State of the Celestial Sphere

Current State of Constellation Developments

Simon Spannagel, DESY

2nd EDDA Hackathon

13/05/2024

What do we expect from a “flexible DAQ system”?

- **Useful to control single laboratory setup**
  (e.g. radioactive source measurement)
- **Possibility to integrate multiple setups**
  (Detector DAQ, TCT laser control)
- **Lab supervision mode**
  (multiple setups monitored but control not ceded)
- **Synchronized operations**
  (test beam environment, coordinated start/stop, central control)
- **Scalability for small experiments**
  (many detectors, multiple data endpoints & monitors)

Introducing Constellation

Si

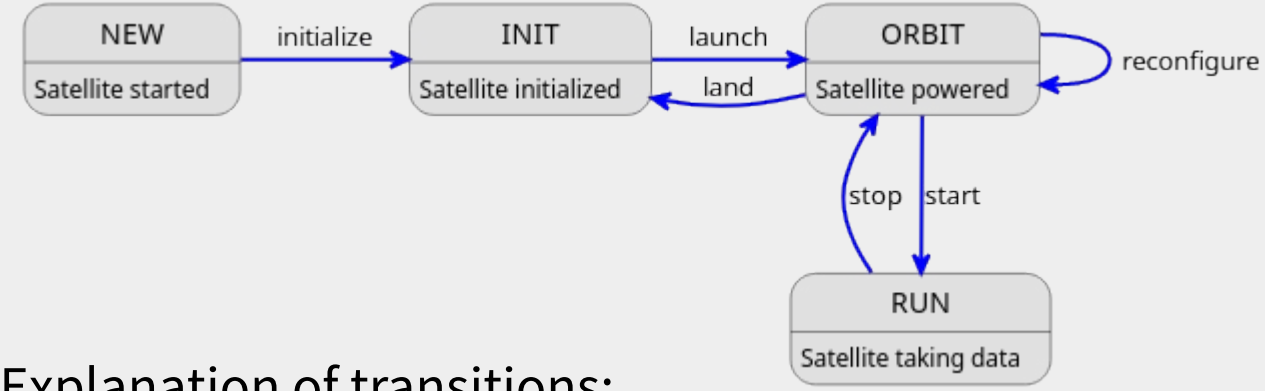
- Project goals:
  - **Easy** to use, easy & fast to integrate new systems
  - **Stable** operation, reliable error handling
  - **Flexible** and applicable for many use cases
- Solid foundation: well-defined communication protocols between components
- Participants are called **satellites** (eudaq: Producers/Collectors)
  - Operation is governed by a **finite state machine**
  - Satellites can operate **autonomously** without active user interface

Communication Protocols

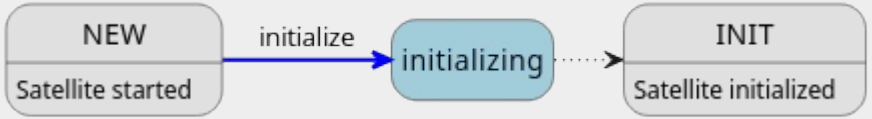
- All interaction is based on five protocols:
  - **CHIRP** (Constellation Host Identification and Reconnaissance Protocol)
    Automatic network discovery of satellites/services – will likely see a revision soon
  - **CHP** (Constellation Heartbeat Protocol)
    Exchange of heartbeats with FSM status, variable sender-defined intervals
  - **CSCP** (Constellation Satellite Control Protocol)
    The controller protocol which sends transition & other commands
  - **CMDP** (Constellation Monitoring Distribution Protocol)
    Monitoring and logging data broadcasting, only subscribed topics on the wire
  - **CDTP** (Constellation Data Transmission Protocol)
    Data transmission with extra features such as sequences, begin- & end-of-run

