Outil auteur pour la réalisation d'oeuvres pluri-artistiques interactives

Tommaso Bianco *ATIAM M2* 2006/2007

Aknowledgments

I would like to thank Olivier Warusfel and Olivier Delerue and the equipe "Acoustique des Salles" for having me welcomed for the internship position.
Particular thanks to Olivier Delerue for the time he devoted to me, both for academic questions and for pleasant company breaks.
Thanks also to our projects partners at Irisa, with a special remark for Gildas who supported my "programming skills growth" during the project development.
Cheers to my master fellows!

Résumé

La réalisation des oevres pluriel-artistiques et interactives reste au jour d'aujourd'hui un complexe et pas très approprié processus pour les artistes. Quand les différents composants du logiciel partagent les données et communiquent ensemble, particulièrement pour échanger la capture, l'interaction et d'autres techniques pour l'analyse et la synthèse audio et visuelles, l'auteur habituellement doit faire face à des caractéristiques de bas niveau pour lier les différents environnements, en implémentent l'exécution ou la communication par des bibliothèques externes la plupart écrites en langages de programmation. Actuellement les paradigmes de communication sont réalisés par des formats standards tels que Open Sound Control ou le Midi, toujours avec une syntaxe de bas niveau, et les " phrases plus abstraites sont récrits chaque fois sur ces formats, habituellement avec la conséquence de rèduire le travail de programmation effectué pour la seule particulière réalisation. Le projet de ConceptMove se concentre principalement sur ce problème, essayant de rapporter un métalangage aussi général que possible, en laissant faciliter la communication et dans le même moment donnant les moyens pour une conception plus élevée pour l-interation complexes.

Abstract

The realization of plural-artistic and interactive performances still remains nowadays a complex and not very suitable process for artists. When different software components come to share the data and communicate together, especially for interchanging capture, interaction and other techniques for audio and video analysis and synthesis, the author usually has to face low level specifications for setting the links between the different environments, venturing on the implementation or on the porting of external libraries for the most part written in programming languages. Currently the paradigms of communication are realized through standard formats such as Open Sound Control or Midi, however still with a low level syntax, and the more abstract "sentences" are re-written each time on these formats, usually with the consequence to retain the overall programming work for the particular ad-hoc realization. The ConceptMove project focuses mainly on this problem, trying to yield a meta-language as general as possible, allowing to facilitate the communication task and in the same time giving the means for a higher conception for complex interactions.

Contents

List of Symbols

AOP	Aspect Oriented Programming
API	Application Programming Interface
CSS	Cascading Style Sheet
DOM	Document Object Modell
DSL	Domain Specific Language
DTD	Document Type Definition
EDA	Event-Driven Architecture
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
MDA	Model Driven Architecture
OSC	OpenSound Control
SAX	Simple API for XML
SOA	Service Oriented Architecture
SOAP	Service Oriented Architecture Protocol
SMIL	Synchronized Multimedia Language
UML	Unified Modelling Language
URI	Unified Resource Identifier
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
XML	Extensible Markup Language
XPath	XML Path language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation

Chapter 1

Introduction

Mixed media, interactive performance, mixed-art-form performance, together with other combinations of digital-era words, have fully entered the universe of paraphrases to refer to the contemporary artistic creation. The blooming of new available miniaturized hardware, as well as the growing of a multitude of software applications for the elaboration of audio and video data, have become nowadays the common mediums for the stage or for the museum profession. Moreover, the digital representation has introduced a gap previously unknown for the human-human dialog, the communications inside the virtual world. As a matter of fact, the virtual world, being completely dependent on an a-priori defined behavior, it must be specified by a finite number of entities and relations. Such a requirement brought about the birth of a number of languages and grammars, in order to store the information for visualization or for future reproduction. More properly called "narrative engines languages", "scenario languages" or "narrative description language", these paradigms allowed the description of virtual worlds in their evolving "story" of connections and events In the same time, the wish to integrate the virtual and the real world, led to the procreation and propagation of different types of languages to interchange data between applications. Among these some have become standards, such as MIDI[1], currently in phase of developing towards more extensive possibilities with the 2.2 Specifications, others have become history because of missing widespread adoption (for example ZIPI elaborated in Berkley[2]), others are still in a testing process by user community, namely the OSC[3] and SKINI[4] formats respectively from Berkley and Princeton Universities. Open Sound Control has gained more popularity in these last years, especially by the reason that its open-ended, dynamic, URL style symbolic naming scheme allowed developers to easily integrate it into pc-mobile TCP-UDP networks with a little effort in terms of platform re-coding and multiple language porting.

Nevertheless the jump between low level specifications for the information transmission over these languages and the overall conceptual virtual and real worlds interaction still remains wide and time to time moulded on the personal tastes and ideas of each author. This glitch may be a consequence of the fact that the level of communication between different softwares, which integrate elaboration and interaction in the media (audio-video) domain, is still rudimental because steady to a low level network protocol.

The work accomplished in the present thesis is part of the project **ConceptMove** between the three partners Ircam/IRISA/Danse 34. A detailed objectives description will be given in Chapter 3

In Chapter 2 we will present some of the most known and commonly used instruments for the realization of artistic productions. Some of them have a long development history and have gained a plurality of functions, also with the help of a large community of user; others are bound to specific

[1] http://www.midi.org/

[2] http://www.cnmat.berkeley.edu/ZIPI/

[3] http://www.cnmat.berkeley.edu/OpenSoundControl/

[4] http://www.cs.princeton.edu/ prc/SKINI09.txt.html

domain uses, but nevertheless proposed interesting resolutions that could turn useful in considering a pluri-artistic application. Along with their description, we will depict some positive features and drawbacks, in order to show how the **ConceptMove** application proposes for their resolution or motivate the adoption of foregoing concepts. Some of the most predominant media languages will be briefly described, both from the graphic and musical field and from the upcoming web technologies.

In Chapter 3 we will detail the **ConceptMove** project objectives and show the progress realized during this thesis internship. We will describe the language elements, the code generation transformations that we performed, and some examples in order to test the functionality of the modules developed.

At the beginning of our experience the specification of the meta-language was still to be completed. The skeleton structure and the main elements were decided, but still some sessions were held for further refinement discussions, and for starting to consider which aspects and how could be translated into a concrete implementation for the following phase. We aquainted skills in markup language manipulation instruments, most of which still in draft version. We looked among literature and current market offer for the most suitable concepts and technologies for performing the code generation process, experimenting and evaluating different methods. The mediums for its realization have been proven mature but still unstable, because of the lack of proper standards and because of a plurality of different underlying philosophies. We believe however that this contest will become more and more discussed, and particular attention should be given in the field, expecially when, as in artistical environments, the facility for the user in computer science development is among the main objectives (clearly always with an eye on performance). The code tranformations undertaken during this thesis revealed the passage of concepts between different language paradigms feasible. If the upstream starting point crisply surrounds the user intentions (expressed by means of the meta-description) the succeeding passage to its low code rendition only concerns with a good algorithmic design study.

Nevertheless, our internship period revealed too short for touchin and resolving all possible issues. Therefore, in the final chapter, we focus on possible future directions, both as accomplishment of the national project targets and beyond for later additions.

Chapter 2

A survey of past and present similar projects

In this chapter we will list some of the systems and formats considered as the most prominent int the conception of a large number of pluri-artistic presentations. In all-accomplishing environments for general digital media elaboration, or simple communication protocols, the technology has gained more and more importance for helping the artists setting down their mixed-reality narratives and installations. However sometime the technical conception of the technology syntax becomes addressed more to other technicians than to a generic user. We will separate the review into three main axes, remarking each description with features and restraints: the all-accomplishing systems, the interaction/relation definition languages, and the communication layer.

2.1 Interactive Multimedia Systems

The Interactive Multimedia Systems represent for the "digital" artist what the glue and cutter could have represented for the plastic arts: the means for the art expression transformed into electronic or computer-generated sound, video, animation, and interactivity, all together in the same application. In the following we will report some inspiring ancestor models as well as current young releases.

2.1.1 HARP

HARP is an hybrid architecture for the representation and planning of real-time processing of music and multimedia knowledge [26] [11]. The creation of this system, originally issued from the researches conducted since year 1983, was motivated by the development of a software for computer assisted composition, performance and analysis, that could be furnished with the fusion of traditional computer science techniques and concepts from the artificial intelligence domain. The HARP software represents the evolution of the ideas implemented into Musical Actors; the latter was a sort of Actors system based on Petri Nets models for the inclusion of certain temporal and causal relations to govern the relations between the states and the transitions entities. The system became more elaborated with the development of a more powerful approach based on semantic network of frames [ref to Key Music], and further with the adoption of a semantic network language KL-ONE.

The hybrid architecture of HARP was intended to give both the development environment, in order to represent the musical and general multimedia knowledge by means of a user-design interface, and the run-time environment. The latter supports the real-time, multimodal interaction and can be thought as a prolongation of the human mind and senses. The system is able to represent and carry out plans in real time for manipulating knowledge according to the user's goals. HARP is grounded on a hybrid integrated agent architecture.

HARP was used to help the development of artistic compositions, in two particular multimedia domains: the theatrical automation project, where the system is delegated to integrate and control sound and computer animation of humanoid figures interacting with real actors on the stage, and for museum application, where the application is faced to perform real time interaction with visitors through a set of sensor inputs.

An example of its application can be found in the project VSCOPE, in which the acquisition of gesture human movement is interfaced with both a sound rendering engine and a visual synthesis of human personages. In particular, a dancer is directly controlling by means of its movements, the different aspects of the sound and virtual face synthesis.

The HARP project was without any doubt a pioneer project in the field of interactive systems, by the fact that included concepts of artificial intelligence in the process of creation of the work as well as in its real time behaviour. We believe that this aspect will be more and more exploited in future applications for artistical applications. As a counterpart, it is too much complex for an artistic user. It demands a too much detailed manual setting in the phase of the conception of the entities network. It has been used with positive results, but only when the artists had a good teoretical knowledge of the system or when they had the possibility to work directly with the application developer.

2.1.2 EyesWeb

The EyesWeb platform represents the will to simplify the concepts developed in Harp in order to allow a more direct utilization. EyesWeb is an open platform originally conceived for supporting research on multi-modal expressive interfaces and multimedia interactive systems. By means of the visual programming language (connection of graphic modules for data and message flow as the MaxMSP and PureData platforms) it facilitate the implementation of computational models in order to analyse and extract features from gesture data. External libraries enable to interface with hardware sensors for the capture of non-verbal expressive communication (e.g., human full-body movement, music), while internal software prebuilt modules allow the processing of expressive gesture in movement, MIDI, audio, and music signals. The interconnection of the software blocks is organized in patches, which can be further used as modules for a higher-level patches. EyesWeb includes a open source and documented SDK which enables the user to extend the core functionalities by coding new modules, data types and libraries. The current version of the software runs on the Win32 COM/DCOM standard API, and it supports plug-in standards (Steinberg VST and FreeFrame) as well as output musical data formats (OSC). Automation support is provided. This let developers invoke EyesWeb from other languages, such as Microsoft Visual Basic. The execution of a patch can be controlled from external applications. Any scripting language that supports automation can be used for this purpose (e.g., Python, VBScript, JavaScript, etc.). The overall set of libraries grouped by functionality can be resumed as follows:

- Input & Output communication: support for frame grabbers (from webcams to professional videocameras), wireless on-body sensors (e.g., accelerometers), audio, MIDI, OSC, TCP/IP, serial, network, keyboard, mouse communication

- Mathematical and filtering algorithms (e.g., operations with scalars and matrices).

- Motion analysis: motion trackers (e.g., feature tracking, tracking of multiple colored blobs), modules for extraction of global features from movement (e.g. amount of detected motion), analysis of the use of the space.

- Mapping of extracted features in real-time generation of audio and visual outputs.

Each module of the above functionality groups may be in turn a passive module (e.g., filters) or an active module with an internal dynamics (i.e., modules which receive inputs as any other module

but may send outputs asynchronously with respect to their inputs). Being active or passive, however every module is settable at runtime by accessing a side object options menu, which allows to change interactively its internal attributes.

EyesWeb has also been widely employed for designing and developing real-time dance, music, and multimedia applications.

2.1.3 Max/MSP+Jitter / Pure Data+Gem

Max/MSP represents the outstanding example of environment for the interactive art. The origin of Max takes place in Ircam, with the development of the *Patcher* editor by Miller Puckette in the mid-1980s, and during the following years evolves into the ISPW real-time synthesizer for the NeXT workstation. At the end of the decade the system undergoes a branch: part of the code get licensed to David Zicarelli, who will extend it ending up to MaxMSP sold by his own company, Cycling '74, while the remaining developers keep on the development of Max/FTS. In 1996 Miller Puckette released an entirely re-designed free software program called Pure Data, which is very similar in scope and design to the original Max program, but has a number of fundamental architectural differences.

Present on the scene for more then a decade, MaxMSP represents the most succeeded visual programming software for the music and audio data elaboration. The complete graphic paradigm exploits its easy to break into the artist community, while its modular, powerful and programmable architecture to please the academic application.

Both the systems can exploit composition (generation of musical structures, also with techniques derived from mathematical and aleatory models), musical accompaniment (production of mixed works where the electronic part is being produced as a reaction to the play of an external instrument), peripheric control (MIDI, OSC, ..), complex audio synthesis and elaboration. The recent addition of real-time video, 3-D, and matrix processing capability (Jitter, Ftm, Gem, GridFlow) opened their operational field to visual manipulation, contributing for the diffusion as sole platform among vjing performances and interactive installations. The two systems are widely used for pedagogical musical workshops and conservatories) and academic (audio analysis-synthesis research) purposes.

2.1.4 VVVV

VVVV is a patched-based visual programming toolkit for real-time graphics, audio and video processing and interaction control. It was originally developed by Sebastian Oschatz e Joreg Diessl at the "Meso Group" of Frankfurt around 1998 as a mean for easily support the thei artistical installation produced in the centre. Its free availability (even though not open source) and an easy graphical interface for the prototyping and development contributed for its rapid diffusion in the all international artistic community. It is similar in modular composition paradigm to the Max/MSP and PureData systems, but it is more focused on real-time visual rendering and and show control. VVVV implements a good hardware acceleration for graphic operations, but because of its dependence on the DirectX libraries, it lacks of portability and can only be runned on Windows platforms. Nonetheless the system is also provided with an audio processing unit (through the AudioIn and FFT nodes) built on the DirectShow multimedia architecture, which provides a basic set of functionalities for the FFT computation and for audio sampling. Besides creating and transforming content for the audio and video media, vvvv is also conceived for a good support for receiving input from and generating output to various external devices. It interfaces easily with a wide range of input and output for low level protocols such as TCP, UDP or RS232 for communication with other computers and software, and for higher level protocols, allowing for communication via MIDI, DMX, ArtNet, OSC, HTTP, IRC etc.

