TUPV009

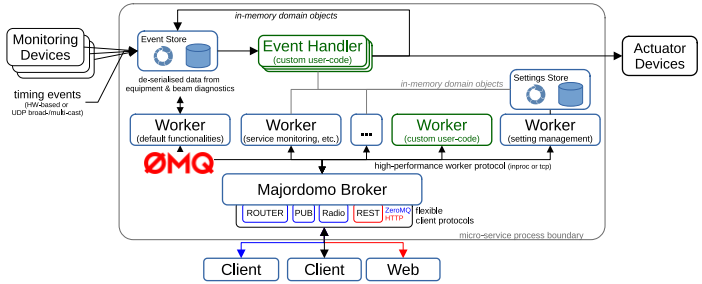
ICALEPCS 2021

A MODULAR OPEN COMMON MIDDLE-WARE LIBRARY FOR EQUIPMENT- AND BEAM-BASED CONTROL SYSTEMS AT FAIR

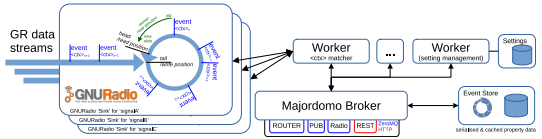
Ralph J. Steinhagen, H. Bräuning, D. Day, A. Krimm, T. Milosic, D. Ondreka, A. Schwinn
GSI Helmholtzzentrum, Darmstadt, Germany

Goals and Architecture

- multi language modular event-driven microservice middle-ware framework based on modern language standards
- backwards compatibility through implementations of established protocols

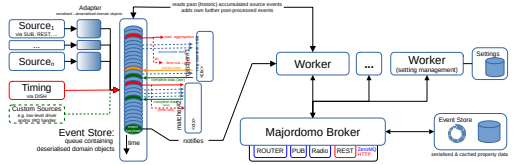


Data Aggregation and Event Processing



Aggregation and processing of data is performed either using GNURadio based continuous streams or event based processing and aggregation.

The majordomo framework facilitates event processing using predefined or custom workers defined by handler-callbacks and input/output domain-objects.



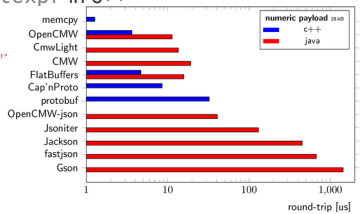
High-performance reflection-based Serialiser

- no interface description language (IDL), instead using reflection based approach
- self-documenting serialisation format and data structures
- integration of mp-units physical units library to prevent scaling or unit errors
- high performance using `Unsafe` in java and `constexpr` in c++

```
struct CppDomainObjectExample {
  Annotated<float, thermodynamic_temperature<kelvin>, "device specific temperature">      temperature     = 23.2F;
  Annotated<float, electric_current<ampere>, "this is the current from ...">              current         = 42.f;
  Annotated<float, energy<electronvolt>, "SIS18 energy at injection before being captured"> injectionEnergy = 8.44e6F;
  std::string                                                                             notAnnotated    = "Hello World!"
  // [...]
};
// refl-cpp-based: targeted and becomes obsolete with the next C++ standard
ENABLE_REFLECTION_FOR(CppDomainObjectExample, temperature, current, injectionEnergy, notAnnotated)

public class JavaDomainObjectExample {
  @MetaInfo(description = "device specific temperature", unit = "K")
  public float   temperature     = 23.2f;
  @MetaInfo(description = "this is the current from ...", unit = "A")
  public float   current         = 42.f;
  @MetaInfo(description = "SIS18 energy at injection before being captured", unit = "eV")
  public float   injectionEnergy = 8.44e6f;
  public String  notAnnotated    = "Hello World!";
  // [...]
}
```

code example: c++ and Java domain objects.



benchmark results: serialisation and de-serialisation of domain-objects using different serialisers.

Open and Lean Development

- development as open source on public GitHub accepting PRs

LGPL Free Software — Free as in Freedom

- test suite, CI/CD, code quality tooling to allow high quality contributions
- small code footprint: e.g. serialiser (without tests)
  - Java 8431 LOCs
  - C++ 1344 LOCs
- low number of dependencies by leveraging standard library functionality

References:

[1] A. Krimm and R. Steinhagen, "FAIR Common Specification - Modular Open Common Middle-Ware Library for Equipment- and Beam-Based Control Systems of the FAIR Accelerators," FAIR, Tech. Rep., 2020. Available: https://edms.cern.ch/document/2444348