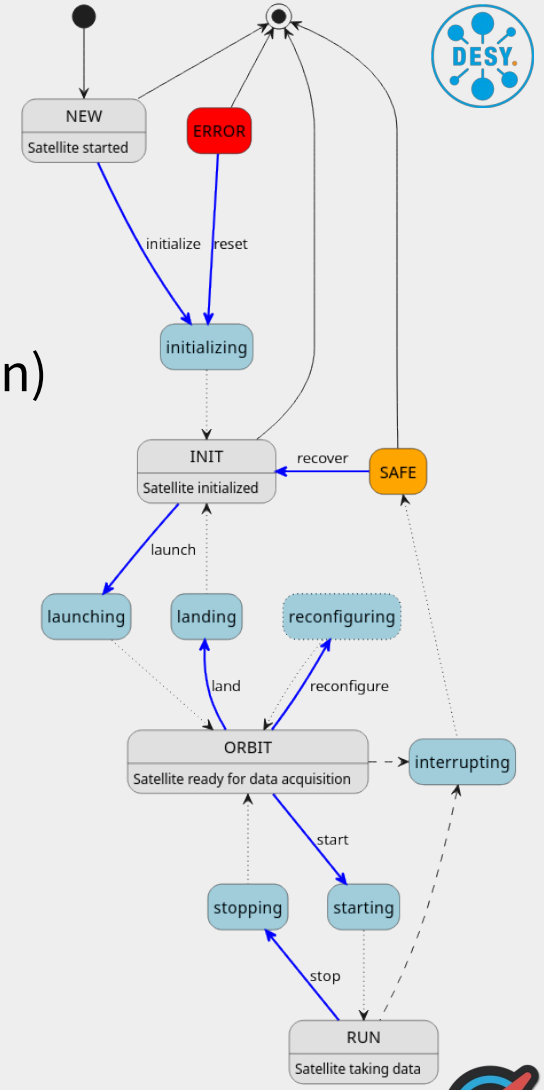
A Central Component: The FSM

- Many discussions, reconsideration & refinement went into this concept
- Diagram shows entire state machine (modulo error transition)
- Simplified diagrams for users:



- Explanation of transitions:





A Standard Interface for Satellites

- Converged on a standard set of commands any satellite has to understand

  Each satellite must be able to understand and answer to the following commands, and it must accept or provide the corresponding payloads. Verbs and commands are always transmitted as native strings, payloads are always encoded as MsgPack objects.

- Documented as
  Satellite Implementation Guidelines
- Forms the basis for controller classes and UIs

Command	payload	verb reply	payload reply
`get_name`	-	Name of the Satellite	-
`get_version`	-	Constellation version identifier string	-
`get_commands`	-	Acknowledgement	List of commands as MsgPack map/dictionary with command names as keys and descriptions as values
`get_state`	-	Current state (as string)	-
`get_status`	-	Current status	-
`get_config`	-	Acknowledgement	Satellite configuration as flat MsgPack map/dictionary
`get_run_id`	-	Current or last run identifier (as string)	-
`initialize`	Satellite configuration as flat MsgPack map/ dictionary	Acknowledgement	-
`launch`	-	Acknowledgement	-
`land`	-	Acknowledgement	-
`reconfigure`	Partial configuration as flat MsgPack map/ dictionary	Acknowledgement	-
`start`	Run identifier as MsgPack string	Acknowledgement	-
`stop`	-	Acknowledgement	-
`shutdown`	-	Acknowledgement	-

Extending the Satellite Interface (C++)

- Satellites can register special commands which can be called via CSCP
- Commands can have (almost) arbitrary arguments & return values
- They can be limited to specific states of the FSM
- Example:

```
int MySatellite::get_channel_reading(int channel) {
  auto value = device_->read_channel(channel);
  return value / 10;
}

MySatellite::MySatellite(std::string_view type, std::string_view name) : Satellite(type, name) {
    register_command("get_channel_reading",
                     "This command reads the current device value from the channel number provided as argument. Since this"
                     "will reset the corresponding channel, this can only be done before the run has started.",
                     {State::NEW, State::INIT, State::ORBIT},
                     &MySatellite::get_channel_reading,
                     this);
}
```

Current Project Status

- Started with 1st EDDA Hackathon in Lund, October/November 2023
- Most basic components are implemented by now
  - Communication protocols / libraries
  - Satellite state machine and user interface class
  - Controller interfaces, data transmission, logging…
- Parallel development of two implementations with continuous feedback
  - Stephan, Simon - C++ implementation
  - Hanno, Joel, Linus - Python implementation
- Infrastructure set up, development in full swing @ DESY Gitlab repositories

Repository Status

Project information

The Autonomous Control and Data Acquisition System for Dynamic Experimental Setups

