FFMM: a Flexible Software Framework for Magnetic Measurements

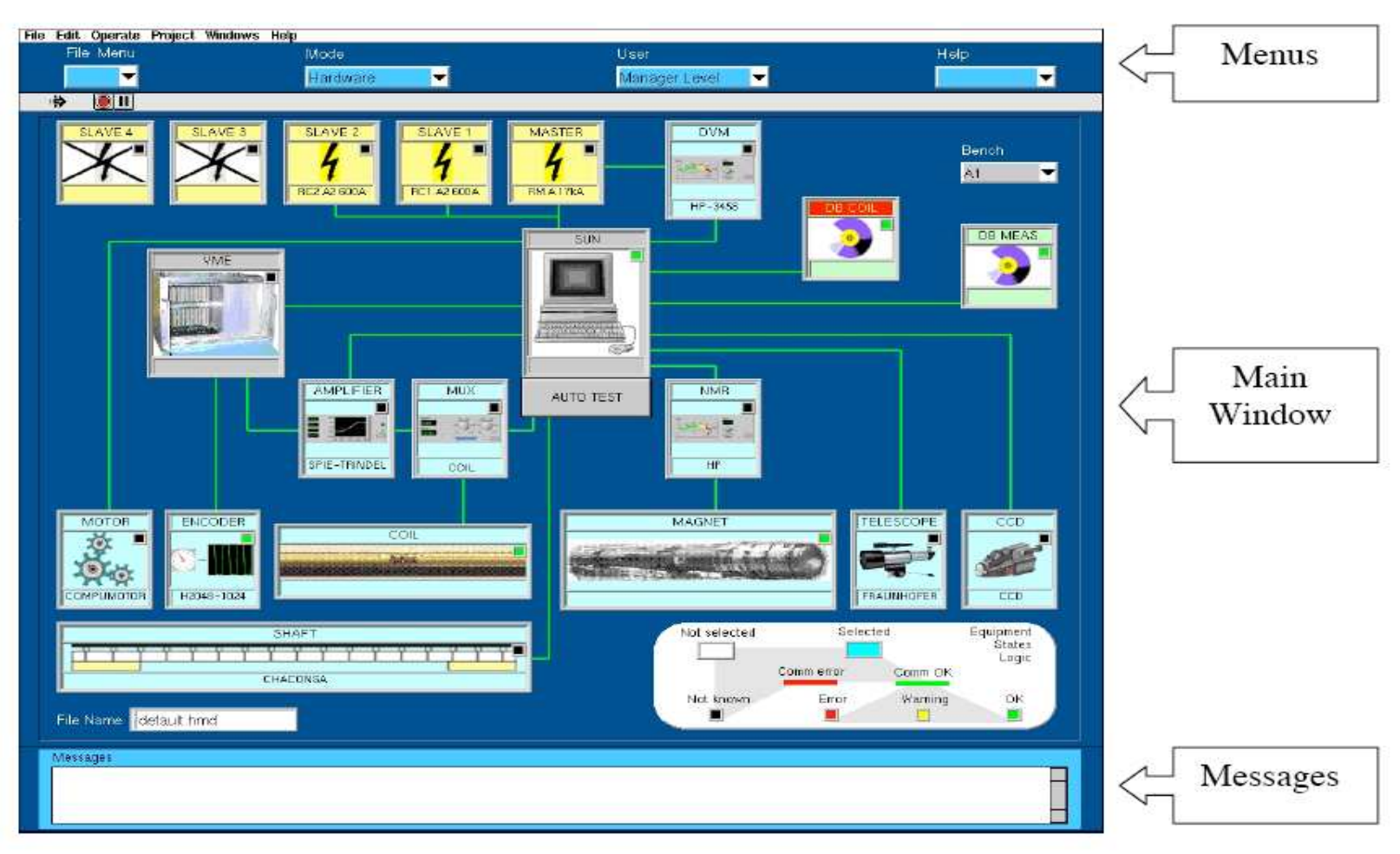
P Arpaia[1], ML Bernardi[1], L Bottura, M Buzio, D Della Ratta[1],
L Deniau, G Golluccio[1], V Inglese[1], G Lucca[1], G Spiezia[1], S Tiso[1]

