ADR-002: Complete Dependency Injection Pattern for Hardware Controllers

Architecture Diagrams

Status: [DONE] Accepted **Date:** 2025-10-30 **Deciders:** Development Team **Technical Story:** Phase 4 - Architectural Consistency & Medical Device Compliance

---

Context and Problem Statement

TOSCA's hardware control architecture evolved incrementally, resulting in **inconsistent controller instantiation patterns** across UI widgets:

1. **ActuatorConnectionWidget** (Phase 2): Constructor injection with centralized controller management
2. **LaserWidget, GPIOWidget, TECWidget, CameraWidget** (Pre-Phase 4): Self-instantiation via `_on_connect_clicked()`

This inconsistency created several problems: - **Architectural Confusion:** Developers unsure whether to instantiate controllers in widgets or MainWindow - **Testing Challenges:** Self-instantiated controllers difficult to mock for unit tests - **Lifecycle Ambiguity:** Unclear who owns controller lifecycle (widget vs MainWindow) - **Medical Device Compliance:** Multiple controller lifecycle patterns complicate IEC 62304

validation

Key Question: Should we standardize on a single controller management pattern across all hardware widgets?

Decision Drivers

Medical Device Context

- **Safety-Critical:** Centralized controller management provides single point of failure handling
- **IEC 62304 Compliance:** Clear ownership model simplifies validation documentation
- **Lifecycle Management:** Predictable startup/shutdown critical for safety watchdog integration
- **FDA Submission:** Consistent architecture eases Design History File (DHF) documentation

Technical Factors

- **Testability:** Constructor injection enables proper unit testing with mocked controllers
- **Separation of Concerns:** UI widgets should only handle presentation, not instantiation
- **Hollywood Principle:** "Don't call us, we'll call you" - inversion of control pattern
- **Code Maintainability:** Single instantiation point in MainWindow simplifies refactoring

Code Review Findings (Comprehensive Review - 2025-10-30)

HIGH Priority Issue #1: Architectural inconsistency across hardware widgets - ActuatorConnectionWidget uses DI pattern (Phase 2) - 4 other hardware widgets use self-instantiation (inconsistent) - **Impact:** Confusion, poor testability, maintenance burden

---

Considered Options

Option 1: Keep Hybrid Pattern (Status Quo)

Maintain mixed instantiation: DI for actuator, self-instantiation for others

Pros: - [DONE] No refactoring required - [DONE] Zero risk of breaking existing code - [DONE] Minimal testing effort

Cons: - [FAILED] Architectural inconsistency confuses developers - [FAILED] Poor testability for 4/5 hardware widgets - [FAILED] Lifecycle management ambiguous - [FAILED] Harder to document for FDA validation - [FAILED] Code review flagged as HIGH priority issue

Decision: [FAILED] **Rejected** - Architectural consistency outweighs refactoring cost

---

Option 2: Service Locator Pattern

Use central registry for controller lookup

Pros: - [DONE] Decouples widgets from MainWindow - [DONE] Flexible controller retrieval - [DONE] Easy to swap implementations

Cons: - [FAILED] Hidden dependencies (not obvious from constructor) - [FAILED] Runtime failure if controller not registered - [FAILED] More complex than needed for our use case - [FAILED] Anti-pattern in modern Python (implicit dependencies)

Decision: [FAILED] **Rejected** - Adds unnecessary complexity, hides dependencies

---

Option 3: Complete Dependency Injection (SELECTED)

Extend DI pattern to all hardware widgets, centralize controller management

Pros: - [DONE] **Architectural Consistency:** All 5 widgets follow identical pattern - [DONE] **Testability:** All controllers mockable for unit tests - [DONE] **Lifecycle Clarity:** MainWindow owns all controller lifecycles - [DONE] **Hollywood Principle:** Inversion of control properly implemented - [DONE] **Medical Device Compliance:** Single instantiation point simplifies validation - [DONE] **Explicit Dependencies:** Constructor signatures show exactly what's needed

Cons: - WARNING: Refactoring required for 4 widgets (~4 hours) - WARNING: Signal connection logic needs extraction (~2 hours) - WARNING: Risk of breaking existing widget functionality (mitigated by thorough testing)

Decision: [DONE] **ACCEPTED** - Benefits significantly outweigh refactoring cost

---

Decision Outcome

Chosen option: Option 3 - Complete Dependency Injection Pattern

Rationale

1. **Architectural Consistency:** All hardware widgets follow single, clear pattern
2. **Medical Device Safety:** Centralized controller management enables systematic shutdown (selective shutdown policy)
3. **Test Coverage:** Mockable controllers enable comprehensive unit testing (IEC 62304 requirement)
4. **Developer Experience:** Clear pattern reduces onboarding time and confusion
5. **Code Review Compliance:** Resolves HIGH priority issue from comprehensive code review

