can be useful for users without this device, or to test the motion in the interaction design phase. By capturing the performer's gestures during the rehearsals or building phase, the designer can simulate the performer's hand motion behavior and explore the interaction, without having to repeat the gesture. This is particularly useful when working without the performer. The application presents other functionalities such as the "convert" option, that translates motion data into a OSC-based text file, which is useful for data analysis. The user can also activate a scale function to map the hand motion values within a normalized interval between 0 and 1. In terms of motion data, the application broadcasts the position and orientation of the hand,

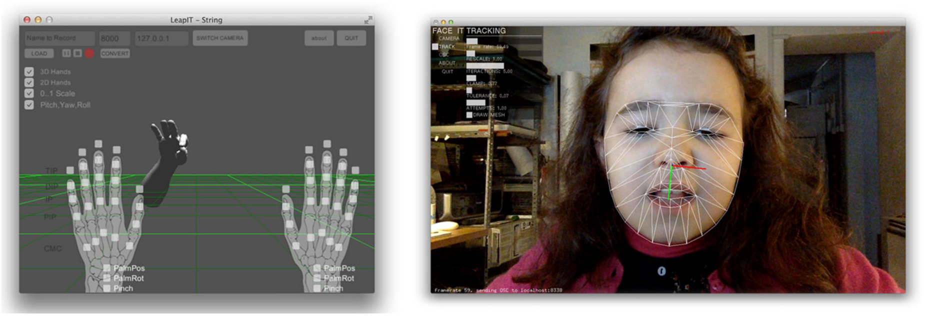


Fig. 10 (Left) Leap IT - Hand strings: A hand motion capture system for Leap Motion. (Right) Face IT - Face strings: Face tracker adapted from the FaceOSC developed by Kyle McDonald

the position of each finger joint, as well as the pinch gesture, which is based on the thumb and index finger distance. The messaging structure is defined in a simple and readable URL style message: /< Which_hand >/< joint ID >< values > (i.e. hand orientation: /HandLeft/PalmOrientation 0.63 0.20 0.56)

An ID number is assigned to each finger as a unique identifier: 0 = Thumb; 1 = Index; 2 = Middle; 3 = Ring; 4 = Pinky. To identify each joint in a finger it was added the name of the respective phalanges Table 4: TIP = tip of the finger; DIP = Distal interphalangeal joint; PIP = Proximal interphalangeal joint; CMC = Metacarpocarpal joint.

For example, to identify the thumb proximal phalange of the right hand the OSC message structure would be: /HandRight/Finger0_pip X,Y,Z.

On the other hand, Face IT (Fig. 10 - Right) is a face tracking application that routes the main facial features through OSC. It uses the OpenFrameworks ofxFaceTracker addon developed by Kyle McDonald, which is based on Jason Saragih's FaceTracker library. Face IT was adapted from the faceOSC example that comes with ofxFaceTracker extending its features providing a graphical user interface. The ofxFaceTracker uses ofxCV which is a OpenCV wrapper for OpenFrameworks [17]

It includes a normalized function to scale all values to a range between 0 and 1. The features are sent via OSC and are grouped into three categories (Table 5): 1) pose; 2) gestures; 3) raw.

1. There are three pose features: the position, the scale, and the orientation. While the position sends a XY screen coordinate, identifying the center point of the face, the scale sends one float with the size of the face, which can be assigned to the Z coordinate of the position. The orientation sends the three direction axes (XYZ) where the user is facing;

Table 4 Finger identification table for Leap IT string

Finger name	Number	Distal	Proximal	Metacarpocarpal	Tip
Thumb	0	Finger0_dip	Finger0_pip	Finger0_cmc	Finger0_tip
Index	1	Finger1_dip	Finger1_pip	Finger1_cmc	Finger1_tip
Middle	2	Finger2_dip	Finger2_pip	Finger2_cmc	Finger2_tip
Ring	3	Finger3_dip	Finger3_pip	Finger3_cmc	Finger3_tip
Pinky	4	Finger4_dip	Finger4_pip	Finger4_cmc	Finger4_tip

Table 5 Features and correspondent OSC messages sent by Face IT string

Category	Description	OSC message
Pose	center position	/pose/position
Pose	scale	/pose/scale
Pose	orientation	/pose/orientation
Gestures	mouth width	/gesture/mouth/width
Gestures	mouth height	/gesture/mouth/height
Gestures	left eyebrow height	/gesture/eyebrow/left
Gestures	right eyebrow height	/gesture/eyebrow/right
Gestures	left eye openness	/gesture/eye/left
Gestures	right eye openness	/gesture/eye/right
Gestures	jaw openness	/gesture/jaw
Gestures	nostril flate	/gesture/nostrils
Raw	points (66 xy-pairs)	/raw

2. Gestures present fundamental data about the facial features with a variety of values that can indicate the facial expressions. For instance, we can identify if the mouth is opened or closed by analyzing the jaw openness or through the mouth height. All gesture values are one dimension float value;
3. The extracted raw values provide 66 XY-pairs of useful information about the facial features, and can be sent directly through OSC.

