FrameWork for Software Production Line
FW4SPL

A. Charnoz, S. Nicolau

Ircad R&D Fr

RMLL July 2011
Outline

1. Introduction
2. fw4spl concepts
3. Framework functionalities
4. Conclusion
Presentation purpose

- Understand why we have developed our framework
- Show main functionalities available in this framework
Ircad context
Ircad R&D needs

- Quick development to create multi medical prototypes/softwares
  multi OS / multi arch
- Maximal source code re-using
- Trainees/phd (students) works integrated but fragmented
- Facilitate collaborations (source code available or not)
fw4spl characteristics

- Component design
- Object/services design
- Developed in C++
- Multi operating system/architecture compatible (osx,linux, windows)(32 and 64 bits)
- Depends on many open source libraries : boost, Qt, wxWidget, vtk, itk, gdc, libxml2, ...

Framework is an open source project (LGPL)

http://code.google.com/p/fw4spl/
fw4spl history

- 2004-2007 : fw4spl project
- 2007-2009 : VRRender 0.7 (free)
- 09/2009 : fw4spl became Open Source
- 2010 : PoC Sofa (Altran-Est), VRRender WLE 0.8.1 (free)
- 2011 : PoC Kinect (Altran-Est), VRRender 0.9 (open)
Outline

1. Introduction
2. fw4spl concepts
3. Framework functionalities
4. Conclusion
fw4spl concepts

- Object/Services approach
- Component-based approach
Limits of classic approach

Classic approach

An object (ex: Image) is represented by a class, and this class contains all functionalities working on the object (reading, writing, visualization, image analysis, ...).

Limits

- Too many functions in the same C++ class
- Many dependencies required (itk, vtk, qt, dcmtk, ...)
- Many developers work on the same file

Solution

Split your code in different files and libraries, split your functionalities.
Functions as services

UML Diagram with all services

Object

IService

Object * m_obj;
Configuration * m_cfg;

void setObject( Object * obj );
void setConfiguration( Config * cfg );

virtual void configure() = 0;
virtual void start() = 0;
virtual void udpate() = 0;
virtual void stop() = 0;

IRreader

IOperator

IVisu

Image

short * m_buffer;
unsigned int * m_size;

void setBuffer( short * buff );
short * getBuffer();
void setSize( unsigned int * size );
unsigned int * getSize();

DcmtkReaderSrv

void configure();
void start();
void udpate();
void stop();

ItkCropOperatorSrv

void configure();
void start();
void udpate();
void stop();

ClmgWindowOperatorSrv

void configure();
void start();
void udpate();
void stop();

VtkQtVisuSrv

void configure();
void start();
void udpate();
void stop();
Xml configuration file

Ok, it is a nice conception, but why factorising all functions?

```xml
<object type="::fwData::Image">
  <service uid="myFrame" impl="DefaultFrame" type="IFrame">
    <gui>
      <frame>
        <minSize width="800" height="600" />
      </frame>
    </gui>
    <registry>
      <view uid="myVisu" />
    </registry>
  </service>

  <service uid="myVisu" impl="vtkSimpleNegatoRenderer" type="IRender" />

  <service uid="myReader" impl="VtkImageReader" type="IReader">
    <filename path="./TutoData/patient1.vtk"/>
  </service>

  <start uid="myFrame" />
  <start uid="myVisu"/>
  <start uid="myReader"/>

  <update uid="myReader"/>
    <!-- Read the image on filesystem -->
  <update uid="myVisu"/>
    <!-- Refresh the visu -->
</object>
```
Representation of OSR/Xml Configuration

Diagram OSR (Object Service Registry) represents the application. This representation permits a better understanding of application design.
Communication in FW4SPL?
Objects receive message and notify all listeners.
Visualisation and OSR Diagram

OSR Diagram

Objects | Services
---|---
<Object> Composite

<Object> Image

<Object> Mesh

<Object> VtkMeshRenderer
uid = myVisu autoColorChannel = yes

<Object> VtkMeshRenderer
uid = myVisu autoColorChannel = yes

<Object> DefaultFrame
uid = myFrame size = 800x600 view = myView

<Object> DefaultView
uid = myView view1 = myVisu view2 = myVisu

<Object> MeshAction
uid = myMesh imaged ui = myImage meshuid = myMesh

<Object> XXXImageReader
uid = myImageSelector node = reader

<Object> XXXImageReader
uid = myImageSelector node = reader

<Object> IOSelector
uid = myImageSelector node = reader

<Object> IOSelector
uid = myImageSelector node = reader
fw4spl concepts

- Object/Services approach
- Component-based approach
What is the component approach?

