EPICS (pvAccess) Transport Layer for the ITER Real-Time Framework (RTF)

Anze Zagar
ITER Organization

Disclaimer: The views and opinions expressed herein do not necessarily reflect those of the ITER Organization

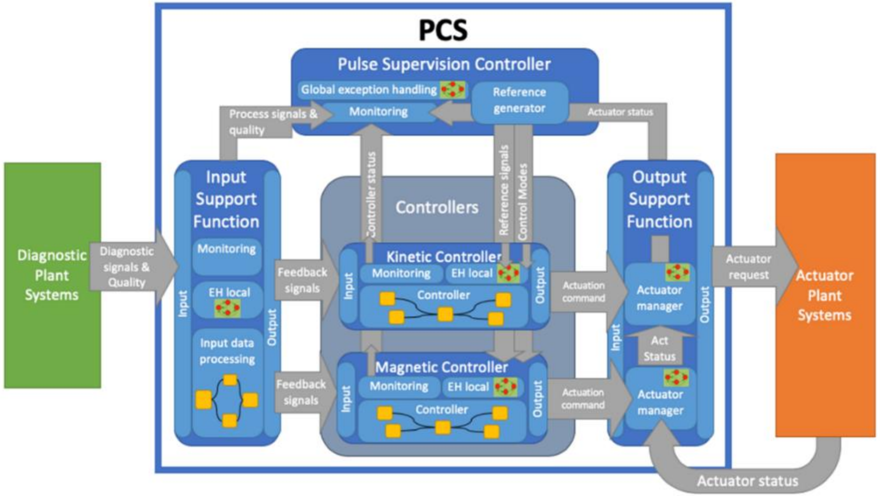
Operation Applications Overview

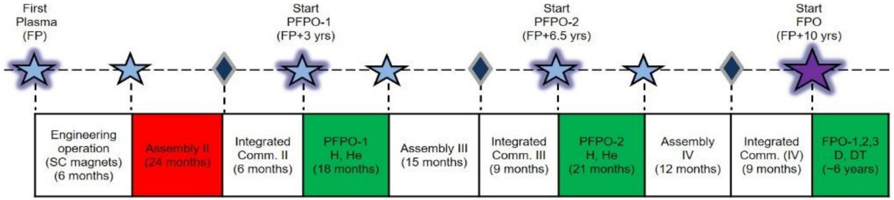
- Pulse Schedule Preparation System (PSPS)
  - Definition of pulse schedules to describe automated operation, with or without plasma*
- Supervisory System (SUP)
  - Coordinate system state across plant systems*
- Plasma Control System (PCS)
  - Deterministic control during pulses using the Real Time Framework (RTF)
- Data Handling (DA)
  - Archiving of POS, SDN data; data visualization
- Remote Participation (RP)
  - Virtual Collaboration tools for ITER members to participate in experiments**

*Bob Gunion Wed 12:10 – ITER Operation Applications: Status and Future Development
**Leonid Lobes Thu 09:35 – Study on EPICS Communication over Long Distance

Plasma Control System

- Final design for first plasma completed in 2020
- Prototype implementation in PCS simulation platform (PCSSP) in Simulink®
- Now being implemented in RTF





