ITER Operation Applications: Status and Future Development

Bob Gunion
ITER Organization

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

Outline

- Operation Applications Overview
- Pulse Schedule Preparation System (PSPS)
- Supervision System (SUP) Overview
- SUP Configuration
- SUP Sequencer

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**

*Anze Zagar Thu 12:25 – EPICS (pvAccess) Transport Layer for the ITER Real-Time Framework (RTF)
**Leonid Lobes Thu 09:35 – Study on EPICS Communication over Long Distance

Challenges in the ITER Control System

- ITER is:
  - Large: 3.5 million PVs at first plasma; 5+ at later stages
  - Complex: >20 plant systems, each with a unique design
  - Diverse: In-kind contributions from 7 members (35 countries) and hundreds of contractors
  - Changing: Staged deployment and operations lead to evolving requirements
- Addressing these challenges:
  - Automate procedures as much as possible
  - Decouple individual plant systems by overlaying a consistent interface
  - Validate, verify, and trace configurations and machine state
  - Provide comprehensive monitoring tools
  - Ensure proper authentication and approval policies
  - Retain flexibility to change procedures and parameters over time
  - Coordinate operational plans through pulse schedules

-> PSPS and SUP solve these by providing a well-controlled, but flexible, solution

Pulse Schedule Preparation System

- Purpose:
  - Allow experiment leaders to define pulse schedules for automated operation of the entire ITER system
  - Provide an approval workflow to ensure pulse schedules match ITER scientific goals, equipment states, plant system requirements, etc.
  - Allow remote creation/editing of pulse schedules
  - Validate configurations through CVVF
  - Execute pulse schedules via integration with SUP
- Currently in early development
  - Initial pulse schedule lifecycle management service has been implemented (C++17)
  - Early prototype of GUI (based on MVVM framework with Qt/C++)
- We conduct demonstrations regularly with our future users (physics, operations, plant systems) and our CODAC colleagues

Integrated PSPS/SUP/PCS Demonstrations:

Dec. 2021
- Read PS, navigate sections, basic authorization
- Transmit to SUP

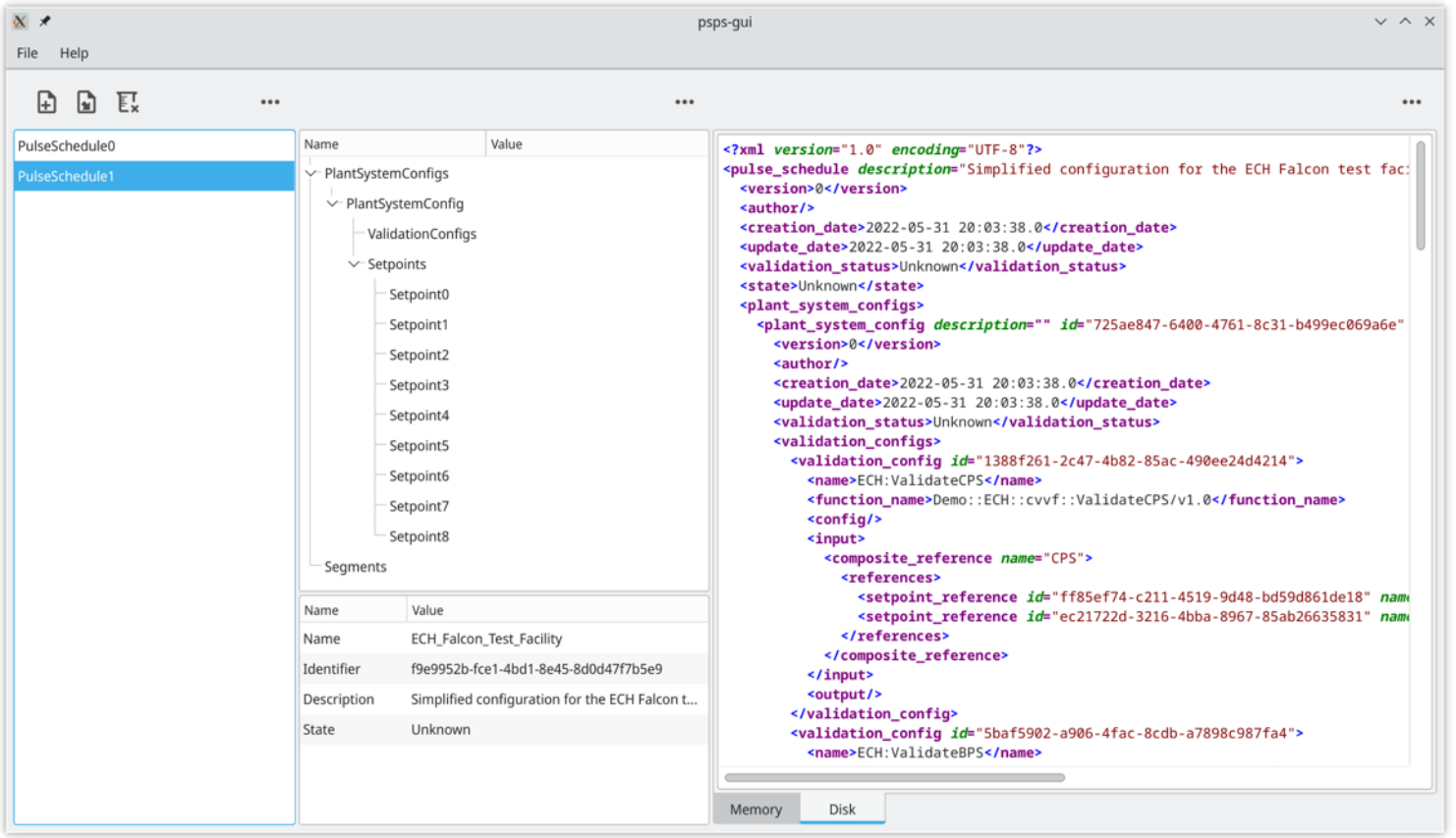
April 2022
- Realistic plant system config, XML and database serialization, validation & transformation
- Transmit complete plant system config to SUP

