DESIGN OF THE HARDWARE ABSTRACTION LAYER FOR THE 3D ACQUISITION SYSTEM OF A.I.L.U.N. AND CRS4

Janusz Kozlowski, AILUN

Gianstefano Monni

Piero Pili, CRS4

INDEX

1 Design of Hardware Abstraction Layer

During development of software we've discovered that would be better divide hardware management from all the other parts of code.
We needed to achieve many results:

- a modular, manageable and portable code (from windows to Linux).
- support of a loto of cameras and runtime choice of cameras
- de-coupling from management of different devices inside th acquisition system
- unified driver handling system (open, close, RTacquire e DSacquire) simplifies drivers development by people who do not belong to the team

To achieve these goals we have re-engineered the code that communicates with the hardware and we've developed a Hardware Abstraction Layer using techniques well known in software engineering (design patterns, petri nets, and so on)

1.1 Design Patterns

1.1.1 Introduction

Christopher Alexander says "*Every pattern describes a problem that we see many times in our work, then it describes the core of the solution such as we could use it thousands times without applying it in the in in the same way*" [AIS 77, pag. x]. Alexander was talking about architectural patterns for cities and houses, what he says it is true even with pattern for object oriented programming. Our solutions are expressed in terms of objects and interfaces but the basic idea is that a pattern is a solution for a problem in a specific context

A pattern is composed of four basic elements:

- **the name of the pattern** : it is a symbolic name that we could use to describe in a word a design problem, its solutions and the solution we've chosen
- **the problem** describes the situation in which we can apply the pattern. It can describe specific design problems or class structures of projects that are not much flexible.
- **the solution:** describes the elements that are part of the project, its relationship, the responsibility and collaboration
- **the consequences:** they're the results and ties that we could obtain with the application of the pattern.

The point of view modifies the interpretations of the pattern nature: what a pattern represents for a person, for an other person could a simple starting point. Design patterns are not linked list or hash tables that we could define with a single class and re-use as they are. They are not complex project of an entire application or sub-system.

Design patterns are descriptions of inter-communicating objects and classes, suitable to resolve a design problem in a particular context.

1.2 The Singleton design pattern

1.2.1 Introduction

The problem that we've discovered during software development

There is a set of parameters that describes geometry, different corrections, and other values that are fundamental for the all other part of application. This set of parameters must be shared from all the object of our application. It must be possible to modify quickly all the parameters of the set.

The pattern: *to have one and only one instance of a class, and a global access point to this class.*

The Singleton defines a Instance operation that permits to the client to approach to the only existing instance of the class. Instance must be a class operation: i.e. it must be a C++ static member function. Instance could be responsible of creation of its only instance.

1.2.2 Advantages

controlled access to the only instance: Singleton class encapsulates its only instance, so it could keep a strict control over times and ways to access to the only instance.
reduction of namespace.

The *Singleton* pattern represents a better approach in respect of use of global variables: Singleton has no impact in namespace pollution.

- it permits refinement of representation operations. It is possible to define sublclasses of Singleton class and it is simple to configure an application

and it is very simple to configure an application in such a way to use an instance of this extended class.

- it permits to handle a variable number of instances
- it has greater flexibility in respect of class operations..

1.2.3 Implementation.

The Singleton patter has been implemented in a class, Cparameters that has been compiled as a DLL, and has been used from all the other objects of our application.

```
class DLL_EXP CParameters
{
private:
        float Wv;
protected:
  static CParameters * parm;

  CParameters(CParameters&){};
  CParameters operator =(CParameters&){};
  CParameters();
  ~CParameters(){};

public

  static CParameters * getInstance();
  static void deleteInstance();
```

```
float getWv(){return Wv;}
}
```

Use case :

```
CParameters * parm=CParameters::getInstance();
float inverse=1.0/parm->Wv;
```

Clients approach to the singleton only by the static member function getInstance. The parm object has been initialized to zero, and the member static function getInstance returns its address initializing it with the instance reference if it is zero: getInstance uses la "*lazy initialization*": the returned value is not created and memorized until an object tries to use it for the first time.