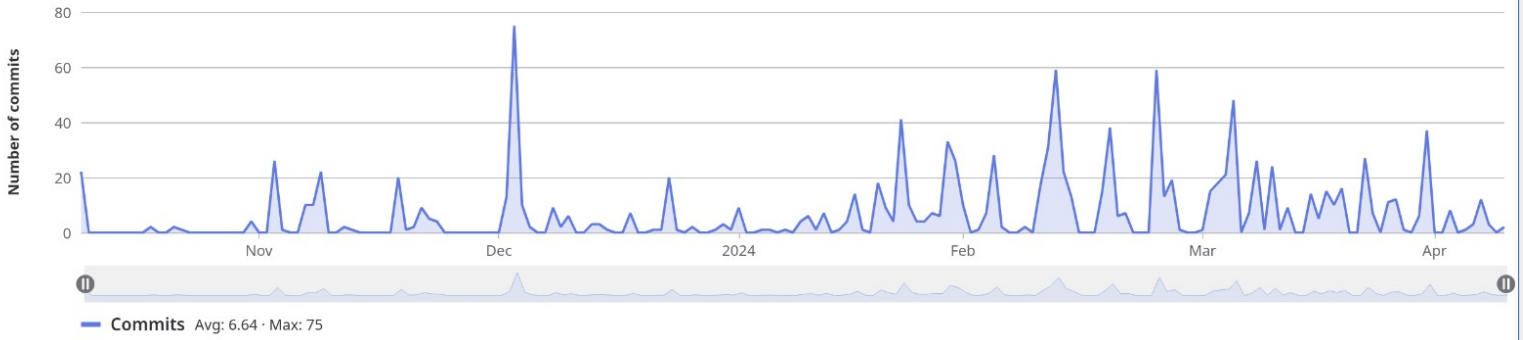
Latest Release none | License EUPL 1.2
REUSE compliant | coverage 81.70%
pipeline failed

- 1,416 Commits
- 23 Branches
- 0 Tags
- 7.7 GiB Project Storage

- README
- European Union Public License 1.2
- CI/CD configuration
- + Configure Integrations

Commits to main

Excluding merge commits. Limited to 6,000 commits.



Constellation / **Merge requests**

Open 17 | Merged 163 | Closed 10 | All 190

Recent searches | Search or filter results...

C++: Implement satellite shutdown command

Website is under construction

Constellation

Autonomous Control and Data Acquisition System

Constellation is a control and data acquisition system for small-scale experiments and experimental setup with volatile and dynamic constituents such as testbeam environments or laboratory test stands.

Get Started | Concepts

See API Reference →

Autonomous

Constellation operates without a central server, satellites exchange heartbeats to keep in touch.

Flexible

Automatic network discovery of satellites make it easy to add and remove satellites on the fly.

Fast Integration

The finite state machine and satellite interface are designed for fast and easy integration of devices.