Components must:

- Be used by different applications without re-compiling the program
- Be independent from another component (to be shared without dependency problem)
- Thus, use a defined protocol to be manipulated by an external program

Component in FW4SPL

Component = data/service package
How the system works

A Component is a folder that contains:

- Xml description file (`plugin.xml`) to describe the content of the dynamic library
- Dynamic libraries (.so, .dll, .dylib)
- Other shared resources (icons, xsd file, sounds, ...)

![Example of a Component folder with files](image.png)
component-based example

Available components

**Specific Components**
- VRRenderUI
- VtkIO
- VolumicVisu
- ComplexOperator
- ARSurgUI
- ItkIO
- ImageVisu
- BasicOperator
- VRAnatUI
- IrcadIO
- MeshVisu
- PipelineManager
- MagneticTracker

**Base Components**
- gui
- io
- visu
- operator
- tracker
- dataReg
- servicesReg

**Libraries**
- FW4SPL Core Libraries
- FW4SPL Specific Libraries
- Extern Libraries
component-based example

Available components

Specific Components
- VRNetUI
- VtkIO
- VolumicVisu
- ComplexOperator
- ARSurgUI
- ITKIO
- ImageVisu
- BasicOperator
- OpticalTracker
- VRenderUI
- IrcadIO
- MeshVisu
- PipelineManager
- MagneticTracker

Base Components
- gui
- io
- visu
- operator
- tracker
- dataReg
- servicesReg

Libraries
- FW4SPL Core Libraries
- FW4SPL Specific Libraries
- Extern Libraries

VR-Render
### Component-based Example

#### Available Components

<table>
<thead>
<tr>
<th>Specific Components</th>
<th>Base Components</th>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>VRAnetUI</td>
<td>gui</td>
<td>FW4SPL Core Libraries</td>
</tr>
<tr>
<td>VtkIO</td>
<td>io</td>
<td>FW4SPL Specific Libraries</td>
</tr>
<tr>
<td>VolumicVisu</td>
<td>visu</td>
<td>Extern Libraries</td>
</tr>
<tr>
<td>ComplexOperator</td>
<td>operator</td>
<td></td>
</tr>
<tr>
<td>ARSurgUI</td>
<td>tracker</td>
<td></td>
</tr>
<tr>
<td>ItkIO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ImageVisu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BasicOperator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpticalTracker</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VRendererUI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IrcadO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MeshVisu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PipelineManager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MagneticTracker</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dataReg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>servicesReg</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### AR-Surg

![Image of AR-Surg](image_url)
Available components

Specific Components
- VRAnatUI
- VtkIO
- VolumicVisu
- ComplexOperator
- ARSurgUI
- ItkIO
- ImageVisu
- BasicOperator
- VRRenderUI
- IrcadIO
- MeshVisu
- PipelineManager
- MagneticTracker

Base Components
- gui
- io
- visu
- operator
- tracker

Libraries
- FW4spl Core Libraries
- FW4spl Specific Libraries
- Extern Libraries

VR-Anat
Outline

1. Introduction
2. fw4spl concepts
3. Framework functionalities
4. Conclusion
### Framework functionalities

- fw4spl
- VR-Render
- ARL Core
fw4spl architecture

- Application (HMI include) configuration available in xml
- Event synchronization
- Data serialization in xml
- System/process memory management (data dumping)
- 3D generic scene in vtk
- Embedded python
fw4spl services

- Patient data structure
- Image and mesh visualization (2D/3D)
  - Window/level edition
  - Zoom/rotation/translation
  - Image/mesh annotations (landmarks/measures)
- Volume image rendering
- Image/mesh reader/writer: inr, vtk, dicom, 3ds, fw4spl
- Widgets to edit, manipulate, interact with data structure or visualization.
Outline

3 Framework functionalities

- fw4spl
- VR-Render
- ARL Core
VR-Render 0.9.0

- Readers and Negato 2D
- Volume image rendering
- 3D patient model
Outline

3 Framework functionalities
   - fw4spl
   - VR-Render
   - ARL Core
Open source
Augmented Reality Library
with FW4SPL

S. Nicolau, L. Goffin, V. Agnus, L. Soler
Augmented Reality

???
ARL Core & ARL Device

ARL Core
- data container
- algorithm

ARL Device
- image processing
- feature extraction
- data filtering
- object tracking
- data synchronization

Device 1
Device 2
Device 3

ARL Visu
- 3D world visualization
- image display
- image fusion (transparency, colour table)
ARL Core & ARL Device

ARL Core
-data container
-algorithm

ARL Device
-image processing
-feature extraction
-data filtering
-object tracking
-data synchronization

Device 1

Device 2

Device 3

ARL Visu
-3D world visualization
-image display
-image fusion (transparency, colour table)

FW4SPL
-fwCore
-fwTools
Purpose:

Management of useful object for AR apps

Contains Algorithms for AR application

Nice and versatile interface

No device management, only data processing
ARL Core: properties

- C++ library
- Linux + Windows + Mac
- Dependencies: VXL, ANN (LGPL)
- Visualization: GNUplot + VTK
ARLCore: basic objects (I)

Camera with distortion (Zhang definition)
- projection
- undistortion
- epipolar lines

Rigid transformation (rotation + translation)
- conversion (matrix/vector/quaternion)
- quick inversion
- covariance
- geodesic distance
- random matrix
ARLCore : basic objects (II)

Point
- multidimensional
- covariance
- visibility

PointList
- random shape
- line, plane, sphere fitting
- closest point research method (ICP)
ARLCore: algorithms (I)

- Camera calibration (from Pointlist)
  - intrinsic parameters with distortion
  - extrinsic parameters

- Multiple acquisition of 2D (or 3D) object

- ML Optimization of 2D reprojection error:

$$\sum_{l=1}^{M} \sum_{i=1}^{N} \| I \ast (\text{Dist}(\text{Proj}(T_l \ast M_i))) - m_i \|^2$$
ARLCore: algorithms (I)

- OpenCV already does it but...
- Initialization may fail without informing
- No choice of optimization methods and parameters
  - LM + Powell + Conjugate Gradient
  - Optimization parameters can be chosen
- All points must be visible in all images
- Tracing error is very difficult due to interface
ARLCore: algorithms (I)

- Multi-Camera calibration
  - assume intrinsic of cameras available
  - optimal extrinsic calibration

- Multiple acquisition of 2D (or 3D) object

ML Optimization of 2D reprojection error:

$$\sum_{l=1}^{M} \sum_{i=1}^{N} \| I \star (\text{Dist}(\text{Pr proj}(T_l * M_i))) - m_{i,l}^{1,l} \|^2 + \sum_{l=1}^{M} \sum_{i=1}^{N} \| I \star (\text{Dist}(\text{Pr proj}(T_{ext} * T_l * M_i))) - m_{2,i}^{l} \|^2$$
ARLCore: algorithms (I)

- Tool calibration
  - centre estimation (several methods)
  - orientation estimation
ARLCore: algorithms (II)

Multi-Camera triangulation
- multi-lines
- reprojection optimization (uncertainty)

3D/3D Point Registration
analytic 3D/3D point registration

$$\sum_{i}^{N} \| T \ast M_i - N_i \|^2$$

3D/3D registration with uncertainty

$$\sum_{i}^{N} (T \ast M_i - N_i) \ast \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{bmatrix}^{-1} \ast (T \ast M_i - N_i)^T$$
ARLCore : algorithms (III)

3D/2D Point registration
  - Homography
  - ISPPC
  - OSPPC
  - EPPC

Optimization method : LM, CG & Powell

Error tracing: RMS available for each point during optimization
Automatic registration of 3D model on real views
ARLCore: Conclusion

- Camera calibration
- Instrument calibration
- 3D/3D Point registration
- 3D/2D Point registration
- Open Source:
  - http://code.google.com/p/fw4spl/
Thanks for your attention
Outline

1. Introduction
2. fw4spl concepts
3. Framework functionalities
4. Conclusion
fw4spl planning

Work in progress

- Store and retrieve medical images on PACS
- Video: readers, 2D visualization, 3D visualization with patient model
- Operator process pipeline for patient modeling
  - itk operators
  - ircad operators: C++ classic, OpenCL, python.
- Volume image rendering: all vtk render with parameters, transfer function editor.
- Generic 2D scene in Qt
- Work around DICOM format

Future work

- Multithreaded applications (augmented reality, simulators, robotics)
- fw4spl on tablet/smartphone (android, ios, maemo/meego)
Ircadb: medical data base

- Patient models (10 cases)
- Livers with hepatic tumors (20 cases)
- Breathing image (1 case)
- In progress
  - parathyroid(20), liver(30), ...
Positioning

Specific framework in medical image analysis
ITK, VTK, GDCM, IGSTK, OpenCV, VxL, Sofa, ...

Specific application in medical image analysis
Osirix, Slicer3d, Mevis, MedInria, InVesalius, Devide, Ginkgo, ...

Framework oriented application
XIP Builder, MAF, MITK, GIMIAS
Thanks for your attention!

FW4SPL Open Source

- source code: code.google.com/p/fw4spl
- discuss: groups.google.com/group/fw4spl-discuss
- chat: IRC server: irc.freenode.net, channel: #fw4spl

Softwares

- VR-Render WLE 0.8.1: www.websurg.com/softwares/vr-render
- VR-Render 0.9: code.google.com/p/fw4spl/downloads

Data base

- liver cancer: www.ircad.fr/softwares/3Dircadb/3Dircadb1
- breathing: www.ircad.fr/softwares/3Dircadb/3Dircadb2