Contents

1 = University of Sannio, Benevento, Italy

Introduction

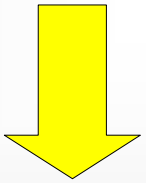
Status: our current software solution for magnetic measurements, i.e. MMP" [1], has successfully seen us through prototype & series LHC tests, however …



1=see M Gateau's talk, IMMW 1411

Introduction

- program developed incrementally over 10+ yrs lifespan by multiple programmers ⇒
  very difficult to debug, maintain and evolve
- overall architecture is becoming obsolete: expensive Sun UNIX workstations + slow VME DAQ hardware ⇒
  not suited for long-term R&D, esp. upcoming fast measurement systems
- LabView-based ⇒
  limited expert programmer base, CERN support staff redirected to new priorities



Need for a new standard software platform for magnetic measurements

Functional specifications

... what do we want ?

- a unified software solution to drive all of our existing and future park of measurement systems (magnetic but also optical, mechanical etc.)
- a flexible system for rapid prototyping in a R&D context ("scriptable" applications)
- a modular architecture to mix and reuse components chosen from an incremental library:
  - power supplies: for magnets and instrumentation
  - magnetic sensors: rotating/fixed coils, hall probes, NMR, stretched wires …
  - other sensors: encoders, torque/force/strain gages, optical …
  - DAQ electronics: voltmeters, multiplexers, integrators …
  - data processing: harmonic analysis, fault detection, logging, database interface …
- a reliable platform expected to last ≥ **10 yrs** (based on experience with MMP)
- performance: more throughput than existing system (typical application needs: control and acquisition of < 100 components; raw data rates < 100 MB/s)

Functional specifications (cont.)

… and what is not included:

- acquisition and control in real time ⇒ synchronization and latencies below ~1 ms must be handled in hardware (irreversible choice !)
- functionality for a production environment (e.g. walkthrough interface): if and when required, department level involvement necessary
- a user-friendly, interactive graphical interface for the framework and the end-user applications: would be nice, but not the first priority

Technical specifications

… how do we think to do it ?

- hardware platform: widespread and inexpensive ⇒ PC + PXI external expansion bus
- operating system: CERN-wide supported ⇒ Linux and /or Windows
- programming language: industry standard, scalable to large scale projects, fast optimized executables, in-built modularity (object-oriented) ⇒ C/C++