Messages from both applications can be routed to Pull-The-Strings, to Stringless or used directly with an application that supports OSC.

5 Pull-The-Strings and stringless evaluation

Configuring a set of tools to work together through the network, even when using just one computer, may not be trivial, in particular for non-expert users. This research is driven by the hypothesis that a middleware visual iterative environment, can provide an intuitive way for non-programming users to establish and test new mappings between puppet and puppeteer. This hypothesis presents two questions: 1) the complexity degree in connecting device controllers to applications through an interoperable approach, 2) the intuitiveness of the mapping process through a Graphical User Interface. Thus, it is important to evaluate the proposed model comparing a set of tools and methods with experienced and non-experienced users. Two mapping methods were selected to be evaluated, as well as two mapping environments to be compared in a digital puppetry workflow:

1. Mapping method: a) through Graphical User Interface (GUI); b) through code.
2. Interoperable environment to support digital puppetry workflow: a) Pure Data environment; b) Pull-The-Strings environment. Comparing the degree of complexity in setting up an interoperable environment for digital puppetry, using two distinct middleware mapping tools that act as an interface: the Pure Data, and the Pull-The-Strings.

The mapping procedure can be resembled to programming, making a scripting approach a natural choice. However, mapping in digital puppetry requires experimentation, typically a trial and error approach. A graphical user interface tends to be more appropriate to an explorative-based mapping. Furthermore, a graphical user interface is more intuitive than the traditional scripting approach, which might be more suitable for non-expert users. On the other hand, when handling with multiple applications and devices simultaneously, it is desirable to use a middleware interface that is capable of orchestrating and routing the data in real-time. Multiple mapping environments were discussed early, but there are few that respond to global requirements such as open-source, cross-platform, compatible with OSC, stand-alone application, and with a visual-programming paradigm. Pure Data is a visual programming environment developed mainly to create interactive computer music that responds to the global requirements. In this way, Pure Data was selected to be the middleware mapping environment to be compared with the proposed Pull-The-Strings, which is more oriented to performance animation.

The evaluation measures quantitative aspects in terms of time taken for each mapping task, as well as parameters, such as level of satisfaction, comprehension, learnability, and the intuitive level.

5.1 Experiment description

Two experiments were conducted to evaluate the a) interactive model and b) the Puppet Tools that support the model. A non-probability sampling technique was chosen for both experiments using a convenience sampling. For comparison purposes, two groups of users were defined, the experienced and non-experienced users. While the non-experienced users were students from the first year of multimedia and audiovisual bachelors using the school computers, the experienced users were students from the third year of the same courses with their own equipment. The first experiment was conducted with the non-experienced users, while the second experimented with the experienced group. On both experiments the users had to follow a digital puppetry methodology by performing four exercises. These exercises were presented in four tutorials, which were available in a video format on each computer, and on a printed document. In these tutorials the participants had to produce the interactive design mapping for a digital hand puppetry performance, in particular mapping the pinch gesture to the mouth of a 2D puppet, as well as to a 3D puppet using the same methodology, but with different frameworks. The exercises were time recorded, to compare how much time each user spent to finish the exercise with a specific framework. Before each experiment there was a brief explication about the workflow and the exercises. Before beginning each exercise the participants had to 1) load a recorded hand performance from the Leap IT string, 2) setup the network connections and 3) start a timer. The first and second exercises were based on the communication between two applications, Leap IT string and Stringless for Unity 3D. The third and fourth exercises required the interaction between three applications, Leap IT string with Animata, combined with Pure Data or Pull-The-Strings.

1. In the first exercise participants had to map the pinch gesture from Leap IT string to the blend shapes of the mouth of a puppet inside Unity 3D using the Stringless GUI. This task required the user to setup the network port and to setup the blend shape parameter in the GUI.
2. In the second exercise it was asked to the participants to make the same mapping as in the previous exercise, using the Stringless plugin but through C# code instead of using the GUI.