July 2022
- PCS configuration, validation in both PSPS and SUP
- Full control chain PSPS -> SUP -> PCS

Nov. 2022
- PS Editor GUI

PSPS Plans

- Add features to core functionality
  - LDAP-based authentication and authorization
  - Version control of pulse schedules
  - Web-based API
  - Complete workflow support
- PS Editor User Interface
  - Demonstrate prototype in Qt/C++ based on MVVM framework
  - Explore web-based UI (React, Angular, ...)



Supervisory System (SUP) Overview

Provides Operations with a central control interface to the entire ITER machine

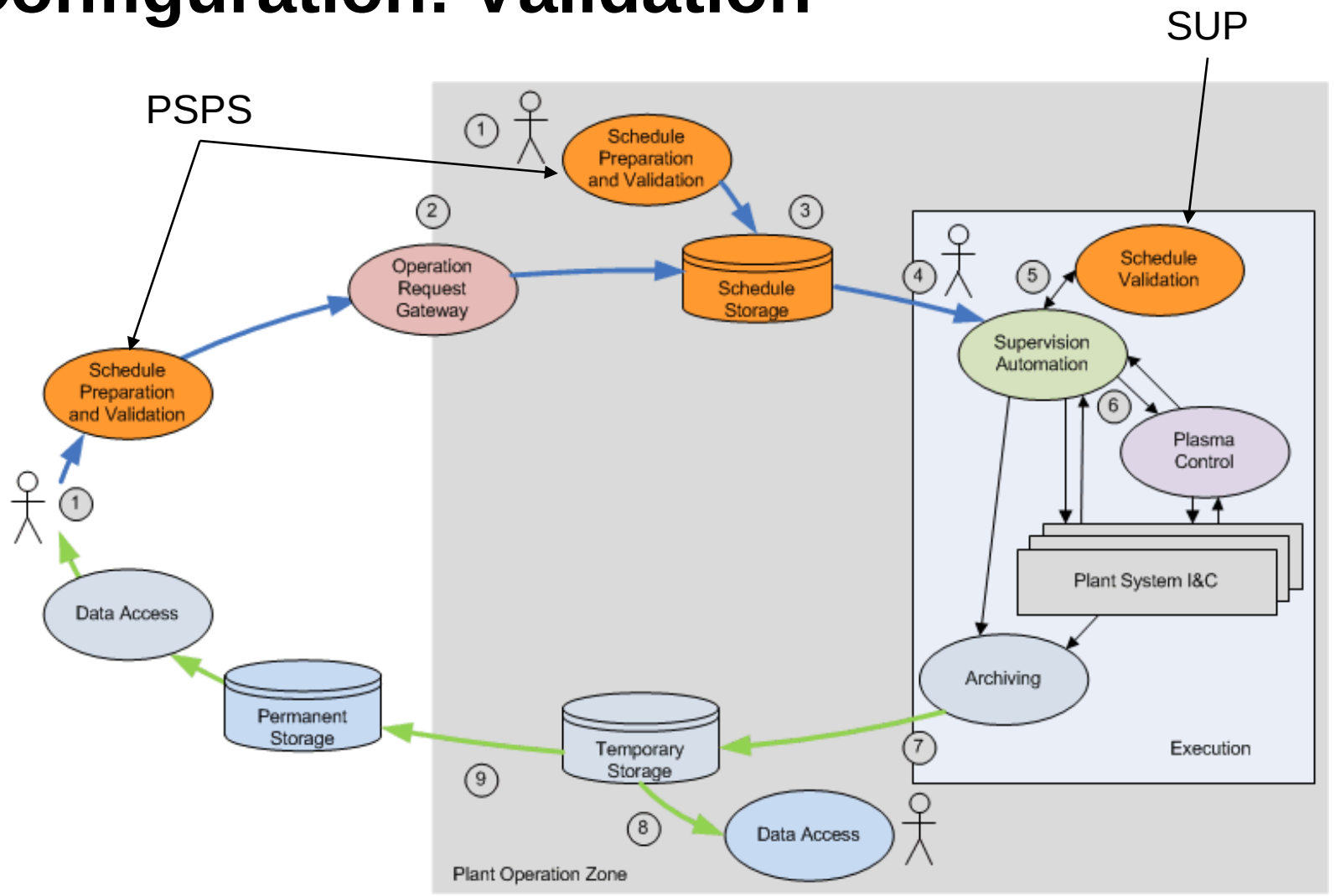
Integration	Provides abstraction of plant system states Common interface for configuration, control and monitoring (common functions)
Automation	Functions for automated activities Control actions based on status monitoring Handle concurrent activities
Operability	Achieve robust and repeatable operation Configure plant system from scratch Access to bare signals
Protection	Lowest tier in defense-in-depth hierarchy Provides permit/inhibit behavior