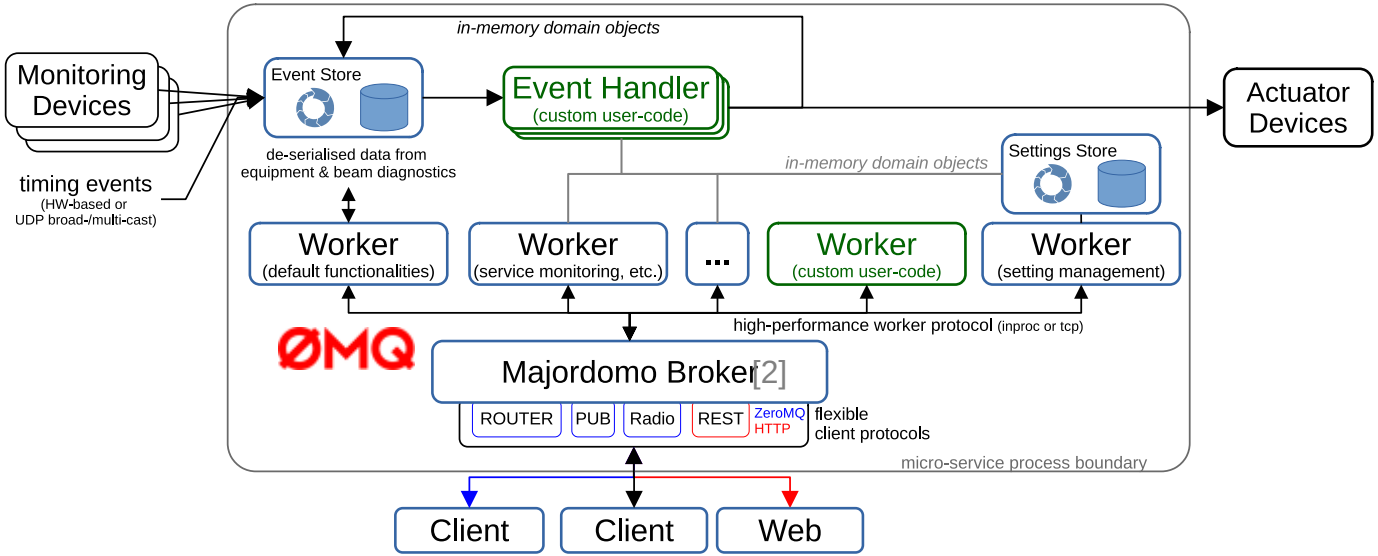
[2] P. Hintjens, "Majordomo Protocol RFC," The ZeroMQ Project, Tech. Rep., 2012. Available: https://rfc.zeromq.org/spec/18/

[3] R. J. Steinhagen *et al*, "Generic Digitization of Analog Signals at FAIR – First Prototype Results at GSI," in Proc. IPAC'19, Melbourne, Australia, 19-24 May 2019, https://doi.org/10.18429/JACoW-IPAC2019-WEPGW021

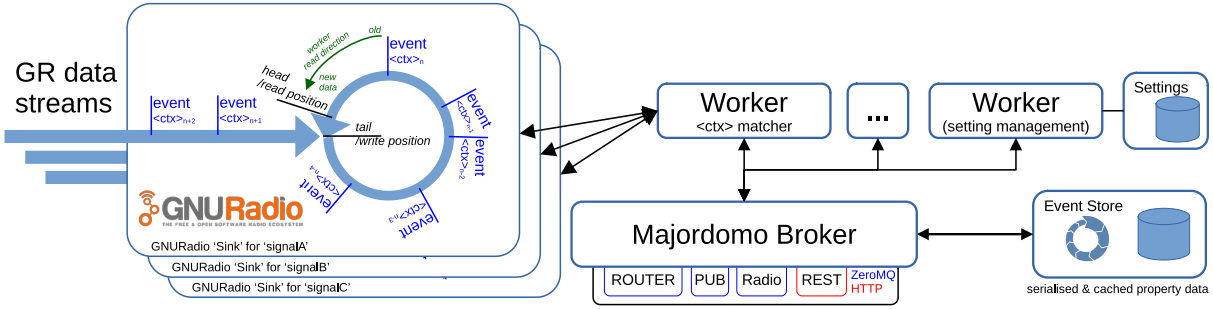
[4] M. Pusz. (2020, Jan) P1935R2 – A C++ Approach to Physical Units. [Online]. Available:http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p1935r2.html

Goals and Architecture

- multi language modular event-driven microservice middle-ware framework based on modern language standards [1]
- backwards compatibility through implementations of established protocols