It must be noted that if the class constructor is declared as protected and a client tries to instance directly the Cparameters class it obtain a compile error. This fact assures oneness of the created instance.

1.3 The Proxy pattern

1.3.1 Introduction

The problem we've seen during development:

We have to support a unified driver handling system: it must be possible to choice, with the same interface, a USB, PCI, Firewire o Frame grabber driver. Drivers for different buses and cameras must have the same interface and behave in the same way, they FSM must be only one for ALL the drivers.

The two patterns: *Define an interface for object creation, leaving to subclasses the decision of what is the actual subclass to be instanced.*
The access to the actual object (IE Firewire or USB2 camera) MUST be controlled.

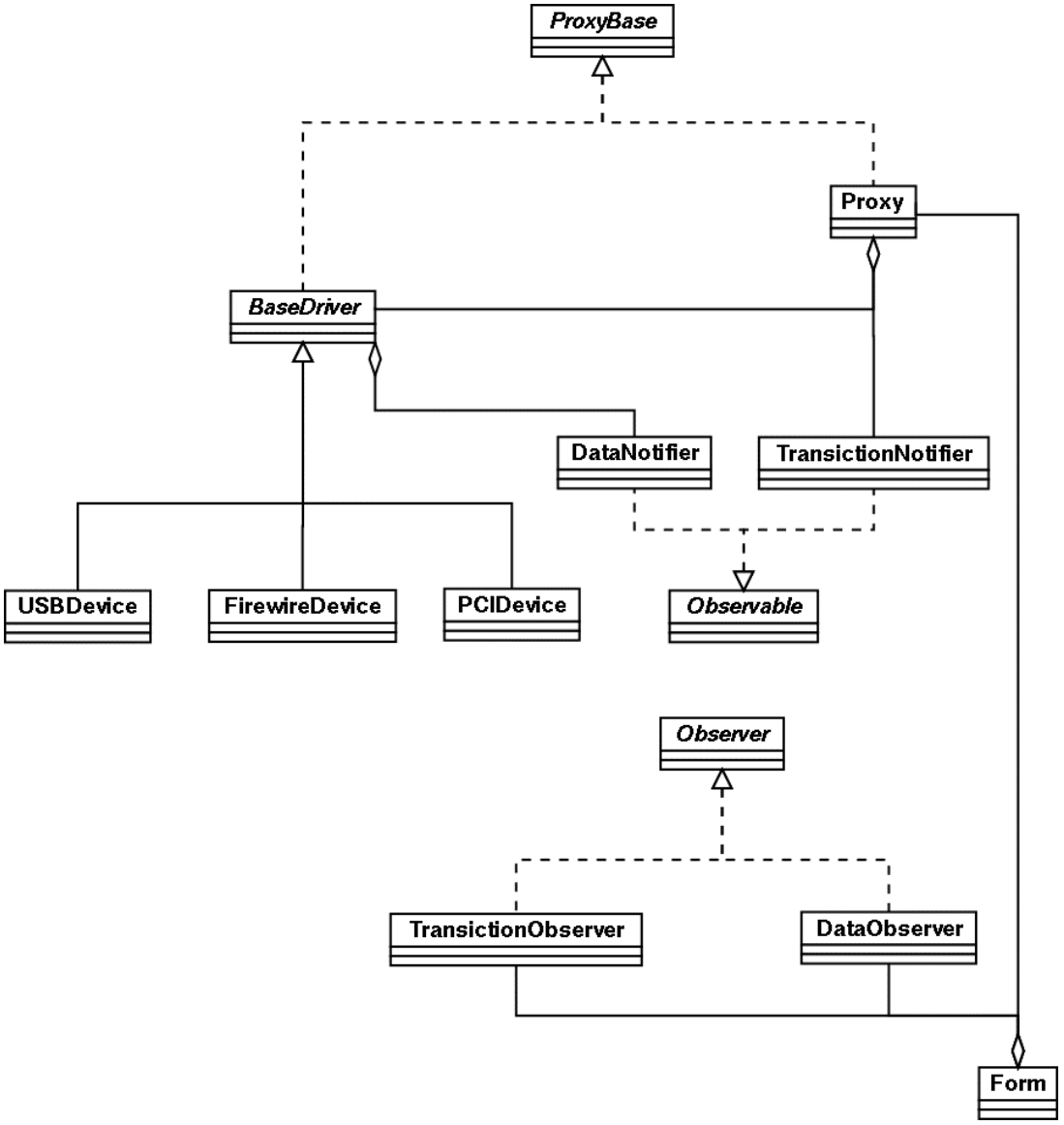


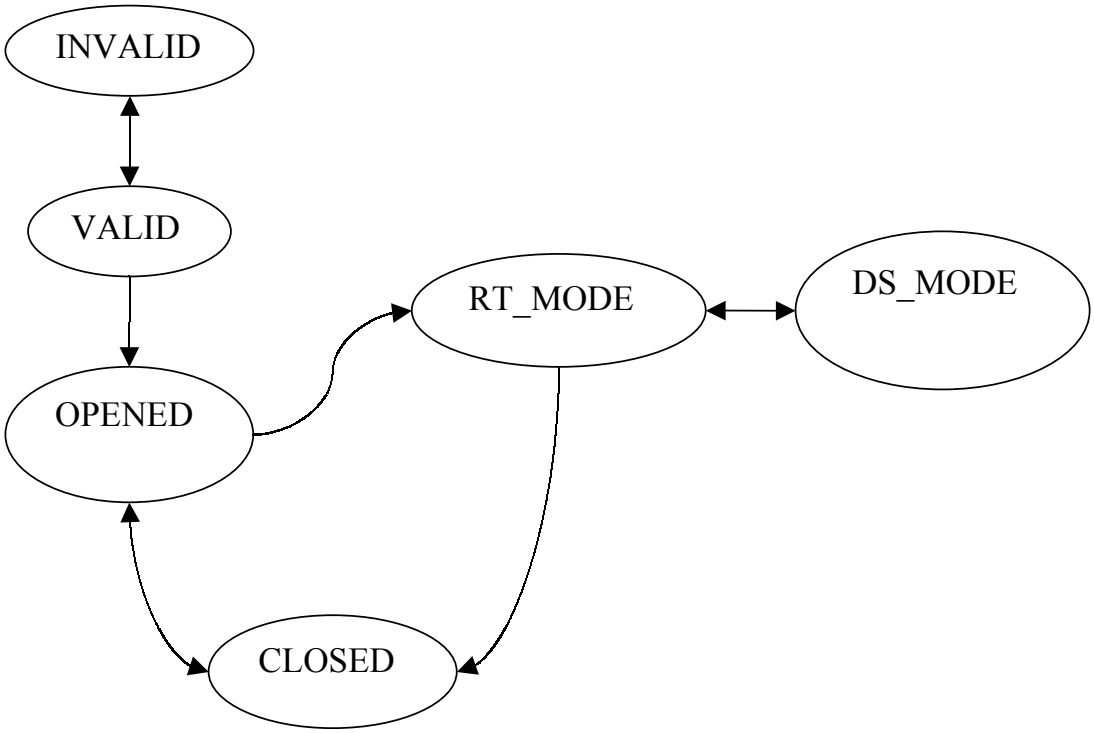
Fig. 1 UML Class Diagram

ProxyBase is the interface common to BaseDriver and Proxy.

