Introduzione a Pure Data

Laurea Magistrale in Ingegneria e Scienze Informatiche

Corso di

Interazione non Visuale

a.a. 2010/2011

Stefano Papetti

Un po' di storia: Il paradigma “Max”

- Nome scelto in onore di Max Mathews
- **Programmazione visuale** secondo il paradigma *dataflow*

- Sviluppo principale nel decennio '80–'90:
  MIT Experimental Music Studio, IRCAM, UCSD
- Fine '80: versione commerciale (poi ***Max/MSP***)
- All'IRCAM ***Max/FTS*** (“Faster Than Sound”, poi ***jmax***)
- Miller Puckette è o è stato il principale ideatore e programmatore di tutte le versioni. Dal '94 alla UCSD Puckette sviluppa ***Pure Data*** (*Pd)*

Pure Data

- Alternativa **open source** e **gratuita** a *Max/MSP*
  - per molti utenti migliore...
  - ...anche se la **documentazione** è decisamente meno completa

- E' un vero linguaggio di programmazione
  - Può realizzare qualsiasi algoritmo (i.e. è Turing completo) ...con un po' di buon senso!
  - Funzionamento “trasparente” (nasconde funzionalità come es. deallocazione degli oggetti cancellati)
  - Scarse strutture per iterazioni, condizioni, branching

Pure Data

- La descrizione grafica di un programma (*patch*) determina completamente le sue funzionalità
- **Utilizzi**:
  - Sound synthesis / analysis / processing
  - Video rendering / processing
  - Physical computing (sensori, attuatori, varie interfacce)
- **Versioni:**
  - Versione “vanilla” (standard): http://www.crca.ucsd.edu/~msp/software.html
  - **Pd-extended**: include decine di librerie e oggetti extra: http://puredata.info/downloads

Pure Data

- Alcune librerie:
  - *external* per programmare in java: **[pdj]** http://www.le-son666.com/software/pdj/
  - *external* per programmare in python: **[py]** (1)
  - Video, 3D: *Gem* (1), *PDP* (1), *Gridflow*
  - Modelli fisici di sistemi dinamici: *pmpd* (1)
  - Sintesi audio (suoni ecologici) per modelli fisici: *SDT* http://www.soundobject.org/SDT/

(1) inclusa in Pd-extended

Concetti base

- **Dati**:
  - Messaggi (es. numeri, simboli, liste)
  - Segnali
- **Oggetti** (box): **[**nomeoggetto arg1 … argn**]**
  - Processi e procedure che agiscono sui dati presentati in ingresso, ed espongono i risultati in uscita
  - 2 tipi:
    - ***Intrinsics***: parte del core di *Pd*
    - ***Externals***: librerie esterne che aggiungono funzionalità (una volta caricate in memoria non sono distinguibili da *intrinsics*)
  - *Inlets* / *Outlets*
    - Hot / Cold *inlets*

Concetti base

- **Connessioni** (cords / wires)
  - Rappresentate da linee rette tra *outlets* e *inlets*
  - Sottili (per messaggi) e spesse (per segnali)
- **Patches** (programmi)
  - Collezione di oggetti interconnessi disposti in una o più *canvas* (finestre)
  - Salvati in formato testuale (file .pd)
- **Data flow** → l'interprete esamina l'albero dei messaggi verticalmente, con ordine *depth-first*

Concetti base

- 2 **modalità di funzionamento**:
  - **Run** → è possibile interagire con GUI (es. number box, sliders)
  - **Edit** → NON interrompe esecuzione e continua a ricevere input dall'esterno

- **Esecuzione** → gestione distinta in base alla natura del flusso dati:
  - Messaggi: sistema *event driven*
  - Segnali: elaborazione continua su blocchi di campioni (default: ciclo DSP di 64 samples, modificabile)

Messaggi

- "**bang**" → è il messaggio fondamentale, usato per attivare un processo
  - Oggetto GUI "bang"

- **float** → in *Pd* i numeri sono sempre float 32 bit (i.e. 1 è 1.00000)
  - Oggetti **GUI** float → funzione di display e input
    - Number box
    - Toggle
    - Sliders
    - Radio box

Messaggi

- **Messaggi generici** → “message box”: **[**message**(**
  - Funzione di display e input
  - Utilizzando un *selector* (header) è possibile esplicitare il tipo di messaggio (es. float, symbol, list)
  - Messaggi multipli con **,**
  - Messaggi multipli a destinazioni multiple con **;**
- **symbol** → “symbol box”: oggetto **GUI** con funzione di display e input

Segnali (audio)