One of the most importants newnesses of the VVVV environment, are the concepts of Spreads/Slices and BoyGrouping. The "spreading" technique is an abstraction that refers to the act of distributing different values across a set of objects, similarly to the concept of vectors, arrays or

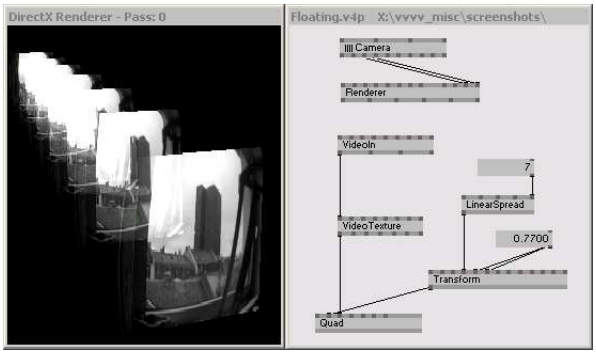


Figure 2.1: Spreaded video floating in space in **vvvv**

lists in other programming languages, but (said to) lacking most of the overhead usually associated with these constructs. Basically any pin or connection throughout the boxes can hold any number of values. These indivudal values are the so called "slices" while the one dimensional list of slices is referred as the "spread". Hence all data can be interpreted as a spread, where normal values are just a special case of a spread with only one element. vvvv contains also many spread graphic renderers; by connecting a spread to a render object make the data to be drawn multiple times, yielding the possibility to master complex behaviours for a large group of objects. One of the main purpose which the vvvv system was conceived for is managing the redirection of media elaboration on different screen systems, as usually desired for multi-projection setups. This intention has leed to the implementation of the so called *BoyGroup* concept. A boygroup is a group of client computer connected via a TCP/IP network. The process for the local connection is made easy by the software, because it demands just a vvvv startup with the prompt information that we're starting a client or server machine and on which node we will be places (something a la `/vvvv /client 129.102.72.133`). Once server and clients are started and inside the server the arbitrary sets of clients are gruped by means of a *Boygroup* node, we are able to perform operation on the aforementioned node and observe the result on the all clients included. The remote control extends to the web protocols, too : a *http* server node allows the direct serving of web content like web-sites and images, a feature that proves to be very useful for remote administration of vvvv installations.

VVVV provides also some nodes for dealing with XML files: it supports the reading of XML inputs as well as the use of XSLT Stylesheets for the transformation of XML meta data into "String with Linebreaks".

In conclusion, even if it doesn't present the power of the combinations Max/Msp+Jitter and PD+GEM, vvvv seems to be more user-friendly and direct than its concurrents. It presents some winning features for the visual and communicating context, and exploits a Windows platform in some aspects left gaunt for artistical production softwares.

2.1.5 Aura

Aura is an evolving software architecture and "real-time middleware" implementation conceived by Roger Dannenberg *et al.* at CMU. Aura can be seen as a kind of a sound synthesis evolution of the real-time object system W [21], a programming environment originally meant to perform network messaging and data stream for the control of sound, graphics, devices and computation. The main underlying concept of the both implementations is to take advantage of developings in network architectures and thus increasing communication performance in order to provide a distributed system capable to perform real-time sound processing by means of messages streaming. The aspects faced during its "evolving" advance direct toward the message passing protocols, the objects inter-

connection, the avoidance of shared memory, the grouping of tasks and objects according to latency requirements, networking and communication issues, debugging, and the scripting language.

As a distributed system architecture, the running software can be splitted into running processes (each organizing its own threading hierarchy) over local-area network interconnected personal computers. The architecture is organized into different layers of abstraction: *spaces* are representing a shared address space with multiple concurrent threads, *zones* consist of a single thread and a set of objects that share the same thread and grouped according to real-time requirements (latency), and *objects*, which are meant to be the atomic elements to be sent or received.

An artistic application of a relatively recent state of the Aura architecture has been presented as the interactive installation "Watercourse Way" for the Selby Gallery in 2002 [19]. The piece is a sort of interpretation of a traditional chinese treatise, which describes a particular attitude and perception as a code of life and livelihood. This concept drove the use of water, as a symbol of freedom and fluidity, in order to encourage the investigation of simultaneity, synaesthetics and the interdependent relationships of time, space, movement and form. The purpose of the installation is to invite the spectator to remove the rigid boundaries and definitions of sound, light and visual art. In order to earn this purpose, the water (symbol of the fluidity) is used as the main interactive link between the dancer/spectator and the light and sound producing system. In a nutshell, the light reflections of a pool of water onto a screen are filmed by a camera and processed by a computer to perform a real-time synthesis of sound (by means of a kind of "scanned synthesis" [52]) and light effects. The Aura system has been used to interconnect the different data domains, such as video audio MIDI and computer graphics. It implements both message passing and scheduling of the running "objects", which are organized in the different "zone" threads; for example, the video frames are processed by the video object to obtain spectra, and the resulting data is sent as an Aura (data) message to the spectral interpolation objects running in another running thread (with higher priority).

As an integrated solution for the design of distributed, real-time, interactive, multimedia programs, experience with Aura offers lessons for designers (and Dannenberg is certainly one of the most pointed people to do it). The combination of a high performance C++ sub-layer with a runtime interpreter for a customized scripting language (Serpent [18]) demonstrates a good concurrency between an optimized effective architecture and a user interaction facet. However, is Aura proposing yet another killer application? The architecture of Aura is indeed intended for interacting with striclty compatible modules or libraries, therefore the presence of external environments must be deeply adapted. When disclosing the underlying implementation for the message passing protocol, we understand that the authors envisioned a programming model with a deep intersection between controlling and controlled processes: each controlling process typically mantains the references to objects and subobjects and therefore send messages directly to the end receiving objects. This reveals somehow in opposition with the OSC "philosophy", where the communication between processes is gluttered into a sole channel (see Fig. 2.2).

2.2 Interactive Multimedia Languages

2.2.1 SMIL

SMIL is the acronim for Synchronized Multimedia Integration Language, and it represents a format description for the synchronized integration of different typed of multimedia data. The purpose of the language is to facilitate the writing of interactive multimedia presentations, with a high level description of its temporal behavior, its associate hyperlinks for media objects and its presentation layout on a screen. It is a XML-based language that reached its 2.1 specification and started the 3.0 draft version in December 2006, with the aim to add more functionalities with a particular consideration for the mobile industry. It is addressed to choreographing multimedia presentations and aims to enable "authors to specify and control the precise time a sentence is

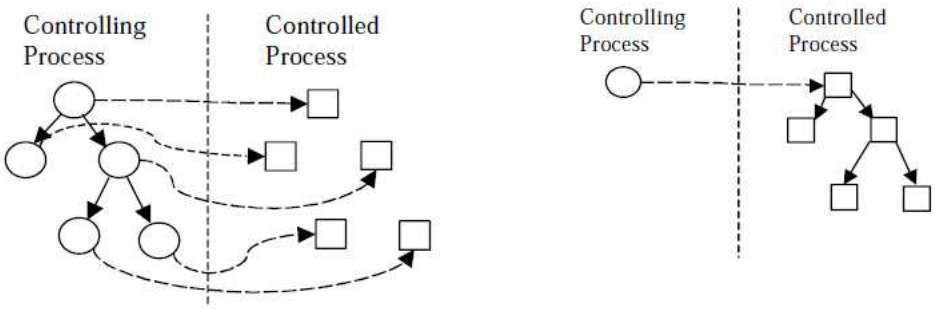


Figure 2.2: The different behaviours of Aura(left) and OSC(right) messaging philosophy for process control

spoken and make it coincide with the display of a given image" [1]. The language syntax, for which we report a simple example in the following lines, is based on the modularization approach, which is derived from the XHTML Modularization concept: a module is a collection of semantically-related XML elements, attributes, and attribute values that represents a unit of functionality. The SMIL 2.1 defined 10 major functionality groups: [Animation, Content Control, Layout, Linking, Media Objects, Text, Structure, Timing and Synchronization, State, Transitions]; of these, the timing and synchronization represents the core of the language specification. Each of the functional groupings represents a collection of individual SMIL modules (in a varying number per group), each of which defines elements and attributes intended for a particular multimedia issues in a reusable manner.

A SMIL presentation consists of a composition of indipendent media objects. The way these objects may be connected in time each other can be essentially resumed by three time container types:

- seq = sequential time container : the children of a seq node container are executed in sequential order, so that for each element the successor child can't bagin before its precedessor child completes.
- par = parallel time container : the children elements are rendered in parallel; this doen not impose that they have to start at the same time, but that they share a common timebase defined by the container and can be active at any time that the parent is active.
- excl = exclusive time container : only one of the children can be active at a time. The starting order is usually imposed by events attached to each child. A button click event, or the activation of another object (see Fig. **??**), could determine the starting of an element "impedendo" the activity for the others.

An example of the different behaviour of the time caontainers is shown in Fig. 2.3.

The timing core modules allow to discretely define the temporal behaviour of media objects. For each media component we can derive its begin moment and active duration (by means of attributes `begin`, `end` and `dur`) as well as the synchronization aspects, being them related to the parent container (for example a parent that ends when a named child ends, thorough the `endsync` attribute) or with user events (as for a `button.activateEvent`). However, the time attribute is not always a straight value, because it can refer to two different behaviours: a resolved or and unresolved time moment. A resolved time has a calculated time relative to the global presentation time; an unresolved time exists logically in the time model, even though it cannot yet be calculated and thus is not concretely part of the scheduling. With an event-dependent activation, a sequence may occur at an unspecified time,

[1] http://www.w3.org/AudioVideo/Activity.html

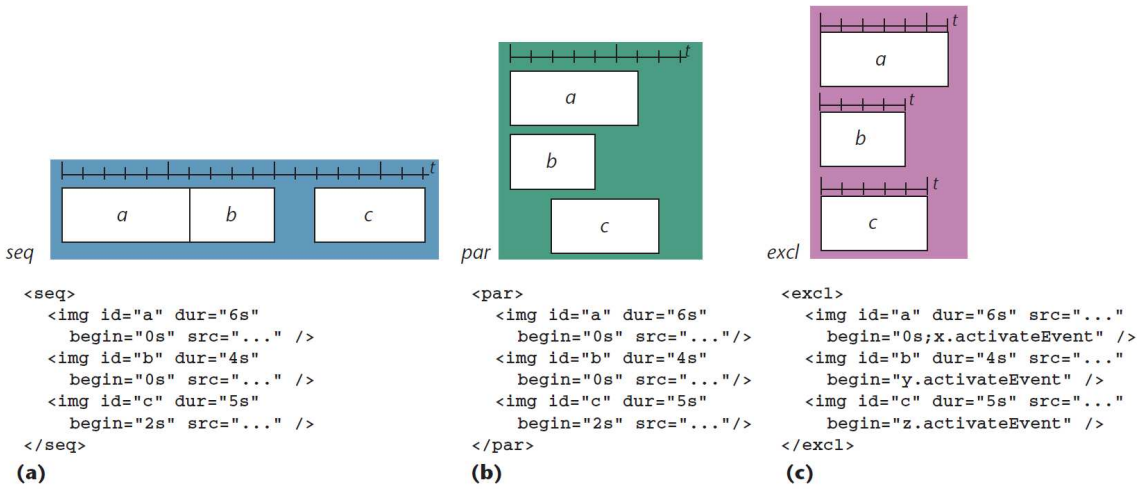


Figure 2.3: The different types of time container : (a)sequential (b)parallel (c)exclusive

implicitly specified by a link traversal or other external activation method, but becoming resolved by the particular activation only at runtime. To face this unpredictability in time synchronization the language furnishes some high level control attributes: `syncBehaviour` for defining the possibility for time slippage in the presentation timeline, `syncTolerance` for the amount of admissible slippage and `syncMaster` to provide the newly master timebase element. The parallel and sequential relations, which are actually a translation of the *equals* and *meets* interval relationships, give the possibility to specify most temporal interactions for the orchestration of a multimedia document [25], but they introduce some congruence issues when using complex cross references, and present a meagre control when going beyond the parent-child relations and trying to estabilish more tangled relations. The introduction of concepts from Petri Nets into Synchronized Multimedia Languages [13] [27] actuated new discussions for the temporal relations verification at runtime, and flew into interesting researches for interference detection, both in the time and spatial (for screen appearance) domain (as the Reachability Tree for spatiotemporal interference detection in [50]). However the option for setting interval based relations in the language syntax still lacks the bacis set of the Allen relations.

Moreover, the layout and compositing graphical techniques are addressed just to one screen output. If dealing with multimedia choreographic works, a multi screen display should be taken in regard, and if composing for different virtual worlds, the artist should be given the possibility to point the output video toward a particular display, with the possibility to easily relate objects inside different 3d systems' coordinates.

In conclusion the SMIL prooves to be a good and promising language for multimedia presentations. It begun to be widely adopted and numerous researches are being conducted in order to add more features for the next 3.0 version. SMIL players have also been designed to play dynamic SMIL documents, which can be modified at run-time with the execution of scripts (such as XML Events and ECMAScript [37]).

2.2.2 VRML and X3D

X3D (Extensible 3D) is a file format for the description of interactive virtual environments developed by the Web 3D Consortium[2] as an evolution of the Virtual Reality Modeling Language (VRML). It is a non-proprietary format and it became ISO standard in 2004. The syntax used for the encoding of the scene graph is based on the XML syntax as well as on the syntax of the VRML97. The scene is substantially encoded by a tree, where the leaves represent the entities of

[2] http://www.web3d.org/

the virtual environment and the internal nodes represent the spatialtransformations between them. The extension of the tree structure to the graph structure is due to the fact that when two nodes refer to the same entity are linked by an arc, thus causing a link between different sub-trees. In the reading phase for the visualization, the player executes a "tree" inspection inserting into the scene each entity as it meets the respective node.

The main purpose of the adoption of the XML syntax was ability to be easily integrated into the web services and distributed systems, allowing an efficient real-time renderingbehaviorr for sharing interaction in network users cooperative tasks, as well as the ease of further extensions[3]. New node entities can be defined by the user through the insertion of new prototype structures, simply performing a hierarchical composition of the node with standard elements or with previously prototyped entities; by naming the new element with the "DEF" command, it can be later reused by recalling (a kind of heredeitance relation) its structure through the "USE" function.

The communication between the entities may be performed by a routing attribute; the nodes may be defined with input and output fields for external interaction with the internal attributes, and the changing of state of its accessible fields may be communicated recursively to the related nodes.

The geometric description of the virtual environment can be furthermore extended by a time behavior. Each node with temporal description (subclassing the abstract node type X3DTimeDependentNode) contains the main input and output fields *startTime*, *stopTime*, *loop*, while *elapsedTime* remains availbale only for output. The former allows to retrieve the elapsed seconds since the activation of the node. A *isActive* parameter can also used for retrieving the running state of the node. Clearly for each node dealing with media supports is mandatory to inherit from X3DTimeDependentNode; therefore the standard classes *AudioClip*, *MovieTexture*, and *TimeSensor* are by default derived from it.

Concerning the user interaction aspect of the language, each X3D scene element is furnished with *..Sensor* typed nodes. Performing an action on this type of nodes causes the scene to change and the user event to be reported by the HTML text. There are different kind of sensors, but they all can be grouped into two types: the ones for events automatically generated by the movement of scene elements (such as an object entering a region) and the ones coming from the hardware user input (mouse and keyboard).

It is clear that the main purpose of the X3D language is the ability to easily construct interactive 3D applications, either stand-alone or web-based. The XML syntax doubtless allows a great efficiency in message routing between the elements of the environment, but however the user interaction level still remains limited to the habitual web and software inputs, such as editors controlled by the mouse and the keyboard, facet that would become compromising when trying to model a choreographic work or an interactive installation. Only recent conferences for the 3d Web showed an interest for developings in this direcion [51].

