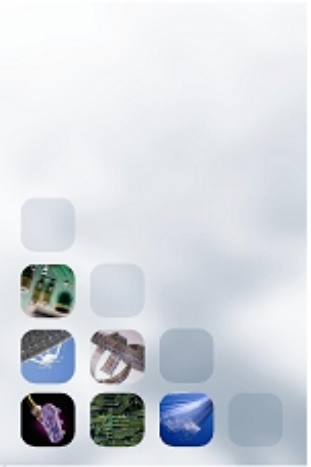
GNSS-SDR
Software Architecture

Carles Fernández Prades
Head of Communication Systems Division and Senior Researcher
Javier Arribas Lázaro
Senior Researcher
Pau Closas
Head of the SI Dept. and Senior Researcher

Communication Systems Division
Statistical Inference for Communications and Positioning Dept.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya



GNSS SDR

Overview

- Introduction
- The Control Plane
- The Signal Processing Plane

CTTC

Centre
Tecnològic
de Telecomunicacions
de Catalunya

Introduction

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Introducing GNSS-SDR

- The acronym stands for '**Global Navigation Satellite System –Software Defined Receiver**'.
- It is a tool for researchers, specially focused in signal processing.
- Flexible, fully configurable, easily extendible.
- Testbed for GNSS synchronization algorithms.
- Programmed in C++. Portable (currently builds in Linux and Mac OS X operating systems. The execution in embedded processors is ongoing work).
- **Open Source**: free as in *free beer and as in free speech.* Released under GPLv3.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Design principles

- Do not reinvent the wheel.
- Follow design patterns.
- Follow a clean coding style.
- Follow standards.
- Try to produce a *well-written* software.
- Test-driven development approach.

Do not reinvent the wheel

GNSS SDR

- On the shoulder of giants:

GNU Radio is a software development toolkit that provides a framework to implement software radios. GPL v3.

FFTW is a C subroutine library for computing the discrete Fourier transform. GPL v2.

Boost is one of the most highly regarded and expertly designed set of libraries for the C++ programming language. Boost Software License.

Google C++ Testing Framework provides tools for writing C++ tests. BSD 3-Clause License.

Armadillo is a C++ linear algebra library aiming towards a good balance between speed and ease of use. Mozilla Public License Version 2.0.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Design patterns

- Software design patterns are **descriptions of solutions to common software problems** arising in different contexts, capturing recurring structures and dynamics among software participants to facilitate reuse of successful, thoughtfully proven designs.
- They generally **codify expert knowledge of design strategies**, constraints and best practices. Following a pattern helps to resolve key design forces such as flexibility, extensibility, dependability, predictability, scalability, and efficiency.
- **They are not code recipes but generalized solutions** to commonly occurring problems.

C. Fernández-Prades, C. Avilés, L. Esteve, J. Arribas, and P. Closas, "**Design patterns for GNSS software receivers**," in Proc. of the **5th ESA Workshop on Satellite Navigation Technologies** (NAVITEC'2010), ESTEC, Noordwijk, The Netherlands, Dec. 2010.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

Coding style

- Following programming guidelines and code conventions not only helps to avoid introducing errors, but cuts maintenance costs and favors effective code reuse.
- The following rules capture the most important aspects of coding style:
  - All should be as **understandable** as possible.
  - All should be as **readable** as possible, except when it would conflict with the previous rule.
  - All should be as **simple** as possible, except when it would conflict with the previous rules.
- Any violation to the guide is allowed if it enhances readability.

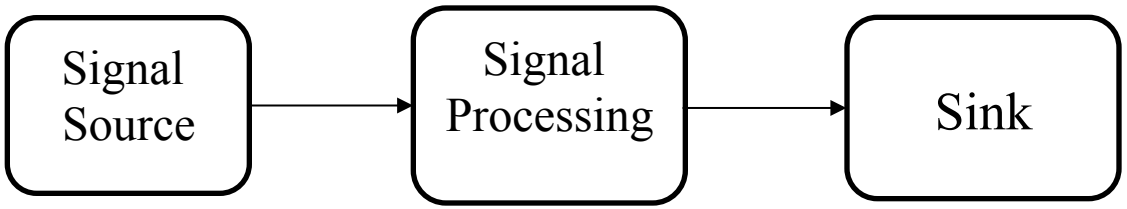
http://gnss-sdr.org/documentation/coding-style

GNSS SDR

Architecture

GNSS-SDR is structured in two layers:

- Control Plane
  - creates a flow graph according to the configuration, and then
  - manages the Signal Processing blocks.
- Signal Processing Plane
  - performs the actual digital signal processing.
  - Signal Processing blocks are the nodes of the flow graph.



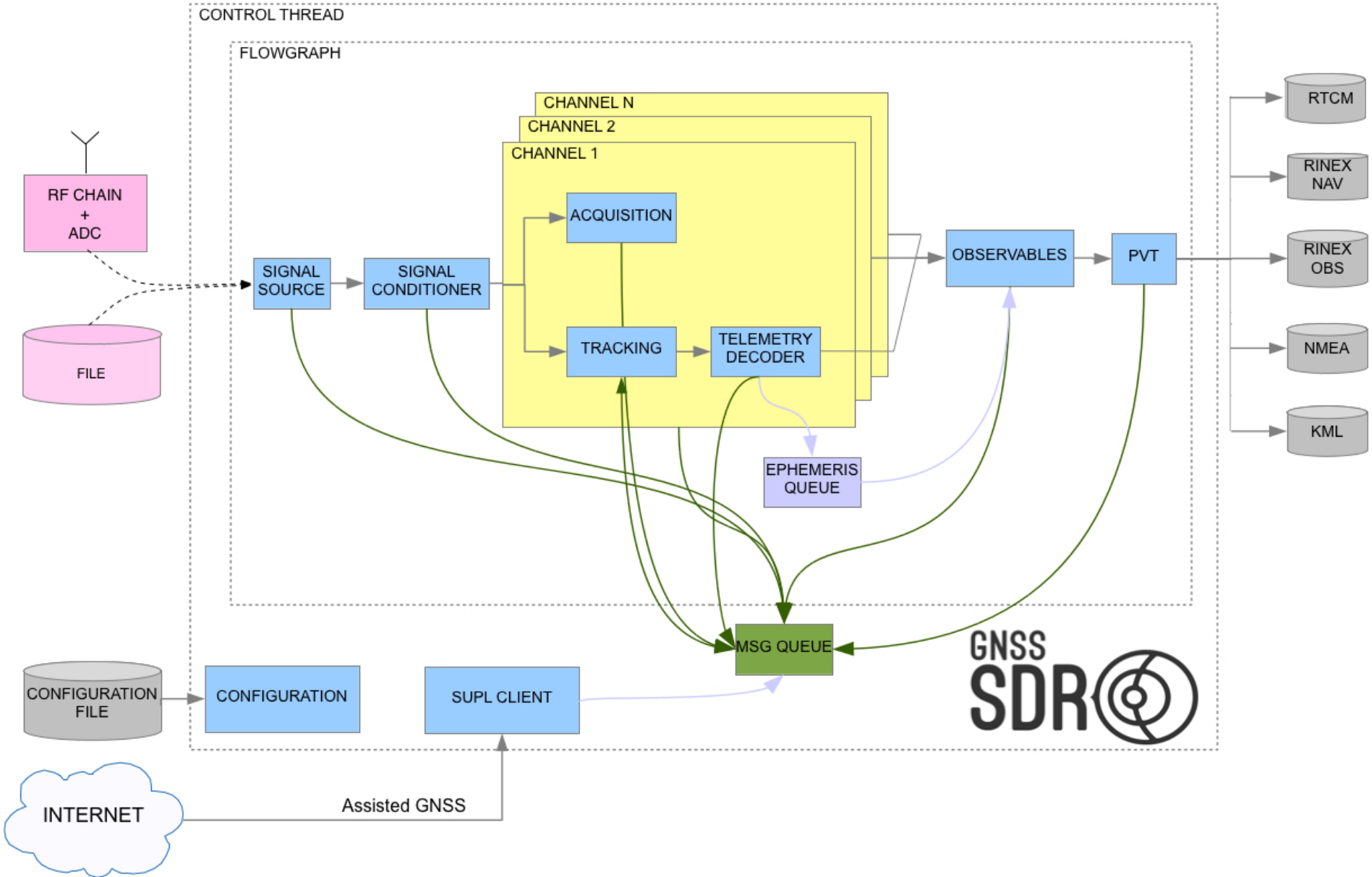
Example of a simple flow graph

C. Fernández-Prades, J. Arribas, P. Closas, C. Avilés, and L. Esteve, "**GNSS-SDR: an open source tool for researchers and developers**", in Proc. of the **ION GNSS 2011 Conference**, Portland, Oregon, Sept. 19-23, 2011.

CTTC
Centre
Tecnològic
de Telecomunicacions
de Catalunya

The Control Plane

GNSS-SDR block diagram



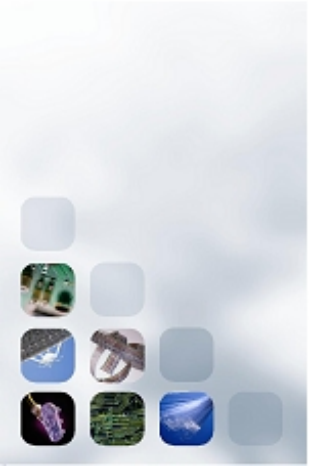
CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Source tree organization

|-gnss-sdr
|---build <- where gnss-sdr is built
|---cmake <- CMake-related files
|---conf <- Configuration files. Each file represents one receiver.
|---data <- Populate this folder with your captured data.
|---docs <- Contains documentation-related files
|---drivers <- Drivers for some RF front-ends
|---firmware <- Firmware for some front-ends
|---install <- Executables
|---src <- Source code folder
|-----algorithms
|-----core
|-----main
|-----tests
|-----utils <- some utilities (e.g. Matlab scripts)

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya



GNSS SDR

Source tree organization

|-gnss-sdr
|---src <- Source code folder
|-----algorithms
|-----core
|-----main <- Everything starts at main.cc
|-----tests
|-----utils

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

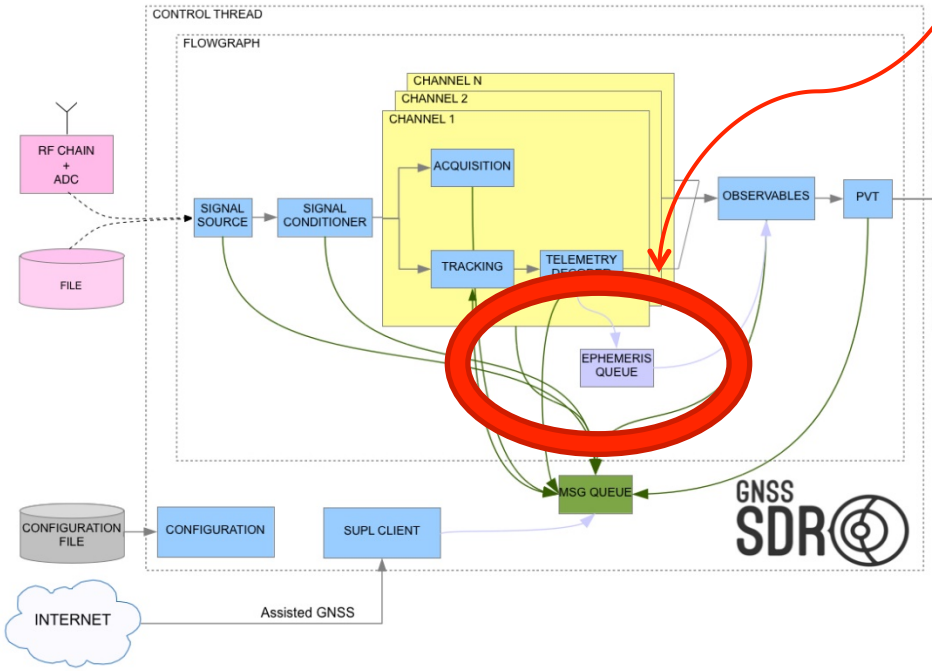
GNSS SDR

main.cc

```
concurrent_queue<gps_navigation_message> global_gps_nav_msg_queue;
int main(int argc, char**argv)
{
   // Initialize logging library
   google::InitGoogleLogging(argv[0]);
   // Process the command line flags
   google::ParseCommandLineFlags(&argc, &argv, true);
   // Instantiate a Smart Pointer to a new ControlThread object
   std::unique_ptr<ControlThread> control_thread(new ControlThread());
            // The constructor reads the configuration
            // Creates a control queue
            // Creates a flowgraph according to the configuration
   control_thread->run();
            // Connects the flowgraph
            // Start running the flowgraph
            // While ( running && !stop) {
                     // read control messages
                     // process control messages }
   }
```

Ephemeris Queue

```
concurrent_queue<gps_navigation_message> global_gps_nav_msg_queue;
int main(int argc, char**argv)
{
```



This is a thread-safe queue that communicates the navigation data block to the module where observables are generated.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Logging

```
concurrent_queue<gps_navigation_message> global_gps_nav_msg_queue;
int main(int argc, char**argv)
{
   // Initialize logging library
   google::InitGoogleLogging(argv[0]);
   // Process the command line flags
   google::ParseCommandLineFlags(&argc, &argv, true);
   // Instantiate a Smart Pointer to a new ControlThread object
   std::unique_ptr<ControlThread> control_thread(new ControlThread());
            // The constructor reads the configuration
            // Creates a control queue
            // Creates a flowgraph according to the configuration
   control_thread->run();
            // Connects the flowgraph
…
```

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Logging

- Logging is important. We need it because software (no matter how experienced you are) is notoriously unpredictable.
- For debugging/tracing purposes.
- You might also want to run some interesting statistics on your logs, and study usage patterns.
- In GNSS-SDR, all those issues are handled by **google-glog**, a library that implements application-level logging. It provides simple yet powerful APIs to various log events in your program. You can:
  - log messages by severity level,
  - control logging behavior from the command line,
  - log based on conditionals,
  - abort the program with stacktrace when expected conditions are not met,
  - introduce your own verbose logging levels, and more.
- https://github.com/google/glog

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Command line flags

```
concurrent_queue<gps_navigation_message> global_gps_nav_msg_queue;
int main(int argc, char**argv)
{
    // Initialize logging library
    google::InitGoogleLogging(argv[0]);
    // Process the command line flags
    google::ParseCommandLineFlags(&argc, &argv, true);
    // Instantiate a Smart Pointer to a new ControlThread object
    std::unique_ptr<ControlThread> control_thread(new ControlThread());
            // The constructor reads the configuration
            // Creates a control queue
            // Creates a flowgraph according to the configuration
    control_thread->run();
            // Connects the flowgraph
…
```

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Command line flags processing

- Command line flags are flags that users specify on the command line when they run an executable.
- Managed by **gflags**, a library that implements commandline flags processing. As such it is a replacement for `getopt()`. It has increased flexibility, including built-in support for C++ types like string, and the ability to define flags in the source file in which they are used.
- https://github.com/gflags/gflags
- The gflags library differs from other libraries, such as getopt, in that flag definitions can be scattered around the source code, and not just listed in one place such as main(). In practice, this means that a single source-code file will define and use flags that are meaningful to that file. Any application that links in that file will get the flags, and the gflags library will automatically handle that flag appropriately.
- There is significant gain in flexibility, and ease of code reuse, due to this technique. However, there is a danger that two files will define the same flag, and then give an error when they are linked together.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Control Thread

```
concurrent_queue<gps_navigation_message> global_gps_nav_msg_queue;
int main(int argc, char**argv)
{
    // Initialize logging library
    google::InitGoogleLogging(argv[0]);
    // Process the command line flags
    google::ParseCommandLineFlags(&argc, &argv, true);
    // Instantiate a Smart Pointer to a new ControlThread object
    std::unique_ptr<ControlThread> control_thread(new ControlThread());
            // The constructor reads the configuration
            // Creates a control queue
            // Creates a flowgraph according to the configuration
    control_thread->run();
            // Connects the flowgraph
             // Start running the flowgraph
            // While ( running && !stop) {
                        // read control messages
                        // process control messages }
}
```

Control Thread

- As shown in main.cc, after initializing the logging library and process the command line flags, the main application's thread starts: **ControlThread**
- Its first task is to read the configuration.
- Then, creates a control queue and the flow graph defined by the configuration.

Source tree organization

|-gnss-sdr
|---src
|-----algorithms
|-----core
|--------interfaces
|--------libs
|--------**receiver <- ControlThread and Flowgraph implementations here**
|--------system_parameters
|-----main
|-----tests
|-----utils

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Notation

- We use a very simplified version of the Unified Modeling Language (UML), a standardized general-purpose modeling language in the field of object-oriented software engineering.
- This document shows some class diagrams used in GNSS-SDR.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Notation

In this document, classes are described as rectangles with two sections: the top section for the name of the class, and the bottom section for the methods of the class.