3. In the third exercise the participant had to map the pinch gesture to the mouth bone of a 2D figure in Animata using the Pull-The-Strings (PTS). First connecting the Leap IT to PTS, then creating the node patch to send the appropriate messages to Animata.
4. The fourth exercise was similar to the third but using the Pure Data as the mapping interface middleware.

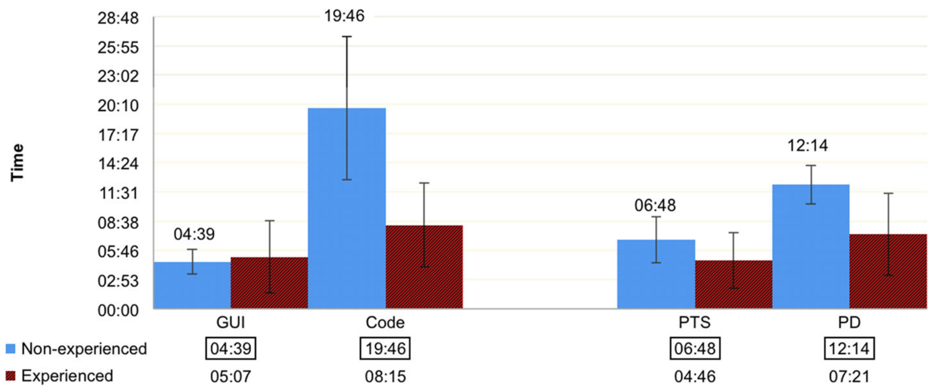
In this way, it was possible to evaluate the interaction model as well as the distinct frameworks and tools. The first two exercises compared the same task using the Stringless GUI or Stringless C# scripting. The last couple of exercises compared directly the Pull-The-Strings with Pure Data environments. At the beginning of the experiment, the participants responded to a survey to evaluate their experience with animation and programming. Then, at the end of each two exercises, the participants had to respond to a questionnaire: reporting the time taken to finish each task; reporting if the exercises worked at the first attempt, and if they would use this method in the future; evaluating from 1 (worst) to 5 (best) the following aspects: intuitive, learning curve, understandable, experience. At the end of the experiment the participants had to respond to another inquiry about the interaction model itself, with a 1 to 5 classification in terms of the design complexity, if they recognized advantages in using the proposed model in the future. The expected duration of this experiment was 2 hours (complete all the exercises, and answer the questionaries).

The first experiment was conducted in a class room with controlled lighting conditions using four Apple iMac computers with 2 Gbytes of memory and all the software already installed. The experiment was conducted with non-experienced users, with 7 participants in pairs and individually with a duration between 1:30 to 2 hours. The age of participants ranged form 18 to 32 years (M = 21.0, SD = 4.97), 4 females and 3 males. All of the participants were starting a multimedia bachelor with few experience in animation and programming.

The second experiment was conducted with the experienced users group in four sessions in different classrooms, with distinct light settings, using the student computers, with variable setups and a variable number of participants in each session. While some of the computers were based on the Windows operating system, others were Mac-based computers. The students had to download, install and configure each tool. There were a total of 25 participants in this experiment with an average age of 23. The students were from the 3rd year of a bachelor in Multimedia and a bachelor in Audiovisual. The diversity of the student's settings allowed to understand how the tools behave in different setups and evaluate the system robustness.

5.2 Results

The overall results obtained from these experiences show that users were able to complete their tasks taking less time with the proposed tools. However, these results are more clear with non-experienced users, which are the main target. The graph (Fig. 11) shows that the participants took less time to complete the mappings with the Stringless GUI than with Stringless C# scripting. While non-experienced users took an average of 4:39 using the GUI, they needed almost 20 minutes to complete the same mapping with code. This difference is not so significant with experienced users, taking 5:07 minutes with the GUI and 8:15 with code. There is a significant difference between non-experienced and experienced users when using the code to establish the mapping, which confirms the hypothesis that GUI is more intuitive than code. This difference can be seen in the questionnaire, with experienced