Some extra features for the time manipulation have been proposed with the addition of a TimeClock node [4]: control of the speed of the animation, backwards play, repetition of specific time interval of animation or instant access to key-frame. However this node addition only addresses to simple time manipulations and doesn't account for more complex qualitative relationships. Moreover the all time architecture is based on the browser timer engine, which could reveal poorly customizable and low performing.

2.3 Music Communication & Control: Midi and OSC

MIDI is a well known standard protocol for communication, control and synchronization between electronic musical instruments, computers and other interfaces. The protocol is addressed to purely event-driven communication, in the sense that it does not transmit audio od signal data, but only

[3] see International Conference on 3D Web Technology for recent developings

event messages. Since its introduction in 1983, it became widely adopted, expecially by industry, and still remains the main used protocol for musical systems interoperability. The MIDI protocol has also been used as for control in applications other than music, including show/theatre control (light and special effects), sound design and recording system synchronization, animation parameter control, computer networking. One of the most outstanding platforms for this format control is Midishare, developed by Grame[4]; Midishare is a real-time, multitasking MIDI based musical operative system which aims providing inter-applications communication by performing the routing of the events between its server layer and the client applications. The underlying idea of its design was to realize a real-time oriented server application, with a routing capability of all the events between the applications or towards the midi ports, maintaining a central time stamp with millisend resolution. It supplies on the same machine the possibility to share the MIDI input and output ports, and to exchange messages or data stream in the MIDI format.

OpenSound Control (OSC) is a "open, transport-independent, message-based protocol developed for communication among computers, sound synthesizers, and other multimedia devices"[56][57] developed at CNMAT[5]. It was admittedly not designed for a particular transport layer[55], and this ductility helped, together with higly encouraged community partecipation, for a wide and rapid use diffusion among real-time sound and media processing environments, becoming fstly integrated into web interactivity tools, software synthesizers and a large variety of programming languages and hardware devices. It easily integrates into modern networking technologies, both for cable based communication (busses, USB and Firewire, Ethernet, ...) and internet protocols(TCP/IP, UDP). As the transfer rate communication over these technologies nowadays reaches velocities of 10+ megabit/sec, it was designed with primary emphasis on easy manipulation and integration rather than on the optimization on bytes number data representation. The information is not sent as a data stream, but is intended as the delivery of "in place packet", in the sense that the communication is performed through self contained chucks, i.e. packets that contain all the relevant data in one place, thus discarding the holding of information about previous communication (as MIDI does). For a detailed description of the representation we send back to literature, but we think useful to report the scheme used for addressing the information. The model to which OSC conforms a hierarchical paradigm: the messages are addressed to a feature of a particular object or to a set of objects through a hierarchical namespace similar to URL notation, e.g., /voices/drone-b/resonators/3/set-Q. Some special character are reserved for pattern matching in order to gain further functionalities, such as range of characters ([a-z] stands for the range of letters), choice (any of the contained words), etc. One of the most interesting features of Open Sound Control relies in the information request. Some character patterns able a *Questioner-Responder* mechanism, in which a message is sent as a request for obtaining some information from the *Responder* endpoint. The timetag contained in the message will help which answer has to be associated to the respective question, therefore avoiding the wait-for-response condition. The features consist of namespace exploration, documentation, type-signature, return-type-signature and parameter constraint specification, current-value polling, identification of common interpretation maps via osc-schema, as well as a general error reporting and reply mechanism[43]. The OSC format reveals winning if considering distributed environments, or Web 2.0 based applications(such as Lily[6]). Further integrations with Web Service Technologies are apparently in a incubation phase.

[4] http://www.grame.fr/

[5] Center for New Music and Audio Technologies, U.C. Berkeley

[6] http://www.lilyapp.org/about/

2.4 Web Service Definition Language

WSDL[7] is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME [8]

The WSDL document defines the public interface of a Web Service creating a multi layered hierarchical XML description of how the user can interact with the given service. This description aims to fully detail the specifications about

- *what* can be used in the communication, i.e. the operations offered by the service;
- *how* to use it, e.g. the communication protocol to use to comply with the service, the format of the messages to be accepted in input and sent in output by the service and the related data carried;
- *where* to use the service, the endpoint of the service, which is usually equivalent to the URI address in which the service could be located;

The operations supported by the Web Service and the messages that could be exchanged in it are described in an abstract way, and their details are thus separated from the concrete network protocol or the data format specifications used. This gives the possibility for the reuse of the abstract definitions with different low level implementations (SOAP 1.1, HTTP, ...). Therefore the modularity of the format can be resumed with two groups: the abstract one, including the XML nodes *Type Message Operation* and *portType*, and the concrete one, including the elements *Binding*, *Port* and *Service*. As this notions will effectively be used in a great extent for the communication layer of our metalanguage too, we retain essential to enumerate the properties of each node element, reporting some code examples to better image their use in a generic case implementation.

- Abstract Elements
  - Types : the *types* element defines all of the non-built-in data types that the service uses, such as arrays and complex structures. Even if, for a maximum interoperability, the XSD canonical type system [9] is recommended for typing elements in WSDL, the language allows to defined more personalized data types by means of the *types* element. A simple example is given in the following lines, where a complex `MathInput` type is build as the aggregation of two doubles canonical types in order to give the `Add` element for a future ready to use inclusion.

```
<xs:complexType name="MathInput">
  <xs:sequence>
   <xs:element name="x" type="xs:double"\>
   <xs:element name="y" type="xs:double"\>
  <\xs:sequence>
<\xs:complexType>
<xs:element name="Add" type="MathInput"\>
```

[7] Web Service Definition Language
[8] http://www.w3.org/TR/wsdl
[9] http://www.w3.org/TR/xmlschema-2/

- Message : the *message* element defines the abstract description of the data to be exchanged between the Web service endpoints. Each message refers to a single piece of information moving between the invoker and the service, and consists of one or more logical parts; each part can be associated with a *type* or *element* attribute from some type system namespace (`tns:` if defined in current local namespace "this namespace").

  In the following lines the message named AddMessage is defined as being composed by a single part, which consists of a pair of doubles (the `Add` element refers to the `MathInput` type previously defined).

```
<message name="AddMessage">
  <part name="parameter" element="tns:Add"\>
<\message>
<message name="AddResponseMessage">
  <part name="parameter" element="tns:AddResponse" type="tns:MathInput"\>
<\message>
```

- Operation : the *operation* defines a specific action supported by the service. It is the transmission primitive for each remote method call. It can contain only two type of nodes: *input* and *output*. These two types of node qualify the direction of the message referenced in the attribute fields, and their order (if both present) in the *operation* parent node determines four different communication behaviors: a *One-Way* transmission when the endpoint just receives a message (solely *input* child node), *Notification* when it just sending a message (solely *output* child node), a *Request-response* architecture when the endpoint receives a message and is required to send a correlated return message, a *Solicit-response* when it performs the opposite operations, e.g. send a messages in order to receive a response. The following example shows a *Solicit-response* operation, where the endpoint is sent a *AddMessage* element (of MathInput type), and is requested to send an equally typed MathInput response message (probably after some mathematical computations).

```
<operation name="Add">
  <input message="tns:AddMessage"\>
  <output message="tns:AddResponseMessage"\>
<\operation>
```

- Porttype [10] : a *porType* is a named set of operations (thus the parent node for each *operation* node) and the abstract messages that are involved. Its name attribute must provide a unique name among all the port types defined in the same enclosing WSDL document. The port defines the connection point to a web service. It can be compared to a function library (or a module, or a class) in a traditional programming language. Each operation can be compared to a function in a traditional programming language.

```
<portType name="MathInterface">
  <operation name="Add">
  ...
  <\operation>
  <operation name="Subtract">
  ...
  <\operation>
<\portType>
```

- Concrete Elements

  - Binding : the *binding* element represents the link between the abstract port (the portType referenced in the type attribute) and the concrete port; it defines the physical protocol and data format that in turn determines how the port type is mapped onto a particular protocol.

```
<binding name="MathSoapHttpBinding" type="MathInterface">
  ...
```

[10] In the WSDL 1.2 draw specification it is substituted by the *interface*node

```
  <operation name="Add">
  ...
  <\operation>
  ...
<\binding>
```

– Port : the *port* defines the physical address (unique) for each binding; it does not specify more than one address, and it must not contain binding information other than address information.

```
<port name="MathEndpoint" binding="MathSoapHttpBinding">
   <soap:address location="http:\\localhost\services\math"\>
<\port>
```

– Service : the *service* groups a set of related ports together. The ports included in the same set could not communicate each other, i.e. with outputs of a port going as inputs of another; ports related to the same *portType* by means of the *binding* element represent "alternatives": each port provides the same semantic behavior (same *portype*) but are implemented with different protocol techniques. A *port* specification for the MathService would be similar to the following lines

```
<service name="MathService">
  <port name="MathEndpoint" binding="MathSoapHttpBinding" >
    <soap:address location="http:\\localhost\services\math" \>
  <\port>
  ...
  <port>...<\port>
<\service>
```

The Web service technologies made a breakthrough in the progress of distributed systems, achieving a good homogeneous interaction in applications developed using different programming languages and running on different architecture machines. The WSDL format has by now became widely used for the service-driven applications, especially where adopting the Publish/Subscribe paradigm [11]. Comparing with previous distributed technologies, they are much easier to use: to establish a communication with a destination endpoint, one just needs to obtain the WSDL specification of the interlocutor web service by specifying its URL address; with this simply query, he gets ready XXX to shape the communication hierarchy, what protocol should be used or which are the operations and data type allowed for the data exchange.

The WSDL format is currently in 2.0 draft version. This new revision will introduce some interesting features; the *portype* will be substituted by the *interface* element; since a *service* could only implement a single *interface*, thus penalizing for a reduction in extensibility, the information inheritance has been added. The format will be provided with further extensibility, allowing to mix XML into WSDL from other namespaces and to define patterns of interaction between agents (Message Exchange Patterns). However, some discussion have recently arose among the users, stressing some weak points of the WSDL format. While remaining the most suitable for the SOA[12] architecture, it reveals too complex for the message-oriented middleware applications, in which the Event Driven paradigm only demands a "notification delivery" paradigm. Indeed, even if WSDL introduced features like *interface* or *operation*, however it seems to make it even harder for the community to reuse "old" concepts and ideas inherited from distributed object technology. The SSDL [13](SOAP Service Description Language) is a XML-based SOAP-centric contract description language for Web

[11] Publish/Subscribe is an asynchronous messaging paradigm where senders (publishers) of messages are not programmed to send their messages to specific receivers (subscribers). Rather, published messages are characterized into classes, without knowledge of what (if any) subscribers there may be. Subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what (if any) publishers there are. This decoupling of publishers and subscribers can allow for greater scalability and a more dynamic network topology. [Wikipedia (August 2007)]

[12] Service Oriented Applications

[13] http://www.ssdl.org/i

Services that aims to overcome this weakness. Unlike WSDL, SSDL focuses on message-oriented contracts for Web Services, assuming SOAP (over an arbitrary transport) as the only protocol to transfer messages between Web Services [36]. Anyway at the present SSDL hasn't been applied on real-world scenarios and it is lacking valuable experience and industry feedback.

2.5 Language Oriented Programming and Automatic Code Generation

The automatic code generation is not a totally new concept in the field of programming languages. The idea of automatically generated pieces of code could make someone remind a procedure that always has gone with the existence of the integrated development environments, i.e. the wizard procedure. The explanation presented by the Jargon[14] dictionary defines the wizard as "...an interactive help utility that guides the user through a potentially complex task..", specifying that "..wizards are often implemented as a sequence of dialog boxes which the user can move forward and backward through, filling in the details required. The implication is that the expertise of a human wizard in one of the above senses is encapsulated in the software wizard, allowing the average user to perform expertly" [15]. The wizard procedure represents just a limited fruition of the automatic code generation concept; if on the one hand it saves the programmer from coding repeated routines for implementing language specific features (such as inheritance or abstraction), on the other hand it stops at a pure customizing point; the programmer at the end is always asked to manually complete the code in order to wholly implement the core algorithm the code is supposed to perform. The purpose of the code generation programming is more ambitious: fully substitute the research, the arrangement and the assembling of the all components of a software with an automatic process. This concept is claimed by many to be the next technology revolution in software development [23] [46]. Intentional programming, Template Meta Programming System, Language Workbenches, Generative Programming, Domain Specific Languages, etc., they all express the different variations of the the effort to give more freedom and coding ease to the nowadays programmers.

Ideally, programming has always been driven by the objective to reduce the gap between the abstract conception of a procedure and its concrete machine understandable realization. However, this theoretical freedom, intended as the choice to do "anything" on a computer, is still restricted in today's mainstream approach to programming. The potentiality of the languages conceived over the years remains anyhow plenary: any general-purpose language like C or Java or Lisp is said to give the possibility to implement anything we want on a computer. But in the practical perspective, they could reveal quite unproductive (at least taking into considerations that they have been used from more than 30 years), because they still force the programmer to be dependent on the features of the particular language adopted. This leads inevitably to a decrease in productivity, in software quality and, at the same time, in loss of the time. Today the greatest part of the developers copes with programming as writing a set of instructions for the computer to follow according to the syntax and formalism of the language chosen, and the most of the times risking to loose the congruence of programming means and software role goals. Some techniques of software engineering for properly run the life-cycle of the project (i.e. the Agile software development for the short-time project evaluations) have been introduced to face this duality problem, but these shortcuts represent just a gloss over an underlying lack, trying to recover in the software testing stage what should be improved during its conception and production.

The quality of a software is also determined by its capacity to evolve and the rapidity in its development. A well conceived program, indeed, must be quickly adaptable to the needs of the user as well as to be easily integrated into new technologies introduced in the market. It should

[14] http://catb.org/jargon/html/
[15] Jargon Informatics 4.2.0

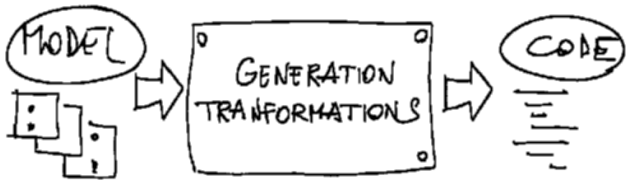


Figure 2.4: The conversion of the *what* into the *how*: a semantic representation of the software with objects and procedures (model) is automatically analyzed and converted into its concrete implementation (code)

be able to exchange data with softwares differently implemented, especially when interfacing to web services. This requirements have deeply changed the way to design and code systems. The developers have to rely on a modular and flexible implementation, based on generic and reusable components. In order to take into account these evolutions, the introduction of new design concepts became mandatory. The Object Oriented programming architecture certainly contributed, with concepts such as encapsulation and inheritance, to the modularization and extensibility issues. But this style is by now insufficient for considering the crosscutting notions transverse to classes.

Therefore a new conception motivated by the distinction of the application domains has arisen, and has developed two main different conceptions: the MDA (Model Driven Architecture) and the AOP (Aspect Oriented Programming)

2.5.1 Model Driven Architecture

This design approach encourage the use of modules that are independent from a specific platform, in order to be transformed later into a more machine dependent code by means of customizable generators, with the possibility to express modules in different paradigms.