Name_Of_The_Class
typeA method1() typeB method2(int a, int b) ….

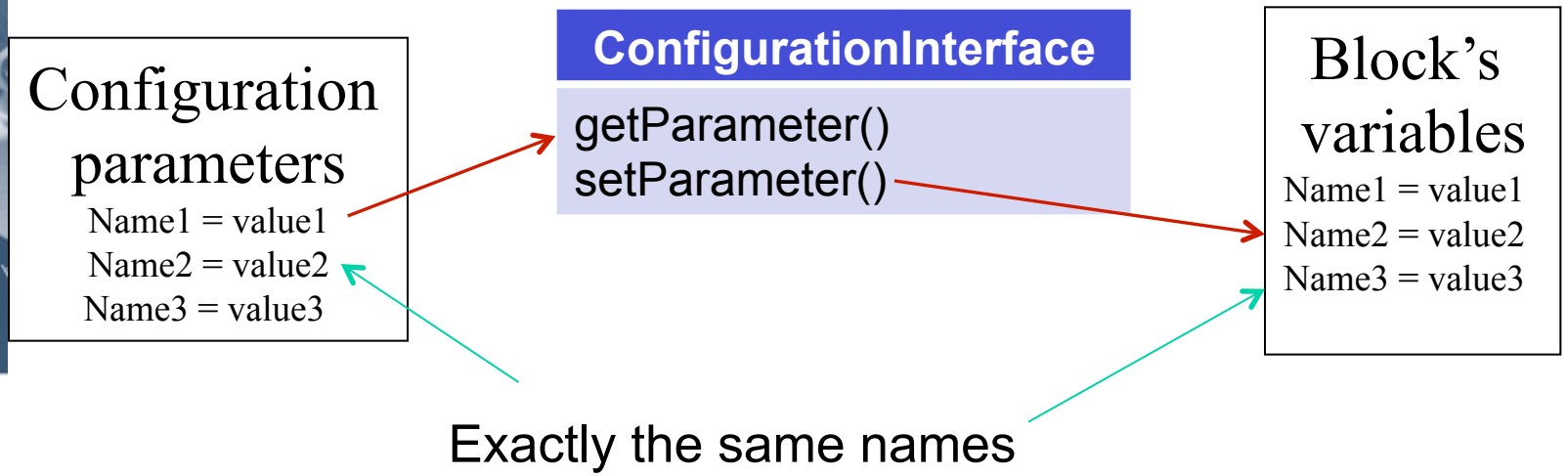
Control Thread

ControlThread

```
Public:
 ControlThread();
 ControlThread(std::shared_ptr<ConfigurationInterface> configuration);
 virtual ~ControlThread();
 void run();
 void set_control_queue(boost::shared_ptr<gr::msg_queue> control_queue);
 void set_control_queue(gr_msg_queue_sptr control_queue);
 unsigned int processed_control_messages()
 unsigned int applied_actions()
 std::shared_ptr<GNSSFlowgraph> flowgraph()
 …

Private:
 void init();
 void read_control_messages();
 void process_control_messages();
 void apply_action(unsigned int what);
 …
```

Configuration Interface



Configuration allows users to define in an easy way their own custom receiver by specifying the flow graph (type of signal source, number of channels, algorithms to be used for each channel and each module, strategies for satellite selection, type of output format, etc.)

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Flow graph

GNSSFlowgraph

Public:

```
GNSSFlowgraph(ConfigurationInterface* configuration, gr_msg_queue_sptr queue);
virtual ~GNSSFlowgraph();
void start();
void stop();
void connect();
void wait();
void apply_action(unsigned int who, unsigned int what);
std::shared_ptr<ConfigurationInterface> configuration_;
std::shared_ptr<GNSSBlockFactory> block_factory_;
std::shared_ptr<GNSSBlockInterface> sig_source_;
std::shared_ptr<GNSSBlockInterface> sig_conditioner_;
std::shared_ptr<GNSSBlockInterface> observables_;
std::shared_ptr<GNSSBlockInterface> pvt_;
std::shared_ptr<GNSSBlockInterface> output_filter_;
bool connected()
bool running()
```

Private:

```
void init();
void apply_action(unsigned int what);
void set_satellites();
```

This method connects all the blocks defined in the flowgraph

These blocks may have several implementations, but their interfaces always meet the minimal GNSSBlockInterface

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Flow graph

- Connections within modules are always the same, no matter the concrete instantiation of each module.
- Example:
  - Use implementation *DirectResampler* for block *Resampler*. In the configuration file:

```
Resampler.implementation=DirectResampler;
```

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Flow graph

ConfigurationInterface

getParameter()
setParameter()

Add here new modules

Factory

getResampler()

BlockInstance

(Resampler implemented by DirectResampler)

A Factory encapsulates the complexity of the instantiation of processing blocks.

Source tree organization

|-gnss-sdr
|---src
|-----algorithms
|-----core
|--------interfaces
|--------libs
|--------**receiver <- GNSSBlockFactory implementation here**
|--------system_parameters
|-----main
|-----tests
|-----utils

CTTC

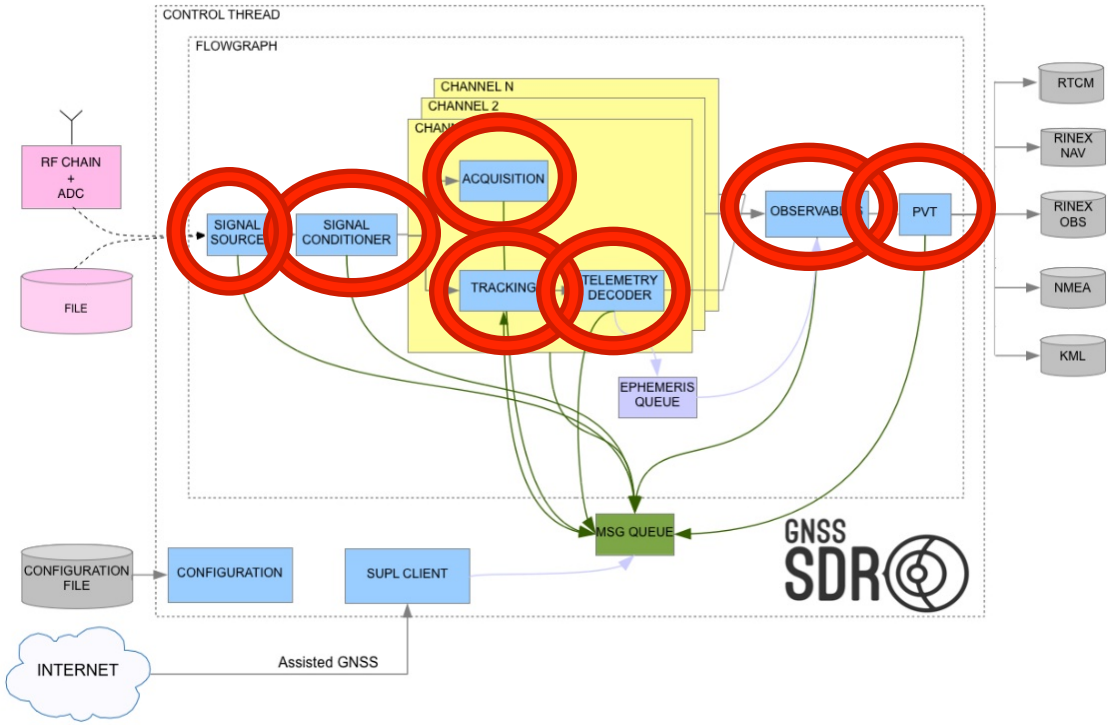
Centre
Tecnològic
de Telecomunicacions
de Catalunya

The Signal Processing Plane

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

Interchangeable implementations

- All the processing blocks (red circles in the diagram) can have different implementations.
- The concrete implementation used in the receiver is defined in the configuration file.
- All the implementations must meet (at least) the corresponding minimal interface.



CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

Signal Processing in Software Radio

GNSS SDR

- Software radio turns radio hardware problems into software problems, in contrast to the traditional approach in which the processing is done with either analog circuitry or analog circuitry combined with digital chips.
- GNU Radio is a free software toolkit for building software radios.

GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

http://gnuradio.org/redmine/wiki/gnuradio

- The programmer builds a radio by creating a graph where the nodes are signal processing blocks and the lines represent the data flow between them.
- The signal processing blocks are implemented in C++. Conceptually, blocks process infinite streams of data flowing from their input ports to their output ports. Blocks' attributes include the number of input and output ports they have as well as the type of data that flows through each.

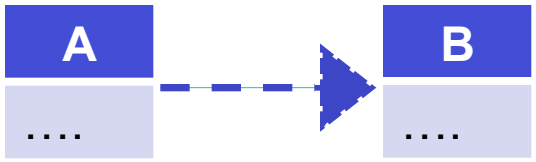
CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

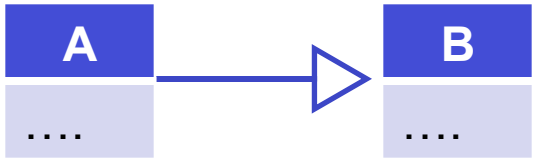
Nice features

- Concurrency
- Efficiency
- Performance
- Portability
- Ability to run in real-time or in post-processing
- Extendibility
- GPL

Notation



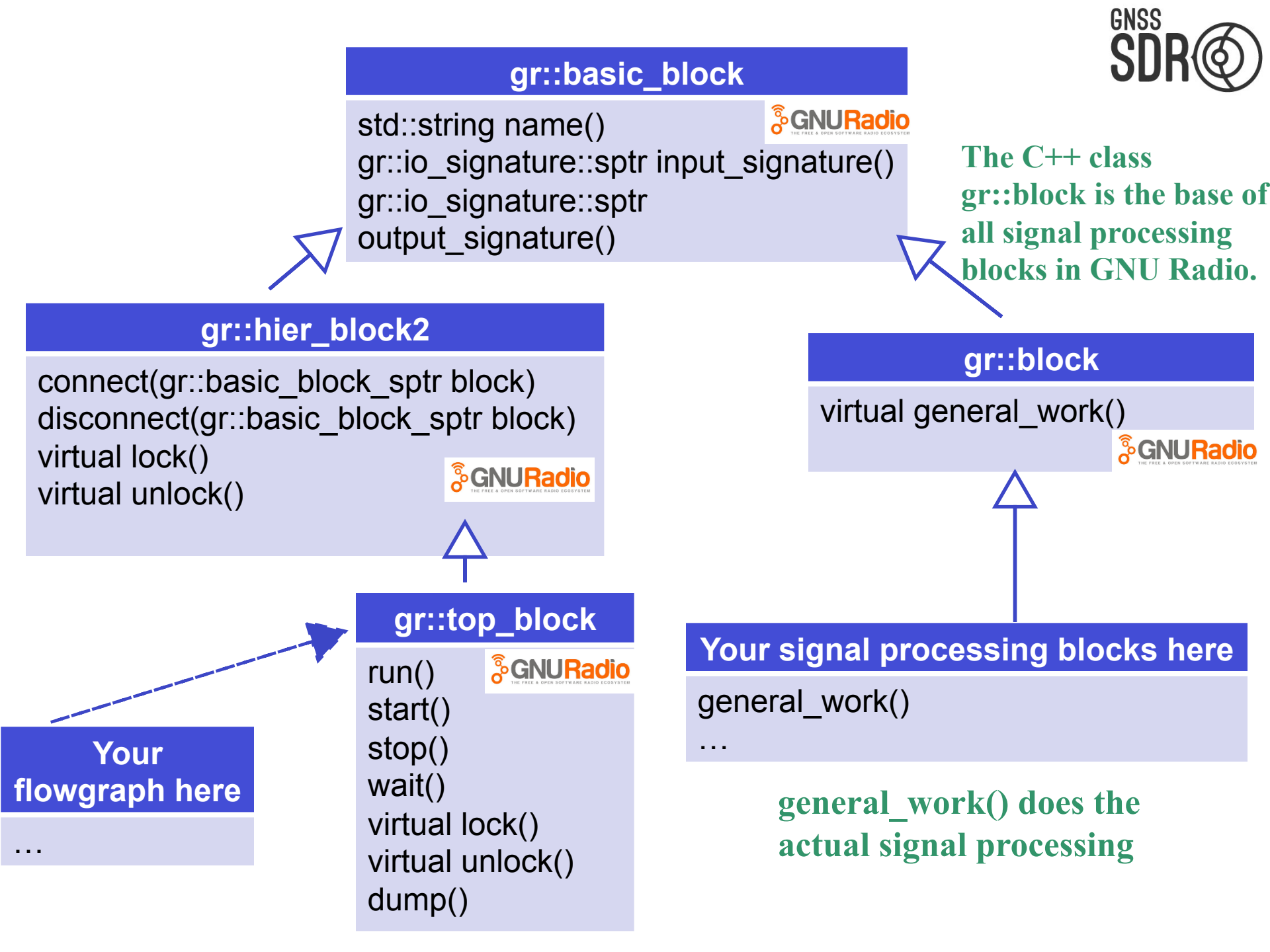
A dashed arrow is the **dependency** relationship. This relationship simply means that class A somehow depends upon class B. In C++ this almost always results in a #include.



Inheritance models “is a” and “is like” relationships, enabling you to reuse existing data and code easily. When A inherits from B, we say A is the subclass of B and B is the superclass of A. The UML modeling notation for inheritance is a line with a closed arrowhead pointing from the subclass to the superclass.

GNU Radio class hierarchy

- Next slide shows the relationship of the principal classes provided by GNU Radio
- Although GNU Radio blocks can be connected together using Python, we have opted to implement the whole receiver in C++.
- This class hierarchy imposes a thread-per-block architecture that allows automatic scheduling in multicore processors.


GNSS SDR
gr::basic_block
std::string name()
gr::io_signature::sptr input_signature()
gr::io_signature::sptr output_signature()
GNURadio
The C++ class gr::block is the base of all signal processing blocks in GNU Radio.
gr::hier_block2
connect(gr::basic_block_sptr block)
disconnect(gr::basic_block_sptr block)
virtual lock()
virtual unlock()
GNURadio
gr::block
virtual general_work()
GNURadio
gr::top_block
run()
start()
stop()
wait()
virtual lock()
virtual unlock()
dump()
GNURadio
Your signal processing blocks here
general_work()
…
Your flowgraph here
…
general_work() does the actual signal processing

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Principles of reusable object-oriented design

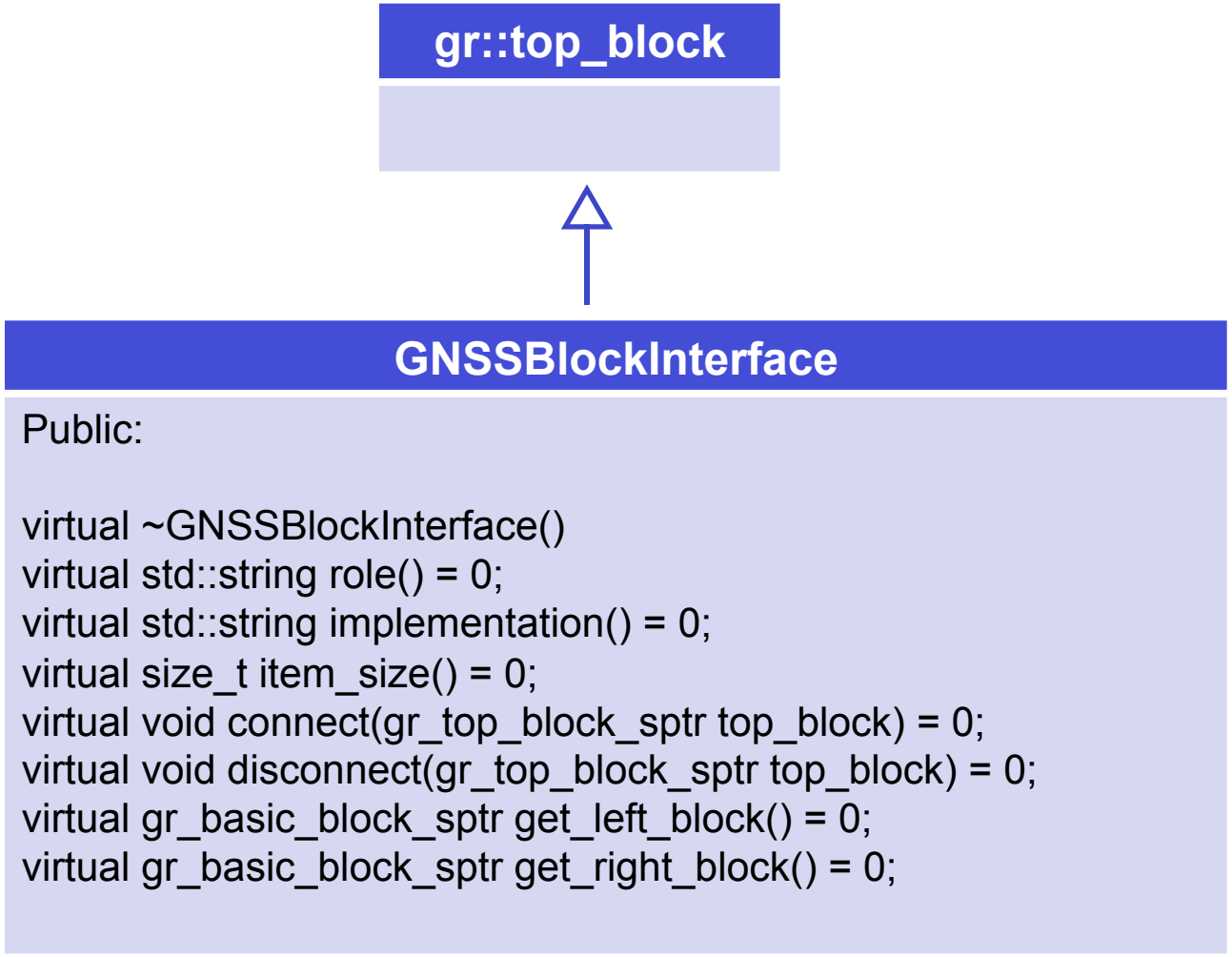
- The first principle is: "**Program to an interface, not to an implementation**."