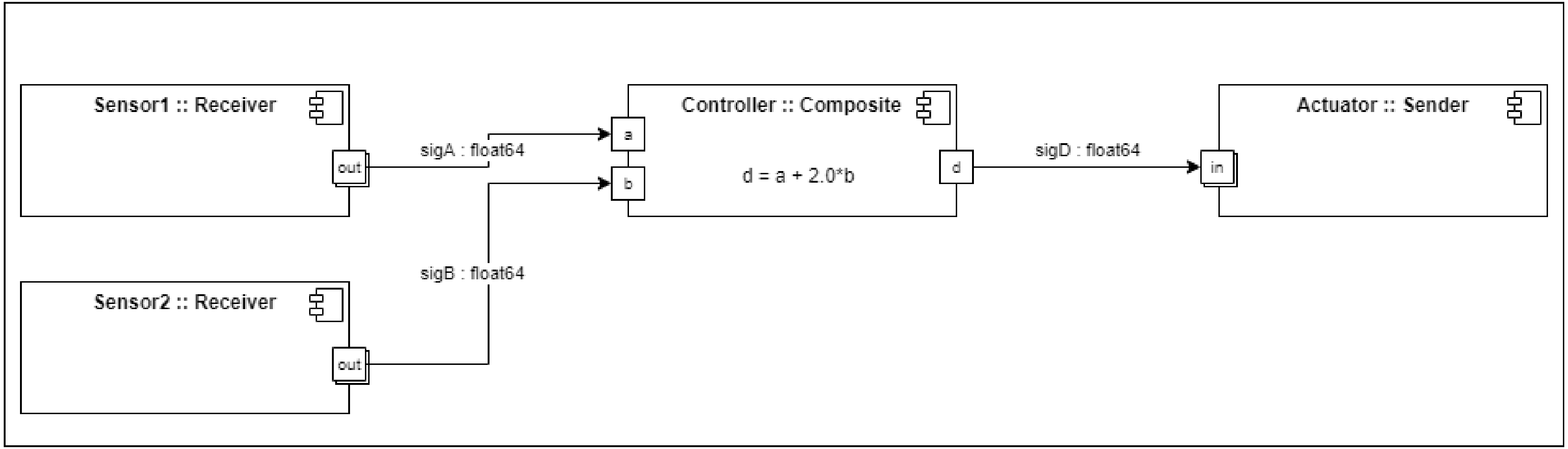
Real-Time Framework

- Software framework for building real-time applications
- Applications at ITER: PCS and plasma diagnostics
- C++14, RHEL MRG-R real-time kernel
- Optimized for real-time performance (minimizing response time and jitter):
  - each control loop in its own isolated CPU thread
  - synchronization done using busy-wait and atomic operations (no mutexes or semaphores)
  - no runtime memory management (memory pre-allocated at initialization time)
  - logging and archiving delegated outside real-time threads using lambda expressions and lambda queues
  - no redundant memory copying (e.g. for in-thread communication)

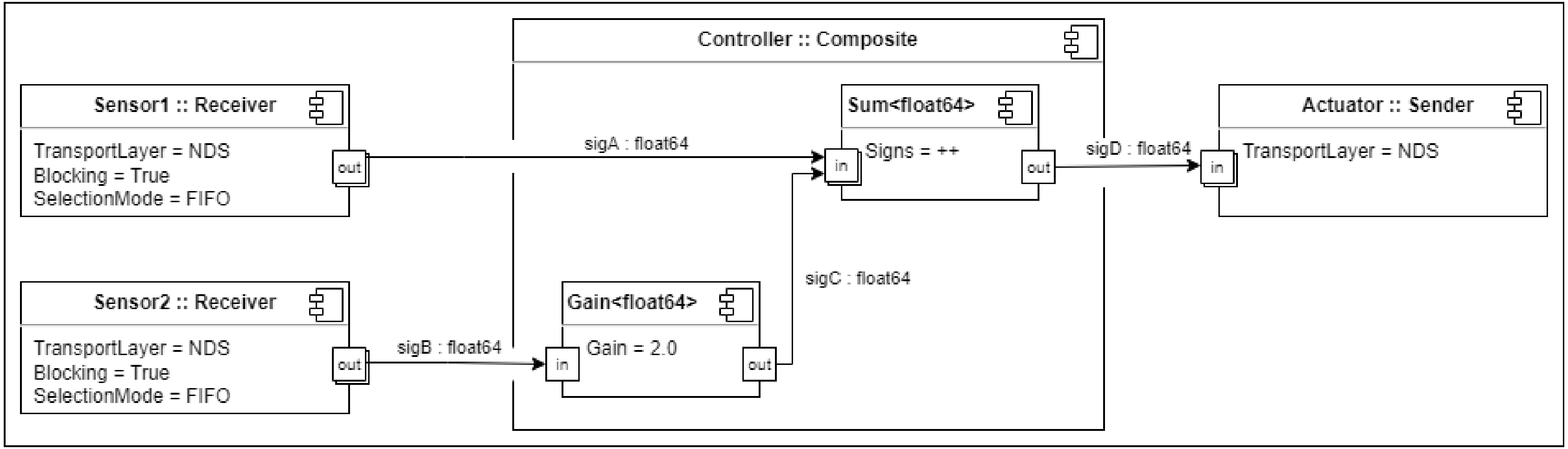
RTF Application

- Instantiated from XML-based configuration files at the initialization time
- Consists of Function Blocks (FB) configurable through parameters and inter-connected with signals and (asynchronous) events
- Allows nesting of FBs in Composite FBs
- Defines signals and signal groups (internal or external)

