MINISTERO DELL'UNIVERSITÀ E DELLA RICERCA SCIENTIFICA E TECNOLOGICA
ALTA FORMAZIONE ARTISTICA E MUSICALE · PADOVA
CONSERVATORIO DI MUSICA
Cesare Pollini

Writing *PureData* Plug–ins

(V.581 2006-05-31)

Nicola Bernardini

nicb@sme-ccppd.org

Conservatorio Cesare Pollini, Padova, Italy

http://www.conservatoriopollini.it

School of Electronic Music

Workshop Overview (1)

- Day 1:
  - *PureData* basics: outside and under the hood
  - Plug–in types (control, audio, graphics)
  - *PureData* plug–in mechanisms
  - The **`simplest`** control plug–in
  - Argument passing and other details

Workshop Overview (2)

- Day 2:
  - *PureData* data types (control, audio)
  - advanced plug–in writing:
    - audio plug–ins
    - driving external devices
    - graphic plug–ins
  - Plug–in libraries

Workshop Overview (3)

- Day 3:
  - Tools for faster and scalable development: the *flext* library
  - Tools for portability (*autoconf*, *automake*, *autoproject*)
  - Tools for concurrent development (*svn*, track managers)
  - Overview of licensing schemes
  - Open issues (non–pluggable elements, multiple data–types, embedding pd in other applications, stand–alone applications)

Workshop Overview (4)

- Afternoon Labs:
  - Day 1:
    - Development environment setup
    - Write a simple control plug–in
  - Day 2:
    - Write an audio plug–in
    - Write a graphic plug–in
  - Day 3:
    - (Re)–Write plug–ins using *flext*
    - Implement the auto–project environment for the projects already produced
    - create (or use an already created) *svn* repository and associate it to a tracking system and put the plug–ins in it

References

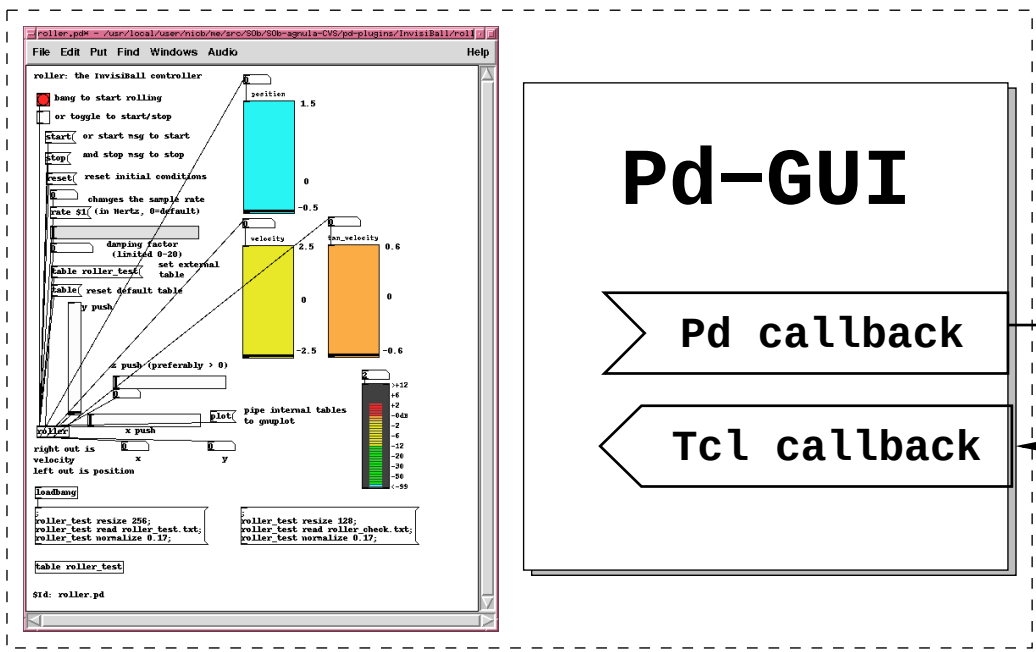
- the sources themselves :) (CVS: http://pure-data.sourceforge.net)

- http://pd.iem.at/externals-HOWTO/index.html

PureData Basics

- *PureData* is a graphical programming language
- It is Free Software (*FreeBSD*–style license)
-  Here is what it looks (and sounds) like; please take note the idiomatic ergonomy rules
- We won't delve much into *PureData* programming per se in this workshop

PureData under the Hood (1)



14_software_Bernardini-pd_scheme.fig,v 0.1 2003/04/06 10:50:16 nicb Exp

PureData under the Hood (2)

- *PureData* has a number of built–in objects
- *PureData* plug–ins follow the same implementation rules
- *PureData* uses standard kernel services to load the plug–ins
- the mechanism is completely transparent and portable

PureData Plug–ins (3)

- There are several types of *PureData* plug–ins:
  - control plug–ins
  - audio plug–ins
  - graphic plug–ins
  - . . .

Example: simplest_00 (1)

- Plug–in Mandatory code structure:

simplest.c

```
#include <m_pd.h>

void simplest_setup(...)
{
  ...

}
```

Called by the Host Application

Example: simplest_00 (2)