Using the MDA methodology, the functionality of each module may be defined using a platform independent model(PIM), where the algorithms are expressed by a formalism consistent with a proper local Domain Specific Language (DSL). Through these "little" languages, we can better handle particular target operations, because they are tailored to be highly productive for the particular domain problems: for performing database queries we will be writing SQL style syntax, for constraint resolution, the variables and methods will simulate the formalism characteristic of the constraint programming paradigm, etc. If on one side they reveal a good performance for specific purposes, on the other side they show their weakness when trying to perform general purpose algorithms, since generic applications may involve many different domains. The PIMs are further supplied with platform definition models (PDM), so as to be translated into platform dependent code (PSMs). The PSM may use different DSLs and General Purpose Languages like Java, C#, etc.. The transformations can be performed in multiple manners: they may point from and to models of different domain type but with the same language, or perform a multi-input/multi-ouput translation.

The MDA model has became integrated with varioues existing standards, among which the Unified Modeling Language (UML) and XML Metadata Interchange (XMI). An open source platform that supports this software design is Eclipse, which developed a plug-in integration for the basic IDE called Eclipse Modeling Framework(EMF).

One of the fundamental consequence of the MDA technique is that it leads the programmer to focus more on *what* to implement then *how* to implement it (as he can express better with a domain oriented formalism), relegating the operative code writing to an automatic and thus faster and less error prone process.

2.5.2 Aspect-Oriented Programming

AOP is another concept for structuring software designs and code. The concepts of procedures, packages, classes, methods, objects are enhanced with the concept of *aspect*, which tries to describe in clear statements the underlying design intents of the programmer as a combination of crosscutting involvement of objects and procedures [31]. To understand what *crosscutting concerns* should refer to, one can consider that when thinking about an algorithm, like `compute notes interval`, such a concept is usually intended tied to the concept of a procedure or a method, while the idea of a `note` is usually associated to an class-object concept. However, things become more complicated with concepts like `interval transformation strategy` or `observer musical pattern`. These don't fit into procedures, nor into objects, nor any other common basic element. Instead, one can easily imagine to write a single class that implements the `interval transformation strategy` by means of agglomerations and manipulations of other classes of the system, because it represents a crosscutting definition. The notion of *aspect* was invented to cleanly capture this concern.

An Aspect Oriented Program is constituted mainly by two sets of constructions: *aspects* and *objects*(custom code); the *aspects* are stand alone modules that from outside the *objects* "observe" the program flow of data generated by the interaction between the *objects*, and have the possibility to modify it when convenient. More properly, an *aspect* module can contain some code (*advice*) joined to some specified points in the program (*joint points*), as the invocation of a method or of a constructor, the access to an attribute or occurrence of an exception; these points are further organized in sets (*pointcuts*). Thereby, the *aspect* operates on the behavior of the base code (the non-aspect part of a program) by applying *advices* at various *join points* specified in a *pointcut*. An aspect can also contain inter-type declarations (structural changes of other classes, like the addition of members or parents).

The Aspect Oriented Programming was developed by Gregor Kiczales and his team at Xerox PARC, where he contributed to the development of AspectJ[16], the first and most popular general-purpose AOP language. Since its introduction, AOP has been ported to other languages (C/C++/C#, Lisp, Ruby). The progresses and researches with the related training courses and consultings are annually presented at the International Conference on Aspect-Oriented Software Development.

Comparison

Both MDA and AOP are similarly coping the complexity of the system architecture through the *separation of concerns*, i.e. breaking the program into distinct features that overlap in functionality as little as possible. In AOP this modularization is achieved separating *crosscutting concerns* from the standard code application, while MDA focuses on a separation based on a technological concerns, in the sense that different targets are addressed by different language grammatics.

An obvious difference between AOP and MDA is that AOP is about programming and MDA is about modeling. So, when dealing with program code, we are effectively more AOP oriented, and when dealing with models we're approaching the MDA method. Still, both program code and a model may describe the same application. They do so in a different manner and on a different level of detail, but they are very similar. Many modern IDEs allow program code to be viewed as a UML model. Alternatively, with things like executable UML and model compilers available, a model can also be seen as a program. The difference between models and code is becoming less important as they grow closer to each other and the distinction between model and code is not large enough to clearly differentiate between AOP and MDA.

[16] http://www.eclipse.org/aspectj/index.php

2.5.3 Template Metaprogramming

The Metaprogamming technique depicts the writing of computer programs able to manipulate or generate other programs, and is commonly related to the concept of *software factory*[17]. The term was introduced by Charles Simonyi while working at Microsoft Resarch during the 90s [46]. Such concept however is not as new as declared. Indeed, the concept of metaprogramming can be largely detected in the long time history of the high-level programming language. Considering the family of interpreted languages such as Lisp, Python, Smalltalk, Ruby, Perl etc., the process is quite similar: a customized high level language expressed in its personal syntax is used to describe the procedures' behavior and entered into the virtual machine, interpreted and executed at runtime. In that limited case, however, no source code is being generated, but the programs are directly modified at runtime. If considering the interpretation-generation of code at compile time, an even older example could be recognized in the couple parser-compiler: a compiler allows a programmer to write a relatively short program in a high-level language and uses it to write an equivalent machine executable language program; parsers and generators, such as the classic UNIX utilities lex and yacc, are used to read a formal description of the language (expressed in a particular grammar) along with actions to be taken for each grammar rule, and they output lexical analyzers. The current use of the term, however, has been mostly associated to the recent trends among the more extended code generation processes.

The Template Metaprogramming is an augmented version Metaprogramming which uses templates[18] to perform code refilling and substitution at compile time. This technique doesn't establish its relevance on the introduction of new concepts, but it only extends their valence to more generic situations. Moreover, the recent increase in the use of templates specifically designed for generating Web pages, and the parallel evolvement of markup language formats for Meta-Models (as MDA and AOP) programming design, generated a promising research in the field. The instruments developed for the Web services and addressed to the manipulation and transformation of XML format files, turned indeed to be useful also for the other type of markup languages, opening the possibility for generic text automatic production and therefore code generation.

Among them there are some quite ambitious, as Intentional Programming (IP) [47] [48]. IP is an extensible and language-independent programming environment developed at Microsoft Research By Charles Simonyi, and then licenced to his new company IP Corporation. Often mentioned to include concepts from domain-specific languages and aspect-oriented programming, the IP aims to improve the level of abstraction, delegating issues such as notation, types and standards to programmers, in such a way that their choices remain compatible with the choice made by other programmers. In order to accomplish this level of "knowledege partecipation", the IP system maintains the source code in a proprietary binary file, where the active code is coded into a tree-like data structure. Graph links are used for connecting instances to their abstractions or references to their declarations. Many declarations can have the same name, but each declaration has a unique identity. The immediate contribution of this arrangement is that editing the name of a declaration immediately reflects the change in every reference.

Styles of generated code

In the process of "custom manual" software production, the programmers are directly handcrafting the code lines of the program, and they are aware that, in order to make future changes easy, they have the duty to furnish a fully understandable code. In the generated programs the design follows a different path. In this instance, the need to understand and change occurs at the specification level (the metamodel representation), not at the program level. This results in greater flexibility in the design of generated programs. There are three styles of programming used in program generators,

[17] as used in the .NET Framework
[18] also an old term

that relate directly to the type of code generated by the program generator [15]. "The OO-approach favors highly structured OO techniques. The code-driven approach favors straightforward code with embedded data. The table-driven approach puts data in a separate data section that is used by the code section." [15] A typical program generator will use some combination of these three techniques.

- Object Oriented - Driven Style (favors highly structured OO techniques) The OO-Driven approach uses standard object oriented techniques such as inheritance, interfacing, and message passing, for organizing the structure of the generated program [15]. Related variables and methods are bundled to create specific objects. Data from specifications are inserted where needed and Classes are created for attributes where needed. This approach is based on stacking object-oriented layers of abstraction. Each layer contains a number of object classes where each top layer refines the layer below it by adding new classes and new methods to existing classes [16].

- Code-driven (favors straightforward code with embedded data) This is the most natural and simplistic approach to program generation. In this method the code is simply generated where it is needed without any concerns about program structure but focuses on simplicity and efficiency of code generation [15]. In this method data from the specification is simply embedded where needed.

- Table-driven (puts data in a separate data section that is used by the code section) Table-driven design is a software engineering methodology that attempts to simplify and generalize by separating program control variable and parameters from the program code and placing them in external tables. Design objectives include an emphasis on modularity and decoupling program control data from application logic. Applications are made more flexible by postponing the time when control values and rules are bound to the processes they direct. This approach to program generation emphasizes the separation between data and the code that performs the generation. Data from the specifications are stored in the tables and other data structures. The code simply refers to the tables when the data is needed rather than embedding the data directly into the code [15].

Chapter 3

The project ConceptMove

An artist developing an interactive numerical work finds himself confronted with the consideration of the role of the spectator in his work: which space of freedom does it leave him and which are the acceptable degrees of freedom in the structure of work? In addition, one should not any more think a work as an individual creative process but as a collective act. This concept of a collective creation is in the heart of our research project. How to create an interactive pluri-cultural work collectively allowing the dialog between individual creations? As much for traditional media such as the cinema, the process can be iterative (one composes the music on images or one films starting from the rate/rhythm of a selected piece of music), as much interactive numerical work requires the existence of a contract signed between the various protagonists of creation. That starts with the definition of the nature of the interactivity and its repercussions on the various components of work. Today, at the end of a phase of discussion and exchange in which each one expressed himself with its own cultural background and its professional vocabulary, it is necessary to make a request to an engineer or a technician who will define, based on his own interpretation and by using low level protocols, the contents of what will be transmitted between software components or specialized libraries. The capture and the restitution can be rich by taking advantage from the advanced technologies in the fields of virtual reality or of the video game. In the same time, the dialog between the universes today is very limited due to a lack of languages supports to this dialog which are at the same time rich, pluri-artistic and open. It thus seems important to us to support the exchanges on this subject between all the interested actors (artists, researchers, engineers) in order to build new open languages allowing a much higher level dialog between the various artistic fields implied within the process of creation and realization of an interactive poly-artistic work. We have noticed the necessity to build a higher level protocol between software components as it is so far just a transmission of integers or floating values. The aim of this project is to model and develop a new meta-language allowing a high level communication between the different software implied in the creation and execution of interactive artistic installations. The aim of this project is to model and develop a new meta-language to describe the pluri-artistic and interactive art pieces in the most generic way and completely independently from the systems (software, programming languages) that will be used for its concrete realization.

In the sections that will follow, we will report the structure of the language proposed, detailed in its functionality segmentation: the hierarchy netween elements, the time and spatial profiles, the communication specifications. Along with its formalization, we will show how we designed and performed some of the code generative transformations, both for the communication and managerial layers of the application. Some hints for further improvements will arise as a consequence for the unsolved aspects, and future directions will be suggested in order to refine the performace results. In the final section, a simple test case used as a test for the parts implemented will be reported.

3.1 Language Specifications

The language developed is intended to give the author the possibility to easily express the concepts and the relations that found its artistic composition. It is evident how its simplicity and the ease of its elements' manipulation in the authoring process drastically decree its success or failure in the community of artists.

In the same way as the WYSIWYG[1] content editing established a new norm in the text editors' design and the visual programming language gained a consistent quote among the environments for media elaboration, the **ConceptMove** language specification is meant as well to be partly fulfilled in a visual compositing editor. Hence since this addition will be performed only in the final stage of the whole system development, a preliminary description that would highly support display and manipulation facilities presents as an obliged choice.

The XML format revealed as the best (and for some aspect mandatory!) solution in order to accomplish our objectives. It is a human and machine readable format, it describes structure and field names as well as specific values, it allows validation using schema languages such as XSD and is the most suitable for documents with an hierarchical structure.

In the following sections we will detach the different fields of the work's description, detailing each particular functionality and reporting some clarifying code examples.

3.1.1 The hierachical containers

- *Work* : it is the root of the language. This entity contains a set of *capsules*, a set of *worlds* and the set of *links* between the contained elements; the work is a kind of title for the all choreographic piece, therefore it cannot contain temporal information. It is a standalone element which doesn't have ancestors.

- *World* : this elements express the multiplicity of the languages used inside the same *work*. More precisely, for every different language used in the performance, being it a general purpose language to be later compiled or a patch to be included in an existing environment (such as Max/MSP), will correspond different worlds. In the code generation phase, each world will follow the specific template for its particular code.

- *Capsule* : the capsule allow to define the hierarchy of the different elements of the language. It can contain a set of other capsules or a set of nucleus. The capsule is the first element in the top-down hierarchy of the language to have the possibility to maintain temporal information, therefore a single instance is usually use to define the overall duration of the choreographic work. Implementing a temporal object, it consequently inherits the entire set of attributes owned by the simple object structure. Capsules included into other capsules are implicitly time dependent, because their existence is conditioned by the existence of their mothers.

- *Nucleus* : the nucleus is the atomic object of the language. It implements an objects, therefore it can maintain temporal information and relations with other elements.

- *Object* : the objetc represents a basic interface from which to derive other elements; it specifies some properties over the attributes, and it imperatively demands the specification about the portypes used for communicating to other elements and the world affiliation.

[1] What You See Is What You Get

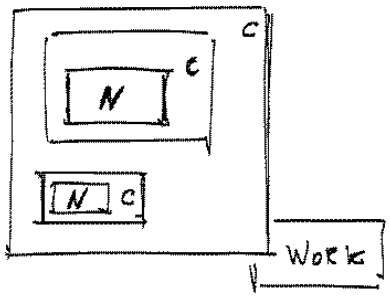


Figure 3.1: The *work* as hierarchy of sub-elements: a *nucleus*(N), the atomic structure, is contained into a hierarchy of including *capsules*(C), with a root *capsule* corresponding to the whole presentation

3.1.2 The communication layer

The communication uses the same syntax of the WSDL markup language, therefore for more detailed information we send back to Section 2.3. The elements for structuring the specifications are:

- *Service* - *PortType* - *Operation* : the *service* contains the access points over the network, which are itemized by a number of different *ports* specifying the concrete physical address for the binding of the *service*. The portType defines the interface as a collection of the available operations that define the functionality of an object. Each portType is connected to at least one port in a service, by means of a *binding* element. The operation represents the active layer of the message. Conceptually it may be thought as the method call used from inside the code. Depending on its nature (presence and order of input and output messages), it can accept the input data to be send and can return a response from the remote endpoint.

  The connection between abstract ports over the network, as well as the direction of their link is described by a `link` elements, which bounds together two operations specifying their sending/receiving roles, as with

  ```
  <cptm:link xlink:from="OperationA" xlink:to="OperationB"\>
  ```

  The `link` (which however is an unnamed element in **ConceptMove**) here reported expresses that the data sent as output by *OperationA* will be directed and processed as input by *OperationB*. The `link` element is extremely important for the tranformation of the meta-data into the concrete "lower level " code implementation. When coding the "sending" functionality of each interface, the destination addresses (ip+port for an UDP connection) for each output message is always retrieved by looking at the operations connected, and "discovering" the network address of the membership parent service.