RTF Application

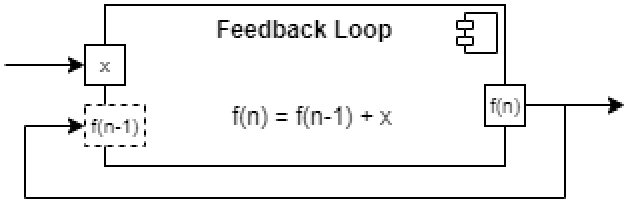


RTF Application



Closed-Loop Transfer Function

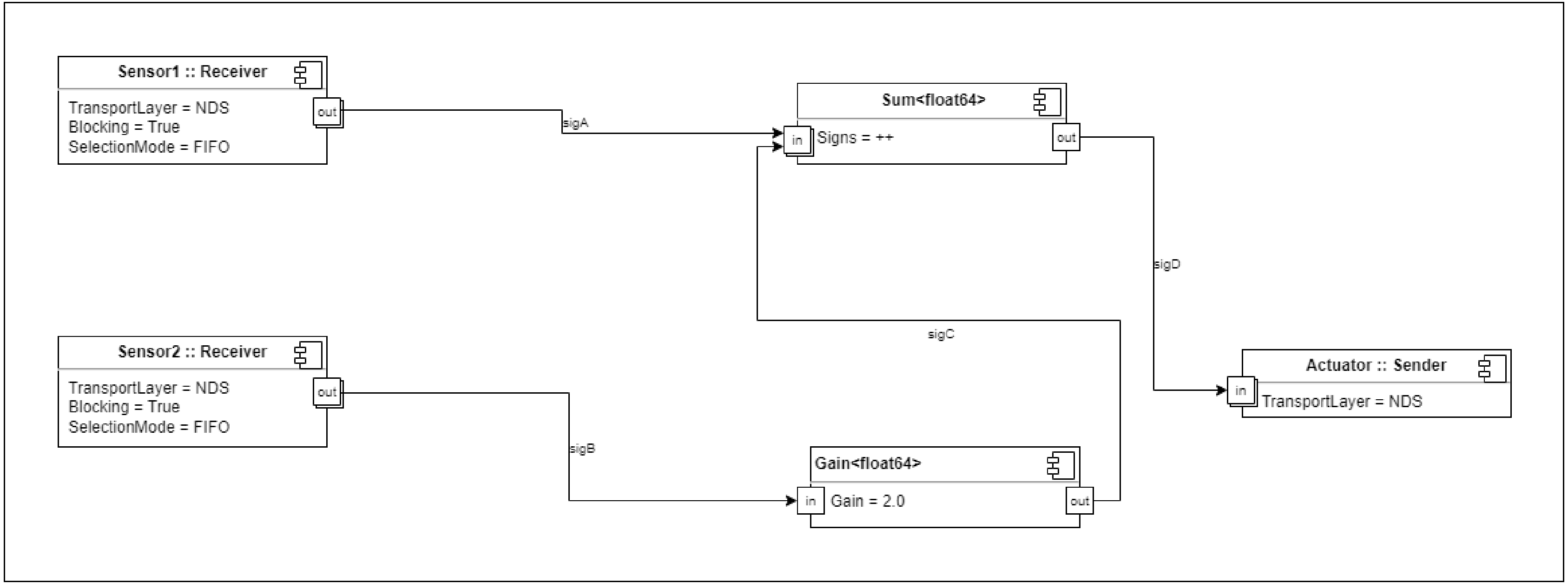
- Closed-loop transfer functions can be implemented by configuring an input port as “Required=False”



