ionpy Framework: Erweiterte Architektur-Spezifikation

Dieses Dokument dient als detaillierte Implementierungsvorlage für die nächste Evolutionsstufe des ionpy-Frameworks. Ziel ist die Transformation von einer reinen Monitoring-Plattform zu einem interaktiven Automatisierungssystem.

1. Dynamische Eingabesynchronisation (The Mute-Pattern)

Problem: Race Conditions zwischen Hardware-Polling und User-Eingaben führen zu springenden Werten in der UI.

1.1 Backend: Implementierung in AbstractDevice

[cite_start]Die Basisklasse `AbstractDevice` [cite: 113] wird um ein Muting-System erweitert.

- **Datenstruktur**: Ein Dictionary `self._mute_until: Dict[str, float]` speichert pro Entity-ID den Ablaufzeitstempel der Sperre.
- **Code-Integration (hardware/base.py)**:
  - [cite_start]In `execute_command(self, entity_id, value)`[cite: 137]: Vor dem Aufruf des Handlers wird `self._mute_until[entity_id] = time.time() + 3.0` gesetzt.
  - [cite_start]In `update_entity(self, entity_id, raw_value, …)`[cite: 128]: Bevor das Sample erstellt wird, erfolgt die Prüfung:

```
if time.time() < self._mute_until.get(entity_id, 0): return # Update verwerfen
```

1.2 Frontend: Universeller Focus-Lock

In der `settings.html` wird ein globaler Schutzmechanismus implementiert, der unabhängig vom Gerätetyp funktioniert.

- **Logik**: Verwendung eines `Set()` in JavaScript, das IDs von Elementen speichert, die den Fokus haben.
- **Code-Hint**:

```
    const lockedFields = new Set();
    containerEl.addEventListener('focusin', (e) =>
lockedFields.add(e.target.id));
    containerEl.addEventListener('focusout', (e) => setTimeout(() =>
lockedFields.delete(e.target.id), 500));
    // Im WebSocket-Handler:
    if (lockedFields.has(`i-${sample.entity_id}`)) return;
```

2. Strukturierte Daten: TableEntity

[cite_start]Ermöglicht die Verwaltung von Profilen, Sequenzen und Speicherplätzen (z.B. M1-M10 für DPS5005 oder Timer-Listen für UDP3305 [cite: 303]).

2.1 Datenmodell (structures/entities.py)

[cite_start]Erweiterung der Entitäten um einen tabellarischen Typ[cite: 413]:

- **TableEntity**:
  - [cite_start]`columns`: Liste von Meta-Definitionen (z.B. `{key: "v", type: "number", unit: "V"}` [cite: 418]).
  - `value`: Liste von Zeilen-Objekten (z.B. `[{v: 12.0, i: 1.0}, {v: 5.0, i: 2.0}]`).

2.2 UI-Repräsentation

- Dynamische Generierung einer HTML-Tabelle.
- Jede Zelle erhält eine koordinatenbasierte ID (z.B. `table-ch1_list-row0-v`).
- Änderungen senden ein spezielles Command-Objekt: `{"row": index, "column": key, "value": new_val}`.

3. Gamepad-Steuerung via Pygame

Integration von Human Interface Devices zur haptischen Steuerung.

3.1 Gamepad-Treiber (hardware/system/gamepad.py)

Ein dedizierter Treiber nutzt `pygame.joystick` zur Abfrage.

- **Threading**: Da Pygame einen eigenen Event-Loop benötigt, wird dieser in einem Thread gestartet:

```
    def pygame_loop(self):
        while self.running:
            for event in pygame.event.get():
                if event.type == pygame.JOYAXISMOTION:
asyncio.run_coroutine_threadsafe(self.update_entity(...), self.loop)
```

- **Entity-Mapping**: Der Controller wird als `GamepadEntity` mit einem Dictionary-Value dargestellt, der alle Achsen und Buttons enthält.

3.2 Discovery

[cite_start]Der Treiber scannt beim Start mittels `pygame.joystick.get_count()` alle verfügbaren Controller und legt pro Controller eine Instanz in der `SystemEngine` [cite: 81] an.

4. LogicService: Die Automatisierungs-Brücke

[cite_start]Ein zentraler Dienst in der `SystemEngine`[cite: 81], der Events zwischen Geräten vermittelt.

4.1 Die Rule-Engine

[cite_start]Der Dienst abonniert den `EventBus` [cite: 85] und vergleicht jedes Sample mit einer Liste von Regeln (`rules.json`).

- **Beispiel-Regel (Mapping)**:
  - **Trigger**: `gamepad_0.axis_1` (Linker Stick Y).
  - **Transformation**: Linear Scaling (Input -1.0 bis 1.0 → Output 0.0 bis 32.0V).
  - **Action**: `engine.execute_command("udp3305_1", "ch1_v_set", calculated_value)`.

4.2 Web-Konfigurator

Ein universelles Frontend-Modul erlaubt das Erstellen dieser Regeln per Dropdown:

- WENN [Gerät wählen] [Entität wählen] ÄNDERUNG > X%
- DANN [Zielgerät wählen] [Zielentität wählen] SETZE WERT [Transformations-Formel].

5. Erweiterter Entitäten-Katalog (Specs)

5.1 LogEntity (Geräte-spezifisch)

- **Zweck**: Ein lokaler Feed für geräteinterne Ereignisse (Fehlercodes, Statuswechsel).
- **Visualisierung**: Terminal-Widget in der Geräte-Tab-Ansicht.

5.2 StatusIndicatorEntity

- **Zweck**: Visuelles Feedback ohne Text (virtuelle LED).
- **Mapping**: Wert 0 = Grau, 1 = Grün, 2 = Rot blinkend.

5.3 XYGraphEntity

- **Zweck**: Darstellung von Kennlinien (z.B. Strom über Spannung beim Batterie-Test).
- [cite_start]**Implementierung**: Nutzt das Waveform-Sample-Format[cite: 425], aber mappt Achse A gegen Achse B statt gegen die Zeit.

5.4 FileEntity

- **Zweck**: Übertragung von Firmware-Dateien oder Konfigurations-Backups.
- **API**: Endpoint für Multipart-Uploads, der direkt an den Treiber durchreicht.

6. Implementierungshinweise für KI-Programmierung

- [cite_start]**Stabilität**: Alle Treiber müssen `AbstractDevice` [cite: 116] [cite_start]korrekt implementieren, insbesondere die Fehlerbehandlung im `loop()`[cite: 137].
- [cite_start]**Performance**: Der `EventBus` [cite: 85] ist das Nadelöhr; Samples sollten nur bei signifikanten Änderungen gesendet werden.
- [cite_start]**Sicherheit**: Automatisierungsregeln müssen “Safe-Guards” (z.B. maximale Spannungsgrenzen) respektieren, die in der `TableEntity` oder `NumericEntity` [cite: 418] definiert sind.