Robust

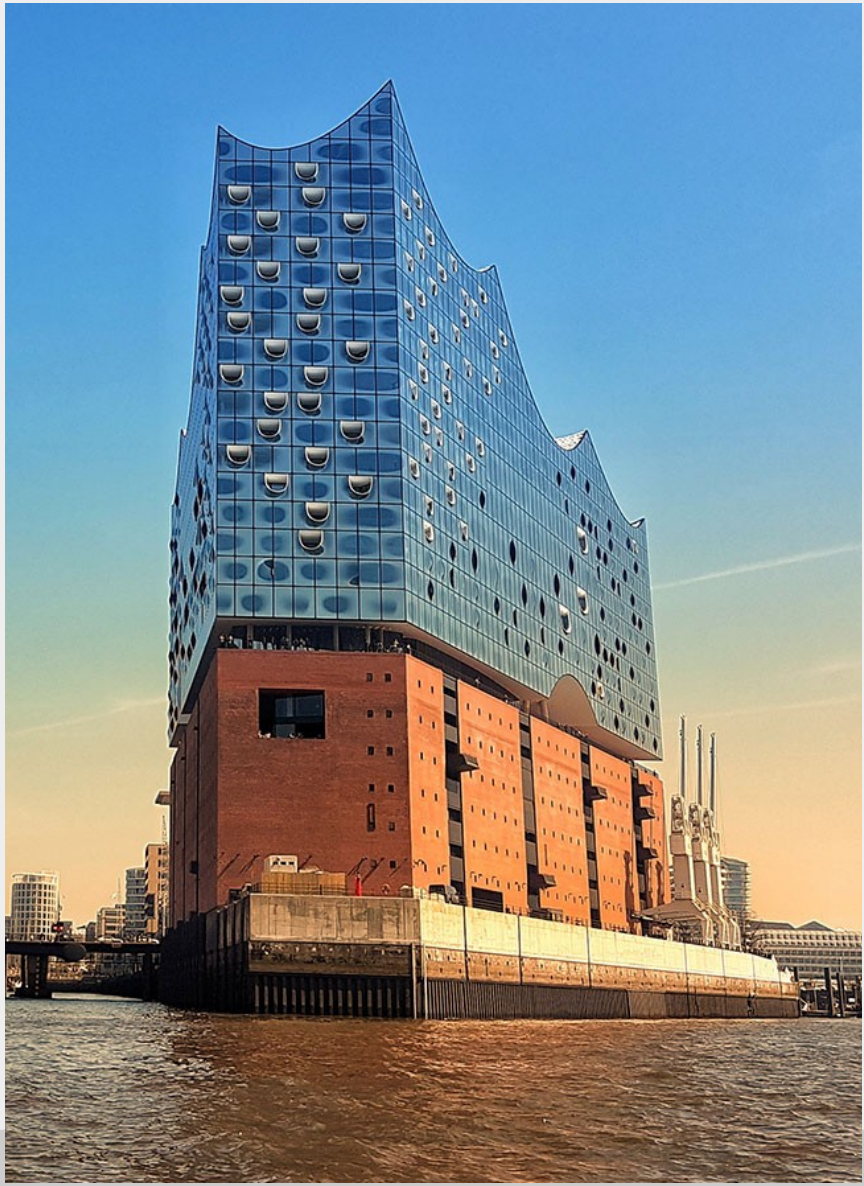
Constellation is based on widely adopted networking libraries such as ZMQ and MsgPack.


Low-Level
& Implementation
Chin-Chia
Satellite
Configuration
Hanno
FSM
calls user functions
Satellite Module
• load
• unload
• reload
• launch
• land
• reconfigure
. . . .
cwtlh::
Satellite
Lene Kristian
Commander
(aka Controller)
Chin-Chia
UI Concepts
Annika
Listener
@HHHackathon
Log Service
spdlog?
Heartbeat Service
HeartbeatCheck Service
Data Sender
DataReceiver
CHIRP Service
Stephan [Hanno]
Message
Log Message
Command Message
Heartbeat Message
Data Message
Stephan [Hanno]
CHIRP Message
documentation of share
Separate C++ & Python implementations of full stack?
TESTS!
CI!
HOSTING?
system)
2N goal:
current knowledge is documented

Goals for the 2nd EDDA Hackathon

- Documentation
  - ...writing documentation (how-to, tutorials, concepts)
  - ...complementing & polishing website (news, about, …)

- Graphical User Interfaces
  - ...graphical interfaces for logging
  - ...graphical interfaces for controlling
  - ...graphical interfaces for monitoring



Documentation

- Website generation up & running
- Documentation structure mostly laid out - follows https://diataxis.fr/
  - **Tutorials:** Teach how to use Constellation, little background explanations
  - **Concepts:** Write up of the thoughts behind the framework structure, functionality, ...
  - **How-To Guides:** Concise answers on how to achieve a specific goal
  - **Reference:** Detailed documentation one can consult to get info on inner workings
- Good examples for this division:
  - Gatsby: https://www.gatsbyjs.com/docs/
  - Read The Docs: https://docs.readthedocs.io/en/stable/index.html

something familiar…
…and working!

Graphical User Interfaces

- Controller: Some work done already with Qt5 (euRun):
  - Probably should be redone completely, more of a demonstrator
  - Base controller class exists but likely has to be extended depending on needs
  - Configuration file parsing missing entirely so far in C++
- Logging: The prime example for a listener component
  - No base class available yet, no layout suggestions yet
- Monitoring: We will not get around Grafana
  - Suggested assumption: an Influx Database and Grafana server are running
  - We need to feed them from a satellite (or listener)

DESY.