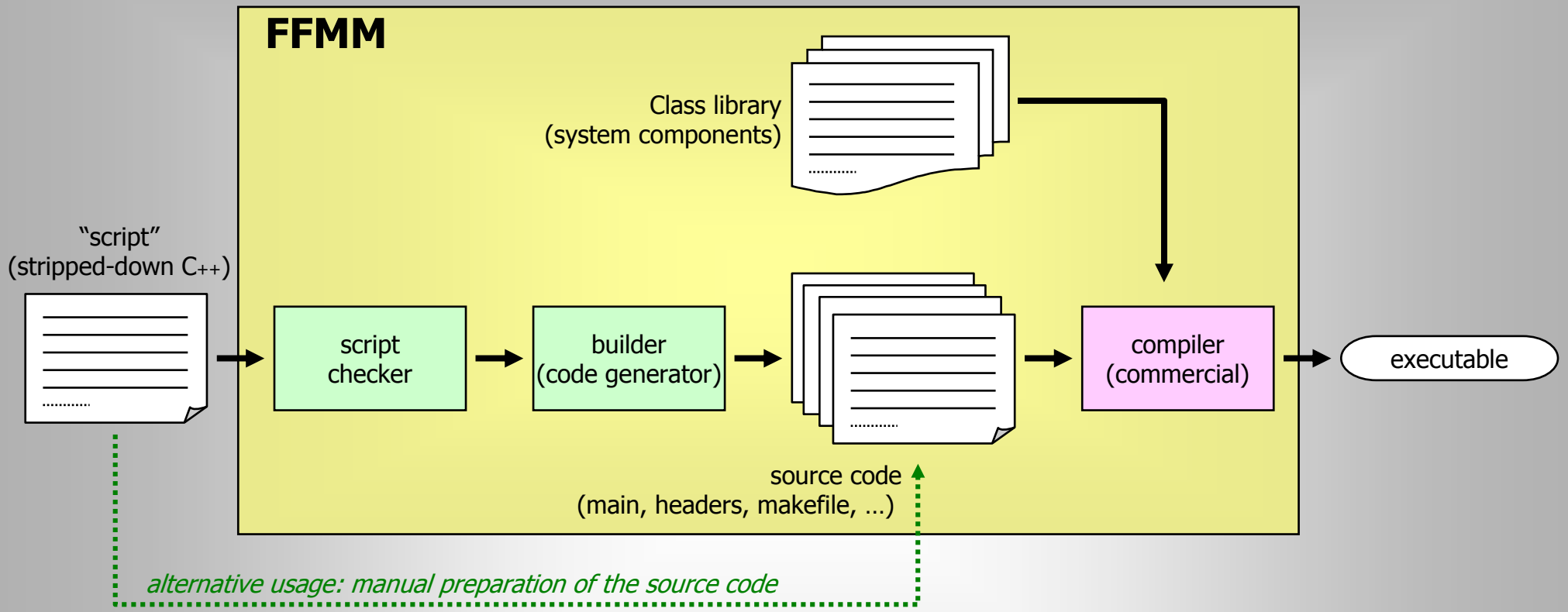
State of the Art

CERN	**FESA** Front end Software Architecture for Accelerator Controls	• Distributed, large-scale, real-time • Intended for accelerator control systems
ESRF, ELETTRA, SOLEIL, ALBA	**TANGO** Control system framework for accelerators and beamlines	*(as above)*
FNAL	**EMS** Extensible Measurement System	• Targets precisely our needs (few, complex custom instruments)
Groningen U.	**Object-oriented C++ measurement framework**	• Distributed, concurrent test & measurement systems • Oriented towards industrial QA (many simple sensors)
National Instruments	**Test Stand**	• Visual sequencer for test modules in LabView/C++/etc. • Oriented towards industrial QA

... why build another then ?

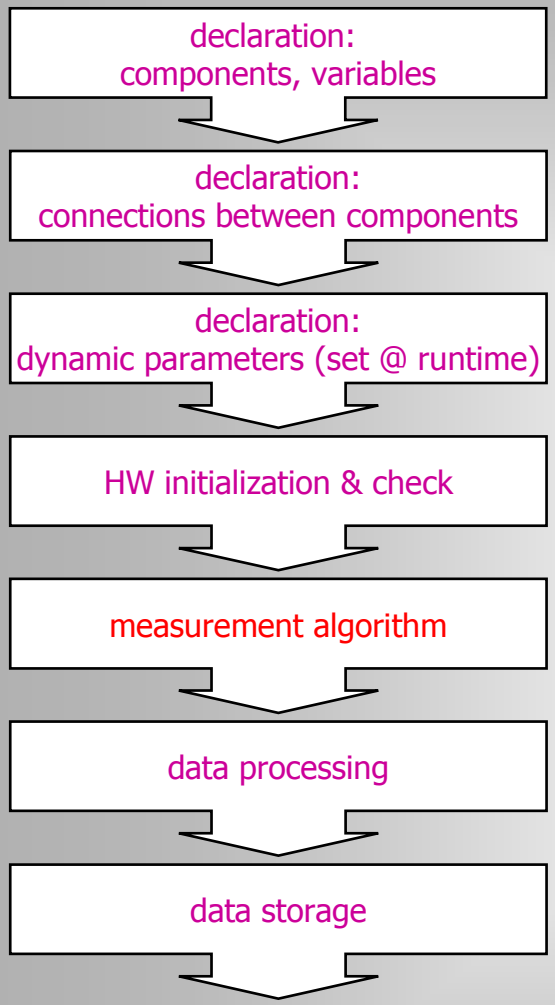
- the only industry-standard system (NI's TestStand) is not really suitable for our purpose;
- no system except EMS matches precisely our goal and is proven in our field.
- in any case, no commercial-grade, full time support can be guaranteed now or in the future.
  We are on our own.