Key Components:

- Configuration (**CVVF**)
- **Sequencer**
- Monitoring
- Pulse Counter
- Timing

SUP Configuration: Validation

- Primary use case: pulse execution
- Validation must be performed at all stages of pulse schedule development and execution
- All validations must use the same code to ensure consistency



Addressing Configuration Challenges

- Provided solution must be capable of:
  - Executing arbitrary codes after approval by appropriate experts
    - Multiple languages (C++/Python/Matlab)
    - Arbitrary datatypes, including structured/array data
  - Linking disparate systems with a common interface
  - Deterministic operations when necessary
  - Parallel execution when appropriate
- RPC service with flexible interface is an ideal solution
- Meanwhile, EPICS is already the established control system

-> Solution: Configuration, Verification and Validation Framework (CVVF) with PVAccess transport layer

Development Status of CVVF

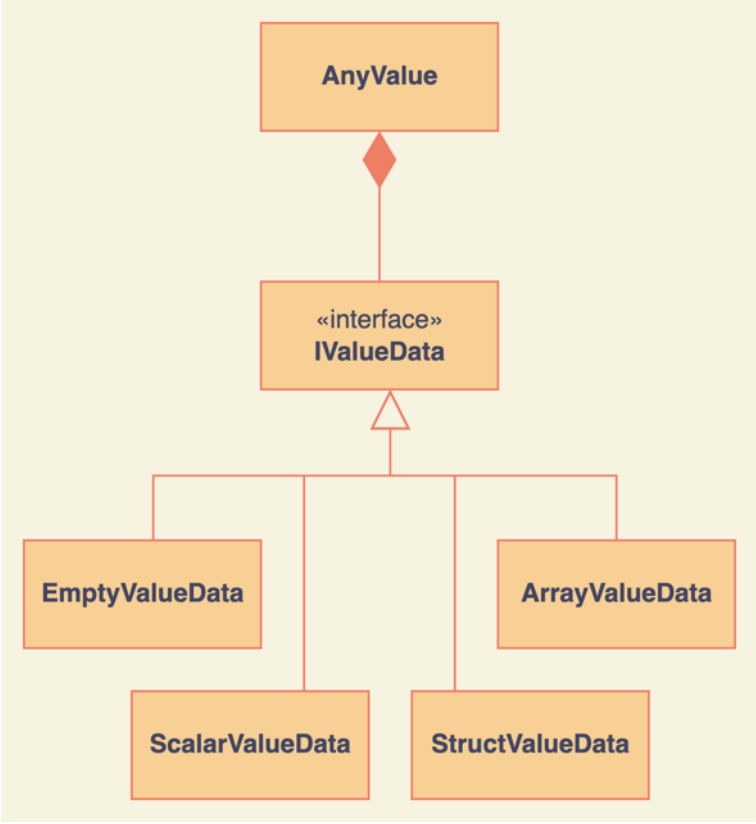
- CVVF is in active use
- 2018: First release as part of CODAC Core 6.0 including:
  - Minimized dependencies on C++11 (compiler support, ?'s about stability)
  - PVAccess-based RPC transport
  - `bool Execute(const char* function_name, const AnyValue& config, const AnyValue& input, AnyValue& output);`
- 2022: Refactoring to address shortcomings of initial implementation:
  - Unnecessary dependencies on CODAC infrastructure
  - Non-intuitive API leads to frustration and confusion among developers
  - Passing everything by reference, including the reliance on a custom shared pointer implementation, leads to unexpected exceptions