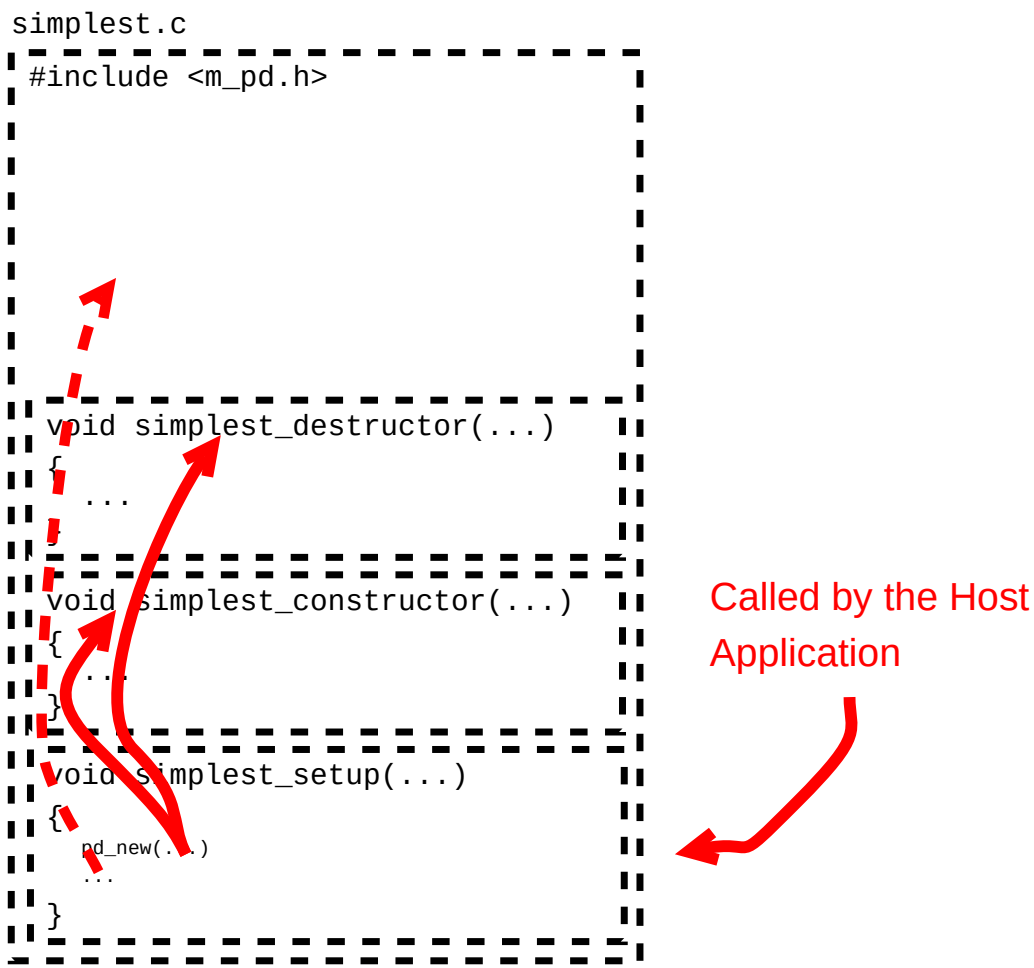
simplest_00.c

```
6   #include <m_pd.h>
7
8   void
9   simplest_00_setup(void)
10  {
11    post("Hello from inside the simplest_00 setup function");
12  }
```

- the only call the host application (*PureData* itself) calls is **<plugin name>_setup()**
- **<plugin name>_setup()** must then define everything else
- it is like the **main()** function call in a **C** program
- does it work? Let's compile it and run it

Example: simplest_01 (1)

- Let's proceed to something slightly less trivial:



Example: simplest_01 (2)

simplest_01.c-end

```
2   static void
3   simplest_01_destructor(void)
4   {
5     post("Sigh. I'm dead. Bye Bye");
6   }
7
8   void
9   simplest_01_setup(void)
10  {
11    post("Hello from inside the simplest_01 setup function");
12    simplest_01_class = class_new(gensym("simplest_01"),
13      simplest_01_constructor, simplest_01_destructor,
14      sizeof(t_simplest_01), CLASS_NOINLET, 0);
15    post("Done with the simplest_01 setting up");
16  }
```

- **CLASS_NOINLET** creates an object without inlets (not the default; other possibilities are: **CLASS_DEFAULT**, **CLASS_PD**, **CLASS_GOBJ**, **CLASS_PATCHABLE**)

Example: simplest_01 (3)

simplest_01.c-beg

```
6   #include <m_pd.h>
7
8   static t_class *simplest_01_class = (t_class *) NULL;
9
10  typedef struct _simplest_01_
11  {
12    t_object parent;  /* this must be first - mandatorily */
13  } t_simplest_01;
14
15  static void
16  *simplest_01_constructor(void)
17  {
18    t_simplest_01 *this_is_us =
19      (t_simplest_01 *) pd_new(simplest_01_class);
20    post("Hello from the *CONSTRUCTOR* now!");
21    return this_is_us;
22  }
```

- does it work? Let's compile it and run it

Example: simplest_02 (1)

- Now let's try to actually get some work done:
- Let's do a plug–in that:

a) has one inlet

b) accepts numbers in its inlet

c) outputs them when received

d) accepts **bang** messages in its inlet

e) outputs the last input number (or zero) when a **bang** message is received

Example: simplest_02 (2)

simplest_02.c setup

```
1  void
2  simplest_02_setup(void)
3  {
4    post("Hello from inside the simplest_02 setup function");
5    simplest_02_class = class_new(gensym("simplest_02"),
6      simplest_02_constructor, simplest_02_destructor,
7      sizeof(t_simplest_02), CLASS_DEFAULT, 0);
8    class_addfloat(simplest_02_class,
9      (t_method) simplest_02_float);
10   class_addbang(simplest_02_class,
11     (t_method) simplest_02_bang);
12   post("Done with the simplest_02 setting up");
13 }
```

Example: simplest_02 (3)

simplest_02.c constructor

```
1  static void *
2  simplest_02_constructor(void)
3  {
4    t_simplest_02 *this_is_us =
5      (t_simplest_02 *) pd_new(simplest_02_class);
6    this_is_us->memory = 0;
7    outlet_new((t_object *) this_is_us, &s_float);
8    post("Hello from the *CONSTRUCTOR* now!");
9    return this_is_us;
10 }
```

Example: simplest_02 (4)

simplest_02.c data structure

```
1  typedef struct _simplest_02_
2  {
3    t_object parent;  /* this must be first - mandatorily */
4    t_float  memory;
5  } t_simplest_02;
```

- we need to add a float element to the structure in order to keep track of the input

Example: simplest_02 (5)

simplest_02.c methods

```
1  static void
2  simplest_02_bang(t_simplest_02 *x)
3  {
4    outlet_float(x->parent.ob_outlet, x->memory);
5  }
6
7  static void
8  simplest_02_float(t_simplest_02 *x, t_floatarg f)
9  {
10   x->memory = f;
11   simplest_02_bang(x);
12 }
```

- does it work? Let's compile it and run it

PureData Data Types

- `t_symbol *gensym(char *s);`
- `SETFLOAT(atom, f)`
- `SETSYMBOL(atom, s)`
- `SETPOINTER(atom, pt)`
- `t_float atom_getfloat(t_atom *a);`
- `t_float atom_getfloatarg(int which, int argc, t_atom *argv);`
- `t_int atom_getint(t_atom *a);`
- `t_symbol atom_getsymbol(t_atom *a);`
- `t_symbol *atom_gensym(t_atom *a);`
- `void atom_string(t_atom *a, char *buf, unsigned int bufsize);`

Setup Functions (1)

- **`t_class *class_new(t_symbol *name, t_newmethod newmethod, t_method freemethod, size_t size, int flags, t_atomtype arg1, ...);`**
- **`void class_addbang(t_class *c, t_method fn);`** (**bang** function: **`void bang_method(t_mydata *x);`**)
- **`void class_addfloat(t_class *c, t_method fn);`** (**float** function: **`void my_float_method(t_mydata *x, t_floatarg f);`**)
- **`void class_addsymbol(t_class *c, t_method fn);`** (**symbol** function: **`void symbol_method(t_mydata *x, t_symbol *s);`**)
- **`void class_addpointer(t_class *c, t_method fn);`** (**pointer** function: **`void pointer_method(t_mydata *x, t_gpointer *pt);`**)