- *Message* : the message defines the complex data that is exchanged between the two endpoints. Each message is in turn composed by a single *documentation* subnode and zero or more *part* subnodes, depending on its duplex nature of simple notification or data dispatch. The *part* subnodes are used to contain the atomic values that form the complex message data, and their type is defined by a proper attribute according the DTD (Document Type Definition) adopted. Because of the use of the OSC paradigm, the inclusion of the `oscpath` inside the message parts has been highly encouraged. However, as the XML syntax of the WSDL format arose some questions when trying to include strings containing forward slash characters (by which the oscpath string is constituted [2]) in node names and attributes , the *documentation* node has

[2] `http://www.cnmat.berkeley.edu/OpenSoundControl/OSC-spec.html`

been shaped for this purpose. A message composed by four floats would thus would result in the following form:

```
<message ... name="Voice1Timbre">
  <wsdl:documentation>
    <oscpath value="\sinth\voice1"\>
  <\wsdl:documentation>
  <part name="a" type="xsd:float"\>
  <part name="d" type="xsd:float"\>
  <part name="s" type="xsd:float"\>
  <part name="r" type="xsd:float"\>
<\wsdl:message>
```

Similarly to the the Aura system(Section 2.1.5), the communication in **ConceptMove** is considered inexpensive and does not need to be scheduled. In a real case of network contention, we might want the threads that are executed at high priority to get priority on the network as well. In our current implementation, no difference is made between information delivery and data flux (i.e. a notification of an event versus the continuous value stream of a variable). The messages are not associated to a priority (which would be specified in terms of allowable latency), so they all receive equal treatment.

3.1.3 The time behaviour

- Temporal Object : the Temporal Object (TO) is a subclass of the basic Object element, to which it adds the temporal specification subnodes, namely, the *begin* and *end* elements (see code example below). Both the introduced subnodes attempt to define the time instant in which a particular event should occur (i.e. the start or the end of a media object), but they are also capable of indeterminate values, such as time intervals in which the event may happen to take place. In order to better clarify this dual behavior, we report in the following lines an example of a capsule with its temporal specifications:

```
<capsule xlink:label="C1">
  <time>
    <begin xlink:label="C1.begin" value="0" type="abs" calcul="sooner" min="0" max="3"\>
    <end xlink:label="C1.end" value="5" type="abs" calcul="later" min="3" max="5"\>
  <\time>
  ...
<\capsule>
```

Each time instant node (`<begin>` or `<end>`) may act in three ways, which are stated by the value of the attribute `calcul` (thus mandatory) :

1. *preferred* : the instant of the time event is determined and defined by the value attribute. No time slice is allowed, and attributes min and max become useless
2. *sooner* : the instant of the time event is no determined but it is able to take place during the time interval [min,max]. If no time relation with other objects will hold, the time instant (for both begin and end attributes) for the runtime execution will be fixed on the minimum value; if some relations with other objects exist, so as to possibly shrink the domain of admissibility, the time instant will be fixed at the lowest value in the restricted domain.
3. *later* : the instant of the time event is undetermined. Its choice methodology follows a construction opposite of the *sooner* case, falling on the maximum value of the allowed domain.

It is important to say that the simultaneous presence of the *sooner*/*later* declaration in the `calcul` attribute and the the `value` attribute will disable the consideration of the former, in the same way as a *preferred* `calcul` will make useless the `min` and `max` attributes.

Moreover, if the `begin` or `end` subnodes of a TO is related to an external event, such as the start/end of another TO or an user interaction synchronization, the `calcul` attribute becomes useless. Thus if an object is said to begin during the interval $[begin_{min}, begin_{Max}]$ with the receipt of a synchronization event, it will wait until $begin_{Max} - 1$ for its activation and if not received will start at $begin_{Max}$.

In the case of a TO other than the root capsule of the interactive presentation, the time value may be defined in two ways (as stated in the `type` attribute): *absolute* or *relative*. An *absolute* value refers to a time instant as a time amount elapsed since the master root capsule start, while a *relative* refers to the time elapsed since the beginning of its parent object.

- Time Relation: The time relations offer the possibility to specify time constraints between objects, so as to maintain a mutual dependence between their occurrence during the artistic representation. The time relation specification, specified in the *allen* sub-node similarly to

```
<cptm:allen allentype="$AllenRelation" xlink:to="$TemporalObjetcLinked"\>
```

conforms the consolidated Allen temporal knowledge representation [2], whose basic (13) relation types are reported in Fig. 3.2. The second term of the relation is expressed in the `to` attribute value. A single objetc may contain multiple relations with other objects. Each relation is recorded internally to each cached temporal object.

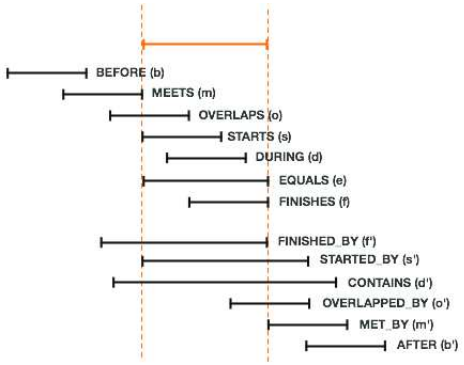


Figure 3.2: The basic thirteen Allen relations, composed by the active and passive declinations

3.1.4 The spatial behaviour

A choreographic representation is a multimedia installation involving different abstract layers as well as different hardware support. For this reason a multitude of different worlds must be considered, because we can have not only the duality real-virtual but also multiple video or sound outputs (sound spatialization). Thus each objects concerning will included in a specified space system. The elements that account for this layer are:

- Spatial Object: this element extends the basic Object element by adding a *position* information (3D Cartesian coordinates). The position represents an attribute in an SO node, and it is intended to be accessible through the communication layer for a reading/writing operations by a request-notification messaging [3]. The aggregation of multiple SO gives change for the definition of more complex objects, namely the *Volume*, *Area* and *Line* triple.

- Reference Mark: it defines a spatial reference orientation. It is used for retrieving the absolute position of the objects that it contains

[3] a sort of *getPosition()* & *setPosition()* methods

- Main Reference Mark: it is the main and the only one spatial reference orientation, to which all the other reference marks belong. The absolute position of each object is given with respect to him.
- Spatial Transformation : a spatial transformation defines the matrix (4x4 in the case of a 3D reference) for the axis transformation employed in the passage between frames.
- Global Spatial Object: each specific *world* represents a global spatial object. Its position refers directly to the main reference mark, but in turn contains a personal Reference Mark which all the objects in it contained are referring to.

3.2 From Meta- to General Purpose Programming language transformation

Once the ConceptMove specifications have been outlined in the form of the XML document, we are ready to transform the metadata for the concrete implementation into a particular general-purpose programming language. Since the starting point of this process consists of a well consolidated markup language, commercial and research softwares offer a wide number of instruments for its manipulation. Among this profusion however, two basic approaches to the problem can be recognized:

1. DOM Look-up: an ordinary general purpose language using DOM to parse data. The XML file is read from the filesystem and it is converted into a tree-structure element, the Document Objetc Model (DOM). The DOM furnishes a neutral [4] interface used to dynamically access and manipulate the tree-structured elements of the XML.

   explain better the birth and uses of the DOM, say it was developed for the internet browsers for the display of meta data, with the purpose of easily change the display layout.

   The DOM API birth was mainly caused by the need to have a simple and fast way to connect web pages to scripts or programming languages. The DOM is most often used in (JavaScript before all [5]); this caused the language to have an objected oriented syntax, therefore all the properties functions and events available are accessible as objects' methods. In what follows we'll furnish a basic understanding of how the XML meta data are organized in memory and the basic methods for accessing its structured elements.

   Once the file is read and transformed into a *Document* objects, the entry point of the tree must be retrieved in order to be able to navigate toward the underlying contained nodes. This top-level element is called the *DocumentElement*, and it can be accessed invoking a proper method on the *Document* class, as in

   ```
   Element root = (dom.Document)doc.getDocumentElement();
   ```

   Each *Node* (which is a subclass of *Element*), starting from the root parent and extending to all the derived nodes of the document, may have one or many *childNodes* (a list of Node objects), a container representing the branches of the tree. The subnodes may be obtained through a single level inspection (Parent-Child) by means of

   ```
   NodeList nList = (Node)n.getChildNodes
   ```

   of by a multi-level inspection, enabling to get the ancestor nodes whose node name conforms an argument pattern tag string

   ```
   doc.getDocumentElement().getElementsByTagName(SERVICE)
   ```

[4] from platform and language used

[5] JavaScript was first introduced and deployed in the Netscape browser version 2.0B3 in December of 1995

The backtracking, however, is accessible only as a single step inspection:

```
(Node)n.getParentNode()
```

The attributes of each node are accessed by the function

```
aNode.getAttribute(‘‘anAttribute’’)
```

We reported this basic set of methods for the reader to be acquainted in understanding the example that will follow in next section, but for the complete set of functions available we send back to the W3C specifications.

Even if in adoption since long time and ported to the majority of programming languages used for the web, the adaption of XML data manipulation to a general purpose programming language has proved difficult because of problems associated with expressiveness and typing. As a matter of fact, the complex manipulation of semistructured data, such as required for the markup languages, does not always fit very well with conventional programming languages; i.e., a manipulation where it would be required to find all the occurrences of a structure matching a structured search pattern whose context may be different in different places of the tree would cause some difficulties. From the other side, if trying to use at runtime a special purpose query language more addressed to XML, an additional interface to the more general programming environment would be required, and this typically would turn into a runtime overhead for type conversions. Therefore, the DOM API present an easy interface for simple look-up inspections, but reveal theirs drawbacks when attempting to more complex manipulations.

2. XSLT by means of XPath : XSLT[6] was in origin developed for the formatting of the XML files, with the purpose of easily transforming and visualizing the tagged data in a customizable way. The “stylesheet” definition for this format is somehow reducing because while real stylesheets formats (such as CSS) apply a particular style to the TAG present in the document, the XSLT transforms the XML document giving as output a new one of different structure. Broadly, the code of a XSLT transformation (which must in turn be a well-formed XML file) works by models, or templates `xsl:template`, that define the rules applied to the node which are specified in the mathing attribute. As an example, the template

```
<xsl:template match="message">
  ...
<\xsl:template>
```

will perform the included code to all nodes named `message`. Inside each `xsl:template` we can use external calls by means of `xsl:apply-templates` (or `xsl:call-template`) to apply named elaboration models defined elsewhere in the document, also with the possibility parameter passing (<xsl:param name="n"/>). Model templates can be used to get different processing for the same node in different situations. `<xslt:copy-of>` instruction can be used to reuse XML from the source document or from the result tree fragment. The values can be accessed through the `<xsl:value-of ..>` element. XSLT provides also conditionally execution (`<xsl:if>` or `<xsl:choose>` nodes), sorting and iterating functions, temporal variables and recursion (a recursive template is a named template that calls itself). Even if originally intended only for the data display reformatting, the XSLT has been proved to be Turing complete [7]. However since an XML document primarily contains encapsulated elements and texts, it must be attended by additional methods to locate specific elements, so that they can be processed, and such a functionality is obtained with XPath. This language operates over a logic representation of the XML document, modelled as a tree structure, and offers a syntax for accessing the nodes

[6] XSL=Extensible Stylesheet Language Transformation

[7] `http://www.mulberrytech.com/Extreme/Proceedings/html/2004/Kepser01/EML2004Kepser01-toc.html`

of the tree, additionally disposing of functions for the manipulation of strings, numbers, and boolean values. The expressions defined by XPath are called Location Path, and conform to the form

```
axis::node-test[predicate]
```

The `axis` component expresses the "family" relationship between the current and the searched node; the `node-test` component specifies the type or the name of the searched node, while `predicate` holds zero or more filters for specifying the selective conditions to be applied to the research. In XPath we can access the nodes in a second way, the *Abbreviate Location Path*. The search expression is constituted by a list of element names of the document, separated by a forward slash (/), where the sequence describes the increasing level of inspection of the XML tree. The mechanism is similar to the one used for filesystem or Open Sound Control. It will take the form of `/A/B/C` to select "the `C` elements that are children of the `B` elements that are children of the `A` element that forms the outermost element of the XML document". XPath disposes also on a wide number of functionalities for controlling the nodes, the strings, etc., such as: counting the number of nodes, returning the position of a certain elements inside a set of nodes, computing the sums, ceiling or floor rounding approximations of numbers, counting characters of and cutting strings, etc..

XPath 1.0 was published as a W3C Recommendation on November 16, 1999 and a 2.0 version was published as a W3C Recommendation on January 23, 2007. The importance of XPath stems from its potential application as an XML query language per se and from the fact that it scales well both with respect to the size of the XML data and the growing size and intricacy of the queries.

We experienced the transformation with both DOM and XSLT methods. Each approach has it's own advantages and drawbacks. XSLT is a more flexible and structured approach: it offers the maximum separation between data, programming logic, and presentation. Its tree-formed inspection permits a consistent saving over verification functions or temporal variables mandatory for generic language paradigms. It allows to define separately a set of functions which can be accessed with parameter attributes, and the possibility to save temporary node sets during the elaboration of the templates. Moreover, its ability to find nodes and traverse through a document using a pattern matching paradigm (by means of XPath) supplies an easier but powerful handling of XML documents. On the other hand it requires careful design. For generic language used programmers, the capture of the data from the XML structure may wander from the real manipulation of data.

The transformation process can in turn be splitted into sub-transformations, which help grouping and organizing the concepts to increase the modularization and understanding of the code. The maximum modularity is synonym of maintainability as well. This turn to be useful if following interventions on localized data are previewed, as they are granted more immediate and less time consuming operations.

The use of XSLT increased prominently in these last years. In the beginning it was critically regarded, especially because of its performance deficiency; for large amount of data, common for commercial web based applications, it revealed slower than common DOM systems[8]. This drawback is however not predominant in our case, where the number of elements is not expected to be too as copious as for commercial situations.

In the sections that follow we will examine the key points for the code generation processes performed. We will analyze the way and the limits we must comform in order to "enter" into the client environments, so as to maintain at the same time a good compromise between their own language paradigms and the communication layer of **ConceptMove** .

[8] At the preset substantial progresses are being made XML querying language (both XPath and XQuery)

3.2.1 Max/MSP and Pd tranformation

Even after over twenty years of their appearance, the encoding of Max/MSP - PureData patch files still resembles to the original implementation (Fig. 3.3). The text form saved patches uses a script paradigm performed by the insertion of each line into the system, first specifying to the system the set of objects present in the patch, and then inserting their connections. There are some general information about the patch visualization, like the dimensions of the canvas or the font used, and box particular specifications, e.g. their (x,y) coordinate position together with other unspecified numbers, text and box RGB color numbers, data set to be displayed for the get info dialog. The language used for the encoding of the information presents some drawbacks: being concerned with graphical display too, the transformation process from the XML specifications to the patch files would have required a complex manipulation, in order to maintain the relations between the objects and their connections, as well as for the computation of a suitable display positioning. In order to result visually discernible, each message would have required at least two connections and a routing objects, therefore yielding a very dense canvas when dealing with a wide communication dictionary.

The integration of Java language into the both frameworks (by means of `mxj` and `pdj`[9] externals) presents a good alternative. The transformation phase comprises a single java file generation and conforms the table-drive style (see Section 2.5.3): the template (which is reported in the following code lines) is extended with the methods that will initialize the class data structures that held the user defined specifications, and which will be constantly used for look-up during the exchanging and filtering of messages.