This principle is really about dependency relationships which have to be carefully managed in a large app.

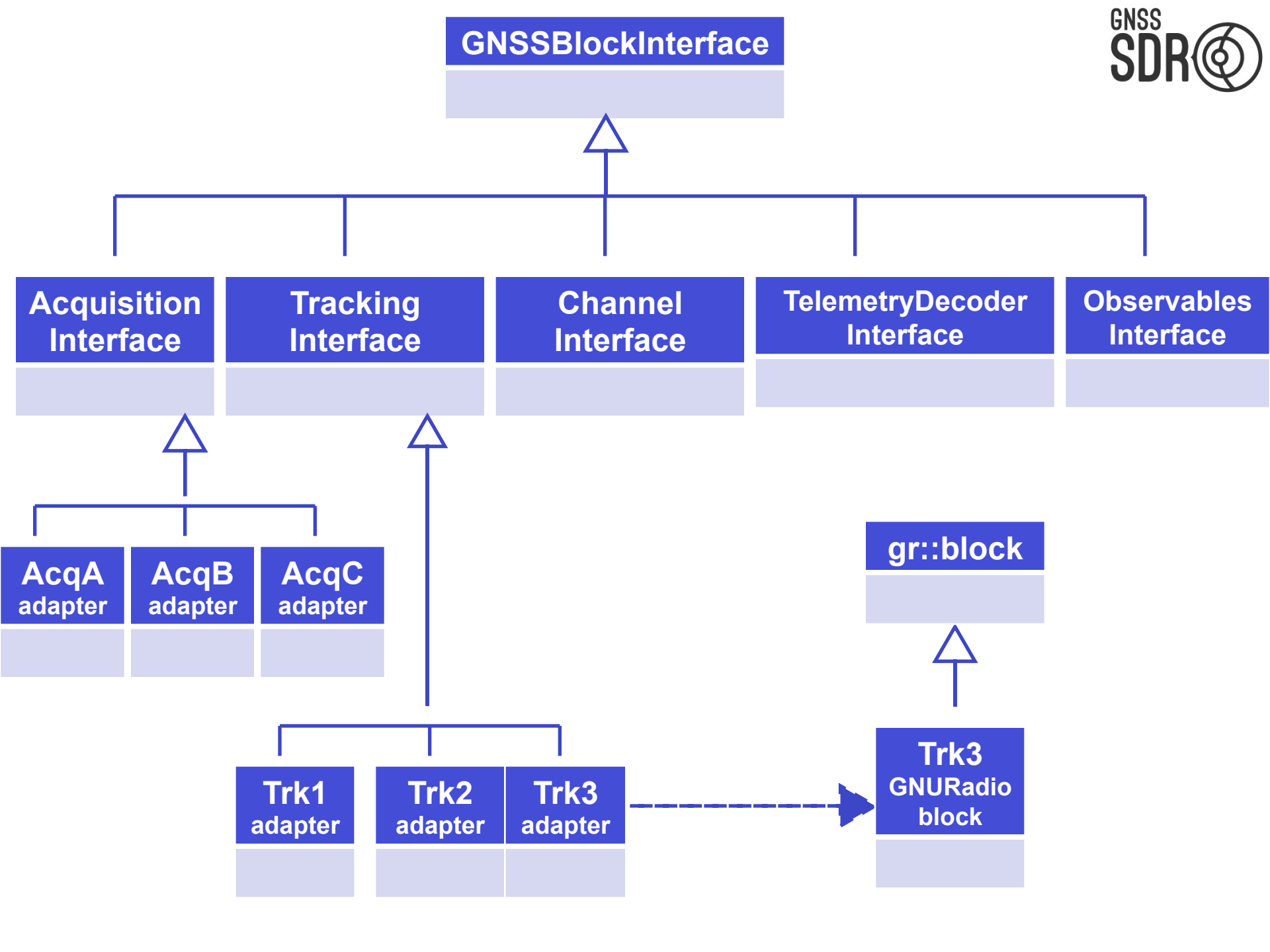
It is easy to add a dependency on a class. It is almost too easy; just add an #include statement. However, the inverse is not that easy and getting rid of an unwanted dependency can be real refactoring work or even worse, block you from reusing the code in another context.

Once you depend on interfaces only, you are decoupled from the implementation. That means the implementation can vary, and that's a healthy dependency relationship. This approach gives you flexibility, but it also separates the really valuable part, the design, from the implementation, which allows clients to be decoupled from the implementation. In fact, an abstract class gives you more flexibility when it comes to evolution. You can add new behavior without breaking clients.

A general interface for processing blocks



A **pure virtual method** is required to be implemented by a derived class. Classes containing pure virtual methods are termed "abstract;" they cannot be instantiated directly, and a subclass of an abstract class can only be instantiated directly if all inherited pure virtual methods have been implemented by that class or a parent class.

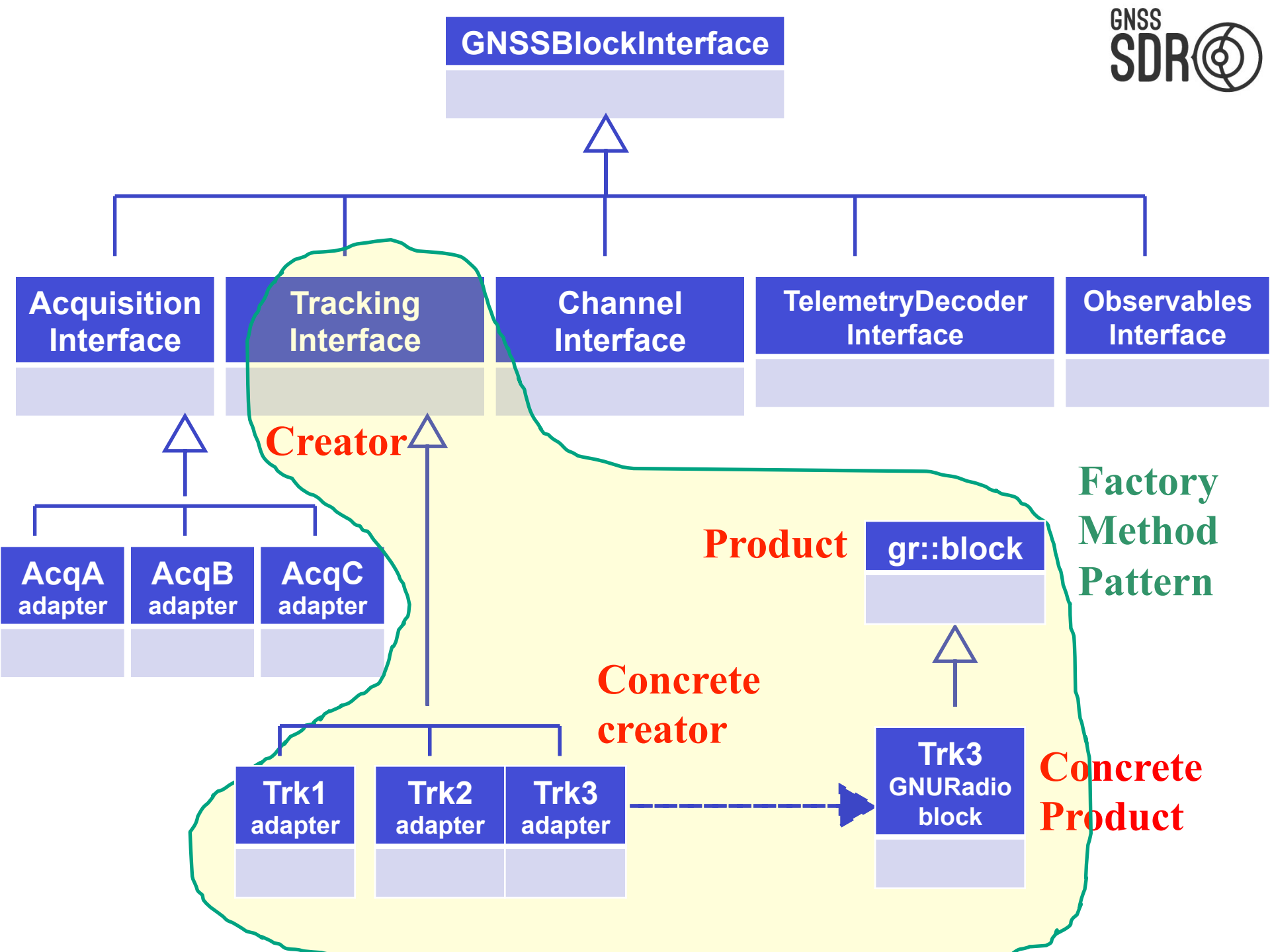

GNSSBlockInterface
Acquisition Interface
Tracking Interface
Channel Interface
TelemetryDecoder Interface
Observables Interface
AcqA adapter
AcqB adapter
AcqC adapter
Trk1 adapter
Trk2 adapter
Trk3 adapter
gr::block
Trk3 GNURadio block

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

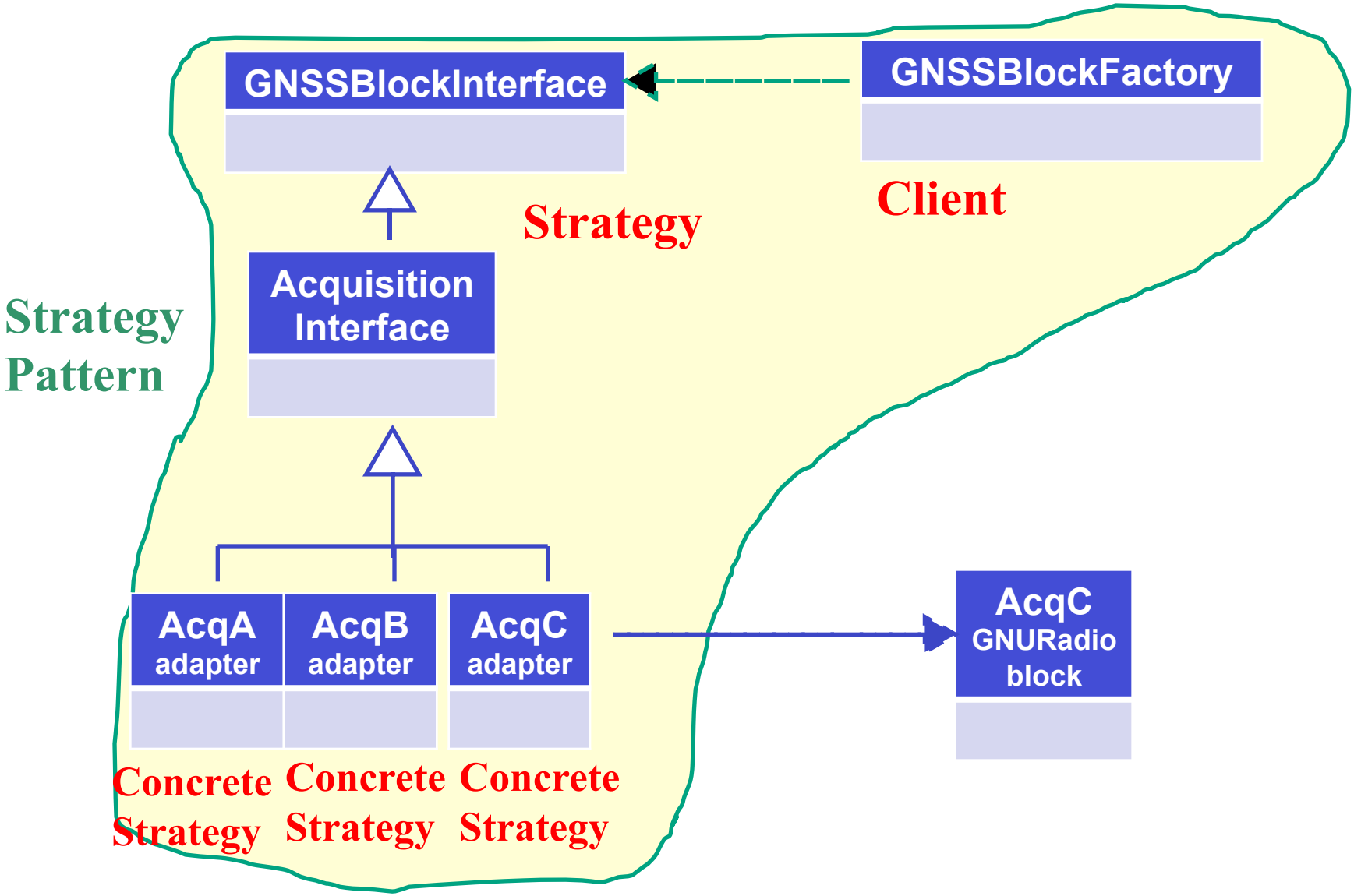
GNSS SDR

Source code organization

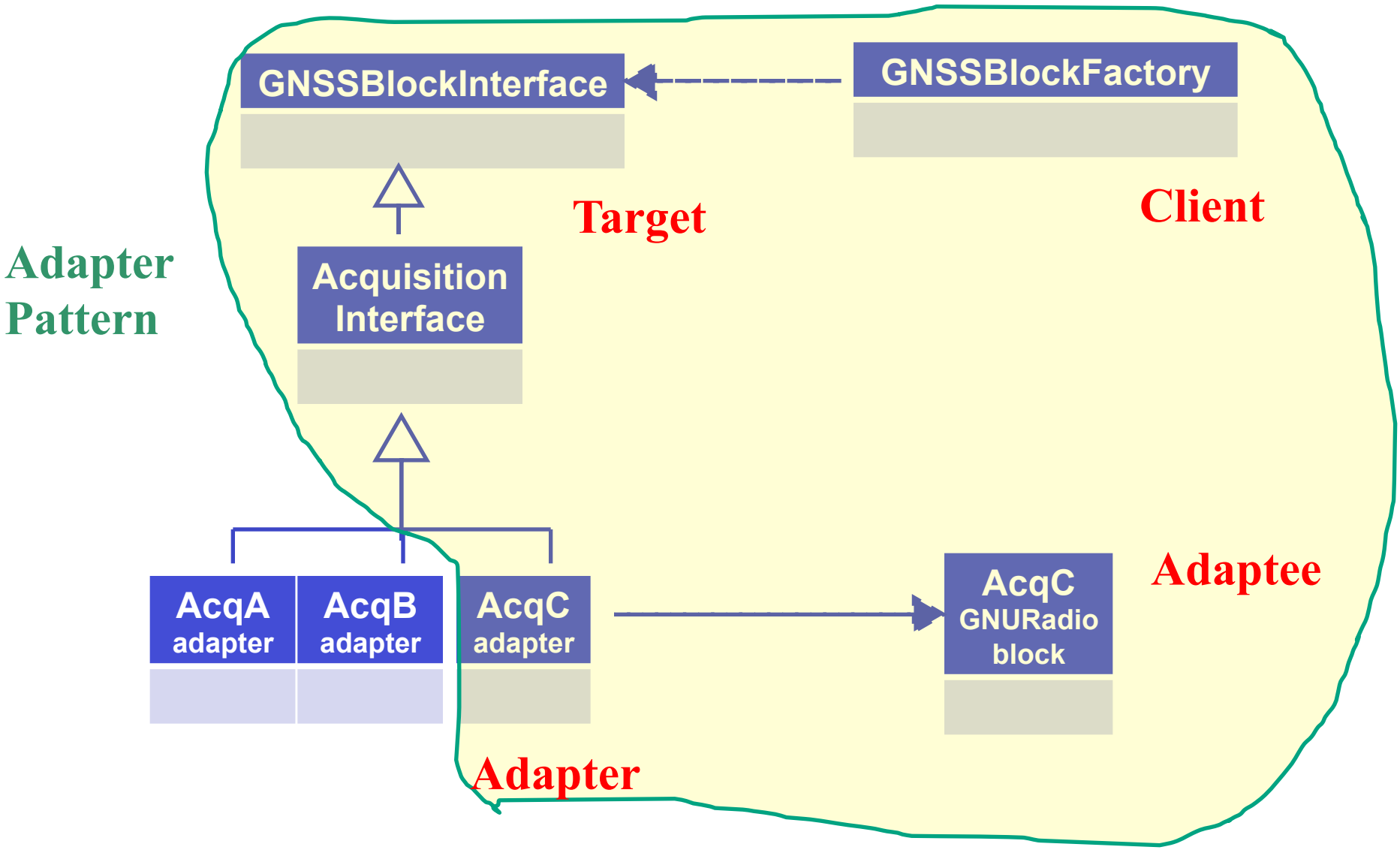
|-gnss-sdr
|---src
|-----algorithms
|-----core
|--------interfaces
|--------libs
|--------receiver
|--------system_parameters
|-----main
|-----tests
|-----utils