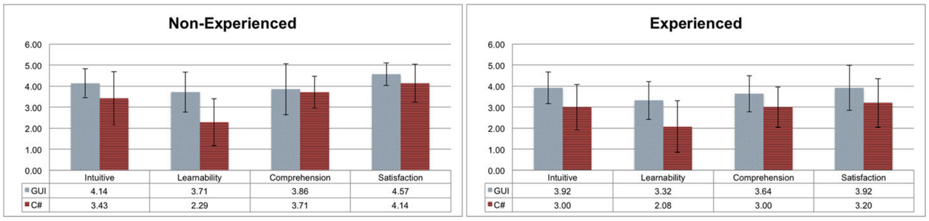
	GUI	Code	PTS	PD
Non-experienced	04:39	19:46	06:48	12:14
Experienced	05:07	08:15	04:46	07:21

Fig. 11 Results from the two experiments, the blue bars report the values from the non-experienced users, and the red bars show the results from the experienced users

users classifying the GUI as an intuitive interface with 3.92 against 3.00 when using the C#. The overall results of learnability, comprehension, and satisfaction from the questionnaire points that both participants prefer the GUI to develop the mappings, as it is possible to see in (Fig. 12).

It is also clear that the participants were faster to complete their tasks with Pull-The-Strings (PTS) than with Pure Data (PD). Again, there are significant differences in the elapsed time with non-experienced users, spending 6:48 minutes to map Leap IT string to Animata using the Pull-The-Strings, against 12:14 minutes using Pure Data. This difference decreases to less than 3 minutes with experienced users. Nevertheless, it is still significant, and by observing the questionnaire results it is possible to confirm that Pull-The-Strings is more intuitive, provides a better learning curve, comprehension, and satisfaction than Pure Data for this type of task (Fig. 13). It is important to clarify that we are comparing different applications and methods for mapping controllers to animation parameters.

Finally, the non-experienced users recognized clearly the advantages of the model with an average of 4,86 in 5. The experienced users had also demonstrated that this model presents advantages for their future work with an average score of 4,08 in 5. On the other hand, the evaluation of the complexity of the interaction model is always based on the previous experiences and knowledge, and must be understood as a reference value. Nevertheless, it is interesting to notice that experienced users considered the model more complex (average of 3,38 in 5) than non-experienced users (average of 2,57). At the end of each experiment, it was possible to discuss with the participants which improvements they would



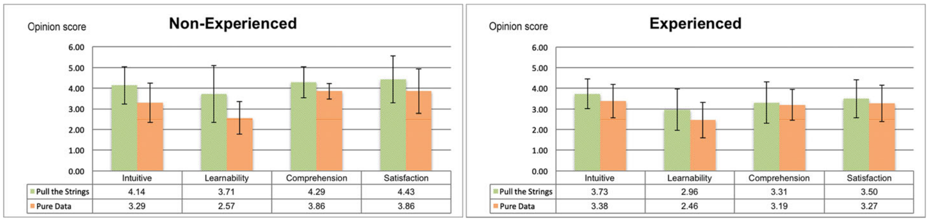
Non-Experienced

	Intuitive	Learnability	Comprehension	Satisfaction
GUI	4.14	3.71	3.86	4.57
C#	3.43	2.29	3.71	4.14

Experienced

	Intuitive	Learnability	Comprehension	Satisfaction
GUI	3.92	3.32	3.64	3.92
C#	3.00	2.08	3.00	3.20

Fig. 12 Comparing two mapping methods with experienced and non-experienced users using Unity Engine: 1) through GUI; 2) through scripting



Non-Experienced	Intuitive	Learnability	Comprehension	Satisfaction
🟩 Pull the Strings	4.14	3.71	4.29	4.43
🟧 Pure Data	3.29	2.57	3.86	3.86

Experienced	Intuitive	Learnability	Comprehension	Satisfaction
🟩 Pull the Strings	3.73	2.96	3.31	3.50
🟧 Pure Data	3.38	2.46	3.19	3.27

Fig. 13 Non-experience and experienced user results comparing the opinion score between: a) Pull-The-Strings environment; b) Pure Data visual programming

like to be implemented, as well new features for future versions. Some of these aspects were included in the last Puppet Tools releases that facilitate the mapping process.

6 Conclusion

The proposed interaction model for digital puppetry allows artists to think in terms of signal flow and help them to create meaningful mappings. A framework was developed to support the interaction model based on the fowling requirements: usability, flexibility, extensibility, and scalability. In terms of usability, the framework was designed as a visual iterative environment allowing non-programming users to establish and test new mappings in a fast and intuitive way. The node-based approach offers flexibility for prototyping. The architecture is extensible, providing a method for adding new functionalities without the need to recompile the code. An interoperable middleware agent was designed to extend the functionalities of applications and devices by combining them in a digital ecosystem. Finally, the system is able to grow when needed, taking advantage of the network environment and the Internet of Things, offering scalability. As a result, it is possible to connect multiple computers and mobile devices, expanding the workflow and creating a collaborative play, where each computer acts as an instrument of a digital orchestra - one marionette controller that expands the manipulation capabilities.