Architecture overview



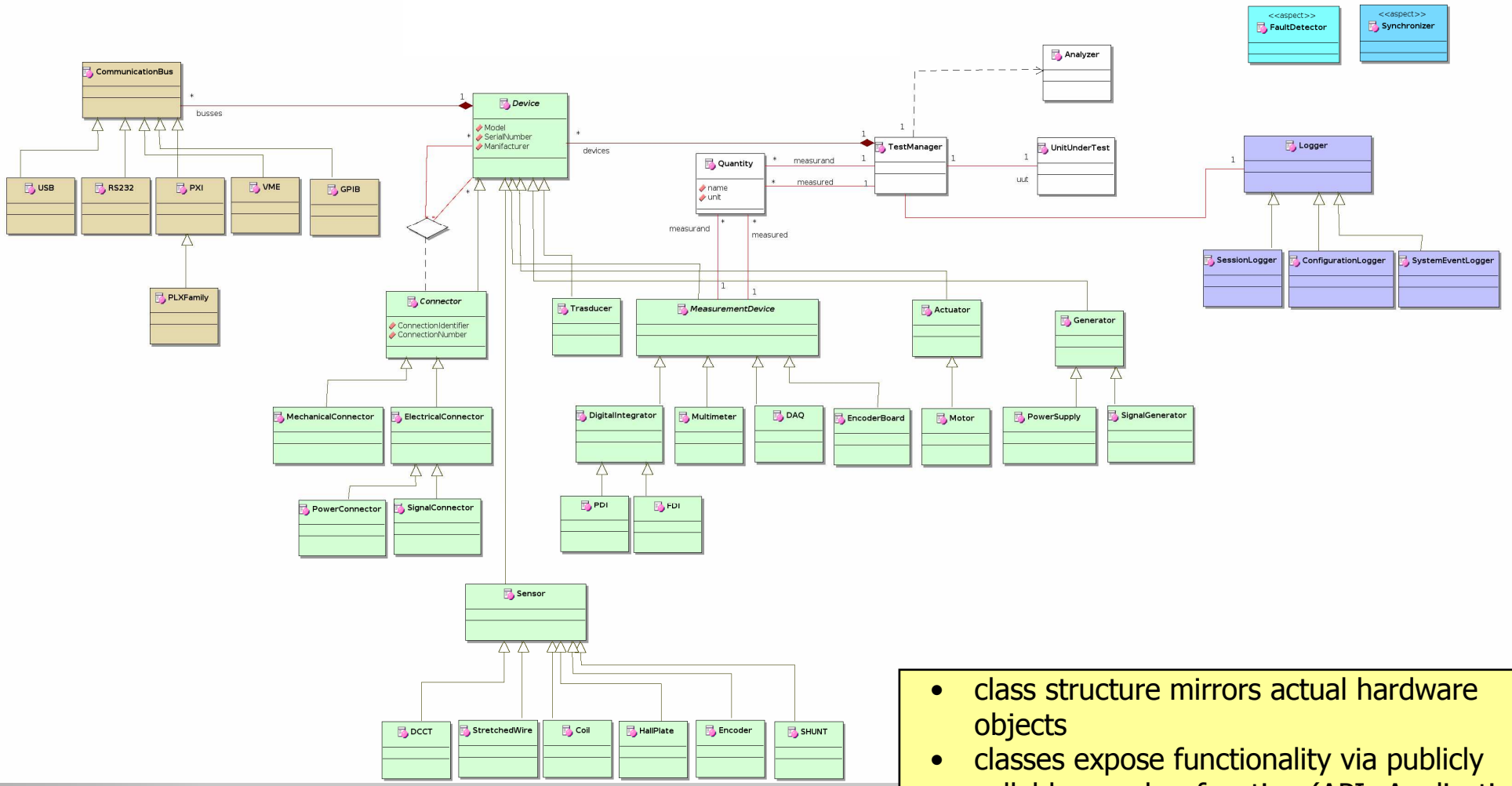
FFMM=component library + (optional) application generator