Setup Functions (2)

- **`void class_addlist(t_class *c, t_method fn);`** (**`list`** function: **`void list_method(t_mydata *x, t_symbol *s, int argc, t_atom *argv);`**)
- **`void class_addanything(t_class *c, t_method fn);`** (function: **`void my_any_method(t_mydata *x, t_symbol *s, int argc, t_atom *argv);`**)
- the generalized call: **`void class_addmethod(t_class *c, t_method fn, t_symbol *sel, t_atomtype arg1, ...);`**, where **`t_atomtype`** can be: **`A_DEFFLOAT`**, **`A_DEFSYMBOL`**, **`A_POINTER`**, **`A_GIMME`**

Instantiation Functions (1)

- **`t_pd *pd_new(t_class *cls);`**
- **`t_inlet *inlet_new(t_object *owner, t_pd *dest, t_symbol *s1, t_symbol *s2);`** This method an additional "active" inlet for the object that is pointed at by owner. Generally, **`dest`** points at **`owner.ob_pd`**. The selector **`s1`** at the new inlet is substituted by the selector **`s2`**. This means
  - The substituting selector has to be declared by **`class_addmethod`** in the setup-routine.
  - It is possible to simulate a certain right inlet, by sending a message with this inlet's selector to the leftmost inlet.
- **`t_inlet *floatinlet_new(t_object *owner, t_float *fp);`**
- **`t_inlet *symbolinlet_new(t_object *owner, t_symbol **sp);`**

Instantiation Functions (2)

- `t_outlet *outlet_new(t_object *owner, t_symbol *s)`; (`s` can be: `&s_bang`, `s_float`, `s_symbol`, `s_gpointer`, `s_list`, `0 (message)`, `&s_signal`)
- `void outlet_bang(t_outlet *x);`
- `void outlet_float(t_outlet *x, t_float f);`
- `void outlet_symbol(t_outlet *x, t_symbol *s);`
- `void outlet_pointer(t_outlet *x, t_gpointer *gp);`
- `void outlet_list(t_outlet *x, t_symbol *s, int argc, t_atom *argv);`
- `void outlet_anything(t_outlet *x, t_symbol *s, int argc, t_atom *argv);`

Afternoon Lab

- Development environment setup
- Write a simple control plug–in
- Some ideas: an input/output swapper, a simple control tempo estimator, a first-order derivative calculator, . . .
- Other ideas welcome!

2nd Day: Overview

- *PureData* data types (control, audio)
- advanced plug–in writing:
  - audio plug–ins
  - driving external devices
  - graphic plug–ins
- Plug–in libraries

Control Data vs. Audio Data

- Control data is *asynchronous* data: events happen whenever they have to, generating or interrupting other events
- Audio data is *synchronous* data: it requires a central control function that gets called ad periodic intervals ($\frac{s_r}{blocksize}$, thus, by default: $\frac{44100}{64} = 689.0625$ Hz)
- The central control function calls all the "audio" functions present in a patch and ordered in a directed graph
- Both s_r and $blocksize$ can be changed, so DO NOT assume any default value! In particular, $blocksize$ may be changed at any time using the **`block~`** object.

Example: simplest_00~(1)

- Let's try to get some sound immediately
- Let's do a plug–in that:

  a) has one audio inlet

  b) has one audio outlet

  c) simply transfer the audio in input to its output

Example: simplest_00~(2)

simplest_00~.c setup

```
1  void
2  simplest_00_tilde_setup(void)
3  {
4    post("Hello from inside the simplest_00~ setup function");
5    simplest_00_tilde_class = class_new(gensym("simplest_00~"),
6      simplest_00_tilde_constructor,
7      simplest_00_tilde_destructor,
8      sizeof(t_simplest_00_tilde), CLASS_DEFAULT,
9      A_DEFFLOAT, 0);
10
11   class_addmethod(simplest_00_tilde_class,
12     (t_method)simplest_00_tilde_dsp, gensym("dsp"), 0);
13
14   CLASS_MAINSIGNALIN(simplest_00_tilde_class,
15     t_simplest_00_tilde, f);
16
17   post("Done with the simplest_00~ setting up");
18 }
```

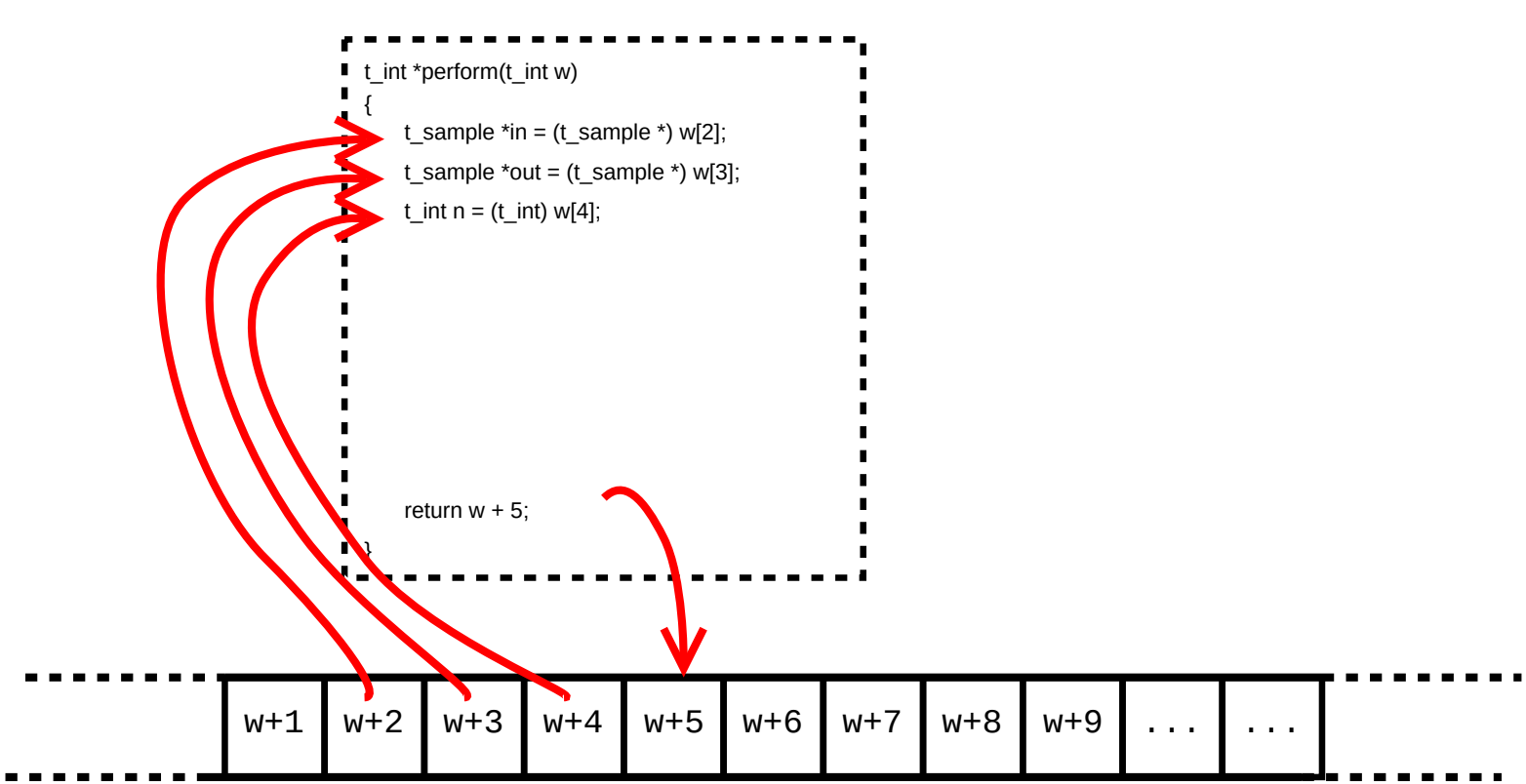
Example: simplest_00~(3)