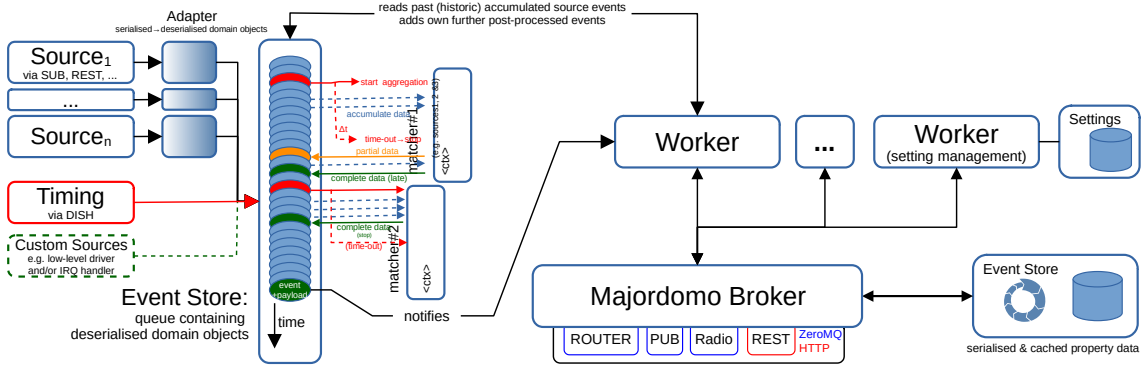


Data Aggregation and Event Processing



Aggregation and processing of data is performed either using GNURadio based continuous streams[3] or event based processing and aggregation.

The majordomo framework facilitates event processing using predefined or custom workers defined by handler-callbacks and input/output domain-objects.



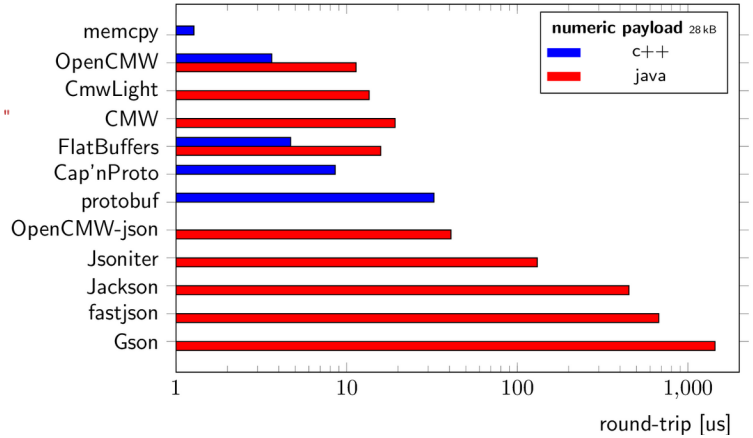
High-performance reflection-based Serialiser

- no interface description language (IDL), instead using reflection based approach
- self-documenting serialisation format and data structures
- integration of mp-units physical units library [4] to prevent scaling or unit errors
- high performance using `Unsafe` in java and `constexpr` in c++

```cpp
struct CppDomainObjectExample {
  Annotated<float, thermodynamic_temperature<kelvin>, "device specific temperature">        temperature     = 23.2F;
  Annotated<float, electric_current<ampere>, "this is the current from ...">                current         = 42.F;
  Annotated<float, energy<electronvolt>, "SIS18 energy at injection before being captured"> injectionEnergy = 8.44e6F;
  std::string                                                                               notAnnotated    = "Hello World!"
  // [..]
};
// refl-cpp-based: targeted and becomes obsolete with the next C++ standard
ENABLE_REFLECTION_FOR(CppDomainObjectExample, temperature, current, injectionEnergy, notAnnotated)
```

```java
public class JavaDomainObjectExample {
  @MetaInfo(description = "device specific temperature", unit = "K")
  public float   temperature     = 23.2f;
  @MetaInfo(description = "this is the current from ...", unit = "A")
  public float   current         = 42.f;
  @MetaInfo(description = "SIS18 energy at injection before being captured", unit = "eV")
  public float   injectionEnergy = 8.44e6f;
  public String  notAnnotated    = "Hello World!";
  // [..]
}
```

code example: c++ and Java domain objects.



benchmark results: serialisation and de-serialisation of domain-objects using different serialisers.

Open and Lean Development

- development as open source on public GitHub accepting PRs

LGPL Free Software
Free as in Freedom

- test suite, CI/CD, code quality tooling to allow high quality contributions
- small code footprint: e.g. serialiser (without tests)
  - Java 8431 LOCs
  - C++ 1344 LOCs
- low number of dependencies by leveraging standard library functionality

References:

[1] A. Krimm and R. Steinhagen, “FAIR Common Specification - Modular Open Common Middle-Ware Library for Equipment- and Beam-Based Control Systems of the FAIR Accelerators,” FAIR, Tech. Rep., 2020. Available: https://edms.cern.ch/document/2444348

[2] P. Hintjens, “Majordomo Protocol RFC,” The ZeroMQ Project, Tech. Rep., 2012. Available: https://rfc.zeromq.org/spec/18/

[3] R. J. Steinhagen, R. Bär, A. Franke, A. Krimm, K. Lüghausen, D. Ondreka, A. Schwinn, and M. Thieme, “Generic Digitization of Analog Signals at FAIR – First Prototype Results at GSI,” in Proc. IPAC’19, Melbourne, Australia, 19-24 May 2019, https://doi.org/10.18429/JACoW-IPAC2019-WEPGW021

[4] M. Pusz. (2020, Jan) P1935R2 – A C++ Approach to Physical Units. [Online]. Available:http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p1935r2.html