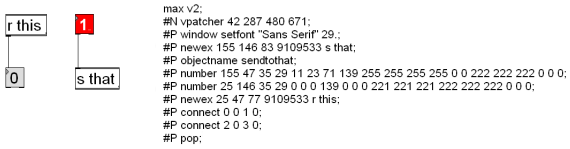


Figure 3.3: An example of Max/MSP patch and its text saved file representation

Runtime Behaviour

The communication gateway is completely provided by the java dependent [mxj CMGateway] box (see Fig.). At the instantiation phase, the object constructor runs the methods for storing the set of all messages that will pass through it, and for setting up the UDP socket layer for establishing the communication over the network.

At first, the functions append at the end of the file template by the code generation process are called: the ports to be opened locally by the set of sockets, the set of incoming messages we're expecting to receive, and the container of messages we are able to send. Each outgoing message is stored with its proper address destination, so as to correctly be sent to the predefined location.

Secondly, some useful state and functionality aids are added (not reported for the sake of space), e.g. the inlets/outlets assistance or a bang response for the displaying on console of the current state (the complete set of ports opened and the messages that will walk through the gateway). Note that the constructor includes the call to a function named `defineReception()`: this method is relevant for the temporal relation of this environment inside the artistic work. Indeed Max/MSP and PureData represent a *world* node in the **ConceptMove** architecture and the working patches could stand for concrete *Time Objects* realizations; the flux of data exchanged with internal objects has to be related to their existence, and the communication gateway has to be able to ban the message from going through for the non-active TOs. This functionality in the CMGateway has been implemented

[9] http://www.le-son666.com/software/pdj/

with an additional message filter. With this purpose the `defineReception()` complies the filling of a HashSet data structure, where the in/out-going messages are coupled with their belonging TO activation (a *(MessageName,Boolean)* pair).

The reception of the messages in Max/MSP and PureData needs no additional box objects. The incoming messages are indeed converted from the OSCMessage java object to a list of Atom type elements and sent to the "system" by means of the MaxSystem.sendMessageToBoundObject method (present also in PureData). The messages and its arguments are captured by the interested patch with the `receive` max object.

Adversely, for the delivering of messages it is mandatory to insert a receiving object (`[receive CMOut]`) in the same patch of the CMGateway , because the environment does not provide (at least at the moment of the code development) the "receiving version " of MaxSystem.sendMessageToBoundObject. Moreover, other additions must be carried out in the patches that are intended to send the data, namely a prefixing to the data with the string message identifier (the OSC path or the message name) and the link to the CMGateway patch (by means of a `[send CMOut]` object).

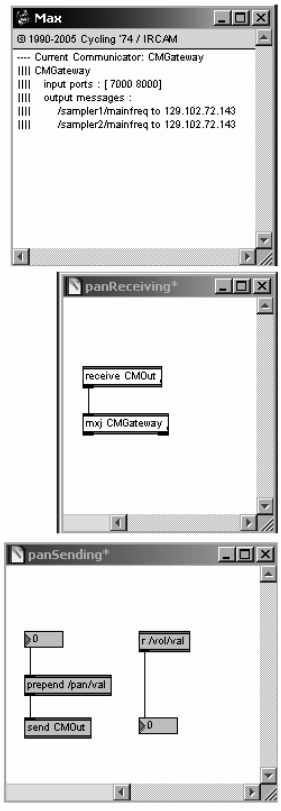
```
public class CMGateway extends MaxObject {
  ...
  private int nPorts = 0;
  private int ports[] = null;
  private Vector<Message>outMessages = null;
  private Vector<Message>inMessages = null;
  private HashSet<String> allMessages;

  public CMGateway() {
    defineInputPorts();
    defineInMessages();
    defineOutMessages();
    defineReception();
    ...
  }
  ...
  \* MAXMSP - PD CODE + SOCKET JAVA CODE *\
  ...

  \\-------------------- Automatic generated code --------------
  \* Store ports to be opened *\
  public void defineInputPorts() {
    int portsArray[] = {7000,8000};
    this.nPorts = porte.length;
  }

  \* Store outgoing messages *\
  public void defineOutMessages() {
    outMessages = new Vector<Message>();
    outMessages.add(new Message("\sampler1\mainfreq", 8000));
    outMessages.add(new Message("\sampler2\mainfreq", 7000));
  }

  \* Store incoming messages *\
  public void defineInMessages() {
    inMessages = new Vector<Message>();
    inMessages.add(new Message("\sampler1\play", 8000, "127.0.0.1"));
    inMessages.add(new Message("\sampler1\stop", 7000, "127.0.0.1"));
  }
}
```



3.2.2 Actionscript transformation

ActionScript is an object-oriented scripting language[10] interpreted by the AVM virtual machine; it is really close to Java and C#, and is used primarily for the development of websites and software using the Adobe Flash Player platform. As the latter has become a leader program for the realization of interactive web sites as well as multimedia presentations and animations, we find useful to include the consideration of this language among the transformations, with the expectation that web applications will be more and more cropping out. Its easy integration of visual, audio and temporal elements contributed for its rapid proliferation among visual designers and artists. ActionScript didn't provide during the last years great support for the free software community (because the compiler was proprietary and closed), however after the buyout of the company by Adobe and to avoid the concurrency of other alternatives (first of all Processing [11]), it opened some "doors" (Adobe open-sourcedits ActionScript Virtual Machine) and made a forward step with the adoption of the third generation of the Actionscript programming language.

We're not extending into a detailed description of the main features of the language (for which re send back to the Adobe official datasheets `http://www.adobe.com/devnet/actionscript/`), however we consider important to better specify an aspect of the language whose concept will be deeply involved in the transformation (and that is common to the most part of recent languages too): the event-driven communication.

The event-based programming is a computer programming paradigm in which the flow of the program is determined by user actions or messages from other programs, rather then predefined by the programmer [34]. This paradigm has been first implemented for the GUI libraries in order to capture the occurrence of some user events, such as the mouse and keyboard key pressions, and to connect them to internal predefined methods for the action elaboration.

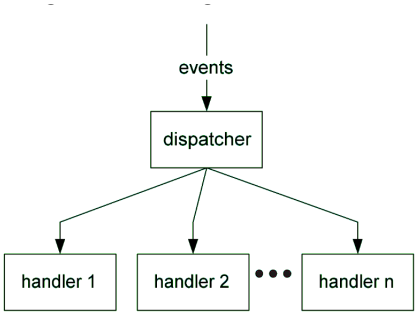


Figure 3.4: The handlers mechanism: a dispatcher receives the events and sends them to the proper handlers

In Fig. 3.4 is reported the basic local flow that the event should follow. A dispatcher takes the event that comes to it, it performs the appropriate analysis in order to determine its type and then send it to the handler that can handle events of that type. For the **ConceptMove** case, the behavior should be similar considering the events to be the messages that are exchanged with the application; the final handlers would be the methods that are supposed to elaborate the data the message delivers. In Actionscript 3 whenever creating an event handler to be called during a certain event we will need to use the EventDispatcher and addEventListener methods. The functionality of the classes created has been though to work as follows:

1. in order to receive a `message`, we want to link a local arbitrary method to the reception of the chosen **ConceptMove** *message*, and we want all the data to be passed directly to the linked

[10] now at version 3 based on the ECMAScript (ECMA-262)
[11] http://processing.org/

method.

2. for the sending of messages, we want to perform such operation in two ways: in an way opposite to the receiving phase, therefore dispatching our custom event (a simple event type augmented with the message data); additionally, with a call to a function that expresses the **ConceptMove** *operation* and that takes as arguments the output messages declared, i.e. in the form `myOperation(myMessage m)`;

Inside the ActionScript transformation we employed XPath and last version of XSLT specifications (by means of the Saxon[12] processor) for a multiple files output and for an architecture more object oriented. Each in-output *message* has been transformed into a different `.as` file for implementing each customized Event ActionScript subclass, while the *operation* methods have been Incorporated in the core module for the message filter cross (a derivation of the *portype* node).

Runtime Behavior

The messages sent to **ConceptMove** *objects* represented in the Flash application must be converted into `Events` to be "listened" from inside the code methods. This process is not performed directly. For network Flash security restrictions, the ActionScript code is not allowed to open UDP sockets for external interfacing. Therefore we resort to a commonly used interface for OpenSoun-Control, the *flosc* [13] Java server; this application implements a simple gateway or bridge between the UDP world of OSC and the local TCP world of the Flash XMLSockets, enabling Flash to use Flash's XMLSocket feature to communicate externally over a UDP socket. The OSC messages are therefore automatically converted in the flosc-Flash passage into a XML representation of the OSC message (XMLSocket data), and then analyzed from inside the CMGateway .

The CMGateway is meant to implement a `portType` element in the meta language semantics, with its set of runnable `operations` that carry out the dispatch and the reception of the `messages`.

Let's imagine a situation in which we want Flash to receive XXX The script code for the incoming concept move message should be quite like the following:

```
myObject.addEventListener(ConceptMoveMessage.AmplitudeValue, processAmplitude);
```

where the node representing a OSC message of the type `\sin_i\amp 4.456` would be of the form

```
<wsdl:message ... name="AmplitudeValue">
  <wsdl:documentation>
    <oscpath value="\sin_i\amp"\>
  <\wsdl:documentation>
  <part name="value" type="xsd:float"\>
<\wsdl:message>
```

The `myObject` object refers to a class of my code, which I want to link to the information that is sent over the OSC protocol.

An alternative way is to declare each different message that is going to be sent to the Flash application as a subclass itself the `Event` AS3 class. In this case, the event listener will be waiting for it by writing

```
myObject.addEventListener(CM_AmplitudeValue, processAmplitude);
```

This would be better because easier to define the number and type of arguments.

[12] http://saxon.sourceforge.net/

[13] flosc : Flash OpenSound Control `http://www.benchun.net/flosc/`

3.2.3 Engine transformation

The engine is the core of the control of all the elements involved in the work, therefore its architecture is the most complex and its generation process the most sophisticated. The main task of the engine is to assure the correct scheduling of the temporal objects, jointly with a constant verification of the occurrence of the interaction events inside the planned time line. It constantly maintains the state of all the objects (running or not-running state) and is charged to notify them when to start and stop.

In addition, it performs the operations for maintaining coherence when relating objects of different spatial coordinates (generating and maintaining the coordinate reference axis for each world), it runs custom scripts developed by the user for additional control or data manipulation.

At the present moment the coded engine architecture only performs the basic temporal functions: it schedules the time instants at which each object is required to start or end, and it performs a meagre control over the incoming interactive events in order to maintain the consistency with running processes. During its architecture design some interesting questions arose and are still in a resolution stage. In the following sections we will discuss the major implemented features as well as some facets that could be added to a final stable version for a more complete management.

From time relations to the CSP formalization

The compositional step of the interactive work ends with the declaration of a set of time dependent objects (capsules,nuclei) and the relations that tie their behavior in the time line. The user profiled these relations by means of the Allen interval equations in Section 3.1.3, but this symbolic representation has to be converted into a numeric representation in order to be verified and finally scheduled in time. During this process and in the case of modifications during the runtime "presentation", a supervisor system has to guarantee that these relations don't hide possible contradictions, which would bring to an unpredicted behavior of the system (e.g. a basic case "A before B before C before A"). This issue has long since been confronted in knowledge based approaches for automatic task planning and scheduling (i.e. for the correct scheduling of resource-sharing concurrent processes). A commonly technique used for its resolution and adopted in **ConceptMove** project is the *constraint reasoning*, or *constraint programming*. This approach, based on the formalization of a *Constraint Satisfaction Problem*, represents an operative method for modelling time relational problems, and it offers a conjunction of established algorithms for its resolution.

Formally a CSP problem is constituted by a finite set of variables, $V = \{v_1, v_2, \cdots v_n\}$, a set of domains $D = \{D_1, D_2, \cdots D_n\}$, one for each variable, and a set of constraints $C = \{c_1, c_2, \cdots c_m\}$, with $c_i : (D_1 \times D_2 \times \cdots \times D_n) \rightarrow \{true, false\}$ a set of equations which define the subset of admissible values for each variable.

The model for the time layer in the following discussed strongly resembles the one proposed in [3] for the interactive musical scores, but the transformation into the concrete time schedule has not been here carried out through a transformation of the CSP into the Time Petri Nets.

Each temporal object TO is translated into one, two or three variables, where these are the resulting CSP translation of the attributes' values in the *time* subnodes *begin*, *end* and *duration* (in the future they will be pointed with the notation $TO_{begin}, TO_{duration}, TO_{end}$); in the case of a single time instant, we remind that the missing "time specification closure" will be deduced by the parent container; on the other hand, a complete specification (with the presence of the all three nodes) is furnished as a facility for the user, even if some redundant information as well as additional control comes to be introduced (the case $TO_{begin} + TO_{duration} = TO_{end}$). The possibility of a zero duration TO is provided, too, and is especially addressed to the modeling of *control* or *interactive* events. We worked with the assumption that any time object is conceived as unbreakable, i.e. that it is not allowed to perform a pause operation or to resume the play from the last position. The looping

condition has also been kept in regard. A *TO* with a loop condition is imported into the constraint problem with ending time given either by the end-piece value or by total-loops value.

The CSP implementation has not been developed from scratch inside the **ConceptMove** project, but has been treated via the external open source java library `Choco`[14]. This caused the inclusion of the library for a compiled working version of application, as well as portions of code dependent on its API (when dealing with temporal relations verification). However the CSP algorithms should not influence the whole engine architecture, but only provide the implementation of an interface meant to be seen by the scheduler as divided (modular) as possible. With this purpose, the XSLT templates try to maintain the concepts (and files too) as separated as possible.

As described in Section 3.1.3, the time instants intended to be undetermined are defined by means of an interval of values $[min, max]$, where for a CSP such a concept traduces into a domain of admissible values, with the min and max attributes depicting the domain boundaries. For the Choco problem implementation, these variables have been declared of type `BoundIntVar (name, lb, up)`, a "finite integer domain variable whose domain is approximated by bounds (lb, ub)". This type of variable is indicated as the more appropriate for large domains, because the CSP algorithms are performed on the bounds and not on single domain values. Such a choice precludes us from additional operations, as the removal of values between the bounds, but the space efficient encoding delivers a faster constraint verification, which is more important if addressing for real time performances. Indeed, when taking into consideration interactive environments, a millisecond resolution (or of tens of ms) becomes mandatory for exhaustive results, and the modeling of variables declared as intervals of millions of integer values (as for time intervals greater than 15 minutes) becomes hardly handleable.

Once all the variables are inserted into the *problem* (and only then), we can perform the insertion of the constraints. Before posting[15] the external constraints (against other objects), all internal implicit relations are inserted and verified: each object, in order to be consistently declared, inserted and afterwards referenced, must be consistent with its bounds.

Finally, it's the turn for the parent-child dependencies and the explicit Allen relations. As a library expressively addressed for Constraint Problems, Choco offers the possibility to add a wide range of predetermined and customizable constraints, thus the insertion of the ones **ConceptMove** is intended to deal of (largely binary constraints) translates straightforwards into the available methods[16].

What kind of solution?