Refactoring begins with a new AnyValue implementation

Changes to CVVF

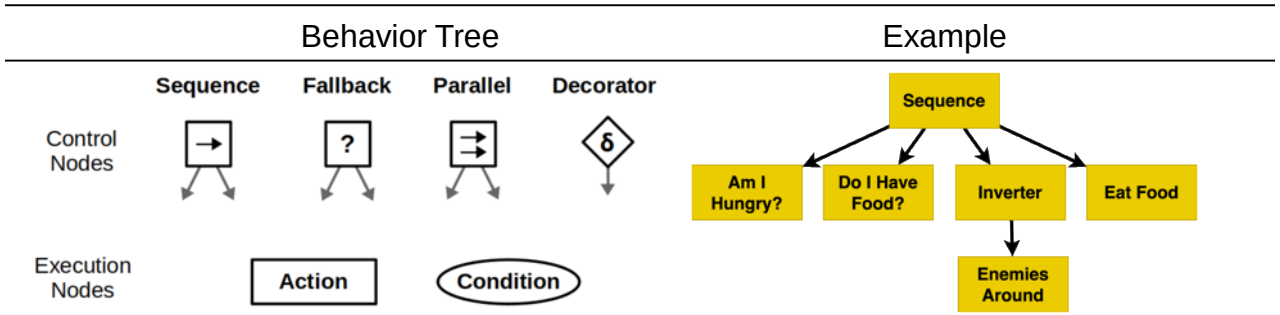
	Old	New
LOC	8k	5k
EXPECTs	300	2300

- New AnyType/AnyValue implementation
  - No more need to avoid C++11
  - Adopt pass-by-value to avoid ownership/reference problems
  - Adopt a more intuitive API, including templated methods
  - Passes quality gates: HICPP, cognitive complexity, memcheck, coverage
- Next: replace the RPC implementation
  - Remove dependency on CODAC Core
  - Still using PVAccess, as it's been very successful