Proxy hides to the external world the choice of what is the driver, and external world uses the setBase (int idDriver) method to be sure that the actual object is correct. Proxy hides to the external world the problems regarding opening and closure of driver, construction of objects, and so on. Proxy implements all the functionalities of BaseDriver, but it makes it using an internal object, and only Proxy can create/destroy the actual driver.

BaseDriver è is the superclass (abstract) of all actual drivers, and it implements some. All methods of Basedriver are virtual, in particular RTAcquire, DSAcquire, open e close (pure virtual methods). The basic idea is that, for a minimal driver realization, these four methods are sufficient, and that these four methods must not be implemented in a class (BaseDriver) that it is not linked to any actual device.
{Firewire/PCI/USB}Driver are three subclasses of BaseDriver and they contain the implementation of the four methods previously stated.

1.3.2 The Proxy FSM



2 Problems with the driver of every camera

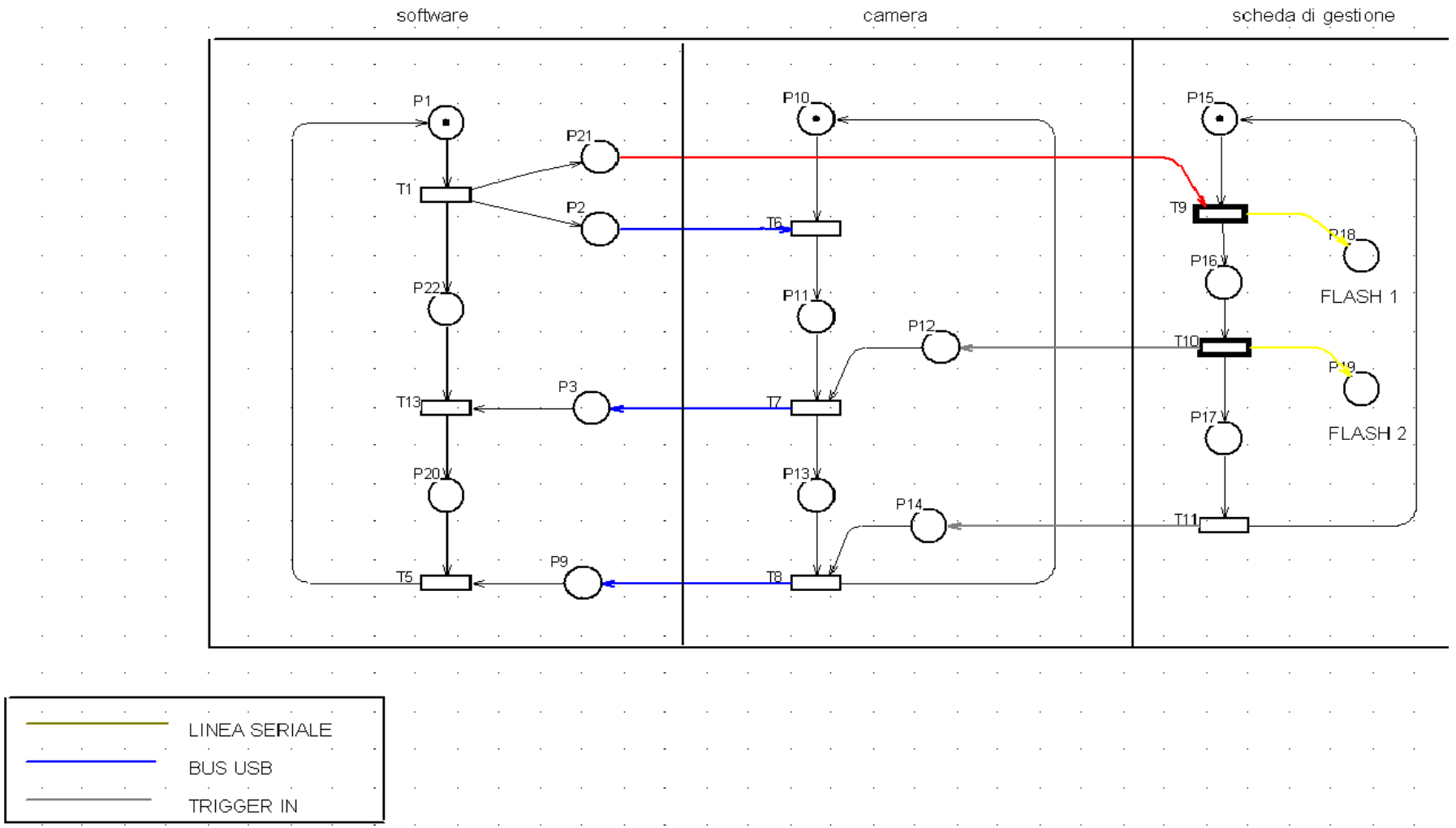
2.1 Introduction

During implementation of actual drivers of camera, we've seen many problems linked with different interpretation (of cameras producer) of VxD (Virtual Device Drivers). In particular we have been involved in two problems :

- design of an asynchronous protocol for camera handle: one of our camera had not (in so, it was out of our minimal hardware specifications) trigger OUTPUT working signal.
- implementation of a GrabTriggered multithread method

2.2 The asynchronous protocol

One of our camera had not a TRIGGER OUTPUT working signal. For this reason the previous synchronous protocol could not work as expected. To be able to implement the driver for this camera, we had to design an asynchronous protocol for management board, camera, and software.



Petri net of our asynchronous protocol

2.3 Implementation of GrabTriggered (SICamera)

Implementation of GrabTriggered requires the realization of a thread that stays in the state BLOCKED until arises a signal that image have been correctly acquired. Implementation of a Thread is remitted to the Cthread class, superclass of SICamera.