simplest_00~.c dsp

```
1   t_int *simplest_00_tilde_perform(t_int *w)
2   {
3     /* w[1] is the object, not needed in this case */
4     t_sample *in1 = (t_sample *) (w[2]);
5     t_sample *out = (t_sample *) (w[3]);
6     int n = (int) (w[4]);
7
8     while (n--) *out++ = *in1++; /* not big deal :) */
9
10    return (w+5);
11  }
12
13  void
14  simplest_00_tilde_dsp(t_simplest_00_tilde *x, t_signal **sp)
15  {
16    dsp_add(simplest_00_tilde_perform, 4, x, sp[0]->s_vec,
17      sp[1]->s_vec, sp[0]->s_n);
18  }
```

Example: simplest_00~(4)

- Make sure you understand how the argument passing works here!



Example: simplest_00~(5)

simplest_00~.c constructor

```
1  static void *
2  simplest_00_tilde_constructor(void)
3  {
4    t_simplest_00_tilde *this_is_us =
5      (t_simplest_00_tilde *)
6      pd_new(simplest_00_tilde_class);
7
8    outlet_new((t_object *) this_is_us, &s_signal);
9
10   post("Hello from the simplest_00~ *CONSTRUCTOR* now!");
11
12   return this_is_us;
13 }
```

Example: simplest_00~(6)

simplest_00~.c data structure

```
1  typedef struct _simplest_00_tilde_
2  {
3    t_object parent;  /* this must be first - mandatorily */
4    t_sample f;     /* required by the audio macros */
5  } t_simplest_00_tilde;
```

- we need to add a **t_sample** element to the structure to comply with the **CLASS_MAINSIGNALIN** macro
- does it work? Let's compile it and run it

Harder Stuff

- Driver plug–ins
- Graphic plug–ins
- They are harder because:
  - There is little or no documentation about them (yes, there is always *the code* :)
  - They were really not thought to be pluggable, but plug–ins started to pop–up just the same

Writing Driver Plug–ins

Written in collaboration with Amalia De Götzen

- To understand what the problem is with drivers, let's go back a few steps
- *PureData* is an event (== callback) – based architecture
- two calls make up the interface with PD (`<plugin_name>_setup` and `<plugin_name>_new`)
- in these two calls you:
  1. establish the definition of a new object (`_setup`)
  2. define the features (number of inputs, outputs, callbacks to be called, . . .) (`_new`)
- `_setup` is run at start (load) time, while `_new` is run at instantiation time

The callback system

- callbacks are functions that get called with given arguments when a certain type of message is sent to the object
- there is also the audio callback which gets called periodically by pd itself (every $\frac{buffersize}{s_r}$ seconds)
- callback functions are non–preemptive (i.e. they cannot be interrupted by another callback)

Where's the Catch?

- if you need to receive data from an external device, you need to read it on file descriptor from a device driver
- *when* is that reading done?
- if you read upon an event (e.g. a bang) and you loose all the data in between and you don't care, then there's no problem
- if you want your object to drool data out as soon as it comes in, then you have to have your call *waiting* for data to come in
- since your call cannot be interrupted, the whole thing (including audio) gets stuck
- you get stuck

What now?

- Unix systems (with some equivalent in other systems too) provide a solution to this common problem: the **`select()`** function call
- **`select()`** allows you to *check* beforehand whether your file descriptor has data, so that you can selectively decide to `read()`
- this solves the problem *but*...
- ... you still have to decide *when* to do the **`select()`**!
- furthermore, it is good practice to have just *one* **`select()`** call per application

PureData comes to the rescue

- *PureData* has the capability of inserting/removing callbacks to its own call to **`select()`**;
- **`sys_addpollfn()`** and **`sys_rmpollfn()`** magic
- the structure of the program becomes a little convoluted :(

Here is what you have to do

1. sync to the incoming data (add to the **`select()`** queue a syncing function that waits for the proper data)
2. as soon as the syncing function is successeful, you de-queue the syncing function and en-queue a reading function
3. you read until you please
4. if you get out of sync, you de-queue the reading function and put back the syncing function
5. when you're done, you dequeue whatever function is in the **`select()`** queue

Example: the `isotrak` Driver (1)

isotrak.c setup

```
1  void isotrak_setup(void)
2  {
3    isotrak_class = class_new(gensym("isotrak"),
4      (t_newmethod)isotrak_new, (t_method)isotrak_free,
5      sizeof(t_isotrak), CLASS_DEFAULT, A_GIMME, 0);
6
7    class_addfloat(isotrak_class,
8      (t_method) isotrak_toggle_read);
9    class_addmethod(isotrak_class,
10     (t_method) isotrak_set_boresight,
11     gensym("boresight"), 0);
12   class_addmethod(isotrak_class,
13     (t_method) isotrak_set_unboresight,
14     gensym("unboresight"), 0);
15   post("Polhemus Isotrak II ($Revision: 0.13 $)\
16       module by Amalia de Goetzen");
17 }
```

Example: the `isotrak` Driver (2)

isotrak.c constructor begin

```
1   static void *
2   isotrak_new(t_symbol *s, t_int argc, t_atom argv[])
3   {
4     char serial_device[ISOTRAK_DEV_SIZE] = { '\0' };
5     const unsigned int bufsize = ISOTRAK_DEV_SIZE;
6     int baud_rate = ISOTRAK_DEFAULT_BAUD_RATE;
7     t_isotrak *x = (t_isotrak *)pd_new(isotrak_class);
8     int i = 0;
9
10    for(i = 0; i< ISOTRAK_OUTPUTS; ++i)
11      x->out[i] = outlet_new(&x->x_obj, &s_float);
```