The modeling of a scheduling problem into a CSP is usually meant to resolve the problem in order to schedule a solution for a concrete time line execution. The resolution techniques of a CSP define the paradigm of *constraint programming*, and according to the depth of solution set pursued could be grouped into two categories:

1. *constraint propagation*: the algorithms for the propagation consist in the deduction of the implicitly defined constraints that are inner to the problem. These cause the elimination of a part of the possible alternatives reducing the time for the solution search. The propagation algorithms have the task to assure the local *consistency conditions* of the problem; in a generic *k-consistency* problem, for each instance (a choice for a variable c_i for a value in its domain D_i) of a *(k-1)-consistent* subset and for each variable x external to such subset it exists a a consistent value for x (that verifies the constraints). In a $k = 2$ consistency satisfaction, commonly known as *arc-consistency*, for any pair of variables c_i and c_j of the problem, for each value of c_i in D_i there is some value y of c_j in D_j that is consistent with x [40]. If, for

[14] http://choco-solver.net/index.php

[15] in Choco a constraint is stated to a problem by using the method `post` available on the `Problem` object : *post(Constraint c)*

[16] $C1\ before\ C2$ $[Allen] \rightarrow C1_{end} \leq C2_{start}$ $[model] \rightarrow problem.post(geq(C2_{start}, C1_{end}))$ $[Choco]$

example, we have two time instants T_i T_k that are expected to fall respectively in the intervals $D_i = [0, 10]$ $D_k = [9, 13]$ and a relation T_i *before* T_k is holding, a arc-consistent problem after the propagation will give the new domains $D_i = [0, 8]$ $D_k = [9, 13]$.

2. *search algorithms*: these algorithms have the task to find a solution to the problem. In order to accomplish this goal, some choices need to be done inside the research space. In a CSP formalization, the choice can be translated into the addition of a further constraint; however, while in the propagation process the constraint added was already implicitly present inside the problem, here the operation is not marked as "necessary" and has the consequence of pruning the set of possible solutions. Consequently, the criterion according to which each variable selected is reduced in its admissible domain (heuristic decisions) reveals extremely important for the approximation to the target solution.

The "resolution" aspect for the CSP has to face in our case (but in general with all problems concerned with running schedules) a concrete strict requirement: the complexity and therefore the computation time for the process completion. Once the presentation has started, there could be still some variables that remain undefined because related to external interactive synchronizations (e.g. waiting for a button click or for a spatial trigger). The system should provide the mean to quickly update the state of all the variables bound with such events and perform once more a consistency verification for the newly updated (or better created) problem. Furthermore, the user should be given the possibility to add or delete constraints or variables in the course of the performance, and if not strictly related to current instant processes, he would be pleased not to stop the all representation for this.

For the schedule rendering of the time linked sequence of objects, we don't even demand for a single solution: if uncertain instants have been defined by the user during the authoring phase, these are supposed to model arbitrary events to happen during a closed time interval $[T_{min}, T_{max}]$. A single value result for them would be too much restraining, as well as useless for the meaning of interaction. On the contrary, the complete set of solutions would require too much time because of its (general case) NP-completeness nature[17].

In **ConceptMove** we hence chose to adopt the only "constraint propagation" operation for the constraints verification. In the following section we will show how we faced the problem of indeterminate variables for a runtime resolved scheduler.

Scheduling with indeterminate timing

The usual methods adopted for structuring multimedia content presentation can be grouped into two distinguished models [44]:

1. pure static scheduling: the presentation is described by a linear timeline; all the time objects descriptions have a specific known value relative to the global presentation timeline, which is uniquely determined. This model cannot support dynamic changes nor interaction that has consequences on the temporal organization, and handling of media with unknown time duration is not admitted. For this reason it is generally associated to pure editing and authoring tools, where all the elements' time endpoints are fixed before the runtime stage. Its implementation is therefore the most straightforward, as it suffices to record each object *start* and *end* time instants only once into the global timer scheduler.

2. pure event-based: the presentation is described as a graph of event bindings; usually adopted for the graphic user interfaces, this method does not provide a time schedule at all, but the flux of events is the governing the passage between processes. It fully supports dynamism and

[17] `http://en.wikipedia.org/wiki/Complexity_of_constraint_satisfaction`

interaction, as each elements follows its independent timeline, with little or not information of other objects.

However, the choice of an unique type between the two would reveal restricted for a presentation that should support any kind of interaction. As a matter of fact, supporting scheduling interaction means that in conjunction with the story line there could be many aspects of the presentation that change or are performed according to the user input or other interaction sources. It becomes unmistakable that for an all-round solution a sort of integration between the two should be attempted.

In the **ConceptMove** application the engine model separates the time description from the runtime values; the overall architecture has been designed as a "dual core" integration between two communicating processes: the CSP problem, which incrementally maintains the set of constraints and variables, and the scheduler, which is charged for the starting and stopping of each temporal object. The set of actions that takes place from the beginning to the end of the presentation can be resumed as a *determining-scheduling* "migration": each time a variable of the constraint problem becomes reduced in its domain to a single specific value relative to the global presentation timeline, it is imported into the scheduler. The moment all the variables have been transferred into the scheduler, the presentation has become linear (an therefore unique), i.e. no more dynamic is allowed, and no more synchronizing interaction can cause any sort of effect (finally we reached a static solution of the CSP). A supervisor algorithm performs a control over the CSP set of variables after each "posting+propagation" call (the result of a scheduling interaction) and it looks for the newly obtained single-value-domain variables. Once the cardinality of all the variables becomes unitary, the "posting+propagation" will be no more performed.

However, there could be situations in which we do never get to shrink all variables to a single value. How should we face the scheduling for such cases? In Section 3.1.3 we defined the time instant node of each capsule as acting in three different ways: *preferred*, *sooner* and *later*, according to the choice made by the user in the authoring stage. When a variable is no more dependent (or it never was) on an external event synchronization and all the constrained variables have become determined, we can deduce that its indeterminacy reveals meaningless. The *calcul* attribute turns here useful for fixing the variable to the value the nearest to the specified one: the domain D_i will hence be reduced to its lower(upper) bound for a *sooner*(*later*) declaration, or to the nearest in-domain value for *preferred*. This operation cannot lead to constraints violation, because we only chose a solution for such variable among the set of solutions the previous constraints were permitting. As a result, it would be useful a tool able to identify among all the remaining temporal objects, the ones whose bounds are allowed to be reduce based on the forementioned condition.

Example

Two capsules $C1$ and $C2$ (Fig. 3.5) last a fixed amount of 2 time units (we'll refer to seconds for brevity) and belong to a parent capsule C with a fixed time position $0 \rightarrow 10$ (it begins imperatively at 0 and ends at 10 seconds). The first capsule, $C1$, is started by an external event (for which we use here the "passive" Allen relation *started by*). It is also simultaneously related to the second capsule $C2$ by a *meets* Allen relation, which implies a synchrony between the former's `end` and the latter's `begin` ($C1_{end} = C2_{begin}$). The relative **ConceptMove** nodes and attributes in the XML specification will appear as follows:

```
<capsule xlink:label="C1">
  <time>
    <begin xlink:label="C1.begin" type="abs" calcul="sooner" min="0" max="10"\>
    <duration xlink:label="C1.duration" value="2"\>
    <allen allentype="started_by" xlink:to="external_event"\>
    <allen allentype="meets" xlink:to="C2"\>
  <\time>
  ...
<\capsule>
```

```
<capsule xlink:label="C2">
  <time>
    <duration xlink:label="C1.duration" value="2"\>
    <end xlink:label="C2.end" type="abs" calcul="later" min=0" max="10"\>
  <\time>
  ...
<\capsule>
```

After the insertion of each capsule in a Temporal Objects container (where the original user-defined XML values are maintained immutable), the data is conveyed to the CSP model; each time instant is converted into a variable, the constraints are inserted and a first *constraint progagation* process is performed, in order to verify before starting that the user information does not incurr in inconsistent declarations; the resulting list of variables and the domains associated will be as follows

```
external_event = [ 0, 6 ]
C_begin  = [ 0, 0 ]
C_end    = [ 10, 10 ]
C1_begin = [ 0, 6 ]
C1_end   = [ 2, 8 ]
C2_begin = [ 2, 8 ]
C2_end   = [ 4, 10 ]
```

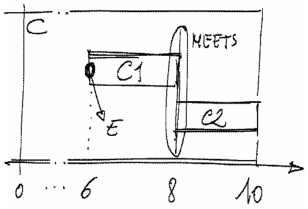


Figure 3.5: An example of parent capsule C containing two child capsules $C1$ and $C2$ linked by the relation $C1 meets C2$

As a consequence for the 2 seconds fixed durations of each capsule, the upper bound for $C1_{begin}$ and the lower bound for $C2_{end}$ have been resized by the *starts* Allen relation (e.g. as the minimum starting time for $C1$ is 0 and it lasts 2 seconds, it implies that the minimum start instant for $C2$ is imperatively 2). The *starts* constraint between the `external_event` and the start of $C1$, has been translated into the CSP as an equivalence between the two intervals, even if the external event has been left by the user (with no explicit definition) to possibly happen whenever during the whole work time line. For a ready-to-execute version of the list of objects, we can infer that only the parent's C_{begin} and C_{end} can be effectively scheduled at a fixed time, because their existence interval has become reduced to a single value, while the remaining variables will only become determined at runtime. One the work is started, the system will suddenly process the parent capsule scheduling it for time 0, and afterwards it will be waiting for two conditions: the arrival of the `external_event` or the reaching of second 6 (the maximum value for the $C1_{begin}$ instant).

The reception of `external_event` at t seconds ($\forall t \leq 6$) causes the constraint $E = t$ to be added and and the problem's set of constraints to be newly verified. In this simple case, the *contraint propagation* leads to the shrink of all the variables' domains to a single value. With these conditions, the scheduler can program the *begin* and *end* of all events and no further dynamic change will be performed till the end of C. On the contrary, what would happen if the `external_event` do not arrive inside the allowed time interval? Or if it even does never arrive?

In **ConceptMove** the communication of interactive events that is meant to supply synchronization with objects is always filtered by the communication layer of the engine. All the time instants are modelled as *listeners* that become available for incoming messages only in the time window defined by their current CSP domain. Such a choice could bring the possibility of some delay for a highly congested communication flow, but this shortcoming is well preferable than stopping the whole performance for consistency violation. In the example formalized above, for example the arrival of the external event at time $t = 7$ would have caused a *constraint contradiction*, because the sequential sum of $C1$ and $C2$ would have overstep the parent C bounds.

3.3 A test case

In order to test the current version of the **ConceptMove** interface, we experimented the formalization of a simple case that could touch the greatest number of features implemented. The idea came out from the frenetic working activity that people usually reserve for the month of August: the virtual actors represented are the enterprise couple *boss-worker*. The entire work timeline, whose time unit is the second, is organized along the 24 hours day (a main capsule of 24 hours), and it is modeled to start at 0:00 o'clock and end at 24:00, with a loop condition for a restart once reached the end of the presentation. The root capsule is in turn composed by a child capsule for each phase of the day (*morning*, *afternoon*, ...), which are organized in a "sequential" order (the end of the *morning* capsule synchronizes the begin of the *afternoon*). Both the acting virtual entities are placed inside two different 2D space bordering zones, whose sum compose the whole rectangular scene of the work. The position of the *worker* is fixed to be inside his own zone, while the *boss* can displace himself from his zone to the *worker*'s one and opposite, arising each time a distinguished spatial trigger ($zone1 \rightarrow zone2$ $zone2 \rightarrow zone1$) when overpassing the linear edge between the zones. The behaviors of the entities are different according to the moment of the day: both are inactive during *night*, both are working during *morning*, both should be working during *afternoon*. When entering (time meaning) into the *afternoon* capsule, the *worker* falls into a great stupor condition, which causes him to easily fall asleep, unless the boss enters his office. During the *evening* he doesn't work nor sleep, while he deeply sleeps during the *night* capsule. The *worker* is the only entity that has an audio rendering (Pd and Max/MSP). The two audio applications are used as sample players for the sounds that represent the sleep condition (a "snoring sound" sample player) or the awake condition (a "keyboard typing" sound). A "no-work+no-sleep" phase will cause no audio output from both the environments. The *boss* in turn has no audio rendering (he never sleeps?), but may be spatially displaced from his zone to the *worker* one during two conditions: the user directly displaces him (here's the interaction aspect of the work), or he perceives that the *worker* is sleeping (a volume threshold in the audio system will notice with an *alarm* the *boss* for a control)

The environments with which we mastered the presentation and for which the code has been generated are the followings:

- ListenSpace[18] [Java]: this environment serves as the user interaction input; the actors of the scene (the boss and the worker) are graphically represented in a bird's eye view as circles together with their rectangular zones. The software receives and sends the positions of the actors and the triggering signals for the passage from one zone to another. Through the mouse the user can directly click and drag the boss picture betwen the two zones, with the meaning of a "user controlled" supervise of the worker.

- Pd + Max/MSP [own+Java]: the audio layer coordinate the playing of the samples corresponding to each attitude of the worker. The audio interfaces are arranged for receiving play and stop messages, and for delivering an alarm to the system whenever the volume of the sound produced oversteps a fixed volume threshold.

- OpenMask [C++]: this environment account the pure visualization of what is taking place on the scene. It has not been designed for a specific data elaboration, as its functionality is already carried out by ListenSpace. However the communication layer has been equally implemented for testing purposes. It does deliver not output, but it only receives the two actors' positions and the starting/ending notification for each capsule.

- Flash [ActionScript]: similarly to OpenMask it is used for the scene visualization, but provides

[18] a graphical authoring tool especially designed for interactive soundscapes in audio-augmented realities developed by Olivier Delerue & Oliver Warusfel

a user input interaction: a click over the *boss* figure will cause him to perform a control over the *worker*

The interface obtained for Action Script and Java revealed very useful for the data manipulation. The position messages of the two agents of the scene is automatically converted from the OSC format (the OSC path string followed by the coordinate number values) into the complex data type `BossPosition`/`WorkerPosition`, where the coordinates can be easily accessible by means of an object oriented paradigm (i.e. `BossPosition.x`). The messaging formalism through event dispatching has proved very intuitive, aware of the fact that for each event the low level network protocol and consequently its destination address has been automatically generated, and therefore for a runtime modification it would demand the regeneration of the all code. Nevertheless, the runtime addition of customized messages is not previewed in **ConceptMove** , as it would be considered as an "hacking" process without the warranty of conflict control with the preexisting message set and socket instantiations.

The example formalized has been tested with relaxed time constraints. The time resolution adopted for the scheduling and the constraint udpating algorithms was indeed based on a second time units, and such a condition left to future tests the performance verification.

Along with the fulfilling of the application protopype and according to the artistical partecipations of the **ConceptMove** project, the most appropriate test will comprise of the "second version" rewriting of "Seul avec loop", a choreographic installation that will conclude the project schedule in October 2007. Conceived by the dance company Danse34 and supported by the collaboration between Irisa and Ircam, its description will be entirely reorganized through the **ConceptMove** meta-language formalism, and the software involved in the media presentation/interaction will be added with the interfaces generated.

Chapter 4

Conclusions

4.1 Summary

The objectives of the project are cross-sectional. During the first part, the meta-language conception, we focused on the analysis and synthesis of the needs of the users. Together with the artistically partners, we tried to model the widest type of problems that the author could be faced to deal with during the conception of the artistic work. Some of them yielded almost noticeably since the beginning of the language specification and others disclosed during the setting of the concrete implementation. We tried to analyze the previous and current similar projects, as well as the most recent communication technologies, in order to adapt where possible to our language model formats already largely accepted and adopted by the engineer/informatics community.