The caller goes in wait state on grabbedEvent (binary semaphore its starting state is not signaled). When SICamera completes execution (inside of a critical section) it signals the semaphore and unblock the caller.

```
void SICamera::Execute()
{
 CRITICAL_SECTION cs;
   InitializeCriticalSection(&cs);
   EnterCriticalSection(&cs)
...

   ResetEvent(grabbedEvent);

//actual grab
PrivateGrab(0);

 SI_DisarmTrigger(theCamID);
LeaveCriticalSection(&cs);
DeleteCriticalSection(&cs)
;
//signal semaphore: acquisition is completed
SetEvent(grabbedEvent);
}
```

2.3.1 Implementation of CThread

To achieve the maximum portability, threads have been incapsulated in a superclass , CThread, that gives minimal support for multithreading

```
/*
constructor of the class: use win32 API to create a thread, assigning propertly
callback functions to methods of the actual object that has been created.
*/
CThread::CThread()
{
  theStatus=tsIDLE;
  t_box= new ThreadBox(this);
   hThread=CreateThread(NULL,
     0,
   (unsigned long (__stdcall *)(void *))&(CThread::_execute),
   (void *)t_box, CREATE_SUSPENDED, NULL);

   if (hThread==NULL)
     throw new Exception("Can't create Thread");
}

CThread::~CThread()
{
  CloseHandle(hThread);
}


void CThread::_execute(void * PtrThis)
{
```

```
  ThreadBox * ptr = (ThreadBox * )PtrThis;
  CThread * this_t=ptr->this_thread;
  this_t->theStatus=tsRUNNING;
  this_t->Execute();
  this_t->theStatus=tsENDED;
}
void CThread::t_restart()
{
  ResumeThread(hThread);
}
void CThread::t_suspend()
{
  SuspendThread(hThread);
}
```

CThread is an abstract class : every real subclass of CThread must implement the Execute() method that contains the code that is run in the thread.
The _execute*(void * PtrThis)* method is the one that is called in callback, from win32API, and this method obtain the pointer to the current actual object. Usiing the "this " pointer we could then call the Execute method (a pure virtual method in the superclass CThread, concrete virtual in its subclasses).