Example: the `isotrak` Driver (3)

isotrak.c constructor end

```
1   switch (argc)
2   {
3     case 2:    baud_rate = atom_getfloat(&argv[1]);
4     case 1:   atom_string(&argv[0], serial_device, bufsize);
5           break;
6     default:
7     case 0:    strncpy(serial_device, ISOTRAK_DEFAULT_DEVICE,
8                bufsize);
9           break;
10
11  }
12  x->fd =  isotrak_start(serial_device, baud_rate, &x->old);
13  x->x_toggle = 0;
14  if (x->fd < 0)
15    post("isotrak open failed");
16  return (void *)x;
17 }
```

Example: the `isotrak` Driver (4)

isotrak.c data structure

```
1  #define ISOTRAK_OUTPUTS (6)
2
3  typedef struct _isotrak {
4    t_object x_obj;
5    t_int    fd;    /* serial port file descriptor */
6    /* x, y, z, azimuth, elevation, roll */
7    t_outlet *out[ISOTRAK_OUTPUTS];
8    t_int    x_toggle;
9    IsotrakData data;
10   struct termios old;
11 } t_isotrak;
```

Example: the `isotrak` Driver (5)

isotrak.c toggle callback

```
1  static void
2  isotrak_toggle_read(t_isotrak *x, t_float f)
3  {
4    t_int i = (t_int) f;
5    if(i ^ x->x_toggle)
6    {
7      if (i)
8      {
9        sys_addpollfn(x->fd, isotrak_sync, x);
10       isotrak_start_querying(x->fd);
11     }
12     else
13     {
14       isotrak_stop_querying(x->fd);
15       sys_rmpollfn(x->fd);
16     }
17   }
18   x->x_toggle = i;
19 }
```

Example: the `isotrak` Driver (6)

isotrak.c sync function

```
1   static void
2   isotrak_sync(void *object, int fd)
3   {
4     t_isotrak *x = (t_isotrak *) object;
5
6     if (isotrak_resync(fd, &x->data))
7     {
8       sys_rmpollfn(x->fd);
9       sys_addpollfn(x->fd, isotrak_fill_buffer, x);
10    }
11  }
```

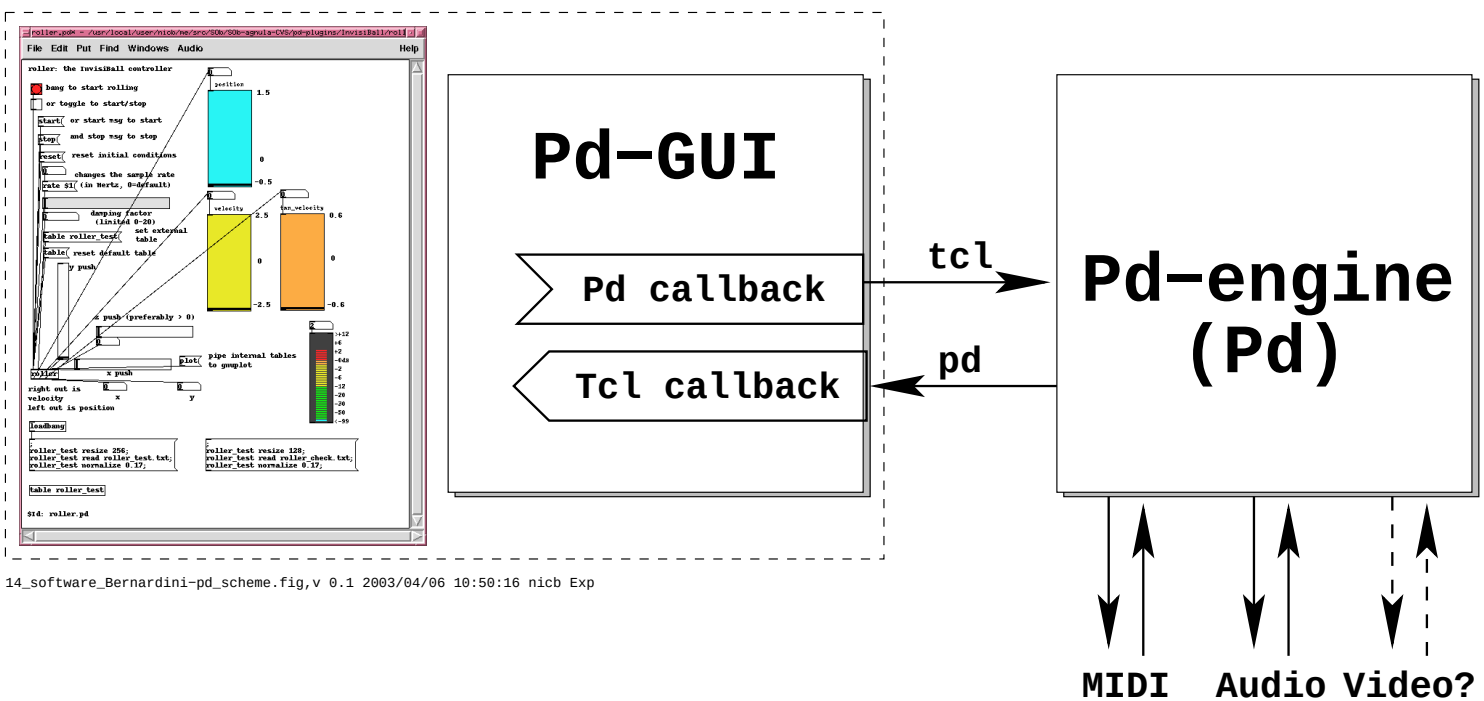
Example: the `isotrak` Driver (7)

isotrak.c run function

```
1   static void
2   isotrak_fill_buffer(void *object, int fd)
3   {
4     t_isotrak *x = (t_isotrak *) object;
5     int rv = isotrak_receive(fd, &x->data);
6
7     if (rv > 0)
8       isotrak_output(x);
9     else if (rv < 0) /* everything has gone out of sync */
10    {
11      sys_rmpollfn(x->fd);
12      sys_addpollfn(x->fd, isotrak_sync, x);
13    }
14  }
```

Writing Graphic Plug–ins (1)

- Writing graphic plug–ins is (possibly) even more convoluted
- Let's remember how is *PureData* running:



14_software_Bernardini–pd_scheme.fig,v 0.1 2003/04/06 10:50:16 nicb Exp

Writing Graphic Plug–ins (2)

- You will have to:
  - Draw on the **`tcl/tk`** part of *PureData* (using the **`sys_vgui()`** call)
  - Provide all the functions for the interaction between the graphic object on the **`tcl/tk`** part and the *PureData* engine
  - Provide all the functions for the normal graphic interaction (e.g. create, redraw, delete, move, resize, etc.)
  - Provide the property dialog
  - etc.