GNSSBlockInterface
Acquisition Interface
Tracking Interface
Channel Interface
TelemetryDecoder Interface
Observables Interface
Creator
AcqA adapter
AcqB adapter
AcqC adapter
Product
gr::block
Factory Method Pattern
Concrete creator
Trk1 adapter
Trk2 adapter
Trk3 adapter
Trk3 GNURadio block
Concrete Product

Different implementations, same interface



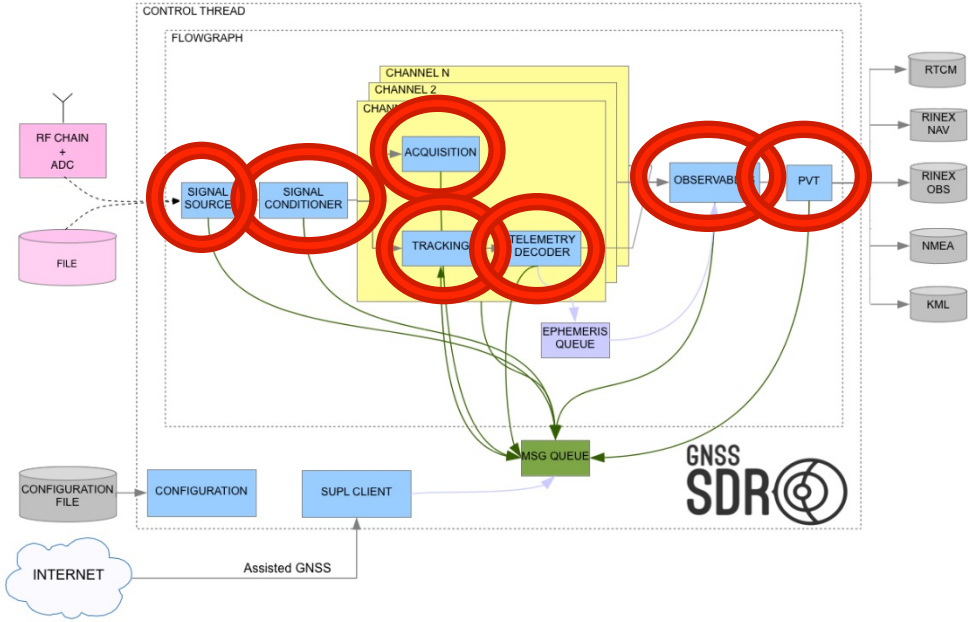
Reuse of existing GNURadio blocks



CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

Source tree organization

|-gnss-sdr
|---src
|-----algorithms
|--------signal_source
|--------conditioner
|--------data_type_adapter
|--------input_filter
|--------resampler
|--------channel
|--------acquisition
|-----------adapters
|-----------gnuradio_blocks
|--------tracking
|-----------adapters
|-----------gnuradio_blocks
...



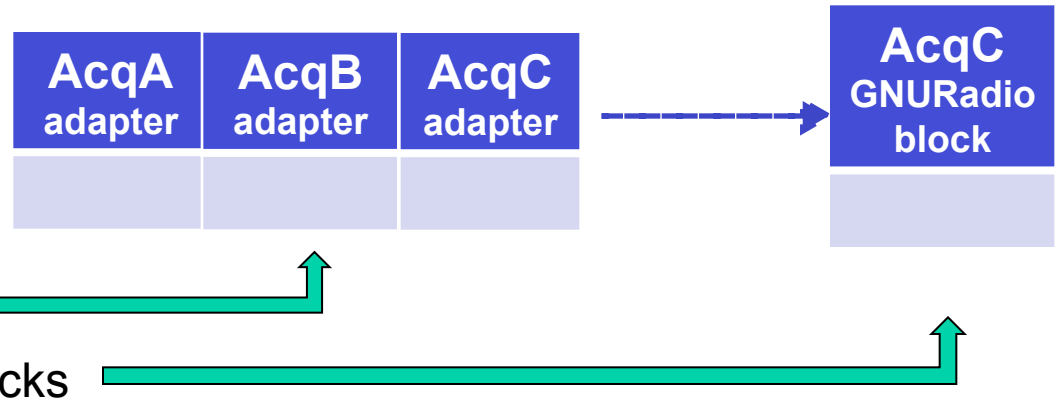
The processing blocks of the general diagram are implemented at gnss-sdr/src/algorithms

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Source tree organization

|-gnss-sdr
|---src
|-----algorithms
|--------signal_source
|--------conditioner
|--------data_type_adapter
|--------input_filter
|--------resampler
|--------channel
|--------acquisition
|-----------adapters
|-----------gnuradio_blocks
|--------tracking
|-----------adapters
|-----------gnuradio_blocks
…

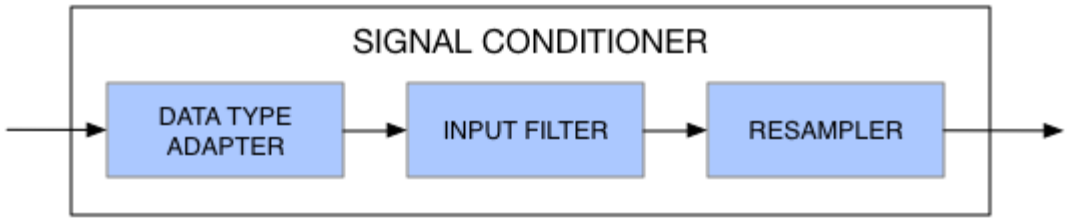


CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Signal Conditioner

- Hierarchical block that contains a data type adapter, an input filter and a resampler.



A Signal Conditioner block is in charge of:
• Adapting the sample bit depth to a data type tractable at the host computer,
• Filtering, and optionally intermediate frequency to baseband conversion, and
• Resampling (to a internal receiver' sampling rate).
Regardless the selected signal source features, the Signal Conditioner interface delivers in a unified format a sample data stream to the receiver downstream processing channels, acting as a facade between the signal source and the synchronization channels.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

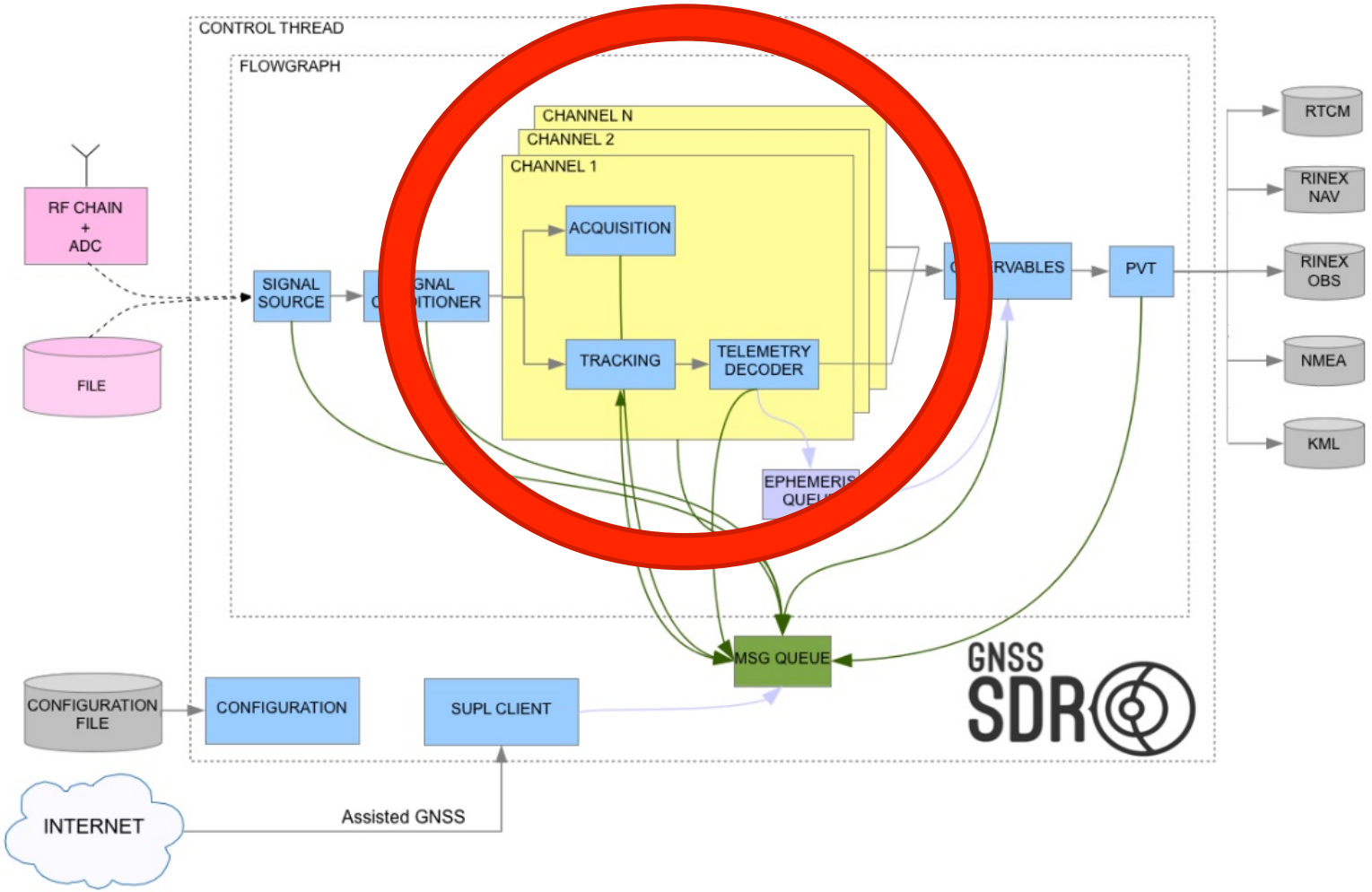
Channel

- Each channel contains an acquisition, tracking and extration of navigation data blocks
  - **Acquisition** searches for visible satellites and makes a rough estimation of their synchronization parameters (time delay and Doppler shift)
  - **Tracking** refines estimation
  - **TelemetryDecoder** demodulates and decodes the navigation data message
- A Finite State Machine handles transitions from acquisition to tracking

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

N parallel channels



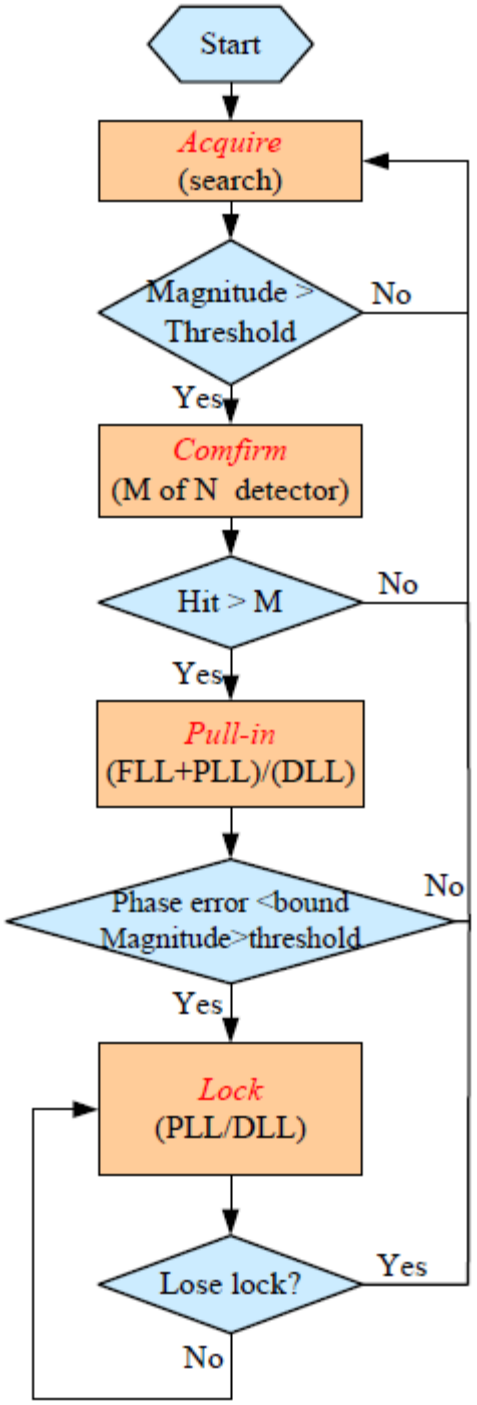
Source code organization

|-gnss-sdr
|---src
|-----algorithms
|--------signal_source
|--------conditioner
|--------data_type_adapter
|--------input_filter
|--------resampler
|--------**channel**
|--------acquisition
|------------adapters
|------------gnuradio_blocks
|--------tracking
|------------adapters
|------------gnuradio_blocks
…

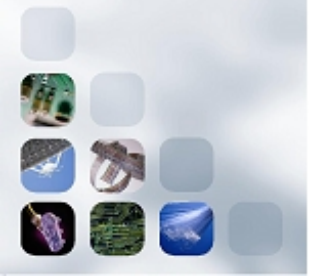
Channel

- The Finite State Machine is programmed using Boost Statechart

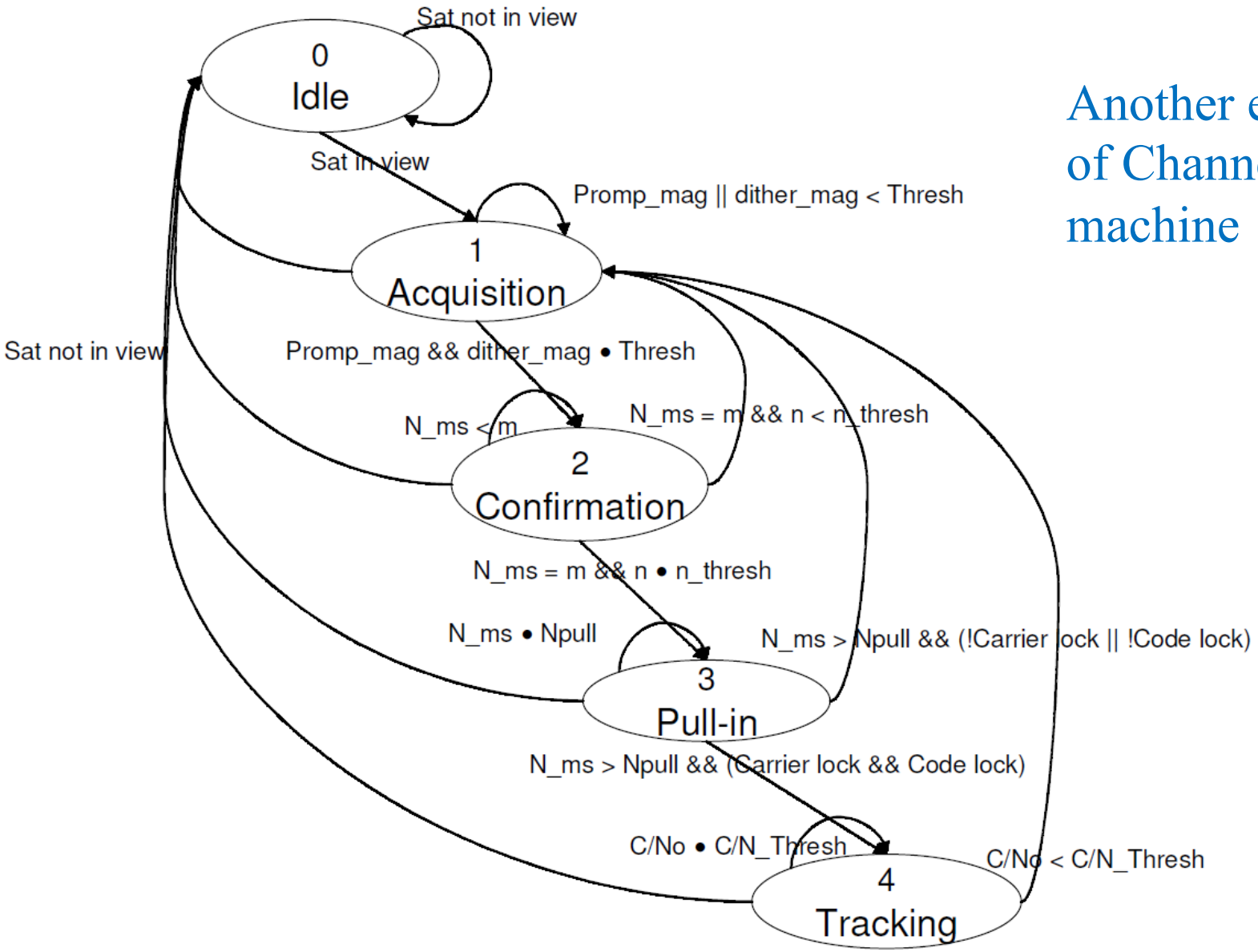
http://www.boost.org/doc/libs/release/libs/statechart/