Implementation Strategy

Phase 4A: Widget Constructor Injection [DONE] COMPLETE

Modified Widgets: 1. **LaserWidget:** Accept `LaserController` via constructor parameter 2. **GPIOWidget:** Accept `GPIOController` via constructor parameter 3. **TECWidget:** Accept `TECController` via constructor parameter 4. **CameraWidget:** Accept `CameraController` via constructor (deprecated setter method) 5. **SafetyWidget:** Pass `GPIOController` to internal GPIOWidget

Timeline: 3 hours (completed 2025-10-30) **Risk:** Low (widgets remain functional during transition) **Result:** All 5 widgets accept controllers via constructor

Phase 4B: MainWindow Centralization [DONE] COMPLETE

Centralized Controller Instantiation:

```
# src/ui/main_window.py lines 75-87
self.actuator_controller = ActuatorController()
self.laser_controller = LaserController()
self.tec_controller = TECController()
self.gpio_controller = GPIOController()
self.camera_controller = CameraController(event_logger=self.event_logger)
```

Timeline: 2 hours (completed 2025-10-30) **Risk:** Medium (requires careful signal rewiring) **Result:** Single source of truth for all hardware controllers

Phase 4C: Signal Extraction [DONE] COMPLETE

Extracted `_connect_controller_signals()` methods: - Separates signal connection from constructor logic -

Called conditionally when controller is injected - Enables testing with and without controller

Timeline: 2 hours (completed 2025-10-30) **Risk:** Low (signal connections tested via GUI smoke test) **Result:** Clean separation of concerns

Phase 4D: Public Connection API [DONE] COMPLETE

Added Public Methods: - `connect_device()` → Programmatic connection without UI events - `disconnect_device()` → Programmatic disconnection - **Benefit:** MainWindow can control connections without calling private methods

Timeline: 1 hour (completed 2025-10-30) **Risk:** Low (backward compatible, private methods retained) **Result:** Clean API for programmatic control

Consequences

Positive Consequences

Benefit	Impact
Architectural Consistency	5/5 widgets follow identical DI pattern
Testability	All controllers mockable for unit tests
Lifecycle Clarity	Single ownership model (MainWindow)
Medical Device Compliance	Simplified IEC 62304 validation documentation
Maintainability	Clear pattern reduces confusion, eases refactoring
Safety Integration	Centralized shutdown enables selective shutdown policy
Code Review Resolution	HIGH priority issue resolved

Negative Consequences

Challenge	Mitigation
Refactoring Effort	8 hours total (acceptable for architectural consistency)
Signal Rewiring Risk	Thorough GUI smoke testing, systematic validation
Backward Compatibility	Deprecated old patterns (e.g., `set_camera_controller()`)
Testing Overhead	Need to create mocked controller fixtures (future Phase 5)

Technical Implementation

Before (LaserWidget - Inconsistent)

```
class LaserWidget(QWidget):
    def __init__(self) -> None:
        super().__init__()
        self.controller = None  # [FAILED] Not injected
        self._init_ui()

    def _on_connect_clicked(self) -> None:
        # [FAILED] Self-instantiation - bad for testing
        self.controller = LaserController()
        success = self.controller.connect("COM10")
        if success:
            # [FAILED] Signal connection inside event handler
            self.controller.connection_changed.connect(...)
```

After (LaserWidget - DI Pattern)

```python
class LaserWidget(QWidget):
    def __init__(self, controller: Optional[LaserController] = None) -> None:
        super().__init__()
        self.controller = controller  # [DONE] Injected from MainWindow
        self._init_ui()

        # [DONE] Conditional signal connection
        if self.controller:
            self._connect_controller_signals()

    def _connect_controller_signals(self) -> None:
        """[DONE] Extracted for clarity and testability"""
        if not self.controller:
            return
        self.controller.connection_changed.connect(...)
        logger.debug("LaserWidget signals connected to LaserController")

    def connect_device(self) -> bool:
        """[DONE] Public API for programmatic control"""
        if self.is_connected:
            logger.warning("Laser already connected")
            return True
        self._on_connect_clicked()
        return self.is_connected
```

MainWindow Pattern (Single Instantiation Point)

```python
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # [DONE] Centralized controller instantiation
        self.actuator_controller = ActuatorController()
        self.laser_controller = LaserController()
        self.tec_controller = TECController()
        self.gpio_controller = GPIOController()
        self.camera_controller = CameraController(event_logger=self.event_logger)

        # [DONE] Constructor injection to widgets
        self.laser_widget = LaserWidget(controller=self.laser_controller)
        self.gpio_widget = GPIOWidget(controller=self.gpio_controller)
        self.tec_widget = TECWidget(controller=self.tec_controller)
        self.camera_widget = CameraWidget(camera_controller=self.camera_controller)

        # [DONE] Protocol engine uses MainWindow controllers directly
        self.protocol_engine = ProtocolEngine(
            actuator_controller=self.actuator_controller,
            laser_controller=self.laser_controller,
            camera_controller=self.camera_controller,
        )
```