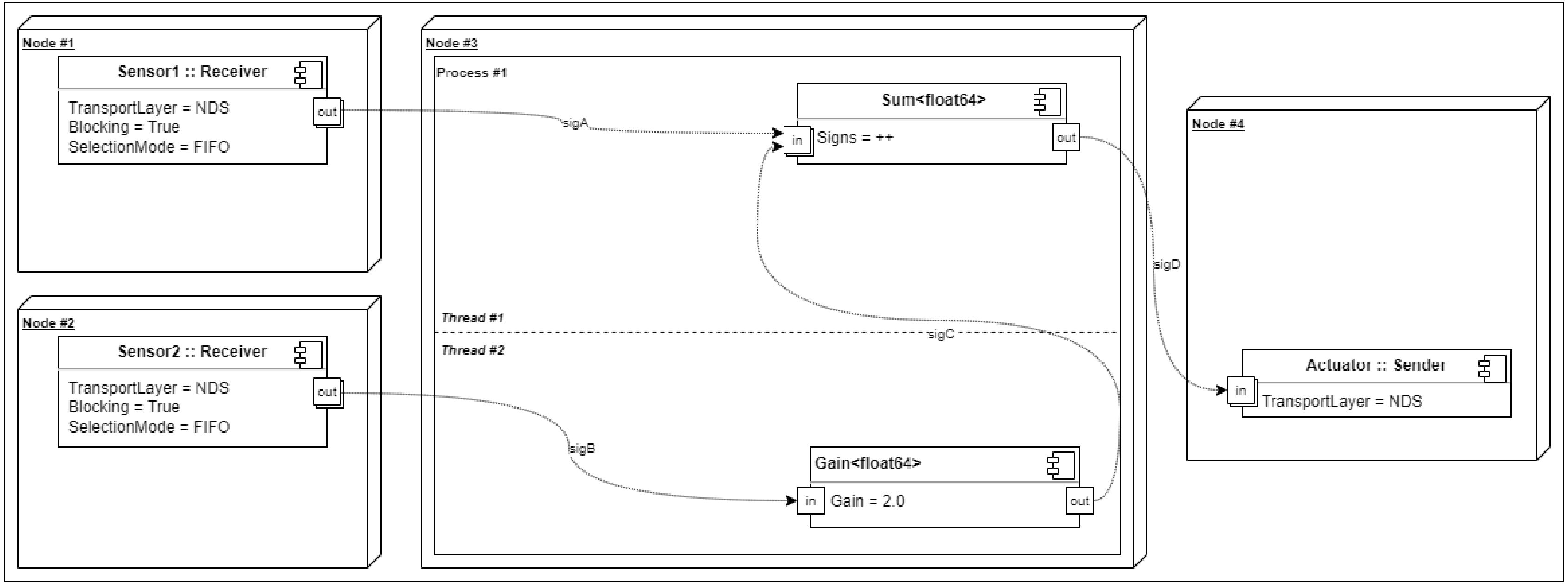
RTF Application Deployment

- Configured separately from RTF Application in deployment configuration XMLs
- Deployment can be local or distributed
- Within the same control loop/thread data is exchanged by sharing the same memory allocation
- Inter-thread communication requires gateways implemented with SPMC real-time queues [Enqueuer/Dequeuer FBs]
- Inter-process/node communication requires gateways implemented with transport layers [Sender/Receiver FBs]
- Gateways are implicitly inserted by the framework