- **Segnali**: data-stream floating point 32 bit [-1, 1]
  - Hardware audio solitamente limita a 16 o 24 bit
- Convenzione: oggetti che agiscono su segnali hanno nome che termina con ~ (tilde)
  - es. **[**osc~ 440**]**
- Elaborazione segnali attiva solo quando il motore audio è “acceso”
- I/O mediante oggetti **[**adc~**]** e **[**dac~**]**

Messaggi e segnali

- Gestione *interleaved*:
  - Messaggi processati all'inizio di ogni ciclo di DSP (es. a 44.1kHz, ogni 64 samples → 1.45ms)
  - Cascata *depth-first* di messaggi elaborata completamente prima del nuovo tick DSP
  - Messaggi mai passati durante un tick DSP (determinismo)
- Conversione tra segnali audio e messaggi: oggetti **[sig~]** e **[snapshot~]**

Subroutines

- *Pd* offre 2 modalità per creare subroutine:
  - ***Subpatch***: **[**pd nome**]**
    - Esiste ed è salvato solo nel patch corrente
    - N.B. ogni istanza è distinta anche se omonima
  - ***Abstraction***: **[**nomefile**]**
    - E' un patch salvato come file .pd
    - Subroutine da riutilizzare (libreria)
    - Passaggio di argomenti: es. **[**nomefile arg1 arg2**]** (v. *dollar signs*)
- I/O: si usano **[**inlet**]**, **[**outlet**]**, **[**inlet~**]**, **[**outlet~**]**
  - La disposizione grafica da sx a dx corrisponde all'ordine nell'object box

Subroutines

- *Abstractions* con proprietà “graph on parent”
  - subroutines con GUI riutilizzabile
  - mostrano elementi GUI e commenti nel patch genitore

Dollar signs $

- Permettono l'utilizzo di **variabili** in “message box” e -- solo utilizzando *abstractions* -- oggetti
  - es. **[**$1-message, $2**(**
  - es. *abstraction* con argomenti **[**abstr arg1 arg2**]**: oggetto interno **[**object $2 $1**]** → **[**object arg2 arg1**]**
- N.B. “message box” in *abstraction* NON riceve gli argomenti dell'*abstraction* → usare oggetti
- **[**message-$1**(** → **modificabile** dinamicamente; **[**obj $1**]** → solo **inizializzabile** dinamicamente

Dollar signs $

- Possibile **concatenazione** in messaggi variabili, o creazione/riordinamento di liste
  - es. **[**con-$1-cat**(**
  - es. **[**$2 foo $1**(**
- $0 in *abstraction* è un **contatore di istanze** → si usa per definire messaggi univoci per ogni istanza
  - es. **[**send $0-message**]**

Array

- *Pd* offre 2 modi per definire **array** di float:
  - Graphs (rappresentazione grafica)
  - Tables (di default hanno proprietà “graph on parent”)
- Indice 0..n-1
- Utilizzi:
  - **Segnali**: oscillatore wavetable, looping sampler, FdT
  - **Controlli**: mapping, sequencing, etc.
- Funzionano anche come GUI: input e display

Pd in uso

- **Finestra principale**:
  - Console
  - Opzioni / Controlli
  - Canvas
- Salvando un patch, *Pd* NON memorizza lo stato interno degli oggetti, ma solo quanto è visibile (N.B. oggetti GUI devono avere proprietà “init” attiva)
- **Undo**: 1 solo livello!

Pd in uso

- **Editing** → selezionare, creare / editare / eliminare oggetti e collegamenti, cut, paste, duplicate
- **Help** ed **elenco oggetti** (poco esaustivo) inclusi in *Pd* → utile integrare con documentazione alternativa: http://en.flossmanuals.net/PureData/ListofObjects
- Esempi messaggi:
  - Patch: hello world!
  - Patch: collegamenti hot / cold, uso trigger
  - Patch: depth-first
  - Patch: counter

Pd in uso

- Esempi segnali:
  - Patch: oscilloscopio + filtro
  - Patch: input audio
  - Patch: oscillatore wavetable
  - Patch: sintesi FM
- Esempi avanzati:
  - Controllo tramite tastiera, game pad, hid
  - SDT

Bibliografia

- Documentazione ufficiale (a tratti obsoleta ma valida): accessibile dall'help di *Pd* (html)
- http://en.flossmanuals.net/PureData/ continuamente aggiornato (html+pdf)
- Andy Farnell, Designing Sound - Practical synthetic sound design for film, games and interactive media using dataflow
  http://aspress.co.uk/ds/
  Versione ridotta (pdf) gratuita: introduzione a *Pd*

- Mailing list: http://lists.puredata.info/listinfo/pd-list