Modular Experiment Control System packages for the CBM experiment

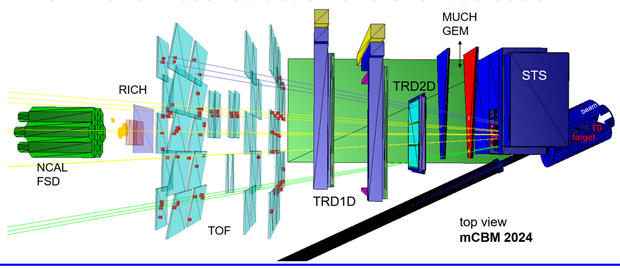
Pierre-Alain Loizeau, ***GSI Helmholtz Center for Heavy Ion Research, Darmstadt*****, for the CBM Collaboration**

The CBM experiment & its mCBM demonstrator

3D model of the CBM setup, ©GSI/FAIR, Zeitrausch

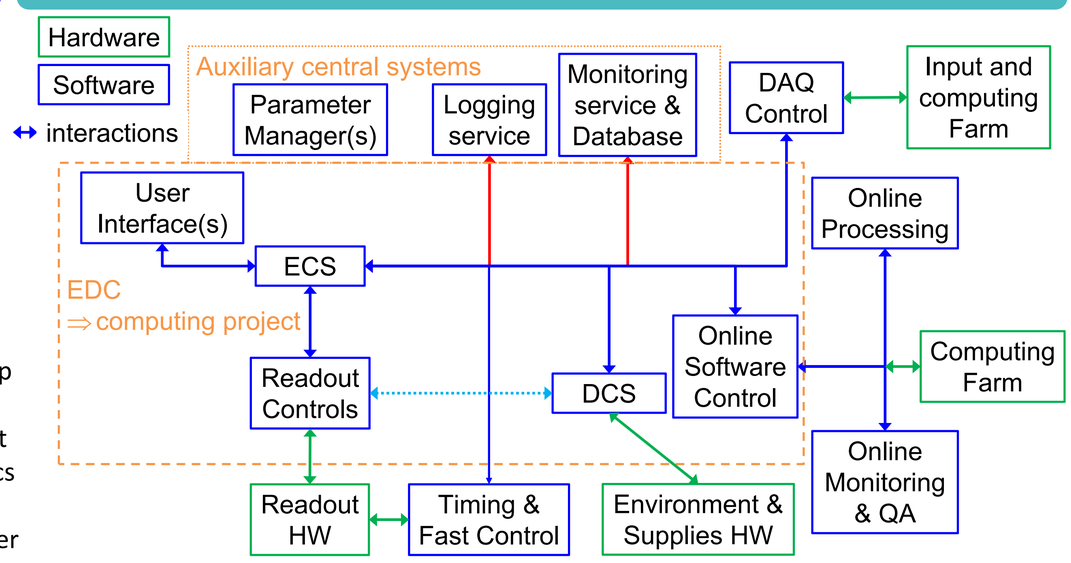
Specifications: **High statistics Heavy-Ions Experiment**

- Fixed target with the following beams (SIS100@FAIR):
  HI from 2 to 11 AGeV (Au, $\sqrt{S_{NN}}$≈4.9GeV), p from 3 to 30 GeV
- High-rate collision environment: 10^5 – 10^7/s (A+A), up to 10^9/s (p+A)
- Physics aperture : $2.5° \leq \Theta \leq 25°$
- Self-triggering front-end electronics with free streaming readout
- Online full reconstruction and event selection



mCBM:
- precursor setup
- SIS18@GSI
- integration test down to physics
- ≈1-5% CBM channel number

Experiment and Detector Controls computing project



Experiment Control System for CBM

→ **Missions:**
- Determination and propagation of experiment state
- Coordination of commands flow
- Propagation/resolution of configuration as tags and sub-tags
- Automatisation of the experiment configuration
- Archiving of all changes to state and configuration

→ **Specifications:**
- Initial version fully in Python
- Scalability from lab setups to full CBM through mCBM
- Flexibility to allow live addition/removal of systems
- Flexibility to allow parallel groups of systems
- Quality: high level of testing and variables typing coverage

ECS: packages

→ **Why?:**
- Separate common parts of code from application specific ones
- Different levels of QA: automatic code testing + typing check
- Flexibility in GUI choices
- Allow future upgrade of core transparently for applied layer

⇒ Example: Conversion of core to C++

⇒ Example: Changes due to dependencies upgrades

→ **QA level goals** (LOCs, flakeheaven/pytest/mypy)**:**
- All: clean lint, only some “same line whitespace” codes ignored
- Core: > 90 % tested, > 90 % typed
- Implementation: > 50-70 % tested, > 90 % typed
- GUI (QT): no CI test, > 50 % typed

Technical classes common to all elements of ECS → CBM ECS core — https://git.cbm.gsi.de/ecs/ecs-core v. 0.3.3

Specializations for each expected element of ECS, using mCBM as example → ECS implementation — https://git.cbm.gsi.de/ecs/mecs-implementation v. 0.1.6