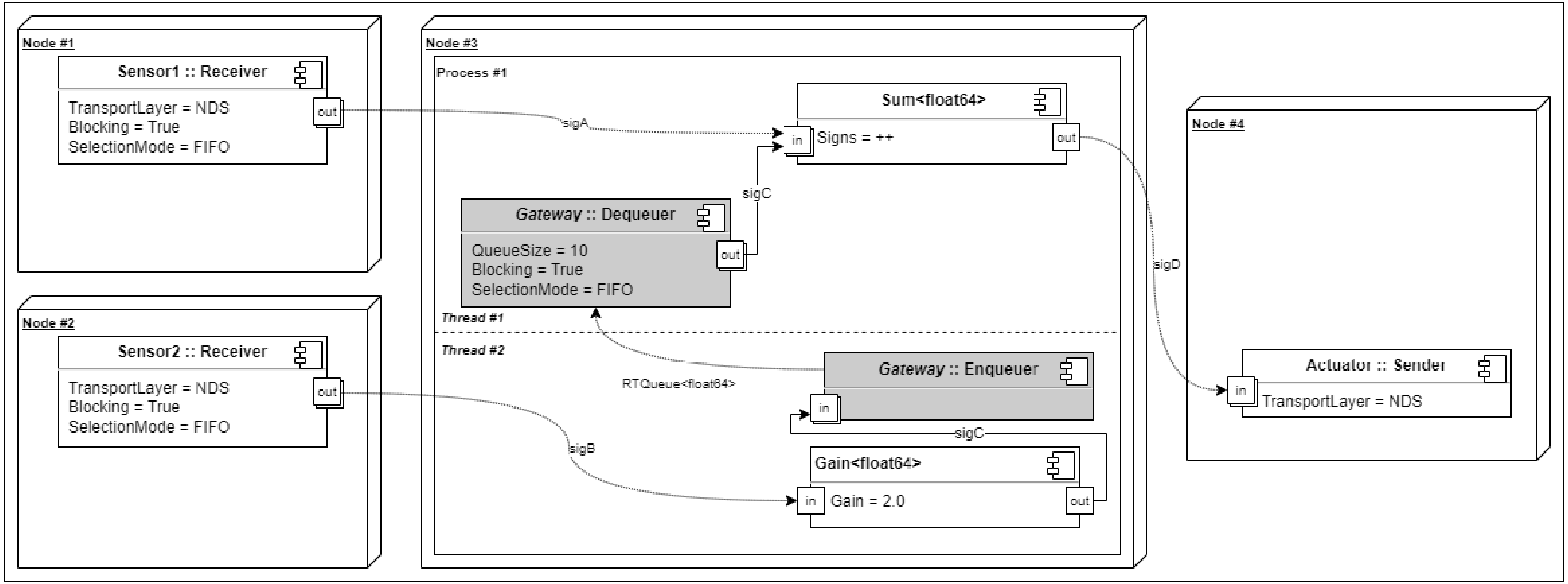
RTF Application Deployment



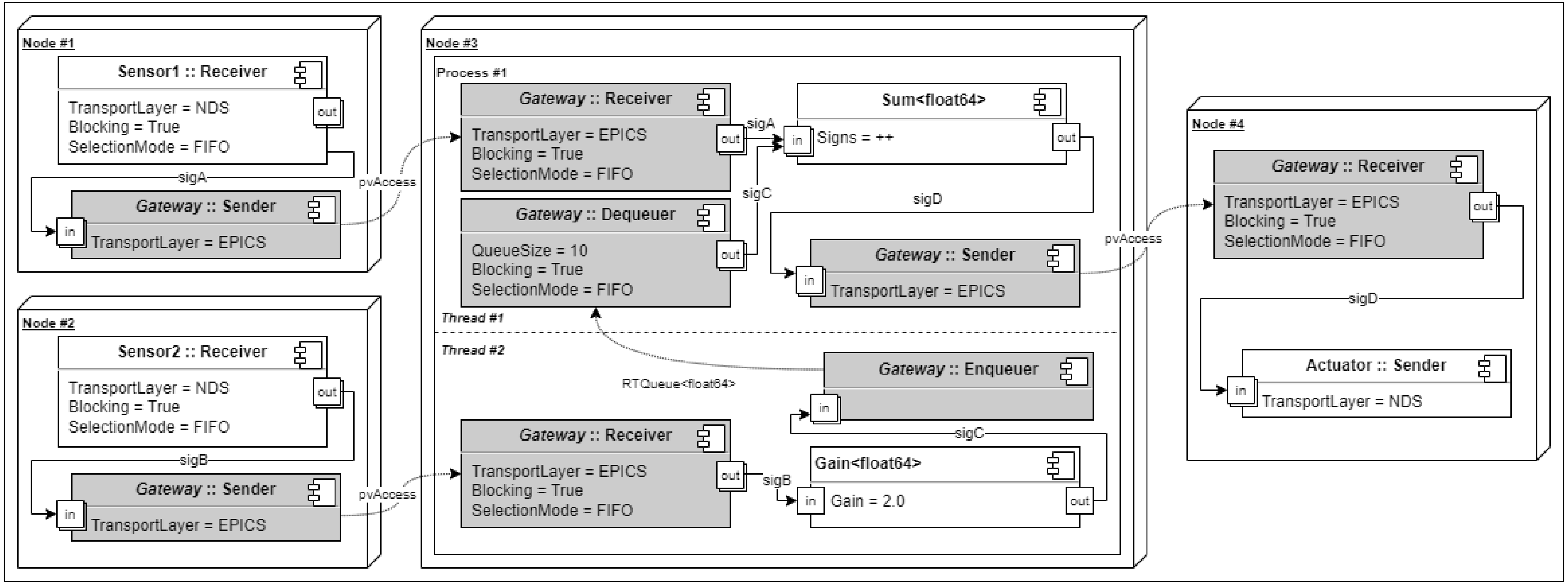
RTF Application Deployment



Inter-Thread Gateways



Inter-Process/Node Gateways



Transport Layers

- Abstraction layer between communication interface and the RTF
- Supports normative and structured data types
- Data types must support:
  - Introspection to allow structure mapping to be performed at initialization time
  - Serialization to allow structure data to be serialized/deserialized at runtime
- Available/foreseen implementations:
  - Ethernet based TLs: SDN, DAN (publish only), EPICS pvAccess
  - Local data exchange TLs: Files, Pipes, Shared memory
  - Hardware/DAQ TLs: NDS
  - User interaction TLs: Console, GUI screens

EPICS Transport Layer

- Originally based on pvAccessCPP but migrated to PVXS in RTF 2.2.4 (March 2021)
- Creates **`EpicsPublisher`** and **`EpicsSubscriber`** objects for **`Sender`** and **`Receiver`** FBs
- Framework recursively climbs the data structure (through introspection) to invoke:
  - **`declare()`** function implemented by both **`EpicsPublisher`** & **`EpicsSubscriber`** to declare/validate pvData structure (at initialization time)
  - **`read()`** function implemented by **`EpicsSubscriber`** to read structure data from pvData (at runtime)
  - **`write()`** function implemented by **`EpicsPublisher`** to write structure data to pvData and **`send()`** to finally send it (at runtime)

Recursive Structure Type Declaration

```
struct Sample {
  uint64_t time;
  size_t quality;