Example of Channel state machine



Channel



Another example of Channel state machine

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Example of configuration GPS receiver with 6 channels

```
;########## CHANNELS GLOBAL CONFIG ############
;#count: Number of available GPS satellite channels.
Channels_GPS.count=6
;#count: Number of available Galileo satellite channels.
Channels_Galileo.count=0
;#in_acquisition: Number of channels simultaneously acquiring for the whole receiver
Channels.in_acquisition=1
;#system: GPS, GLONASS, GALILEO, SBAS or COMPASS
;#if the option is disabled by default is assigned GPS
Channel.system=GPS

;#if the option is disabled by default is assigned "1C" GPS L1 C/A
Channel.signal=1C
```

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Example of configuration: Hybrid GPS/Galileo receiver

;########## CHANNELS GLOBAL CONFIG ##############

;#count: Number of available GPS satellite channels.

Channels_GPS.count=6

;#count: Number of available Galileo satellite channels.

Channels_Galileo.count=6

;#in_acquisition: Number of channels simultaneously acquiring for the whole receiver

Channels.in_acquisition=1

Channel.system=GPS, Galileo

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Acquisition

1: Compute input signal power estimation:
$\hat{P}_{in} = \frac{1}{N} \sum_{n=0}^{N-1} |x_{\text{IN}}[n]|^2$.
2: **for** $\check{f}_d = f_{min} : f_{step} : \check{f}_d = f_{max}$ **do**
3: Carrier wipe-off:
$x[n] = x_{\text{IN}}[n] \cdot e^{-(j2\pi \check{f}_d n T_s)}$, for $n = 0, ..., N-1$.
4: $X[n] = \text{FFT}_N \{x[n]\}$
5: $Y[n] = X[n] \cdot D[n]$, for $n = 0, ..., N-1$.
6: $R_{xd}(\check{f}_d, \tau) = \frac{1}{N^2} \text{IFFT}_N \{Y[n]\}$
7: **end for**
8: Search maximum and its indices in the search grid:
$\{S_{max}, f_i, \tau_j\} \Leftarrow \max_{f,\tau} |R_{xd}(f, \tau)|^2$
9: Compute the GLRT function with normalized variance:
$\Gamma_{\text{GLRT}} = \frac{2 \cdot N \cdot S_{max}}{\hat{P}_{in}}$
10: **if** $\Gamma_{\text{GLRT}} > \gamma$ **then**
11: Declare positive acquisition and provide $\hat{f}_{d_{acq}} = f_i$ and
$\hat{\tau}_{acq} = \tau_j$.
12: **else**
13: Declare negative acquisition.
14: **end if**

C. Fernández-Prades, J. Arribas, L. Esteve, D. Pubill, P. Closas, "**An Open Source Galileo E1 Software Receiver**", in Proc. of the **6th ESA Workshop on Satellite Navigation Technologies** (NAVITEC 2012), ESTEC, Noordwijk, The Netherlands, Dec. 2012.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Acquisition

Satellite detected!

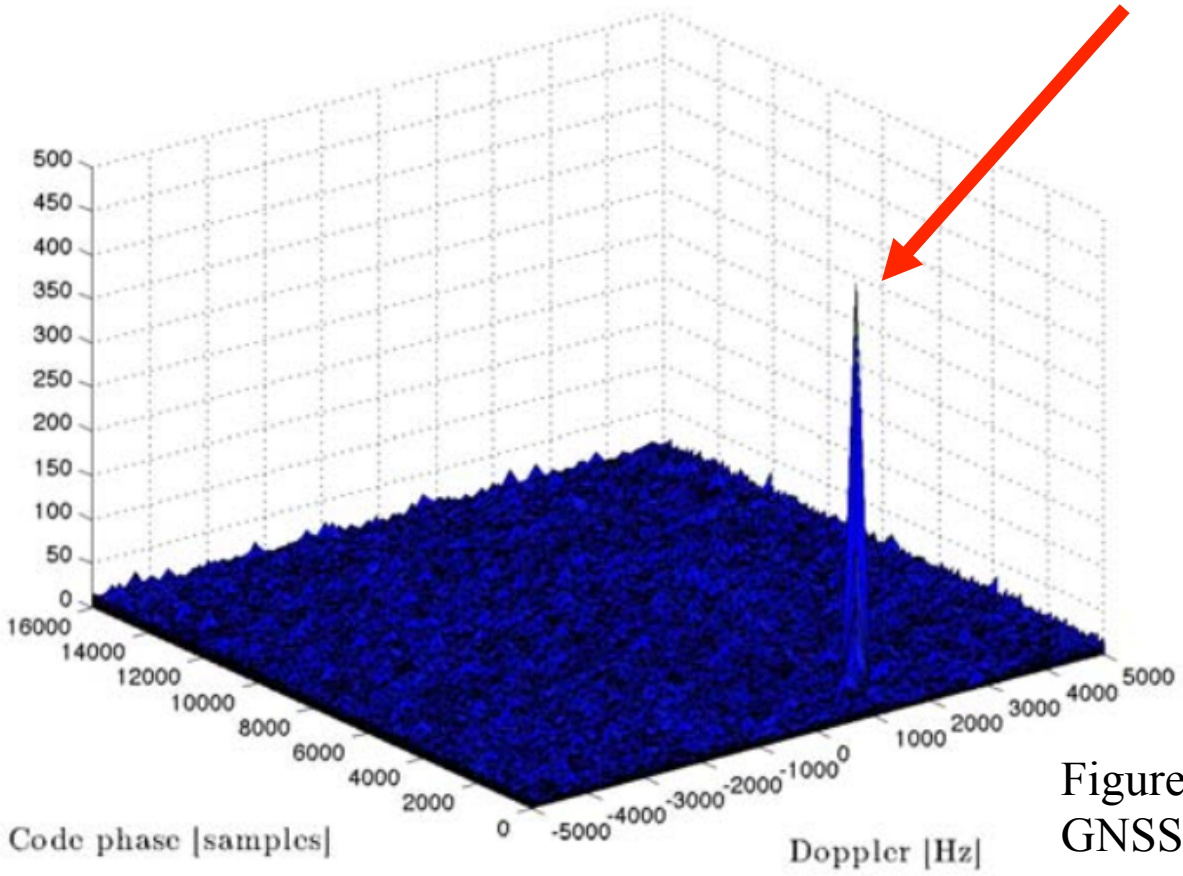
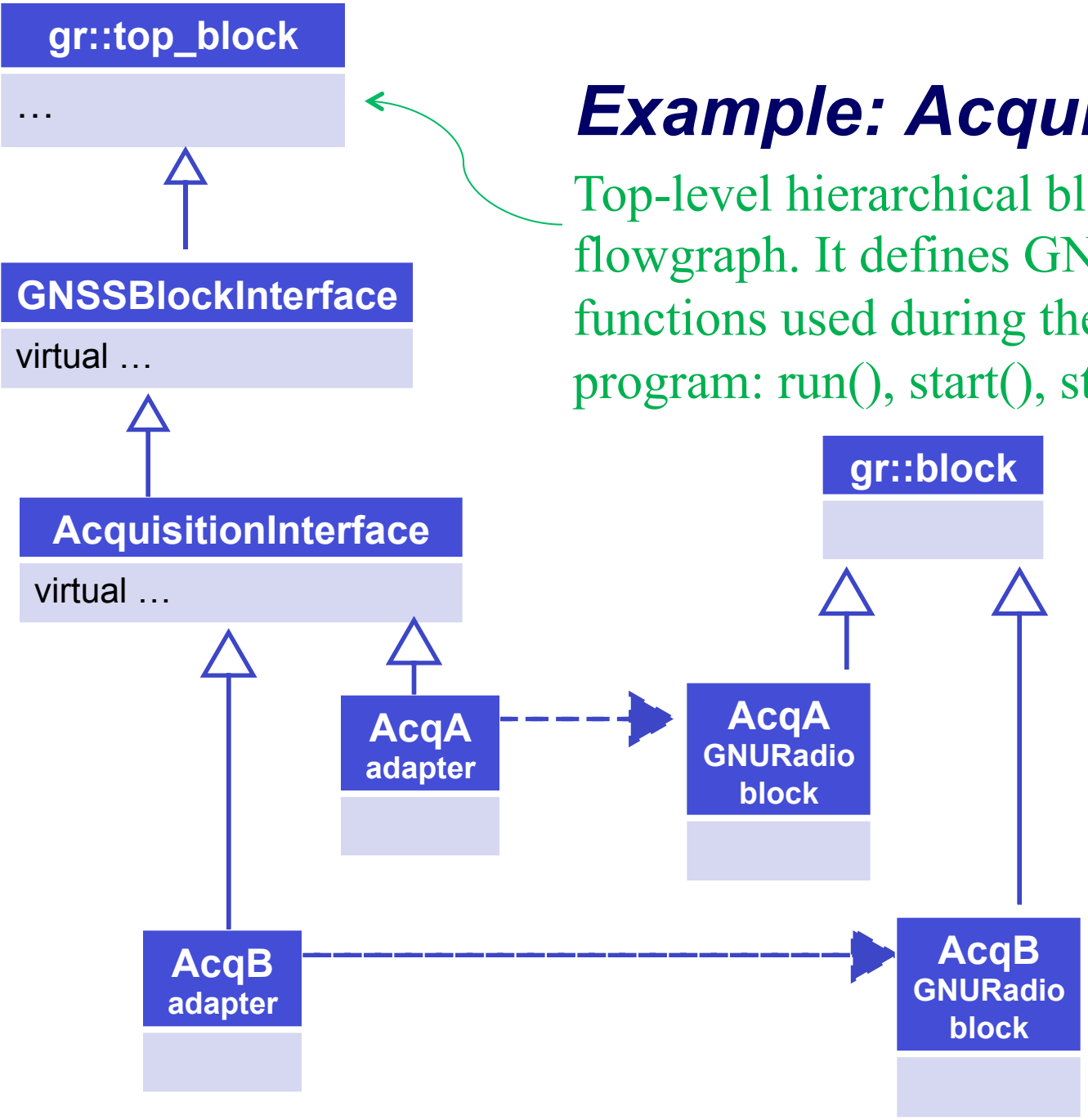


Figure obtained with GNSS-SDR + MATLAB

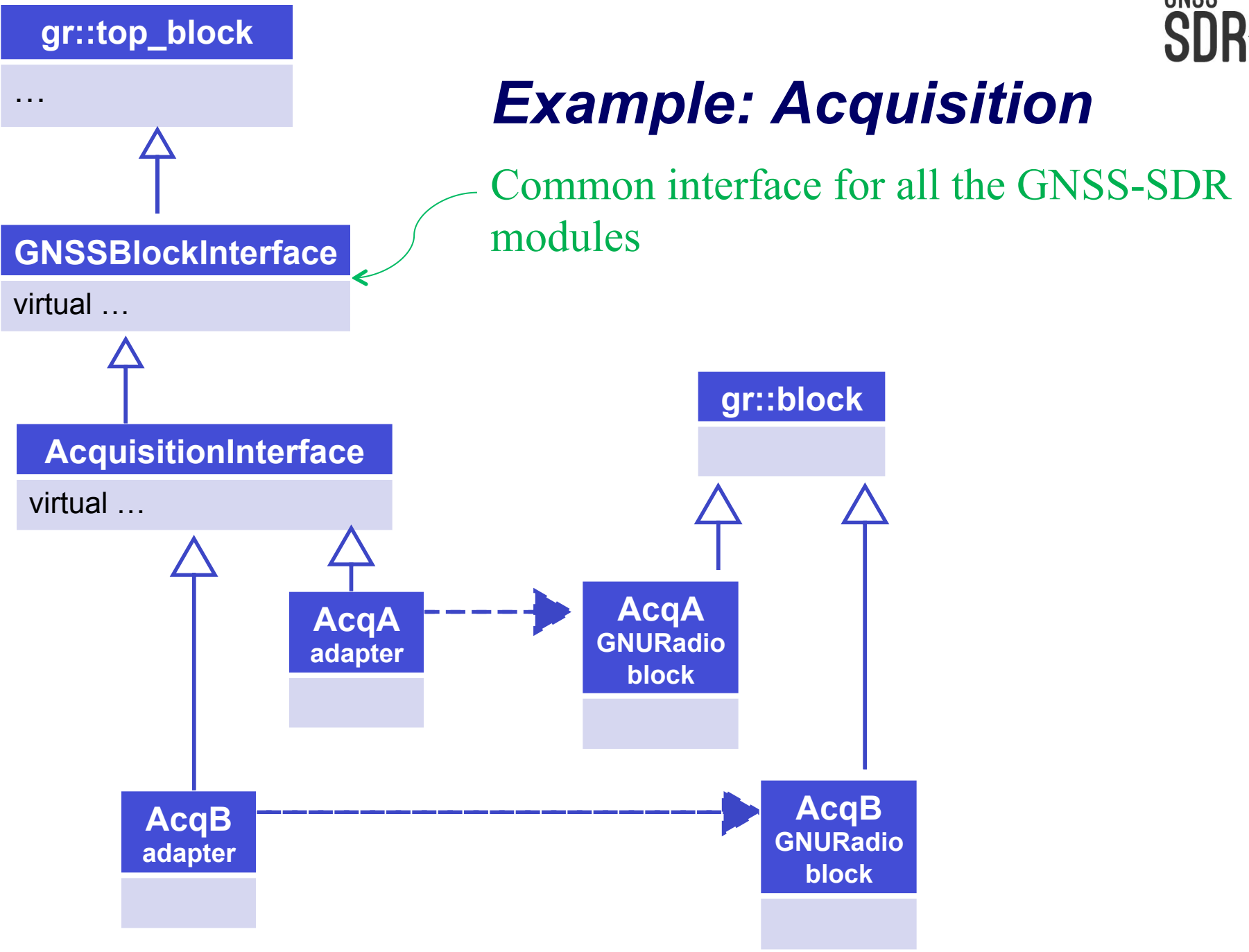
C. Fernández-Prades, J. Arribas, L. Esteve, D. Pubill, P. Closas, "**An Open Source Galileo E1 Software Receiver**", in Proc. of the **6th ESA Workshop on Satellite Navigation Technologies** (NAVITEC 2012), ESTEC, Noordwijk, The Netherlands, Dec. 2012.

Example: Acquisition

Top-level hierarchical block representing a flowgraph. It defines GNURadio runtime functions used during the execution of the program: run(), start(), stop(), wait(), etc.