Example: simplestg (1)

simplestg.c head

```
1   #include <stdarg.h>
2   #include <m_pd.h>
3   #include "g_canvas.h"
4
5   static t_class *simplestg_class = (t_class *) NULL;
6   static t_widgetbehavior simplestg_wb;
7
8   typedef void (*t_drawfun)(void *x, t_glist *glist, int mode);
9
10  typedef struct _simplestg_
11  {
12    t_object  x_obj;
13    t_drawfun draw;
14    t_glist   *context;
15  } t_simplestg;
```

Example: simplestg (2)

simplestg.c setup

```
1  void
2  simplestg_setup(void)
3  {
4    simplestg_class = class_new(gensym("simplestg"),
5      simplestg_constructor, simplestg_destructor,
6      sizeof(t_simplestg), CLASS_PATCHABLE, A_NULL, 0);
7
8    simplestg_set_widget(&simplestg_wb);
9    class_setwidget(simplestg_class, &simplestg_wb);
10 }
```

Example: simplestg (3)

simplestg.c widget setup

```
1  static void
2  simplestg_is_visible(t_gobj *z, t_glist *glist, int v)
3  {
4    t_simplestg *x = (t_simplestg *)z;
5    if (v)
6      (*x->draw)((void *)z, glist, 0);
7  }
8  static void
9  simplestg_set_widget(t_widgetbehavior *wb)
10 {
11   wb->w_getrectfn  = NULL; /* no dimensions */
12   wb->w_displacefn = NULL; /* no motion */
13   wb->w_selectfn   = NULL; /* no selection */
14   wb->w_activatefn = NULL; /* no activation */
15   wb->w_deletefn   = NULL; /* no deletion */
16   wb->w_visfn      = simplestg_is_visible;
17   wb->w_clickfn    = NULL; /* no clicking */
18 }
```

Example: simplestg (4)

simplestg.c constructor

```
1   static void *
2   simplestg_constructor()
3   {
4     t_simplestg *us = (t_simplestg *)
5       pd_new(simplestg_class);
6
7     us->draw = simplestg_draw;
8     us->context = (t_glist *) canvas_getcurrent();
9
10      outlet_new(&us->x_obj, &s_bang);
11
12    return us;
13  }
```

Example: simplestg (5)

simplestg.c drawing function

```
1  static void
2  simplestg_draw(void *obj, t_glist *glist, int mode)
3  {
4    t_simplestg *us = (t_simplestg *) obj;
5    int x = text_xpix(&us->x_obj, glist);
6    int y = text_ypix(&us->x_obj, glist);
7    t_canvas *c = glist_getcanvas(glist);
8
9    write_to_gui(us, ".x%lx.c create rectangle %d %d %d %d\
10     -fill #%6.6x -tags %lxBASE\n",
11     c, x, y, x + SIMPLESTG_WID, y + SIMPLESTG_HT,
12     SIMPLESTG_RCOL);
13   write_to_gui(us, ".x%lx.c create oval %d %d %d %d\
14       -fill #%6.6x -tags %lxBUT\n",
15     c, x+3, y+3, x + SIMPLESTG_WID-3, y + SIMPLESTG_HT-3,
16     SIMPLESTG_OCOL);
17 }
```

Example: simplestg (6)

simplestg.c actual drawing

```
1  static void
2  write_to_gui(t_simplestg *us, char *fmt, ...)
3  {
4    t_canvas *canvas = (t_canvas *) NULL;
5    int xs = 0, ys = 0, xe = 0, ye = 0, col = 0;
6    va_list ap;
7    va_start(ap, fmt);
8
9    canvas = va_arg(ap, t_canvas *);
10   xs = va_arg(ap, int); ys = va_arg(ap, int);
11   xe = va_arg(ap, int); ye = va_arg(ap, int);
12   col = va_arg(ap, int);
13   va_end(ap);
14
15   post(fmt, canvas, xs, ys, xe, ye, col);
16   sys_vgui(fmt, canvas, xs, ys, xe, ye, col, us);
17 }
```

Example: `simplestg` (7)

- does it work? Let's compile it and run it
- of course, since we provided only the basic drawing function, it won't do much but appear :(
- doing a bit more involves writing many more methods
- the source is the only reference (in particular, this one was strongly influenced by the **`g_bang.c`** code)

Writing Entire Libraries (1)

- Building plug–in libraries couldn't be simpler:

1. Create a file of the same name of the library you want to create
2. Create a **`setup`** function for that file
3. Call all the **`setup`** functions of the objects you want to include in the library from *that* **`setup`** function
4. And you are done!

Writing Entire Libraries (2)

simplest library

```
1  const char *announce =
2  "simplest library ($Revision: 579 $), (C) 2006 Nicola Bernardini\n\
3  simplest comes with ABSOLUTELY NO WARRANTY\n\
4  This is Free Software, and you are welcome to redistribute it\n\
5  under certain conditions";
6
7  void
8  simplest_setup(void)
9  {
10   post(announce);
11   simplest_00_tilde_setup();
12   simplest_00_setup();
13   simplest_01_setup();
14   simplest_02_setup();
15   simplest_00_setup();
16   simplestg_setup();
17 }
```

- Let's try if this one works

Afternoon Lab

- Write an audio plug–in
- Write a driver plug–in
- Write a graphic plug–in
- Put all your plug–ins in a single library
- Other ideas always welcome!

3rd Day: Overview

- Day 3:
  - Tools for faster and scalable development: the *flext* library
  - Tools for portability (*autoconf*, *automake*, *autoproject*)
  - Overview of licensing schemes
  - Tools for concurrent development (*svn*, track managers)
  - Open issues (non–pluggable elements, multiple data–types, embedding pd in other applications, stand–alone applications)

Using `flext` (1)

- **`flext`** is a **`C++`** library conceived by Thomas Grill (gr@grrrr.org)

- its aims are to produce plug–ins that:
  - are simpler to write
  - can be hosted by *Max/MSP* applications or by *PureData* applications without changing source code
  - can run on a variety of platforms without changing source code

Using `flext` (2)

- **`flext`** pros:
  - simple plug–ins are much simpler to write
  - host/system independency *can* be achieved
- **`flext`** cons:
  - complexity gets hidden under a carpet
  - its build system is overly complicated and bugged (don't use it!)