  struct {
    float64_t x;
    float64_t y;
  } position;
  uint64_t values[10];
}
```

`Sample.time`

=> declares “time”, depth 1

`Sample.quality`

=> declares “quality”, depth 1

`Sample.position.x`

=> declares “x”, depth 2

`Sample.position.y`

=> declares “y”, depth 2

`Sample.values[10]`

=> declares “values”, depth 1

EPICS Transport Layer

- `EpicsPublisher/EpicsSubscriber::declare(const char* name, const size_t& depth, const std::type_index& data_type, const size_t& size, const size_t& count)`
  - `child = pvxs::Member(pvxs::TypeCode::Struct, convert_name(name))`
  - `child = pvxs::Member(toTypeCode(data_type).arrayOf(), convert_name(name))`
  - `child = pvxs::Member(toTypeCode(data_type), convert_name(name))`
  - `parent.addChild(child);`

- `size_t EpicsPublisher::write(const size_t& position, const void* data, const size_t& count, const size_t& total_size)`
  - `pvxs::client::PutBuilder::set(field[position], *(static_cast<const T*>(data)))`

- `size_t EpicsSubscriber::read(const size_t& position, void* data, const size_t& count, const size_t& total_size)`
  - `std::memcpy(data, field[position].as<T>().data(), total_size);`

Conclusion

- Transport layers are also used for events (through Event Distribution Service)
- Besides for transport layers, pvAccess is also used for RTF process Life Cycle Management Service
  - LCMS could ultimately use TLs rather than its own implementation
- Transport layers currently provide no configurability for the topic (PV) and structure (pvData) field names

QUESTIONS