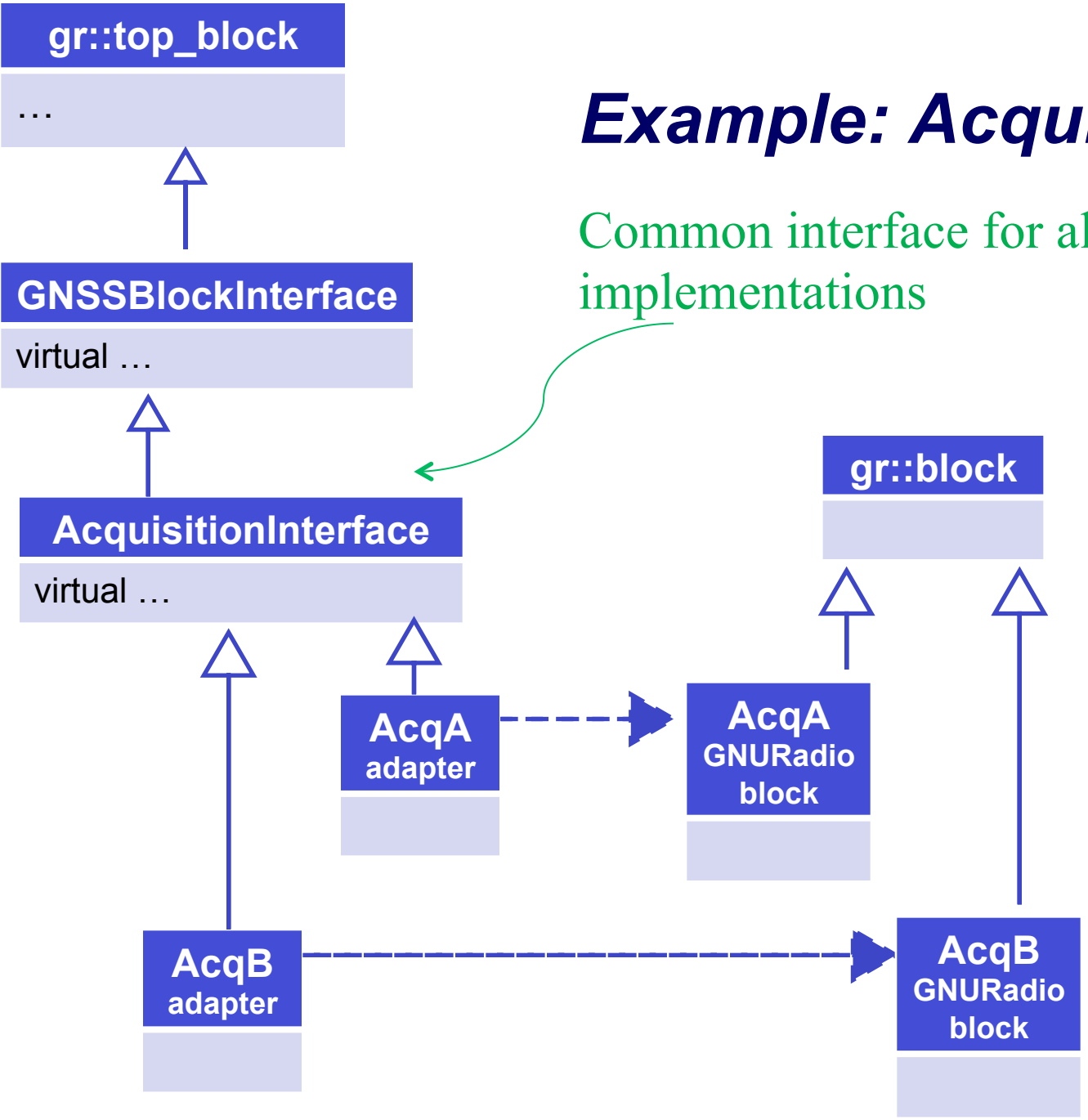


Example: Acquisition

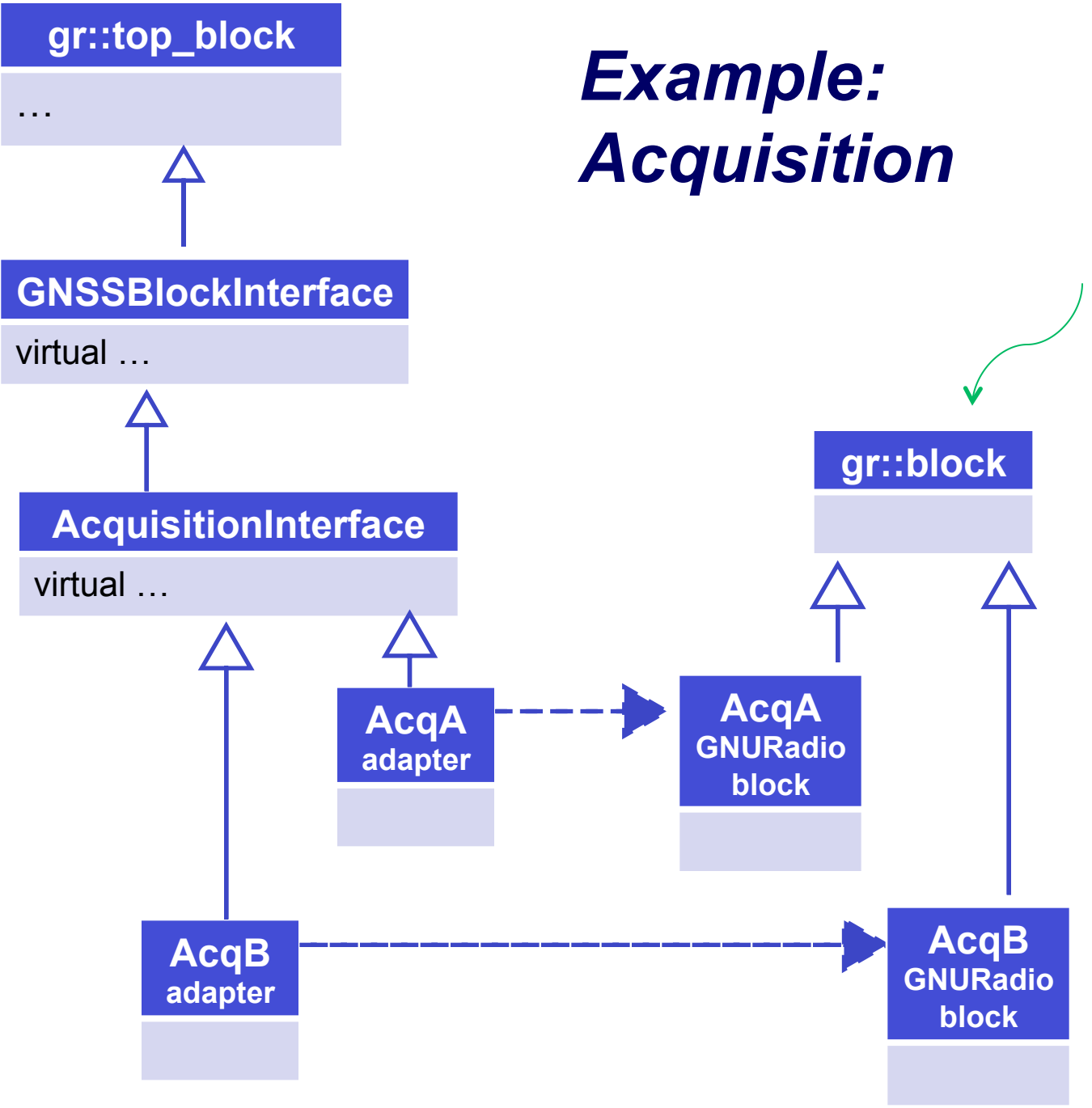


Example: Acquisition

Common interface for all the Acquisition implementations

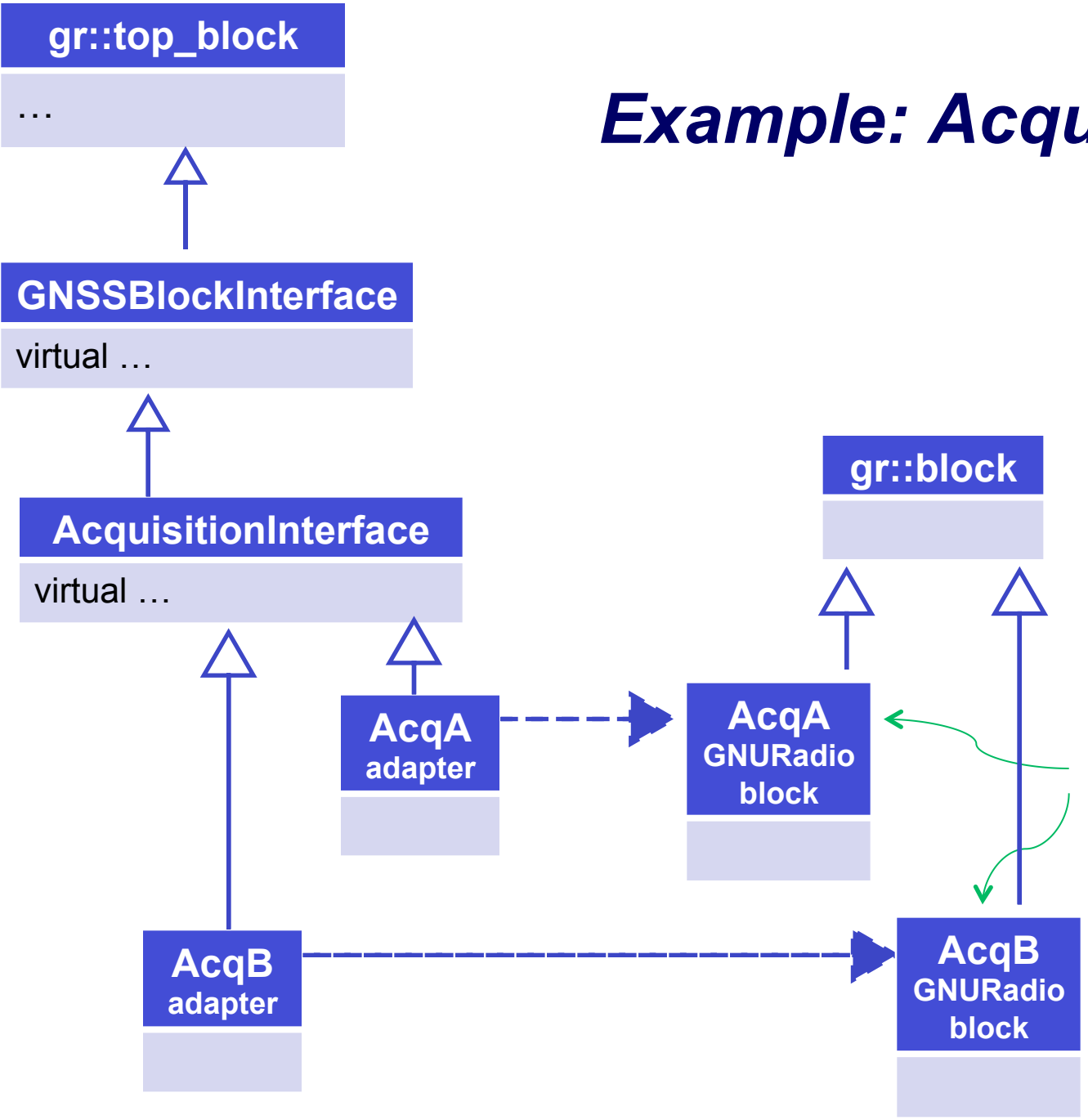


Example: Acquisition



This is the abstract base class for all the processing blocks.
A signal processing flow is constructed by creating a tree of hierarchical blocks, which at any level may also contain terminal nodes that actually implement signal processing functions. This is the base class for all such nodes.

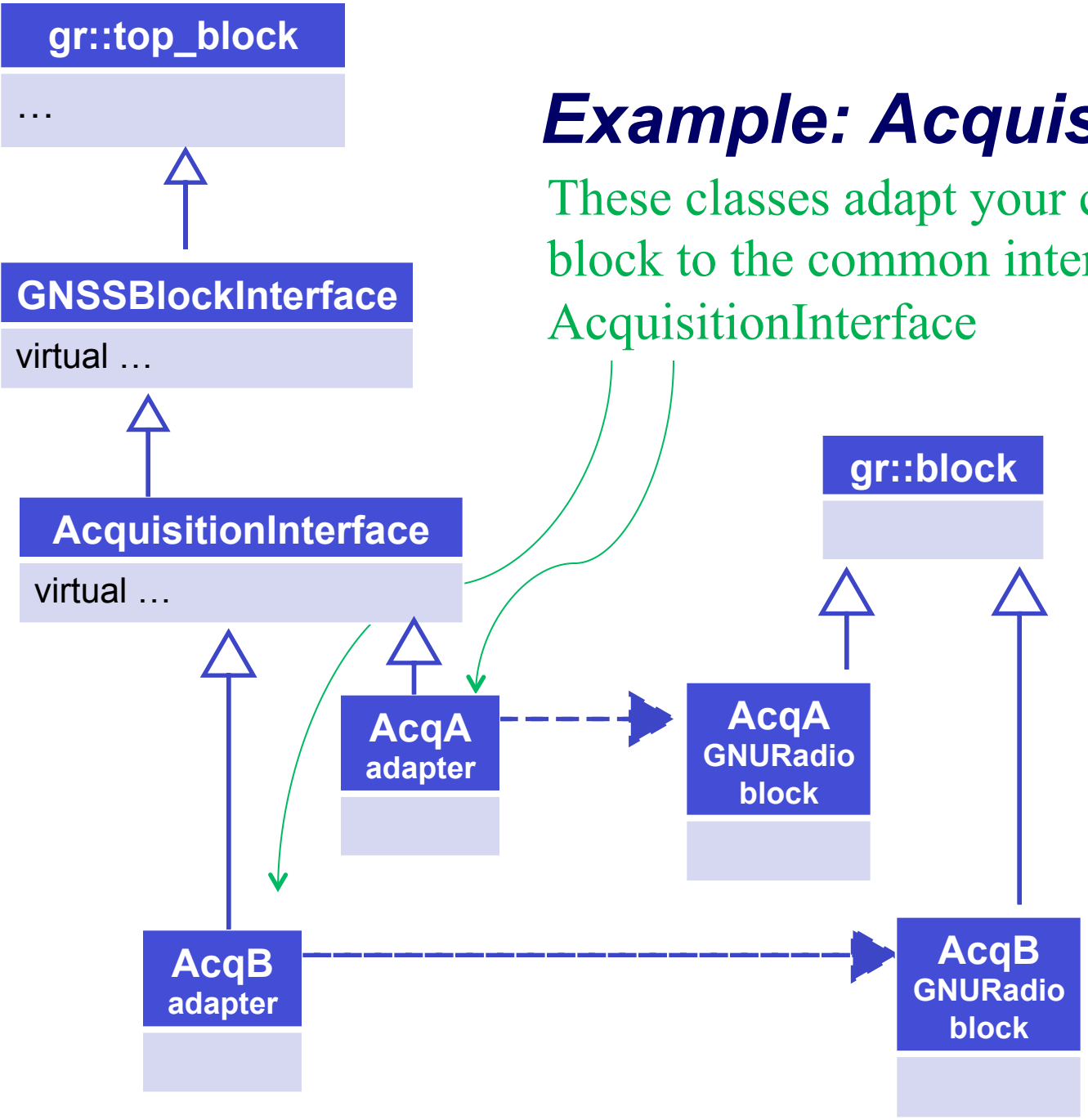
Example: Acquisition



These are different implementations of acquisition. Actual signal processing is done here. You can use existing GNURadio blocks or implement a new one by yourself.

Example: Acquisition

These classes adapt your custom acquisition block to the common interface expected by AcquisitionInterface



CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Adding new algorithms

- I have a nice acquisition algorithm and I want to test it in the framework of GNSS-SDR. What should I do?
  - Create a GNURadio block, derived from gr::block, implementing your algorithm (.cc and .h files) and puttting it in src/algorithms/acquisition/gnuradio_blocks/
  - Create the adapter of such new block to AcquisitionInterface (.cc and .h files) and putting it in src/algorithms/acquisition/adapters/
  - Tell the system that a new block is available and include it in the 'production line': add the instantation of your new block in src/core/receiver/GNSSBlockFactory.cc
  - Update the CMakeLists.txt accordingly to tell the compiler that a new block exists.