2.3.2 Portability considerations

GrabTriggered and Cthread are, of course, not completely portable, but the encapsulation in a class a the " GrabTriggering " requirements and multithreading ones simplifies a lot the portability in other system POSIX compliant.

3 Communication by Com ports (CcommLib)

During the development of project we had to develop a class to handle communication with serial port. The driver, then, had been used to handle flash lamps and step motor. The driver encapsulates com-handling code and a simple FSM (for open, close and trigger events)

3.1 Implementation of COsiride

Osiride is the name of management board. Cosiride is the driver, implemented with a C++ class. Cosiride inherits from Cthread, so it can be run as a thread. Cosiride uses Commlib to communicate with the board.
A interesting method of Cosiride is Execute (the pure virtual method of Cthread is implemented by Cosiride to make the class concrete).

```
void COsiride::Execute()
{
  //open com and set some parameters.
   comm->open();
   comm->SetTimeout(500);
   comm->writeChar(actual_commands[g_cmd]) ;
   comm->writeChar(CR);

   BYTE val=comm->read();

//check what's the value osiride has sent back
   if (val==ACK_Y)
   {
     // OK: sw must sleep, wait for transmission to be completed and close port
      Sleep(total_time);
```

```
    comm->close();
    return      ;
  }
  else if (val==ACK_N)
  {
  // something went wrong: throw an excpetion
    Application->MessageBoxA("can't understand g command", "error",MB_OK);
  }
  else
    throw Application->MessageBoxA("trouble in comm", "error",MB_OK);
}
```

4 Data format

Importance of memorization of two-dimensional data, in our application, has two main effects:

1. the "*information quantity*" in a 2D-image is more than the quantity in 3D data extracted by the image. If one has the software it is possible to extract 3D shape in two seconds from the image, but the inverse is not possible.
2. Files containing images are of a size reduced of a factor of ten in respect of size with 3D shape.

Taking into account points 1. and 2., it is clear that it is very important to find a LOSSLESS data format for bidimensional data that supports a dynamic greater than 255 levels .

One important point, discovered during development, is to couple image files with hardware information of device with which image has been acquired.

4.1 JPEG 12 bit

To be compliant with specification of par. 4, initially we started to use 12 bit JPEG, a standard extension of IJG. 12 bit JPEG supports 4096 gray levels. Monochrome 12 bit jpeg is lossy, but its error seems (seemed) enough little for our application.

An advantage of 12 bit JPEG is its support for OOB markers: we have 7 markers of 64KB everyone. The use of marker is responsibility of programmers: standard viewers (as Paintshop, Coreldraw, and so on) do not process them.

So, we have encapsulated IJG libray (taken from www.jpeg.org) in a class, compiled them for 12 bit and markers support

After all this stuff, we've discovered that 12 bit JPEG support is not common: and its error is not so little, moreover the marker support is not suitable for our needings. At the end we decided to use a custom image format, really lossless, compressed and with arbitrary precision.

4.2 The ZBI format: Zipped Binary Image

After "12 bit JPEG experiment", we've defined a new lossless image format : the ZBI image format. ZBI, basically is a arbitrary precision (8, 12, 16 or more bits), with a very complex header with ALL hardware and software parameters indispensable for a perfect 3D reconstruction extracted from 2D data.. ZBI, after having been built as a bytes stream is compressed using a win32 implementation of zlib.

In this way ZBI results, in memory occupation terms, comparable to its correspondent JPEG without, however, all jpeg ties and drawbacks.

4.3 Memorization of 3D shape

To save tridimensioal data extracted from image, and make them visible to rhe rest of the world, we've implemented two filters for two standard formats :

1. VRML
2. STL

For both of them we've implemented an ecoder-decoder. Then, we've implemented a piece of code for translation between them and for extracting the "*points cloud*" from VRML and STL.