Proof of concept GUI in QT ⇒Check if feature complete → GUI Demonstrator — https://git.cbm.gsi.de/ecs/ecs-gui-emulator v 0.1

ECS core

Design choices

Technical:
- ZeroMQ as inter-process communication layer
- Separate commands processing from state tracking and tag propagation

Functional:
- Only one command executed per agent at a time (sequential execution)
- Only one command emitted per agent at a time (reply wait)
- Buffered commands executed per priority and time of reception order
- Commands and replies routed from source to destination
- State updates broadcasted to all connected agents
- Configuration Tag resolution through request-reply pattern
- Set of “Global States” common to all agents
- “Local States” specific to each Agent for granularity within Global states
- Agent state defined as pair of <Global State, LocalState>
- Dynamic transitions table mapping two Agent States to function handle
- Two auxiliary states used to define recording mode:
  - Configuration state (e.g. “CleanTag”, “Manual”, …)
  - Configuration Stability state (e.g. “Stable”, “Tuning”, “Testing”)
- Agent-SubAgent relation:
  - Agent configuration tag resolves to list of SubAgents tags
  - State defined by default as lower common states of SubAgents
  - Agents track Subagents availability through “ping” special command
  - Optional termination of SubAgents upon Agent Termination
- One “top” agent as default master to avoid orphan agents

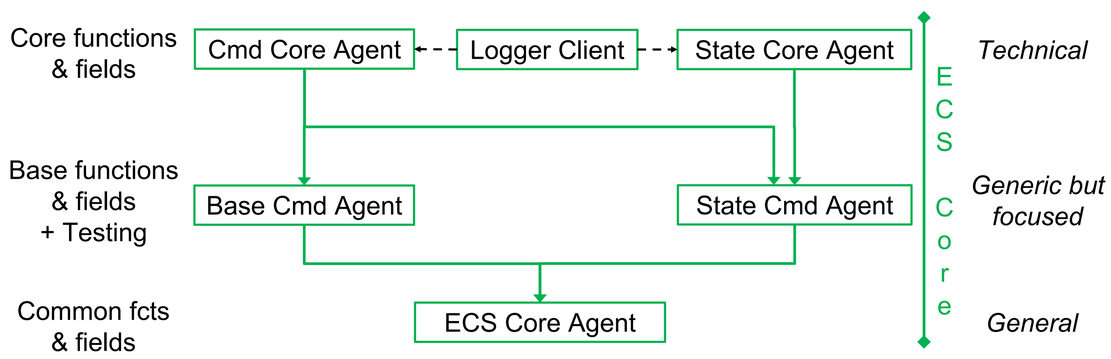
Recording mode examples

⇒ In each case, “Local” configuration/stability state resolved to “Global” one before determination

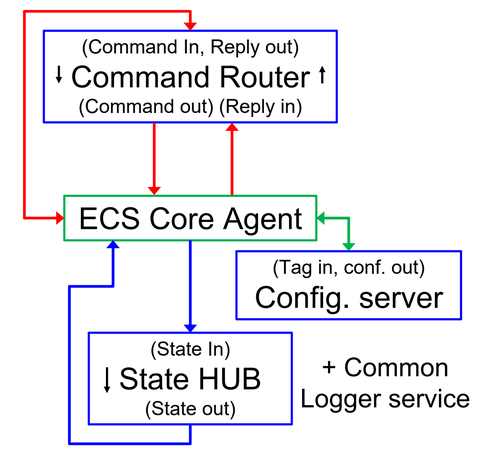
⇒ Recording mode = lower among agents

		Configuration state	
		Dirty/Manual	Clean
Stability state	Unstable	Technical	Unstable
	Stable	Dirty	Production