The script: typical structure



- essentially a simplified C++ program
- hardware components declared as *class* objects
- order and nature of operations can be chosen freely: a script can be
  - a list of parameter values
  - the physical description of a hardware setup
  - a sequence of test steps and checks
  - any combination of the above
- multiple script files can be cloned, modified and chained to manage effortlessly a multiplicity of similar test cases
- standard variables, programming flow constructs and calculations can be mixed freely with framework API calls (pre-defined component functions)
- interactivity can be programmed in (e.g.: measurement paused waiting for manual operations; parameter values input at run time; real-time visual feedback to operator)
- Two basic utilization modes:
  manual: the script is the basis for a conventional C++ source file, which the user compiles along with the class framework to generate an application (more flexibility, but user must be a programmer)
  automatic: the builder accepts directly the script as input and generates automatically the executable (the user is required to master only the measurement technique)

The class framework diagram



- class structure mirrors actual hardware objects
- classes expose functionality via publicly callable member function (API=Application Program Interface)

FFMM: using the framework at different level

required competences:	C/C++ language	FFMM architecture	FFMM API	Specific hardware	Magnetic measurement
Framework developer	✓ (advanced level: OOP, AOP, templates, concurrency...)	✓	✓	✓	
Driver developer	✓ (HW programming, quasi-real time)	✓	✓	✓	
Application developer	✓ (basic level: use the framework to compile an executable)	✓	✓		✓
Script developer	✓ syntax only		✓		✓
End User					✓

Aspect Oriented Programming

- **AOP** = software technique to increase the modularity of code

- "Aspects" are crosscutting concerns, i.e. routines appearing transversally (*scattered*) throughout the class hierarchy, cluttering the "core business" code with spurious (*tangled*) instructions

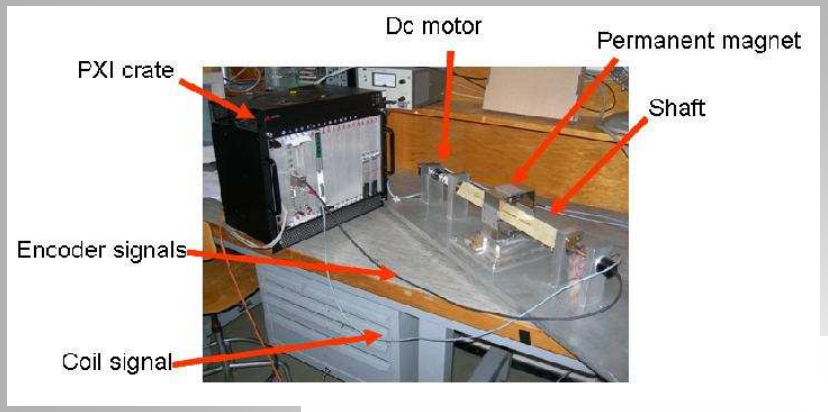
- Typical examples of Aspects:
  - code to detect and react to faults
  - code to write logs and tracing (debugging) information
  - code to synchronize hardware and/or software events

  these routines appear in multiple classes at different *join points* and can be effectively abstracted using AOP tools.