SUP: Sequencer

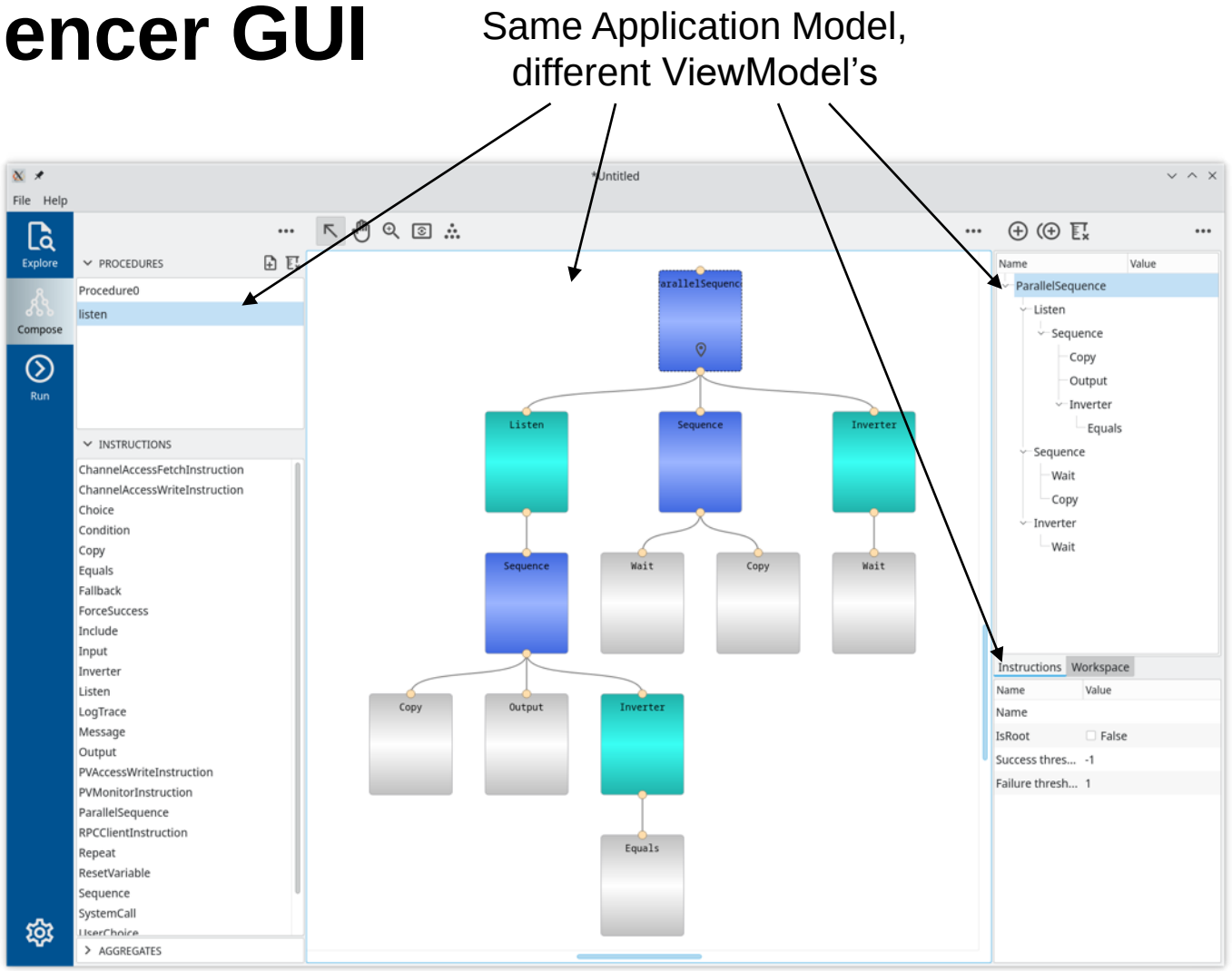
- The Sequencer is a tool for the creation, adaptation, approval and execution of operational tasks
- Coordinates multiple services for integrated tasks including commissioning, maintenance, etc.
  - Generally uses PVAccess for remote communications
- Based on a behavior tree model
- All steps of a task can be defined in a procedure, saved as an XML file
- Sequencer GUI is provided as an HMI for defining, debugging and running procedures




```
<?xml version="1.0" encoding="UTF-8"?>
<Procedure xmlns="http://codac.iter.org/sup/sequencer" version="1.0"
           name="Procedure for configuration of system and running a test
scenario"
           xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
           xs:schemaLocation="http://codac.iter.org/sup/sequencer
sequencer.xsd">
    <RegisterType
jsontype='{"type":"TestSUPPlugin::types::FilterSettings/v1.0", ...
    <Plugin>libsequencer-sup-cfg.so</Plugin>
    <Sequence name="Main sequence">
        <CVVFClient name="Verify filter settings through CVVF"
            service="test-sup-plugin@cvvf-service"
            function="TestSUPPlugin::cvvf::VerifyFilter/v1.0"
            input="config"
            output="result"/>
        <Fallback name="Verification succeeded or override">
            <Equals lhs="result" rhs="true"/>
            <Sequence>
                <Input output="override" description="Override verification
result?"/>
                <Equals lhs="override" rhs="true"/>
            </Sequence>
        </Fallback>
        <LoadConfiguration name="Load configuration for filter"
            service="test-sup-plugin@cfg-service"
            configName="TestSUPPlugin::Filter0::settings"
            input="config"/>
        <Include name="Execute test" file="TestScenarios.xml"
path="FilterTests.test01"/>
    </Sequence>
    <Workspace>
        <Local
name="config" type='{"type":"TestSUPPlugin::types::FilterSettings/v1.0"}'
                value='{"lowercutoff":20.0,"uppercutoff":11000.0,"mode":2}'/>
        <Local name="result" type='{"type":"bool"}'/>
        <Local name="override" type='{"type":"bool"}'/>
        <Local name="true" type='{"type":"bool"}' value="1"/>
    </Workspace>
</Procedure>
```

Sequencer GUI

- Based on MVVM concept of GUI software architecture
- C++17, Qt5, CMake
- Multiple views of the same model allow user to select the most appropriate interaction
- Graphical view of behavior tree nodes allows drag/drop, auto-arranging, etc.