Easy, examples are available

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Example of configuration GPS L1 signal acquisition

```
;######### GPS ACQUISITION CONFIG ############
;#dump: Enable or disable the acquisition internal data file logging [true] or [false]
Acquisition_GPS.dump=false
;#filename: Log path and filename
Acquisition_GPS.dump_filename=./acq_dump.dat
;#item_type: Type and resolution for each of the signal samples. Use only gr_complex in this version.
Acquisition_GPS.item_type=gr_complex
;#if: Signal intermediate frequency in [Hz]
Acquisition_GPS.if=0
;#sampled_ms: Signal block duration for the acquisition signal detection [ms]
Acquisition_GPS.sampled_ms=1
;#implementation: Acquisition algorithm selection for this channel
Acquisition_GPS.implementation=GPS_L1_CA_PCPS_Acquisition
;#threshold: Acquisition threshold
Acquisition_GPS.threshold=0.0075
;#pfa: Acquisition false alarm probability. This option overrides the threshold option.
;Acquisition_GPS.pfa=0.01
;#doppler_max: Maximum expected Doppler shift [Hz]
Acquisition_GPS.doppler_max=10000
;#doppler_max: Doppler step in the grid search [Hz]
Acquisition_GPS.doppler_step=500
```

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

Example of configuration Galileo E1 signal acquisition

GNSS SDR

```
;######### GALILEO ACQUISITION CONFIG ############
;#dump: Enable or disable the acquisition internal data file logging [true] or [false]
Acquisition_Galileo.dump=false
;#filename: Log path and filename
Acquisition_Galileo.dump_filename=./acq_dump.dat
;#item_type: Type and resolution for each of the signal samples. Use only gr_complex in this version.
Acquisition_Galileo.item_type=gr_complex
;#if: Signal intermediate frequency in [Hz]
Acquisition_Galileo.if=0
;#sampled_ms: Signal block duration for the acquisition signal detection [ms]
Acquisition_Galileo.sampled_ms=4
;#implementation: Acquisition algorithm selection for this channel: [GPS_L1_CA_PCPS_Acquisition] or [Galileo_E1_PCPS_Ambiguous_Acquisition]
Acquisition_Galileo.implementation=Galileo_E1_PCPS_Ambiguous_Acquisition
;#threshold: Acquisition threshold
;Acquisition_Galileo.threshold=0
;#pfa: Acquisition false alarm probability. This option overrides the threshold option. Only use with implementations: [GPS_L1_CA_PCPS_Acquisition] or [Galileo_E1_PCPS_Ambiguous_Acquisition]
Acquisition_Galileo.pfa=0.0000008
;#doppler_max: Maximum expected Doppler shift [Hz]
Acquisition_Galileo.doppler_max=15000
;#doppler_max: Doppler step in the grid search [Hz]
Acquisition_Galileo.doppler_step=125
```

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Tracking

- The Tracking block is also receiving the data stream from the Signal Conditioner, but does nothing until it receives a “positive acquisition” message from the control plane, along with the coarse estimations of code phase and frequency shift performed by the Acquisition block.
- Then, its role is to refine such estimations and track their changes along the time.
- Three parameters are relevant for signal tracking: the evolution of the code phase, Doppler shift and carrier phase.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Example of configuration GPS L1 signal tracking

```
;########## TRACKING GPS CONFIG #############

;#implementation: Selected tracking algorithm:

Tracking_GPS.implementation=GPS_L1_CA_DLL_PLL_Tracking
;#item_type: Type and resolution for each of the signal samples.
Tracking_GPS.item_type=gr_complex
;#sampling_frequency: Signal Intermediate Frequency in [Hz]
Tracking_GPS.if=0
;#dump: Enable or disable the Tracking internal binary data file logging [true] or [false]
Tracking_GPS.dump=false
;#dump_filename: Log path and filename.
Tracking_GPS.dump_filename=../data/epl_tracking_ch_
;#pll_bw_hz: PLL loop filter bandwidth [Hz]
Tracking_GPS.pll_bw_hz=45.0;
;#dll_bw_hz: DLL loop filter bandwidth [Hz]
Tracking_GPS.dll_bw_hz=4.0;
;#fll_bw_hz: FLL loop filter bandwidth [Hz]
Tracking_GPS.fll_bw_hz=10.0;
;#order: PLL/DLL loop filter order [2] or [3]
Tracking_GPS.order=3;
```

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

Example of configuration Galileo E1 signal tracking

GNSS SDR

```
;########## TRACKING GALILEO CONFIG #############

;#implementation: Selected tracking algorithm:

Tracking_Galileo.implementation=Galileo_E1_DLL_PLL_VEML_Tracking
;#item_type: Type and resolution for each of the signal samples. Use only [gr_complex] in this version.
Tracking_Galileo.item_type=gr_complex
;#sampling_frequency: Signal Intermediate Frequency in [Hz]
Tracking_Galileo.if=0
;#dump: Enable or disable the Tracking internal binary data file logging [true] or [false]
Tracking_Galileo.dump=false

;#dump_filename: Log path and filename.

Tracking_Galileo.dump_filename=../data/veml_tracking_ch_
;#pll_bw_hz: PLL loop filter bandwidth [Hz]
Tracking_Galileo.pll_bw_hz=15.0;
;#dll_bw_hz: DLL loop filter bandwidth [Hz]
Tracking_Galileo.dll_bw_hz=2.0;
;#fll_bw_hz: FLL loop filter bandwidth [Hz]
Tracking_Galileo.fll_bw_hz=10.0;
;#order: PLL/DLL loop filter order [2] or [3]
Tracking_Galileo.order=3;
;#early_late_space_chips: correlator early-late space [chips]. Use [0.5] for GPS and [0.15] for Galileo
Tracking_Galileo.early_late_space_chips=0.15;

;#very_early_late_space_chips

Tracking_Galileo.very_early_late_space_chips=0.6;
```

Galileo E1 signal tracking

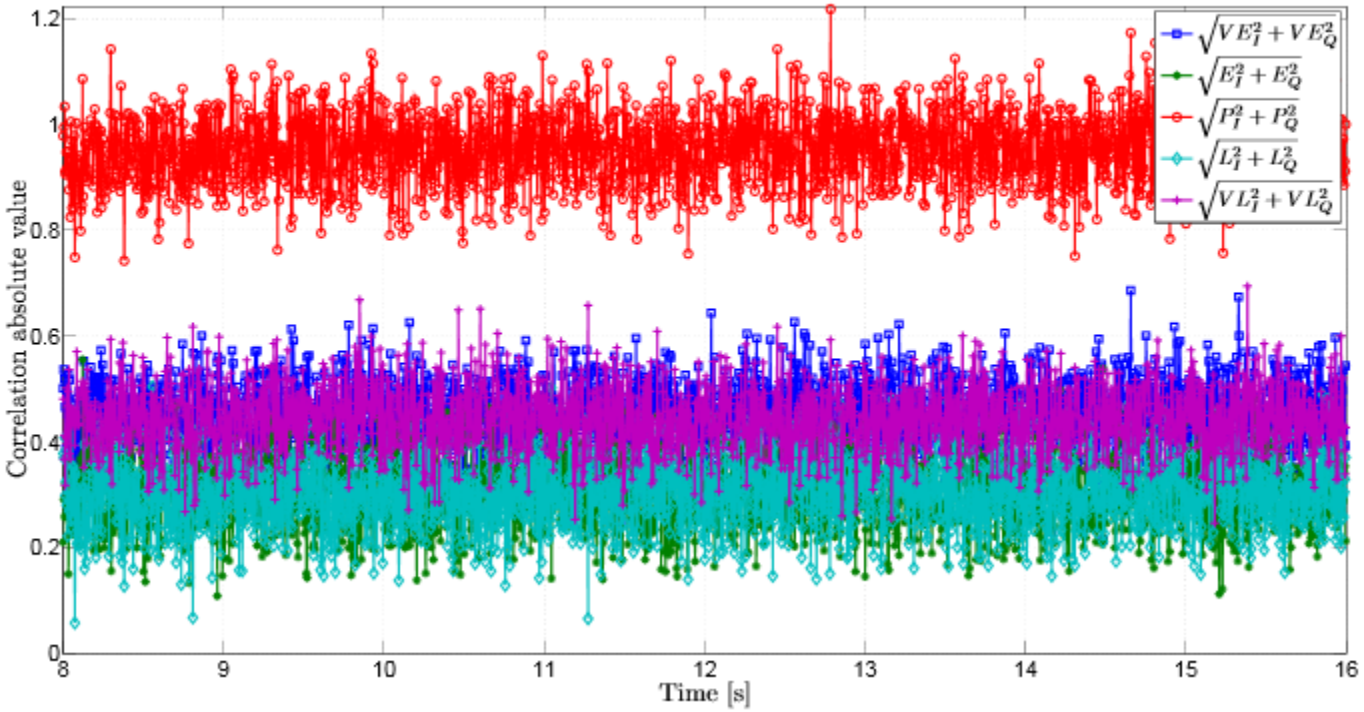


Figure obtained with
GNSS-SDR + MATLAB

C. Fernández-Prades, J. Arribas, L. Esteve, D. Pubill, P. Closas, "**An Open Source Galileo E1 Software Receiver**", in Proc. of the **6th ESA Workshop on Satellite Navigation Technologies** (NAVITEC 2012), ESTEC, Noordwijk, The Netherlands, Dec. 2012.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Telemetry Decoder

- Most of GNSS signal links are modulated by a **navigation message** containing the time when the message was transmitted, orbital parameters of satellites (also known as ephemeris) and an almanac (information about the general system health, rough orbits of all satellites in the network as well as data related to error correction).

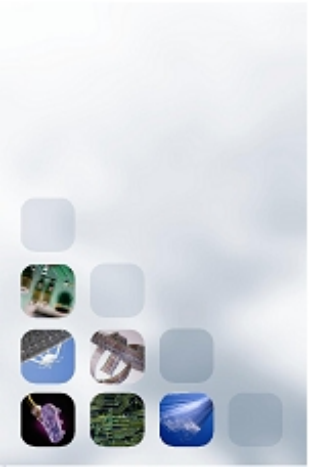
- Navigation data bits are structured in words, pages, subframes, frames and superframes. Some times, bits corresponding to a single parameter are spread over different words, and values extracted from different frames are required for proper decoding.

- All this decoding complexity is managed by a finite state machine implemented with the Boost.Statechart library

Example of configuration GPS L1 NAV message

```
;########## TELEMETRY DECODER GPS CONFIG #############

TelemetryDecoder_GPS.implementation=GPS_L1_CA_Telemetry_Decoder
TelemetryDecoder_GPS.dump=false
;#decimation factor
TelemetryDecoder_GPS.decimation_factor=1;
```



CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

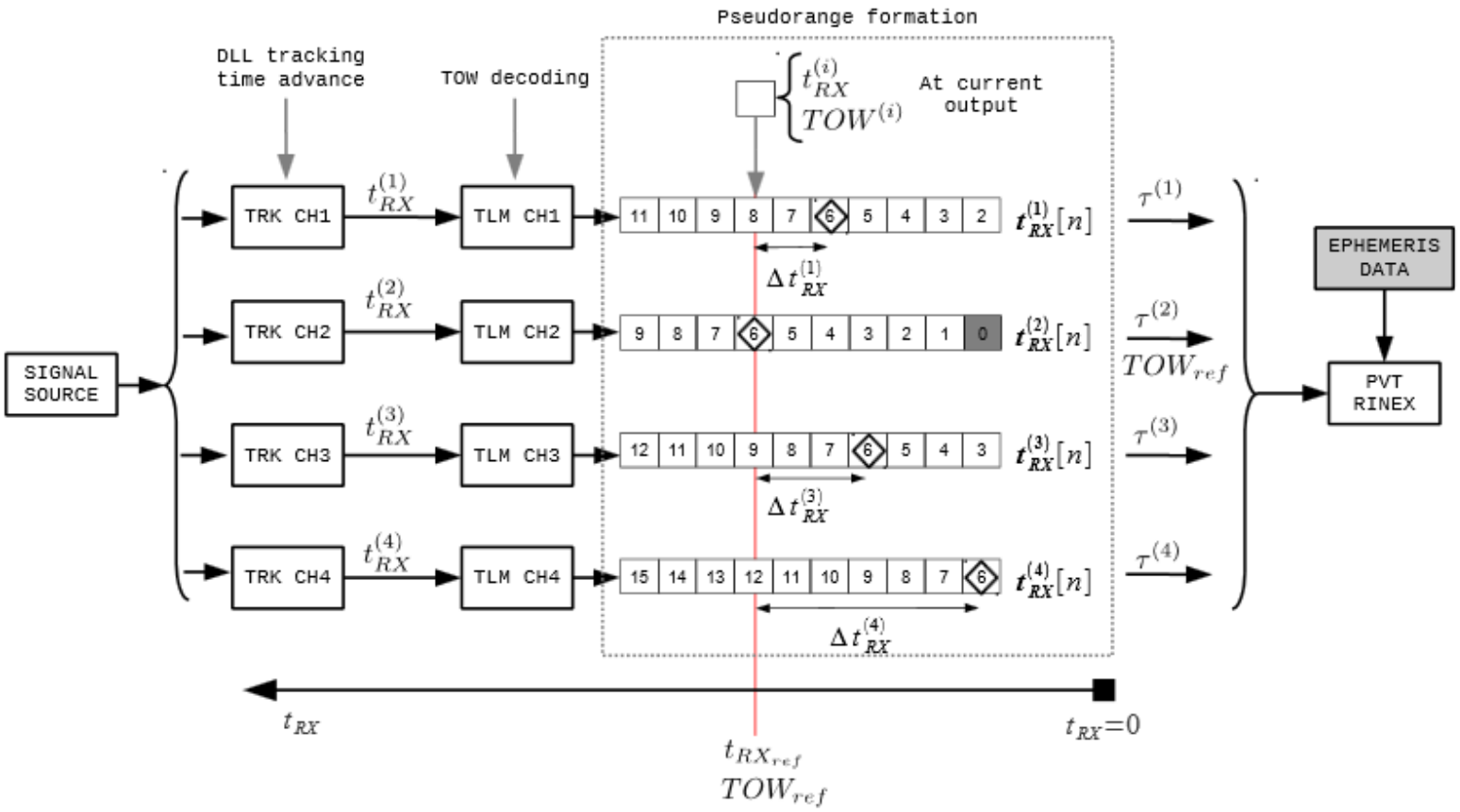
GNSS SDR

Example of configuration Galileo E1 INAV message

```
;########## TELEMETRY DECODER GALILEO CONFIG ##########
;#implementation
TelemetryDecoder_Galileo.implementation=Galileo_E1B_Telemetry_Decoder
TelemetryDecoder_Galileo.dump=false
```

Observables

Pseudorange computation based on common reception time



M. Petovello M. Rao, G. Falca, “Code Tracking & Pseudoranges,” Inside GNSS, vol. 7, pp. 26–33, January 2012

Example of configuration GPS L1 receiver

;########## OBSERVABLES CONFIG ##############

Observables.implementation=GPS_L1_CA_Observables

;#dump: Enable or disable the Observables internal binary data file logging [true] or [false]

Observables.dump=false

;#dump_filename: Log path and filename.

Observables.dump_filename=./observables.dat

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Example of configuration Hybrid GPS/Galileo receiver

```
;########## OBSERVABLES CONFIG ##############
;#implementation:
Observables.implementation=Hybrid_Observables

;#dump: Enable or disable the Observables internal binary data file logging
Observables.dump=false

;#dump_filename: Log path and filename.
Observables.dump_filename=./observables.dat
```

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

PVT (Position – Velocity – Time)

- A simple, memoryless least squares solution is implemented.
- The receiver also produces RINEX 3.02 for further processing. RTCM 3.2 is ongoing work.
- RINEX files can be processed by other software packages (such as RTKLIB), applying high-accuracy techniques (DGPS, RTK, PPP).

Example of configuration GPS L1 receiver

;######### PVT CONFIG ############

;#implementation: Position Velocity and Time (PVT) implementation algorithm: Use [GPS_L1_CA_PVT] in this version.

PVT.implementation=GPS_L1_CA_PVT

;#averaging_depth: Number of PVT observations in the moving average algorithm

PVT.averaging_depth=100

;#flag_average: Enables the PVT averaging between output intervals (arithmetic mean) [true] or [false]

PVT.flag_averaging=true

;#output_rate_ms: Period between two PVT outputs.

PVT.output_rate_ms=10

;#display_rate_ms: Position console print (std::out) interval [ms]. Notice that output_rate_ms<=display_rate_ms.

PVT.display_rate_ms=500

;#dump_filename: Log path and filename without extension. Notice that PVT will add ".dat" to the binary dump and ".kml" to GoogleEarth dump.

PVT.dump_filename=./PVT

;#nmea_dump_filename: NMEA log path and filename

PVT.nmea_dump_filename=./gnss_sdr_pvt.nmea;

;#flag_nmea_tty_port: Enable or disable the NMEA log to a serial TTY port (Can be used with real hardware or virtual one)

PVT.flag_nmea_tty_port=true;

;#nmea_dump_devname: serial device descriptor for NMEA logging

PVT.nmea_dump_devname=/dev/pts/4

Example of configuration Hybrid GPS/Galileo receiver

;######### PVT CONFIG #############

;#implementation: Position Velocity and Time (PVT) implementation algorithm.