The proposed workflow focus on logic and semantic programming rather than on hardware issues. It provides interchangeable devices and allows the recycling of mappings in an ecological thinking. This workflow is supported by a set of Puppet Tools, based on Open Sound Control (OSC) that were developed as a proof of concept. Designed as middleware, the Puppet Tools act as gears of a controlling mechanism extending the functionalities of applications. These tools were evaluated and compared with other frameworks. The results show that users were able to complete their tasks in less time with the proposed methodology using the Puppet Tools. The results also point to the relevance of this interoperable mapping approach in the production of animation. These open-source tools have been already used by artists to explore digital puppetry and create digital storytelling. Pull-The-Strings facilitates the collaborative process of creating novel and alternative experiments using multi-sensorial digital puppetry techniques and recycle device controllers that are obsolete. It provides a customised and personal environment for performance animation, where the artists are free to combine and choose their favorite tools and devices. Hopefully, this approach will stimulate storytellers to design, create and perform exciting digital puppet plays in real-time, and explore their imagination by producing fictional content turning their dreams into a reality - an alternate reality.

References

1. Anson E (1982) The device model of interaction. In: Proceedings of the 9th annual conference on computer graphics and interactive techniques. ACM, New York, pp 107–114
2. Bleser FD, Smedt TD, Nijs L (2002) NodeBox
3. Bodenheimer B, Rose C, Rosenthal S, Pella J (1997) The process of motion capture: Dealing with the data. In: Thalmann D, van de Panne M (eds) Computer animation and simulation '97: Proceedings of the eurographics workshop in budapest, Hungary, September 2-3, 1997. Springer Vienna, Vienna, pp 3–18
4. Card SK, Mackinlay JD, Robertson GG (1990) The design space of input devices. In: Proceedings of the SIGCHI conference on human factors in computing systems. ACM, New York, pp 117–124
5. Coduys T, Ferry G Iannix: Aesthetical/Symbolic visualisations for hypermedia composition. In: International conference sound and music computing, The Hague, Netherlands, pp 194–196
6. Francis P (2011) Puppetry: a reader in theatre practice. Readers in theatre practices palgrave macmillan
7. Hinckley K, Wigdor D (2012) Input technologies and techniques Jacko J (ed)
8. Jacob RJK, Girouard A, Hirshfield LM, Horn MS, Shaer O, Solovey ET, Zigelbaum J (2008) Reality-based interaction: A framework for post-WIMP interfaces. In: Proceedings of the SIGCHI conference on human factors in computing systems. ACM, New York, pp 201–210
9. Leite L (2018) Virtual Marionette: Interaction Model for Digital Puppetry. Ph.D. thesis. University of Porto, Porto
10. Leite L, Amândio A (2020) Solitária - Gestural Interface for Puppetry Performance. In: International conference on live interfaces. trondheim, Norway
11. Leite L, Orvalho V (2017) Mani-pull-action: Hand-based digital puppetry. Proc ACM Hum-Comput Interact 1(EICS):2:1–2:16
12. Leite L, Torres R, Aly L (2018) Common spaces: Multi-modal-media ecosystem for live performances. MATLIT Materialities of Literature 6(1):187–198
13. Leite LM, Lafontana M (2016) Digital theatrograph: Cinematographic puppetry. In: Proceedings of the 1st international workshop on multimedia alternate realities. ACM, New York, pp 3–8
14. Morrison JP (2013) Flow-Based Programming application developers' news (1)
15. Rudraraju V (2011) A Tool for Configuring Mappings for Musical Systems using Wireless Sensor Networks. Chulich School of Music McGill University pp 1–90
16. Walther-Franks B, Malaka R (2014) An interaction approach to computer animation. Entertainment computing, Elsevier, pp 1–37
17. Wang Y, Lucey S, Cohn JF, Saragih J (2010) Non-rigid face tracking with local appearance consistency constraint. Image and Vision Computing 28(5):781–789
18. Wright M, Freed A, Lee A, Madden T, Momeni A (2001) Managing complexity with explicit mapping of gestures to sound control with OSC. In: ICMC