Conclusions

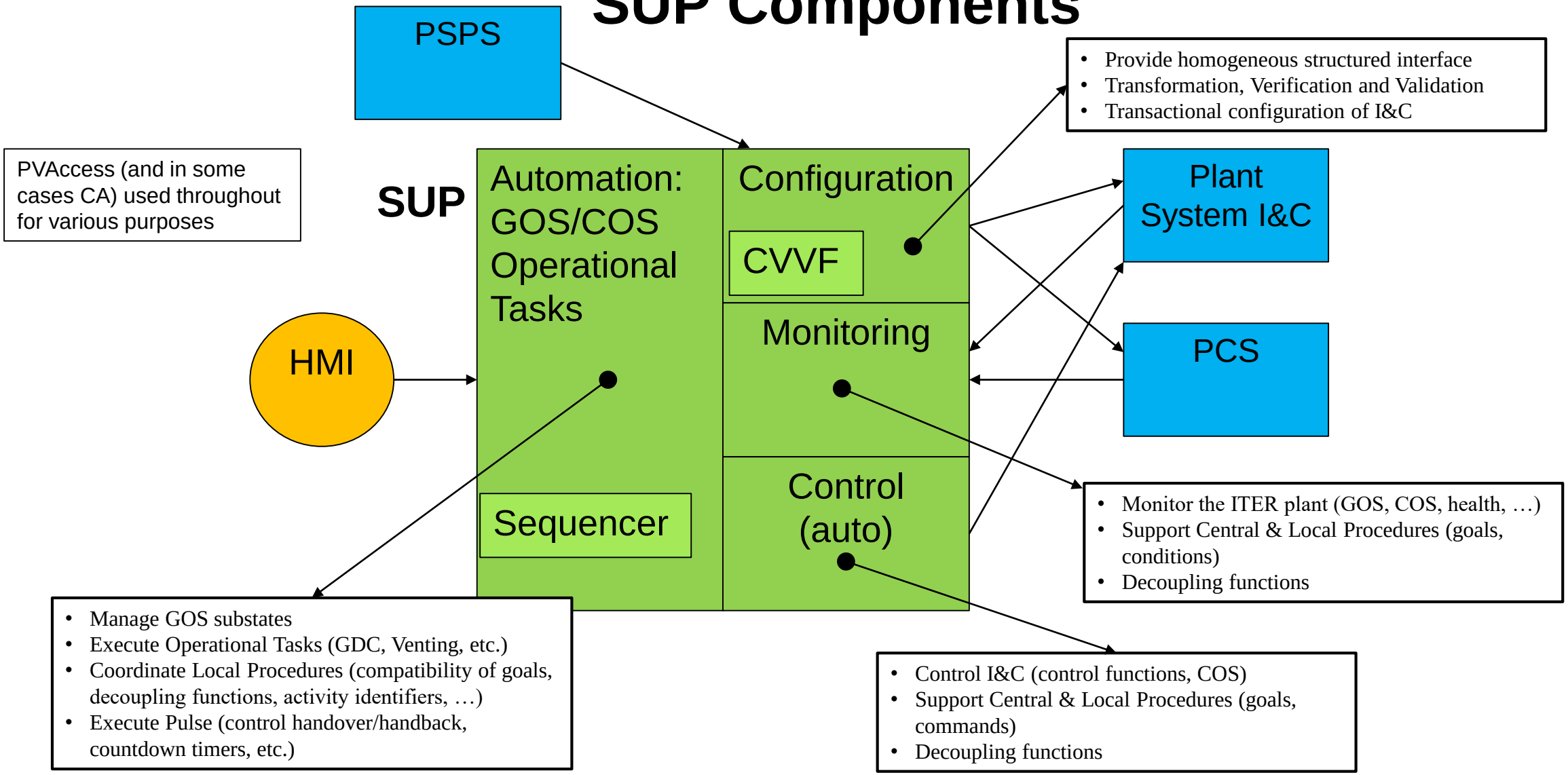
- Operation Applications development is active and ongoing
  - Use of PVAccess is widespread
- PSPS development getting off to a good start
- CVVF has proven to meet its design goals
  - But, improvements are possible
  - Refactoring of the underlying data objects will ensure greater robustness and performance
  - PVAccess has proven to be easy to use, flexible, and performant
- SUP Sequencer provides a flexible and robust solution for automated task execution
- SUP Sequencer GUI provides an intuitive UI for users' benefit