PVT.implementation=Hybrid_PVT

This implementation makes use of GPS and Galileo observables

;#averaging_depth: Number of PVT observations in the moving average algorithm

PVT.averaging_depth=10

;#flag_average: Enables the PVT averaging between output intervals (arithmetic mean) [true] or [false]

PVT.flag_averaging=false

;#output_rate_ms: Period between two PVT outputs. Minimum period is equal to the tracking integration time (for GPS CA L1 is 1ms) [ms]

PVT.output_rate_ms=10;

;#display_rate_ms: Position console print (std::out) interval [ms]. Notice that output_rate_ms<=display_rate_ms.

PVT.display_rate_ms=500;

;#dump: Enable or disable the PVT internal binary data file logging [true] or [false]

PVT.dump=false

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

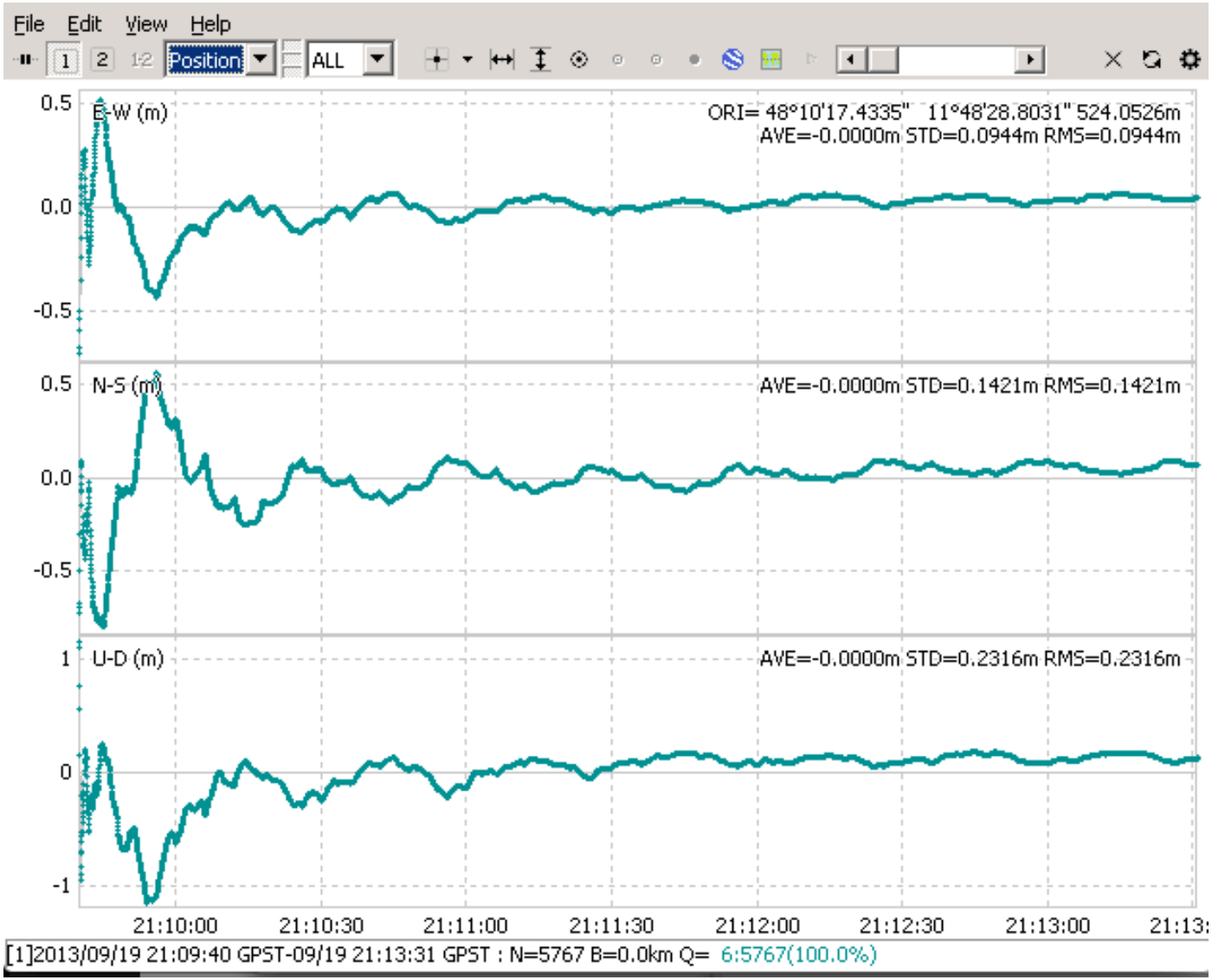


Figure obtained with GNSS-SDR + RTKLIB

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

Testing

We suggest a test–driven development approach. This methodology is claimed to offer valuable benefits to software development:

- it facilitates change,
- simplifies integration,
- automatizes documentation,
- helps separate the interface from the implementation, increases developers productivity, and
- plays a central role in the software quality assurance process.

For unit testing, we find the **Google C++ Testing Framework (gtest)** useful and lightweight.

CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

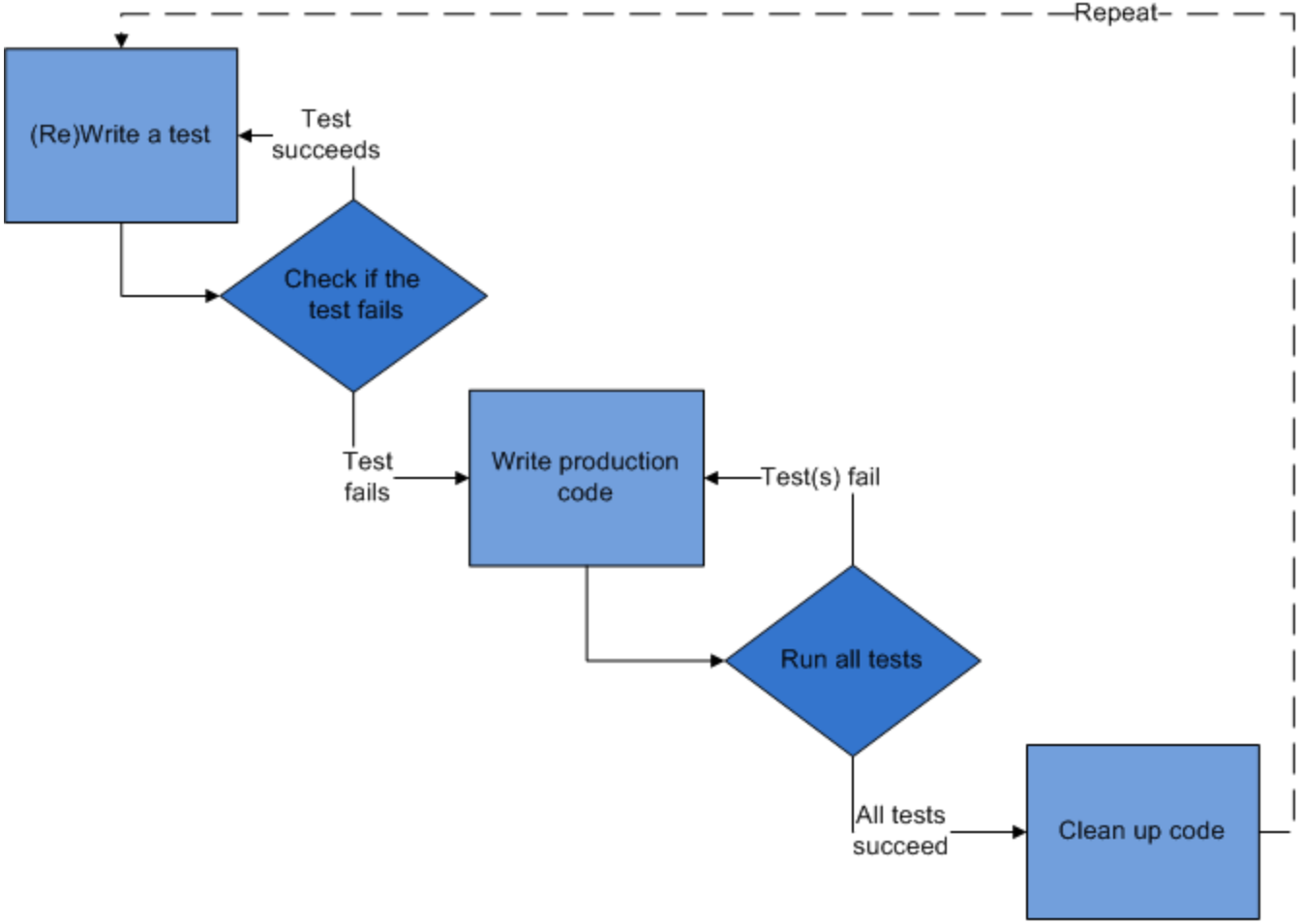
Source code organization

|-gnss-sdr
|---build
|---cmake
|---conf
|---data
|---docs
|---drivers
|---firmware
|---install <- Executable run_tests
|---src
|-----algorithms
|-----core
|-----main
|-----tests <- tests are written here
|-----utils

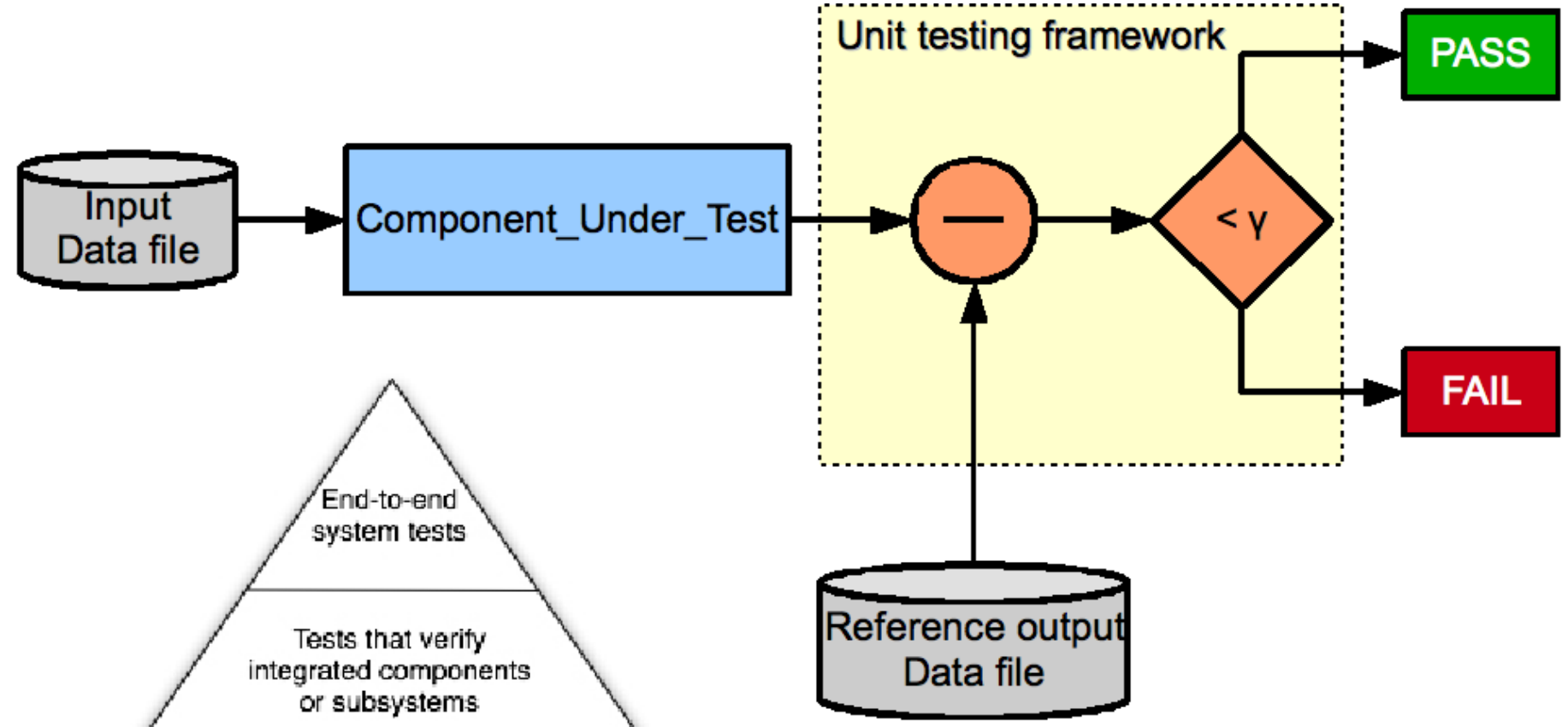
CTTC
Centre Tecnològic de Telecomunicacions de Catalunya

GNSS SDR

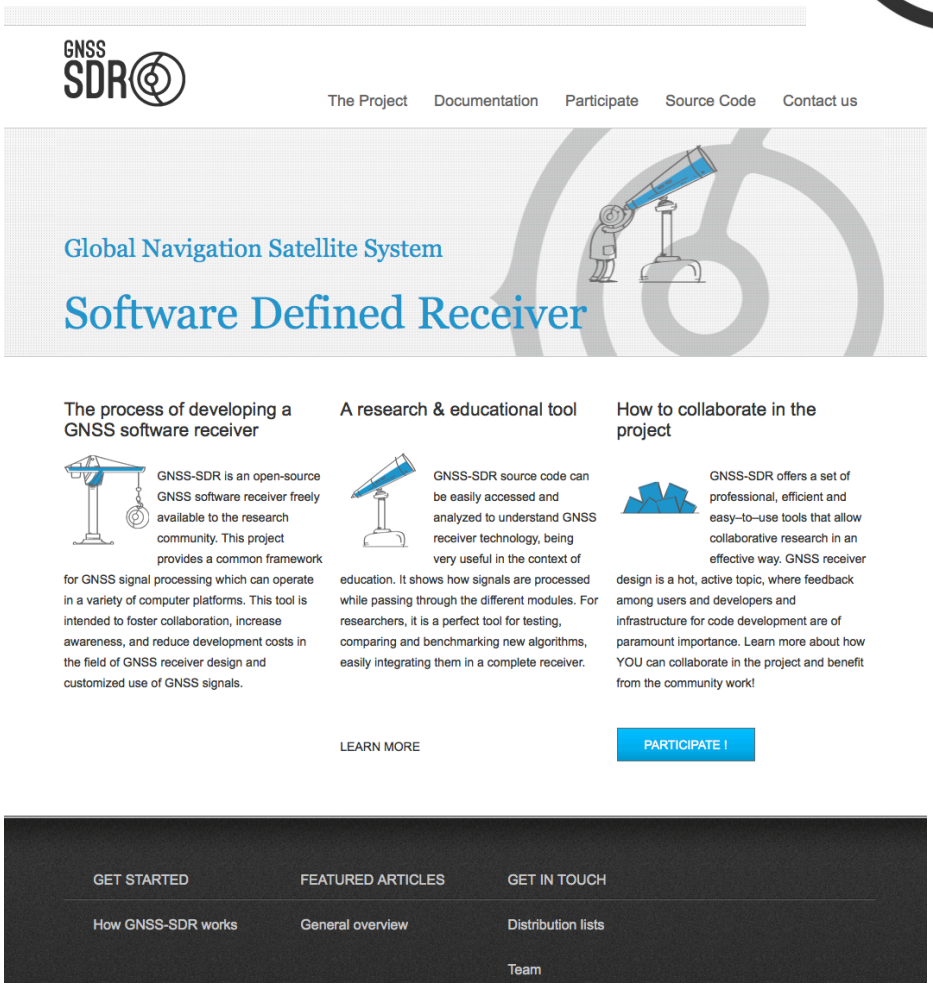
Test-driven development



Testing


Unit testing framework
PASS
Input
Data file
Component_Under_Test
< γ
FAIL
Reference output
Data file
End-to-end
system tests
Tests that verify
integrated components
or subsystems
Tests that verify components in isolation

Thank you!



Please visit **http://gnss-sdr.org** and find:

- ✓ An overview about the project.
- ✓ Instructions for downloading and building the source code.
- ✓ Documentation and tutorials.
- ✓ Links to the mailing list.
- ✓ Suggestions for participation.

GNSS-SDR forms part of the GESTALT® Testbed, and has been partially supported by ERDF.

European Union

European Regional Development Fund

Investing in your future

AENOR ER Gestión I+D+i UNE 166002 IDI-0024/2011