Compliance Implications

Medical Device Software (IEC 62304)

Positive Impact: - [DONE] **Reduced Complexity:** Single instantiation pattern simplifies Class B validation - [DONE] **Clear Ownership:** Unambiguous controller lifecycle management - [DONE] **Testability:** Unit tests can verify controller behavior in isolation - [DONE] **Traceability:** Clear dependency graph for requirements mapping

Risk Management (ISO 14971): - [DONE] **Centralized Control:** Single shutdown point enables systematic safety response - [DONE] **Lifecycle Predictability:** Clear startup/shutdown sequence reduces failure modes - [DONE] **Selective Shutdown:** DI pattern enables safety watchdog to disable laser while keeping monitoring active

FDA 510(k) Preparation

Documentation Benefits: - [DONE] Simpler Design History File (DHF) - single controller management pattern - [DONE] Clearer software architecture diagram (single ownership model) - [DONE] Reduced architectural complexity in submission materials

---

Validation Plan

Pre-Implementation Validation [DONE] COMPLETE

1. [DONE] Code review identified HIGH priority issue (comprehensive review)
2. [DONE] Documented all widget controller patterns
3. [DONE] Created comprehensive refactoring plan (Phase 4A-4D)
4. [DONE] Wrote ADR-002 (this document)

Phase 4A-4B Validation [DONE] COMPLETE

1. [DONE] Syntax validation: All modified files pass `python -m py_compile`
2. [DONE] Import validation: No broken imports
3. [DONE] Signal connection verification: Extracted methods tested
4. [DONE] GUI smoke test: All widgets display and function correctly

Phase 4C-4D Validation [DONE] COMPLETE

1. [DONE] Public API testing: `connect_device()` / `disconnect_device()` work correctly
2. [DONE] MainWindow integration: Connect/Disconnect All buttons functional
3. [DONE] Error message improvements: User-friendly guidance implemented
4. [DONE] Documentation updates: WORK_LOG.md and REFACTORING_LOG.md updated

Future Phase 5 Validation (Unit Tests) [PENDING] PENDING

1. [PENDING] Create mocked controller fixtures
2. [PENDING] Unit test widgets with mocked controllers
3. [PENDING] Verify signal connections
4. [PENDING] Test error handling paths
5. [PENDING] Coverage target: 80%+ for all hardware widgets

Related Documentation

- **Refactoring Log:** `docs/REFACTORING_LOG.md` - Phase 4 details
- **Work Log:** `WORK_LOG.md` - Implementation timeline (2025-10-30)
- **Protocol Consolidation:** `docs/architecture/ADR-001-protocol-consolidation.md`
- **Safety Shutdown Policy:** `docs/architecture/SAFETY_SHUTDOWN_POLICY.md`
- **Lessons Learned:** `LESSONS_LEARNED.md` - DI pattern benefits (to be added)

Pros and Cons of the Options (Summary)

Option	Pros	Cons	Decision
1. Keep Hybrid	No refactoring	Architectural inconsistency, poor testability	[FAILED] Rejected
2. Service Locator	Flexible	Hidden dependencies, anti-pattern	[FAILED] Rejected

3. Complete DI	Consistency, testability, clear ownership	Requires refactoring (8 hours)	[DONE] **Accepted**

Links

- **Hardware Controllers:**
  - `src/hardware/actuator_controller.py`
  - `src/hardware/laser_controller.py`
  - `src/hardware/tec_controller.py`
  - `src/hardware/gpio_controller.py`
  - `src/hardware/camera_controller.py`
- **Modified Widgets:**
  - `src/ui/widgets/laser_widget.py`
  - `src/ui/widgets/gpio_widget.py`
  - `src/ui/widgets/tec_widget.py`
  - `src/ui/widgets/camera_widget.py`
  - `src/ui/widgets/safety_widget.py`
- **Orchestration:**
  - `src/ui/main_window.py` - Centralized controller instantiation

ADR Status: [DONE] Accepted **Implementation Status:** [DONE] COMPLETE - All 4 Phases Finished
Document Owner: Development Team **Last Updated:** 2025-10-30 **Implementation Completed:** 2025-10-30
Next Review: 6 months (or before FDA submission) **Related Commits:** 63c06f0, 552b2c3, e77c6d0, 10a7382