Example: simplest_flext_00.cpp (1)

simplest_flext_00.cpp Part 1

```
1  #include <flext.h>
2
3  #if !defined(FLEXT_VERSION) || (FLEXT_VERSION < 400)
4  #error You need at least flext version 0.4.0
5  #endif
6
7  class simplest_flext_00: public flext_base
8  {
9    FLEXT_HEADER(simplest_flext_00,flext_base) // mandatory
10
11 public:
12   simplest_flext_00()
13   {
14     AddInAnything();  // add one inlet for any message
15     AddOutFloat();  // add one float outlet (has index 0)
16     FLEXT_ADDMETHOD(0,m_float);
17   }
```

Example: simplest_flext_00.cpp (2)

simplest_flext_00.cpp Part 2

```
1   protected:
2     void m_float(float input)  // method for float values
3     {
4       float result;
5
6       if(input == 0) {
7         post("%s - zero can't be inverted!",thisName());
8         result = 0;
9       }
10      else
11        result = 1/input;
12
13      ToOutFloat(0,result);
14    }
15
16  private:
17    FLEXT_CALLBACK_1(m_float,float)
18  };
```

Example: `simplest_flext_00.cpp` (3)

simplest_flext_00.cpp Part 3

```
1  // instantiate the class
2  FLEXT_NEW("simplest_flext_00",simplest_flext_00)
```

- Let's try if this one works

Example: `Makefile` (1)

Makefile

```
1  S=pd_linux
2  TARGET=simplest_flext_00.$S
3  SOURCE=$(TARGET:.$S=.cpp)
4  OBJ=$(SOURCE:.cpp=.o)
5  CC=gcc
6  CXX=g++
7  INCLUDES=-I/usr/local/include/flext
8  CXXFLAGS=-DFLEXT_SYS_PD -DFLEXT_SHARED $(INCLUDES)
9  LIBS=-lflext-pd
10 LD=g++
11
12 $(TARGET):  $(SOURCE)
13
14 clean:
15   $(RM) $(TARGET) $(OBJ) *.cpp-[A-Z]*
16
17 .SUFFIXES:  .pd_linux
18 %.pd_linux:  %.o
19   $(LD) -Wl,-Bdynamic -shared -o $@ $< $(LIBS)
```

So you want to write plug–ins, huh?

- Knowing how to build plug–ins is not sufficient to succeed in having them picked up and used by other users
- If that is what is seeked, then:
  1. dowload $\Rightarrow$ compile $\Rightarrow$ install procedures complexities must compare well to the complexity of the task
  2. they must work on *any* platform
  3. they must be *idiomatic* for developers
  4. licensing rights for doing all this must be clearly stated and idiomatic for developers
- Nowadays, failure to comply to these rules is a safe bet for disaster

Autotools (1)

Thanks to Davide Rocchesso and Carlo Drioli for this section

- since the ”80s onward a number of tools to provide idiomatic dowload ⇒ compile ⇒ install sequences have been devised and built
- they are generally termed as the **Autotools** set and they include:
  - **autoconf**, a set of **m4** macros to produce **configure** files out of **configure.ac** files (and **Makefile** files out of **Makefile.in** files)
  - **aclocal**, a set of **m4** macros to setup the work of **autoconf** properly on every instance
  - **automake**, a set of **m4** macros to produce **Makefile.in** files out of **Makefile.am**
  - **acheader** a set of **m4** macros to produce config.h.in files (to be transformed later by **configure** into config.h)
  - etc.

Autotools (2)

- **Autotools** pros:
  - they produce setups that are idiomatic for other developers to pick–up
  - they enhance portability among platforms and setups
  - they are well documented
- **Autotools** cons:
  - they are overly–complicated for casual developers
  - the learning curve is *very* steep
- Luckily, some *easy* tools that will carry out most of the setup work for you are now available. Let's check **acmkdir**, for example (http://autotoolset.sourceforge.net).
- Other such tools are **autoproject**, etc.

Licensing Schemes

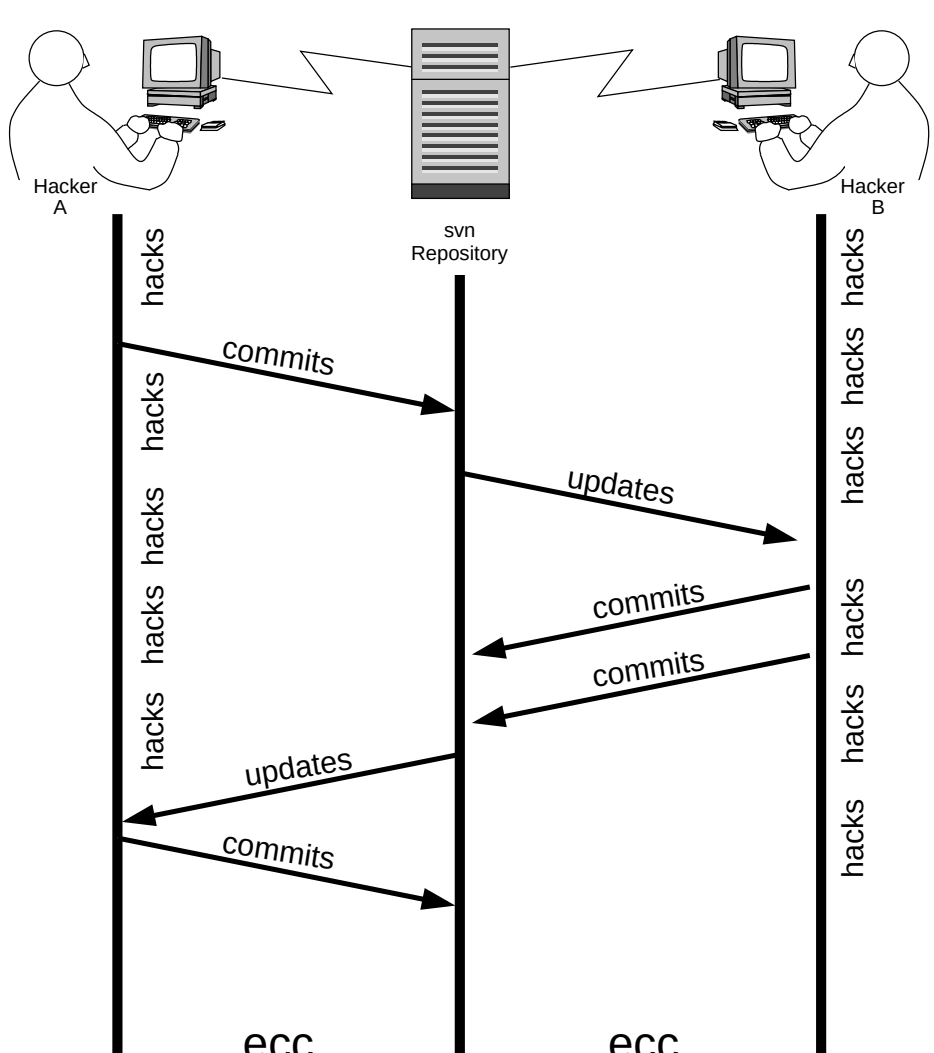
- The importance of *how* do you license your code, documentation, setup etc. cannot be stressed enough
- Failure to provide clear licensing is bound to disaster
- Stick to idiomatic licensing! Developers do not have the time to check whether they can accept your special license or not
- **`GPL v.2.0`** for source code, **`CC BY-SA v.2.5`** for papers, documentation, etc.
- **`GPL v.3.0`** may cater plug–in code differently in the future