Backup Slides

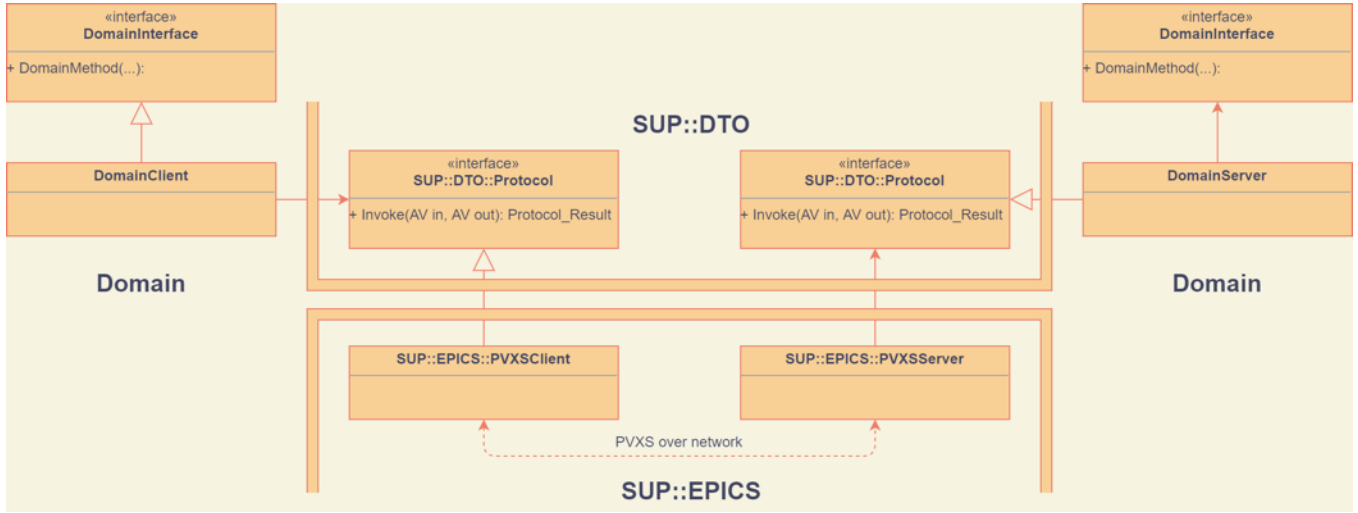
Roadmap for Operation Applications

- Yearly releases
- Dec. 2022
  - Pulse Schedule Editor with CVVF support
  - Sequencer, Pulse Counter, GOS management, documentation
- Dec. 2023
  - PS Editor: integrated plant system commissioning, segments, PCS configuration
  - SUP: Coordinated configuration, implementation of operational tasks
- Dec. 2024
  - PS Editor: Full workflow support
  - SUP: Completion of all SUP related HMIs