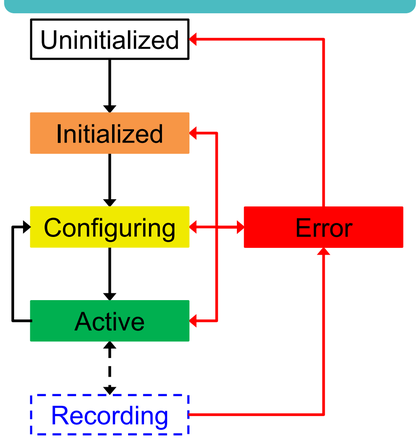
Classes links and purpose



Central Services



Default Global States



⇒ “Active” automatic if “Configuring” success

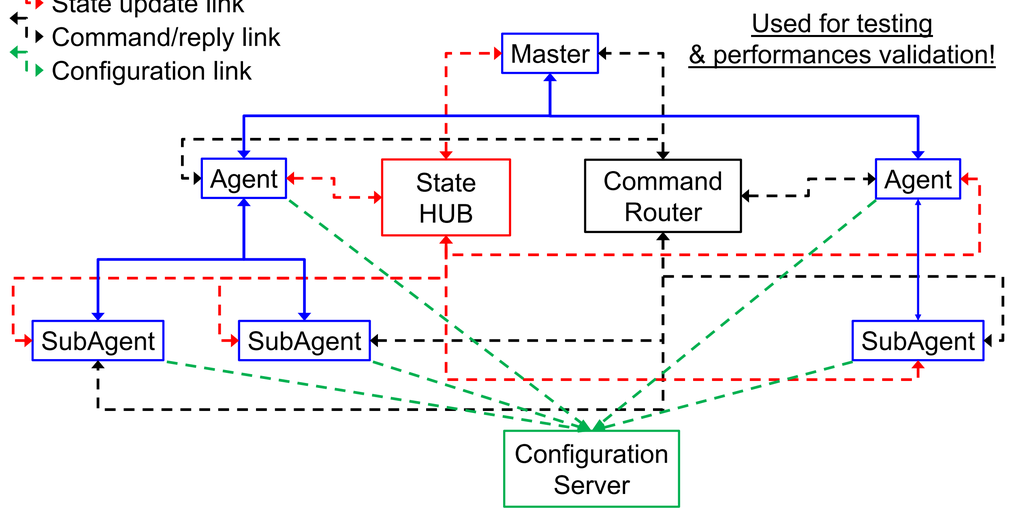
⇒ “Recording” only for specific agents

⇒ Re-configuration from “Active”

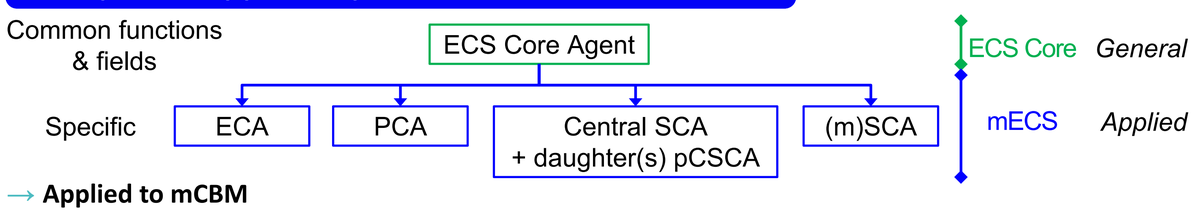
⇒ “Error” recovery except for “Recording”

⇒ Recovery to lower state if not possible

Instance links = test topology



ECS prototype implementation

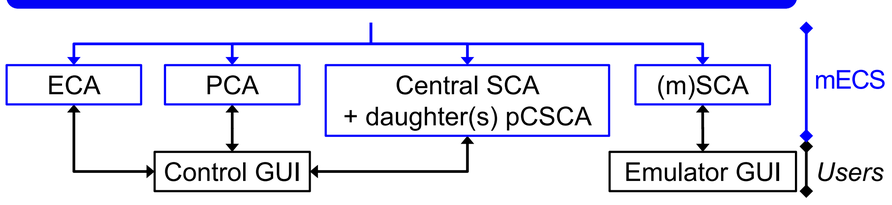


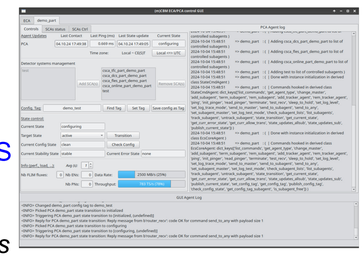
→ **Applied to mCBM**

→ **Agents:**
- 1 Experiment Control Agent = ECA, mandatory
- n Partition Control Agent(s) = PCA(s), dynamic
- m System Control Agent(s) = SCA(s), dynamic and defined by topology
- k Central System Control agents = CSCAs, static, e.g. DAQ, online farm, …
- n x k Partition CSCA(s) = pCSCA(s), dynamic

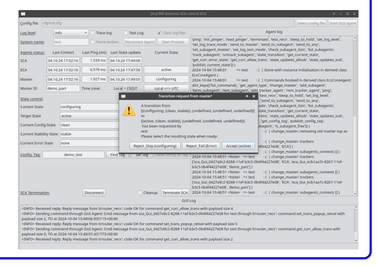
⇒ Share of CSCA resources assigned to PCA

Demonstrator/Emulator GUI





Example screenshots, live demo for details



→ **Key features:**
- QT6 with Pyside6
- One GUI for ECA, PCA and central systems

⇒ testing presence and compatibility of planned mECS features
- One GUI for standard SCAs = detectors with two modes

⇒ Emulator: Automatic transitions, test features completeness

⇒ Demonstrator: Pop-up on transition, replace System with user

ECS Project:

Achieved keypoints:
- First test versions of the core package with all 3 components and all QA goals
- Implementation package for mCBM with skeleton of all planned agent types
- GUIs allowing both to check usage flow (demonstrator) and help with operation of next mCBM beamtime campaign (emulator)

Future plans:
- Deployment in mCBM and 1st users: Feb. 2025
- Systems integration: 2025 – 2026 (mCBM setup as test-bench)
- Release version for CBM: latest 2027 (installation & commissioning)

p.-a.loizeau@gsi.de