Repositories, Tracking, Ticketing

Thanks to Damien Cirotteau for the **`tracker`** hint

- Nowadays it is customary to share development among a community of developers
- This cannot be done without the proper tools and setup
- Currently, the upcoming most commonly used tool for shared repository development is *subversion* (**`svn`** – http://subversion.tigris.org/)
- **`svn`** builds upon the concepts set up in a long sequence of source managers, starting in the ”70s with **`SCCS`**, through **`RCS`** and **`CVS`**
- **`svn`** can easily be connected to a **`trac`** (http://trac.edgewall.org) website for tracking and ticketing (here is one **`trac`** instance)
- Other tracking and ticketing systems are popping up; check, for example, **`tracker`** (http://www.rousette.org.uk/projects/)

svn (1)


Hacker A
svn Repository
Hacker B
hacks
commits
updates
commits
commits
updates
commits
ecc.
ecc.

svn (2)

- For help on **svn**:

```
$ svn help
$ svn help <subcommand>
```

- To create an **svn** repository:

```
$ svnadmin create <repository path>
$ svnadmin help
$ svnadmin help <subcommand>
```

- **svn** repositories may be accessed via a stand-alone **svn** server (both plain and *SSL*), and via an *Apache2 WebDAV*

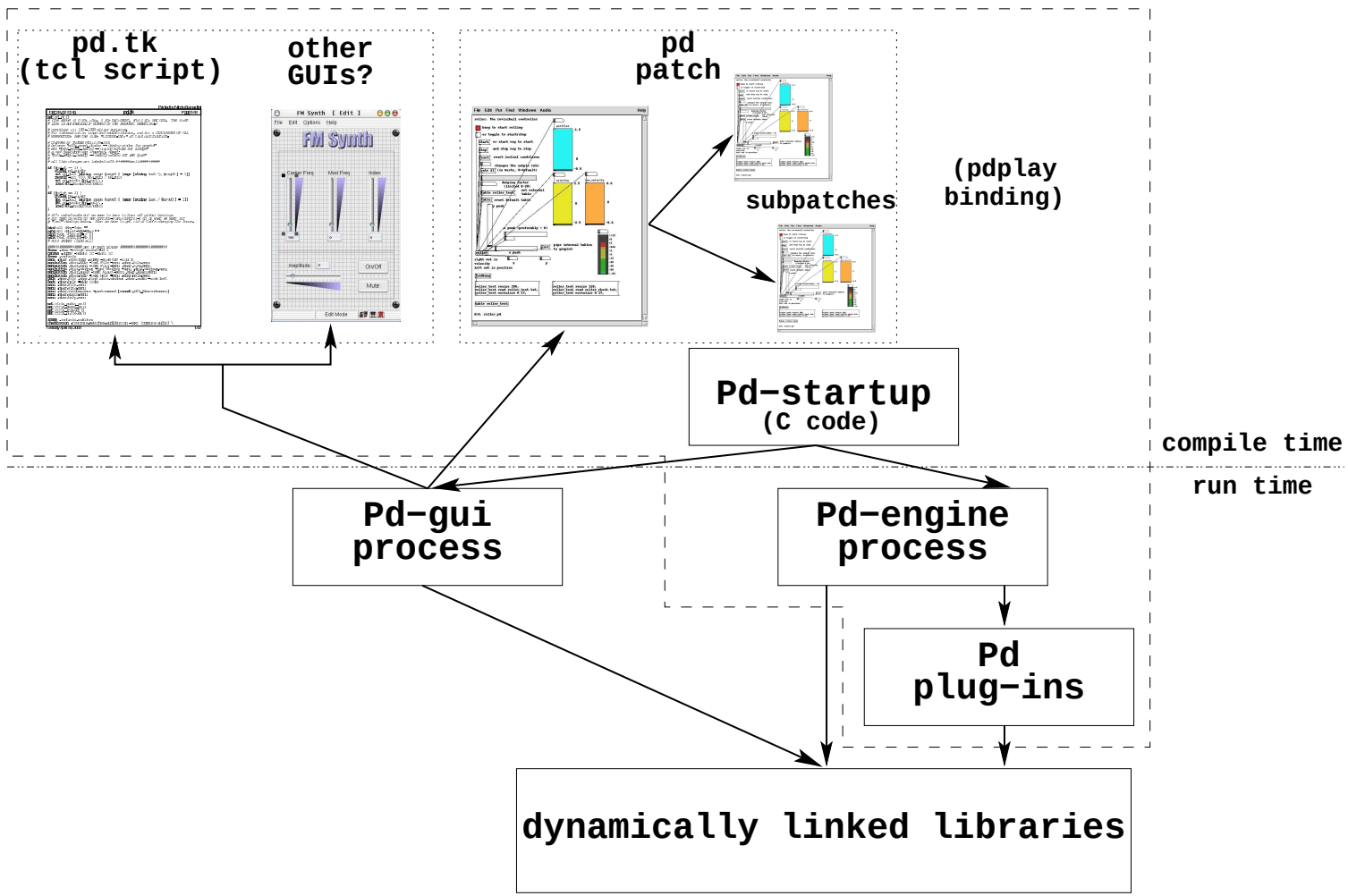
svn (3)

- **`svn`** commands are mostly like **`cvs`**'
- different behaviours:
  - revision is not for the single file, but for a given snapshot of an entire repository
  - **`svn status`**
  - file *properties*
  - **`move`**, **`delete`**, symbolic links allowed
  - most operations are carried out off–line: on–line operations are concentrated and bundled so that on–line time is minimized
  - there is *no* tagging feature (tagging and branching is done in another way)

PureData plug–ins: open issues

- call *PureData* from other applications (doable)
- stand–alone *PureData* applications (**`pdplay`**)
- multiple data–types plug–ins (i.e. video streaming, complex frequency domain data–types, etc.)
- *PureData* web–serving
- . . .

A look at pdplay (1)



14_software_Bernardini-pd_full.fig,v 0.0 2003/04/08 22:07:05 nicb Exp

A look at pdplay (1)

- Incorporate several processes in a single application file
- Incorporate dynamic libraries in a single application file (i.e.: link statically)
- Incorporate plug–ins in a single application file
- Some processes run on interpreters! (i.e. they get never compiled)
- Interpreters may run abstractions, which must all be incorporated in a single application file
- . . .