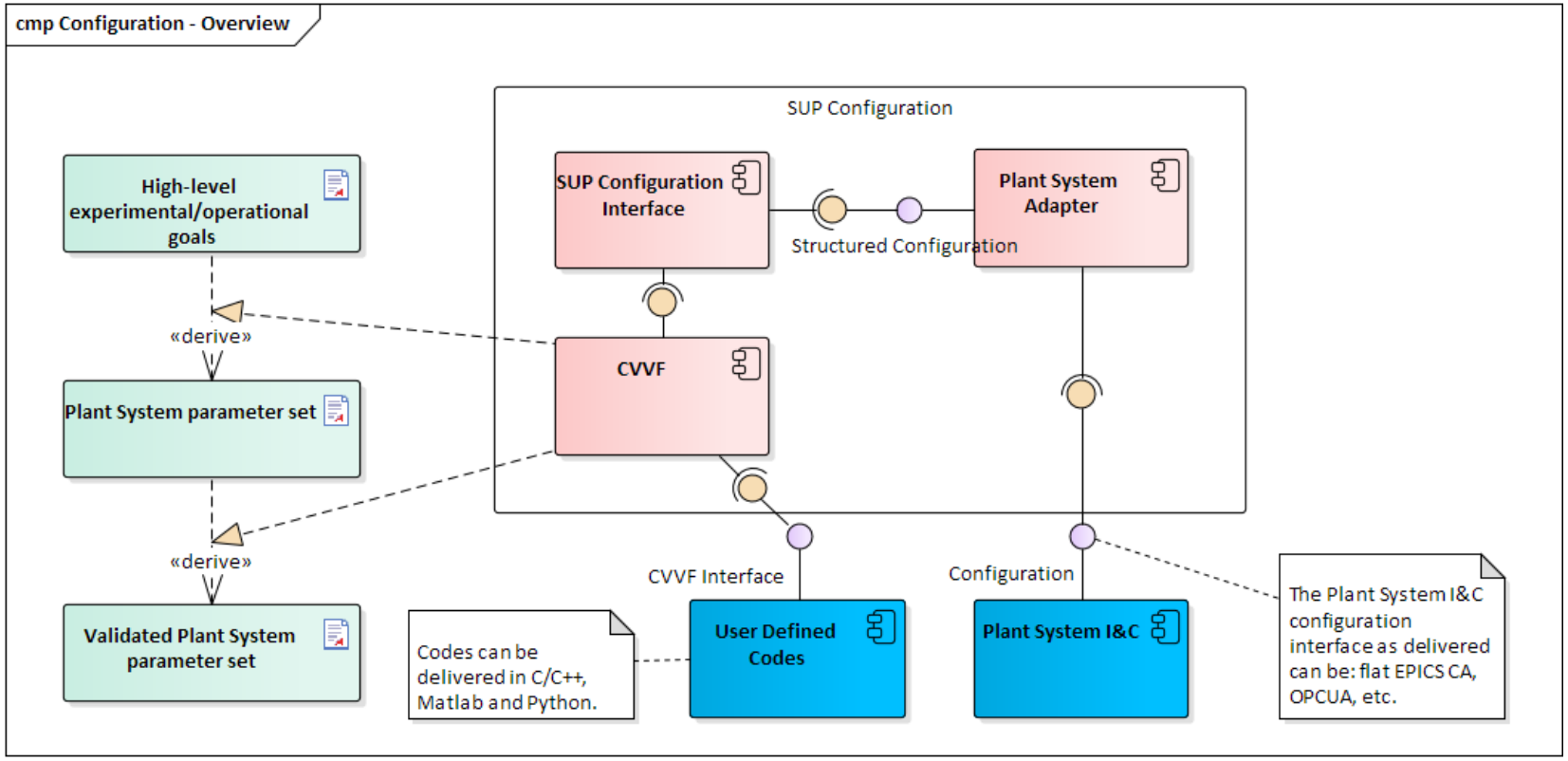
SUP Components



Data Transport Objects



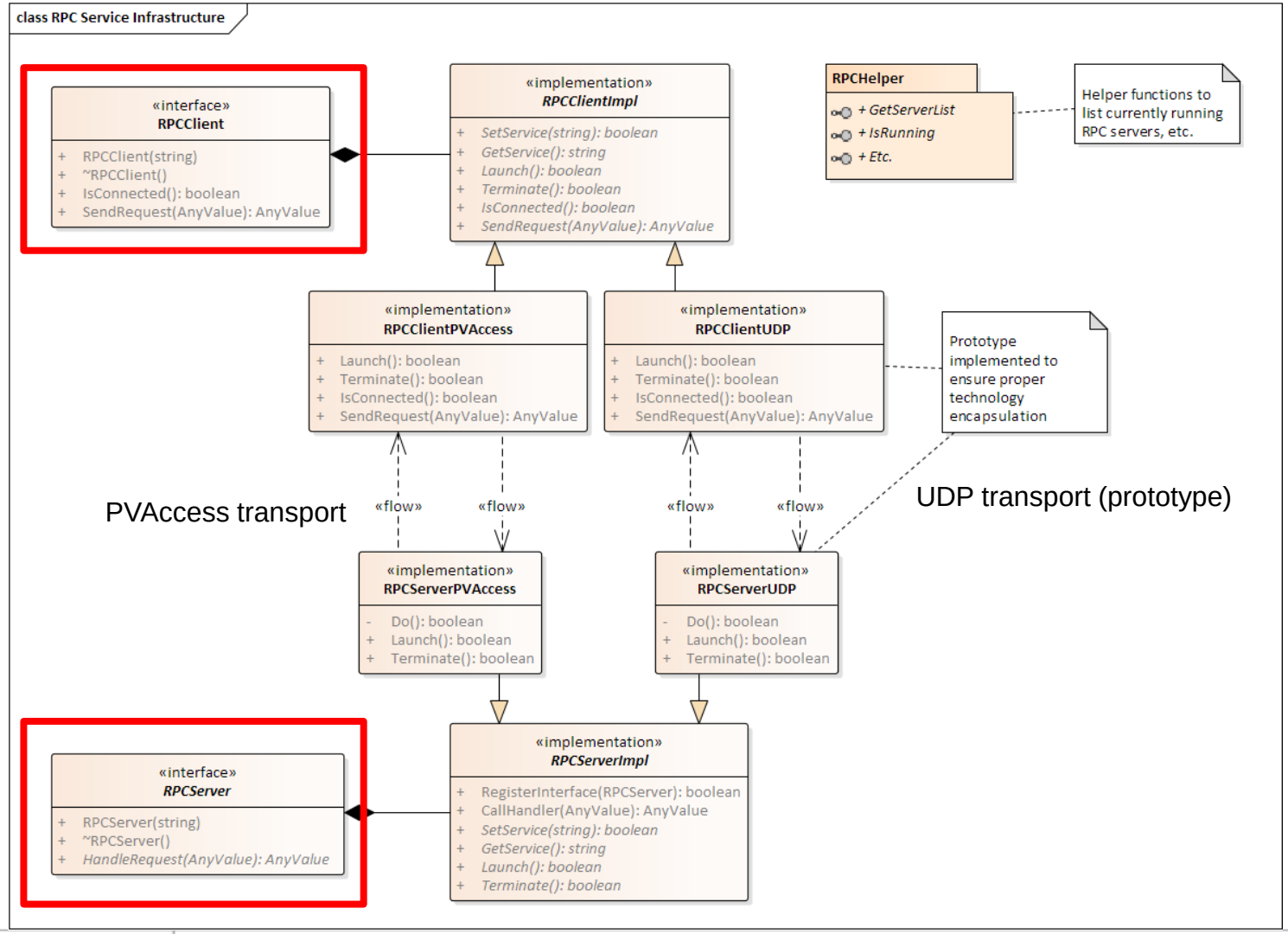
CVVF Information flow



SUP Components

- SUP Configuration provides a common interface to all plant systems
- SUP Automation provides central coordination of automated tasks
- SUP Pulse Counter provides a unique ID for each pulse
- SUP Timing records and publishes pulse countdown, start, end times
- SUP Monitoring coordinates GOS/COS (Global/Common Operating States) across plant systems
- Interfaces with PSPS to execute pulse schedules
- Hands over control to PCS (Plasma Control System) during pulse execution
- Software Integrity Level 1

RPC design



Multiple Environments

- All validations should run the same code