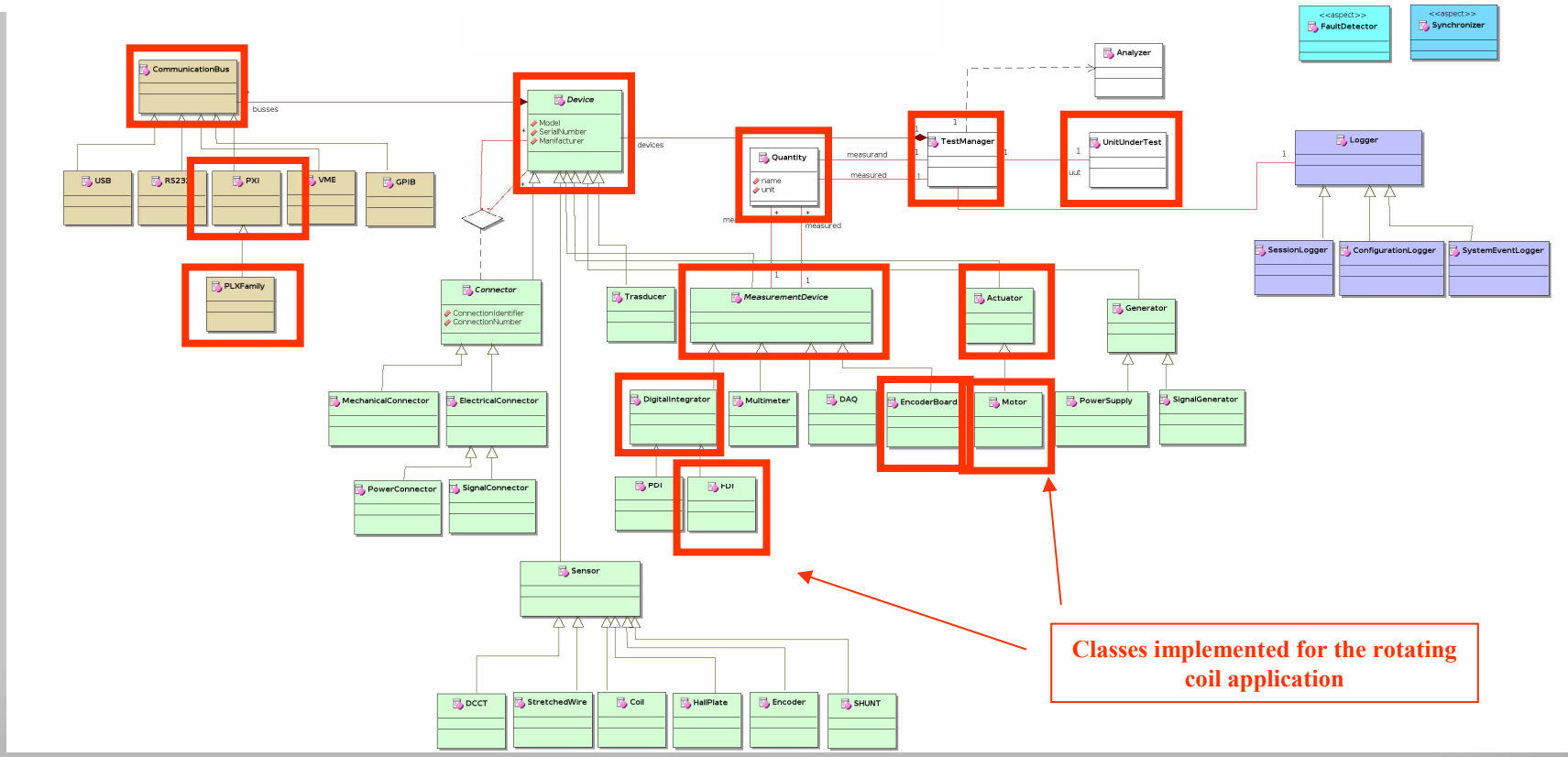
- Key advantages:
  - substantial reduction of source code length (similar to what can be obtained with templates)
  - improved code reliability, readability and maintainability
  - higher programmer productivity

see also: http://www4.informatik.uni-erlangen.de/DE/~lohmann/download/SDJ05_en.pdf

Example of implementation



- Demo test bench including the basic functionality needed for standard rotating coil measurfements
- System commissioning & soft debugging ongoing



Status & prospects

achieved:

- 1st iteration of framework library, including most typical components
- *DataLogger* aspect included, *Synchronizer* and *FaultDetector* designed
- Rotating coil demo (debugging ongoing)

next objectives:

- gradual increment of the framework library;
- finalize design of synchronization components in multi-threaded context
- develop the *Builder* framework component
- development of realistic applications for rotating coils, fluxmeters, SSW etc.
- design interactive aspects
- explore possible integration/synergy with Fermilab's EMS