During the implementation phase we addressed towards the pure technical issues. We considered the different methods in the design of interfaces for the existing softwares so as to interpret our language formalism, and considered the pros and cons between different possibilities as a motivation for the choices made. We are aware that some of them made along the prototyping of the current version could have to be re-examined in the future. The release of the upgraded versions of protocols or libraries, as well as the upcoming of new ones, could lead to substitutions or integrations for some modules inside the architecture designed.

We admit that our position might not be universal and well suited for every interactive pluri-artistic situations. Especially situations requiring fine grained interactivity such as score following might be hard to specify in our formalism, or require a better system performance for real-time reactive responses. However we believe that our approach is well suited to a large number of interactivity situations such as for instance augmented realities or interactive narration.

The whole **ConceptMove** project results were indeed not meant to be closely developed and completed with the end of the project partnership. The main objective, as we have stressed more than once during the thesis argument, is established in the community feedback between the user experience and the system architecture evolvement. Therefore the improvement suggestions that disclosed to us during the test cases rehearsed, could be proposed for future discussions together with external comments. We have however to keep in mind that a good compromise between usability and treatability decrees the success of an artistic technologies. Indeed the direction to keep, side by side with an engineered optimization and new technologies integrations, should always be pointing to a grade of comfort between the user intentions and the authoring formalism, otherwise risking to generate an nth labyrinthic proposal.

4.2 Suggestions for future work

Beyond the author's internship the most impelling direction to point at is the fulfillment of the whole project objectives. The system should be enriched in code templates for the generative process, in order to support further programming languages and environments. Primarly for considering different programming languages, both in the functional paradigm family as Lisp for OpenMusic and in the imperative one (as for example Chuck[1]). This task would help the generative programming sector of the informatics to acquire more examples in the translation of concepts and procedures between different computation paradigms. Second, the insertion of packaged patches and modules into existing platforms would be also useful if attempting a widespread usage. As the Java and C/C++ transformations have been completed, environments such as Processing[2], Clam libraries[3], or some of the ones displayed in Chapter 2, could gain in functionality being easier connectable and acting as instruments of a common shared score.

From the "user perspective" the authoring process should be given an intuitive editor in order to graphically compose the work specifications, and a visualizing layer for the running presentation displaying (as in Fig. 4.1). The text filling of a XML file is prohibitive for error proneness and for the lack of a global visualization. A first version is being prototyped by the **ConceptMove** project partners, and it is based on the Eclipse open-source Java IDE. With the operability of the modeling framework (EMF) and a **ConceptMove** customized plug-in, it will provide the possibility to visually structure the data model and to perform the code generation and/or compilation.

From an "engineering perspective", some key points of the engine architecture still could gain improvements. We showed in Section 3.2.3 how we faced the integration between the scheduling and interaction requirements, and how we avoided possible issues by imposing a kind of "time windowing" reception filters for the external synchronization events. This trick saved us from some cases of relations inconsistency during real time execution. However, someone could scorn this choice, claiming that the interaction aspect of a work is far more important than the exact time dispositions and relations among elements. A different philosophy may be adopted, but we must be aware that this would lead to schedule modifications, and in order to better react to inconveniences, more information about the resolutions to act would be asked to the author. Some concepts of supervised scheduling control (usually referred as Job Shop Scheduling problems) could turn to be useful for our case [32] [12] [7]. A choice for the methods to adopt for "repairing" the constraints consistence violation should be adopted, automatically selected by the system or according to the user specified preferences. The system could for example undergo with would then act with one of the following approaches:

- *reactive approach*: the solution would be corrected as soon as a new event generates a contradiction with the existing rules (an error returned by the addition to the CSP problem of the new constraint), trying to recover the situation. It is the most simple approach, as it doesn't compute a middle-long term prediction of the actions to execute, but simply performs a greedy choice for each different troublesome point. The most direct technique used for the solution repairing is the simple *right shift*, in which the activity is postponed. This method is similar to what we adopted in our current version. In Section 3.2.3 and in the example shown, when missing an external event synchronized to *start* a capsule, we proposed the *begin* delay until the upper bound value of its domain of existence. This type of resolution can also be called local repair method [12], because it tries to locally adjusts the solution so as to avoid further events

[1] http://chuck.cs.princeton.edu/
[2] http://processing.org/
[3] http://clam.iua.upf.edu/

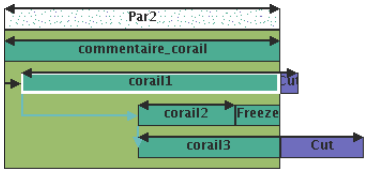


Figure 4.1: A fragment of the graphical user interface of LimSee, an open source authoring tool for multimedia documents using the SMIL format

for the nearest scheduled operations. As it doesn't consider the effects of the modifications on the whole scheduled activity, it is mostly used for domains in which there is a low number of relationships between different elements of the time line. There are more elaborate methods that consider the problem in its entirety and perform the *regeneration of a new schedule*. The re-planning operation should be however joined by a supervisor for choosing the new timeline considering the differences with the "broken" one.

- predictive approach: on the contrary the objective of these methods is to realize solutions that are able to "cushion" unforeseen events. Notorious approaches of this type are the Redundancy Based, which exploits a redundant allocation in time (and in resources if constraining the scheduling), so as to react to to eventual hazards, and the Contingent Scheduling, which tries to anticipate the hazard generating multiple solutions, so that it switches towards the most appropriate solution when the current schedule becomes no more executable.

The propositions mentioned are only a brief overview of the methods available in literature, and they should be not so poser to integrate to the current system, as they have already been tested in concrete implementations for critical system. In conclusion, even for not mentioned methods, the main concepts that should be kept in regard when formalizing an algorithm for uncertainty scheduling account for:

- a specific definition of the uncertainty
- the consideration of the uncertainty during the execution
- the capability to recover unforeseen events during execution

Even if the consideration of a more complex reasoning-based sheduler would be very intriguing for our project, this should compulsory imply the inclusion of ulterior attributes or nodes in the meta-language specification. Without this additional definition, the system could surely behave adopting automatic strategies predefined by the developer, but this "transfer of decision" could get the artist ailing for having a limited control over his own creation. As a final side effect, the overhead introduced in the specification fulfillment would probably result in a lack of naturalness, discouraging him from his original intentions.

Bibliography

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web.* Morgan Kaufmann Publishers, 2000.

[2] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[3] A. Allombert, G. Assayag, and M. Desainte Catherine. A system of interactive scores based on petri nets. *sms*, 2007.

[4] O. Belloc, M. Cabral, and M. Zuffo. Timeclock: flexible animation control in x3d. *Proceedings of the twelfth international conference on 3D web technology Web3D '07*, 2007.

[5] G. J. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of xslt. *Proc. CL*, pages 1137–1151, 2000.

[6] A. Bonardi and F. Rousseaux. Composing an interactive virtual opera : The virtualis project. *Leonardo, Journal of the International Society for the Arts, Sciences and Technology*, 35, 2002.

[7] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, pages 329–341, 1994.

[8] E. Brandt and R. Dannenberg. Time in distributed realtime systems. *Proceedings of the International Computer Music Conference*, pages 523–526, 1999.

[9] D. Bulterman. Standards: Smil 2.0. part 1: Overview, concepts and structure. *IEEE Multimedia*, -:–, 2001.

[10] D. Bulterman. Standards: Smil 2.0. part 2: Example and comparisons. *IEEE Multimedia*, -:–, 2002.

[11] A. Camurri, A. Catorcini, A., C. Innocenti, and A. Massari. Music and multimedia knowledge representation and reasoning: the harp system. *Computer Music Journal*, 19, 1995.

[12] A. Cesta, N. Policella, and R. Rasconi. Coping with change in scheduling: Toward proactive and reactive integration. *Intelligenza Artificiale (Italian Journal on Artificial Intelligence)*, 2006.

[13] S. M. Chung and A. L. Pereira. Timed petri net representation of the synchronized multimedia. *Proceeding of International Conference on Information Technology: Computers and Communications*, 2003.

[14] Nigro Cicirelli F., Furfaro A. Distributed simulation of modular time petri nets: An approach and a case study exploiting temporal uncertainty. *Real-Time Syst*, 2007.

[15] C. J. Cleaveland. *Program Generators with XML and Java.* New-Jersey:Prentice-Hall, 2001.

[16] K. Czarnecki and U.W. Eisenecker. *Generative Programming Methods, Tools and Applications.* New York: Addison-Wesley, 2000.

[17] R. Dannenberg. Aura as a platform for distributed sensing and control. *Symposium on Sensing and Input for Media-Centric Systems*, pages 49–57, 2002.

[18] R. Dannenberg. A language for interactive audio applications. In *Proc. of the Int. Computer Music Conf.*, 2002.

[19] R. Dannenberg, B. Bernstein, G. Zeglin, and T. Neuendorffer. Sound synthesis from video, wearable lights, and "the watercourse way'. *Proceedings of the The Ninth Biennal Symposium on Arts and Technology*, pages 38–44, 2003.

[20] R. Dannenberg and E. Brandt. A flexible real-time software synthesis system. *Proceedings of the International Computer Music Conference*, pages 270–273, 1996.

[21] R. Dannenberg and D. Rubine. Toward modular, portable, real-time software. *Proceedings of the International Computer Music Conference*, pages 65–72, 1995.

[22] R. Dannenberg and P. van de Lageweg. A system supporting flexible distributed real-time music processing. *Proceedings of the International Computer Music Conference*, pages 267–270, 2001.

[23] Sergey Dmitriev. Language oriented programming: The next programming paradigm. 2004.

[24] S. Donikian, F. Devillers, and G. Moreau. The kernel of a scenario language for animation and simulation. *Eurographics Workshop on Animation and Simulation*, 1999.

[25] A. Duda and C. Keramane. Structured temporal composition of multimedia data. *Proceeding of International Workshop on Multi-media Management Systems*, 1995.

[26] A. Camurri et al. Music and multimedia knowledge representation and reasoning: the harp system. *IEEE Computer, Revised and extended in D. Baggi, ed. 1991. Readings in Computer Generated Music. New York: IEEE Computer Society Press, pp. 95-1 15*, 24, 1991.

[27] Ping-Yu Hsu et al. Strpn: A petri-net approach for modeling spatial-temporal relations between moving multimedia object. *IEEE Transactions on Software Engineering*, 29, 2003.

[28] A. Sametti G. Haus. Scoresynth: a system for the synthesis of music scores based on petri nets and a music algebra. *IEEE Computer*, 24:56–60.

[29] Y. Huang and D. Gannon. A flexible and efficient approach to reconcile different web services-based event notification specifications. *International Journal of Web Services Research (JWSR)(submitted)*, 2007.

[30] S. Kepser. A simple proof for the turing-completeness of xslt and xquery. *In Proceedings of Extreme Markup Languages*, 2004.

[31] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, and John Irwin Cristina Videira Lopes, Jean-Marc Loingtier. Aspect-oriented programming. *Proceedings of the European Conference on Object-Oriented Programming*, 1241:220–242, 1997.

[32] C. le Pape and S. Smith. Management of temporal constraints for factory scheduling. Technical Report CMU-RI-TR-87-13, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, June 1987.

[33] Edward A. Lee and Thomas M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83:773–801, 1995.

[34] B. Meyer. The power of abstraction, reuse and simplicity: An object-oriented library for event-driven design. *From Object-Orientation to Formal Methods: Essays in Memory of Ole-Johan Dahl, eds. Olaf Owe, Stein Krogdahl, Tom Lyche, Lecture Notes in Computer Science 2635, Springer-Verlag*, pages 236–271, 2004.

[35] I. Norman, Bonnie L. Webber Badler, and D. Reich Barry. *Towards Personalities for Animated Agents with Reactive and Planning Behaviors*. Robert Trappl and Paolo Petta, Editors, Creating Personalities for Synthetic Actors: Towards Autonomous Personality Agents, 1997.

[36] S. Parastatidis, S. Woodman, J. Webber, D. Kuo, and P. Greenfield. Asynchronous messaging between web services using ssdl. *IEEE Internet Computing*, 10(1):26–39, 2006.

[37] K. Pihkala. Extension to the smil multimedia language. *PhD Thesis, Helsinki University of Technology*, 2003.

[38] O. H. Roux and A. M. Deplanche. A T-time petri net extension for real-time task scheduling modelling. *JESA Modelling of Reactive Systems*, 36(7):973–986, 2002.

[39] E. Rukzio. A generic extension mechanism for x3d to define, implement and integrate new first-class nodes, component and profiles. *PhD Thesis, Dresden University of Technology*, 2003.

[40] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach(Second Edition)*. Prentice Hall, 2002.

[41] L. Rutledge. Smil 2.0: Xml for web multimedia. *IEEE Internet Computing*, 5(5):78–84, 2001.

[42] G. Scali. Xml coding of dramatic structure for visualization. *Space S.p.A., Italy and Graham Howard, System Simulation Ltd, UK*, 2004.

[43] A.W. Schmeder and M. Wright. A query system for open sound control. *Open Sound Control Conference 2004*, 2004.

[44] P. Schmitz. Unifying scheduled time models with interactive event-based timing, 2000. (W3C Internal note), `http://www.ludicrum.org/plsWork/papers/UnifyingNote.html`.

[45] T. Shirai, M. Takano, H. Miyahara, S. Kodama, K. Tajima, K. Kandori, and S. Shimojo. Agent enabled scenario language for production of interactive tvprogram. *Communications, Computers and Signal Processing, IEEE Pacific Rim Conference*, -:530 – 533.

[46] C. Simonyi. The death of computer languages, the birth of intentional programming, 1995. `citeseer.ist.psu.edu/simonyi95death.html`.

[47] C. Simonyi. Interview with charles simonyi, 2004. `http://www.codegeneration.net/`.

[48] C. Simonyi. Anything you can do, i can do meta, 2007. `http://www.techreview.com/Infotech/17969/page5/`.

[49] S. Smith. Reactive scheduling systems. *Intelligent Scheduling Systems, Brown, D. E. (Eds.)*, pages 155–192, 1994.

[50] J.H. Yang Stephen, C.Y. Kuo Kevin, W.Y. Shao Norman, and J.D. Wu Ben. Modeling and analysis of spatiotemporal behavior of multimedia in smil. *Proceeding of International Computer Symposium*, pages –, 2004.

[51] John A. Stewart, Sarah J. Dumoulin, and Sylvie Noël. Binding external interactivity to x3d. In *Web3D '07: Proceedings of the twelfth international conference on 3D web technology*, pages 109–112, New York, NY, USA, 2007. ACM Press.

[52] W. Verplank, M. Mathews, and R. Shaw. Scanned synthesis. *Proceedings of the International Computer Music Conference*, pages 368–371, 2000.

[53] W3C. Web services description language, http://www.w3.org/tr/wsdl. `http://www.w3.org/TR/wsdl`.

[54] M. Ward. Language oriented programming. *Software - Concepts and Tools*, 15(4):147–161, 1994.

[55] M. Wright. Open sound control: an enabling technology for musical networking. *Org. Sound*, pages 193–200, 2005.

[56] M. Wright and A. Freed. Open sound control: A new protocol for communicating with sound synthesizers. *International Computer Music Conference*, 1997.

[57] M. Wright, A. Freed, and A. Momeni. Opensound control: State of the art. *Proceedings of the New Interfaces for Musical Expression*, 2003.

[58] Jianghui Ying. An approach to petri net based formal modeling of user interactions from x3d content. *Proceedings of the eleventh international conference on 3D web technology Web3D '06*, 2006.