ionpy Framework: Erweiterte Architektur-Spezifikation (Vollständig)

Dieses Dokument beschreibt die integrale Architektur-Erweiterung des ionpy-Frameworks. Es dient als verbindliche Grundlage für die Implementierung neuer Entitätstypen, haptischer Steuerungen und geräteübergreifender Automatisierung.

1. Dynamische Eingabesynchronisation (Race Condition Schutz)

Um zu verhindern, dass Hintergrund-Polling Benutzereingaben im Frontend überschreibt, wird ein duales Sperrsystem implementiert.

1.1 Backend: Mute-Timer (AbstractDevice)

[cite_start]In der Klasse `AbstractDevice` (hardware/base.py) wird eine zeitbasierte Sperre pro Entität eingeführt[cite: 117, 120].

- [cite_start]**Mechanismus**: Ein Dictionary `self._last_command_time: Dict[str, float]` speichert den Zeitpunkt des letzten Schreibbefehls[cite: 11, 54].
- **Logik**:
  - [cite_start]Sobald `execute_command()` aufgerufen wird, erhält die `entity_id` einen Zeitstempel[cite: 13, 138].
  - [cite_start]Die Methode `update_entity()` prüft diesen Zeitstempel: Liegt er weniger als **3,0 Sekunden** in der Vergangenheit, wird das Sample verworfen und nicht auf den Bus publiziert[cite: 87, 136].
- **Ziel**: Die Hardware hat Zeit, den Wert intern zu setzen, und der Polling-Loop liest keine “alten” Werte mehr zurück, während der User noch interagiert.

1.2 Frontend: Universeller Focus-Lock (JS)

In der Web-UI (settings.html) wird eine automatische Erkennung aktiver Eingabefelder implementiert.

- **Mechanismus**: Nutzung eines `Set()` namens `activeInputs`.
- **Event-Delegation**:
  - `focusin`: Fügt die Element-ID zum Set hinzu.
  - `focusout`: Entfernt die ID nach einer kurzen Verzögerung (ca. 300-500ms).
- **WebSocket-Logik**: Die Funktion `channel.onmessage` prüft vor dem Update eines HTML-Elements, ob dessen ID im Set vorhanden ist. [cite_start]Falls ja, wird das Update verworfen[cite: 53, 54].

2. Strukturierte Daten: TableEntity (Deep Dive)

Die `TableEntity` ist das Herzstück für komplexe Geräteeigenschaften wie Speicherplätze (M1-M10), Profil-Listen (Sequenzer) oder Zell-Übersichten.

2.1 Datenstruktur & Schema

Eine `TableEntity` kapselt nicht nur Daten, sondern auch deren Bedeutung.

- **Schema (columns)**: Definition der Spalten-Metadaten.
  - Jede Spalte definiert: `key`, `name`, `type` (number/text/toggle/action), `unit`, sowie Constraints (`min`, `max`, `step`).
- **Daten (value)**: Eine Liste von Dictionaries, wobei jedes Dictionary eine Zeile darstellt.
- **Typen**: Unterscheidung zwischen `fixed_size` (z.B. genau 10 Speicherplätze) und `dynamic_size` (Zeilen hinzufügbar/löschbar).

2.2 Erweiterte Interaktions-Logik

- **Row-Updates**: Das Frontend sendet Koordinaten-Pakete: `{ "row": r, "col": "key", "val": value }`.
- **Atomic Row Actions**: Unterstützung einer Spalte vom Typ `button`. Dies ermöglicht "Apply"-Buttons pro Zeile, um einen kompletten Parametersatz (z.B. Volt und Ampere eines Presets) gleichzeitig an die Hardware zu senden, um instabile Zustände zu vermeiden.
- **Active Row Tracking**: Ein zusätzliches Attribut `active_row_index` markiert die Zeile, die das Gerät aktuell tatsächlich verwendet (z.B. welcher Speicherplatz gerade geladen ist).
- **Zell-basiertes Muting**: Die Mute-Logik aus Kapitel 1 wird auf Zellebene angewendet, sodass eine Bearbeitung in Zeile 1 nicht die Live-Updates von Zeile 2 blockiert.

3. Gamepad-Integration (HID-Steuerung)

Haptische Steuerung via USB-Controller, realisiert durch das `pygame`-Subsystem.

3.1 GamepadManager (hardware/system/gamepad.py)

Ein neuer Treiber-Typ, der autonom nach Controllern sucht.

- [cite_start]**Discovery**: Nutzt `pygame.joystick.get_count()` und `get_id()`, um Controller dynamisch zu finden, ohne Hardcoding in der Config[cite: 63, 64].
- [cite_start]**GamepadEntity**: Eine neue Entitätsklasse, die den Zustand (Axes, Buttons, Hats, Triggers) als Snapshot-Objekt im `value`-Feld hält[cite: 135].

3.2 Haptisches Feedback & Visualisierung

- [cite_start]**UI-Widgets**: Spezielle Web-Komponenten für Joysticks (Fadenkreuz) und Trigger

(Druckempfindliche Balken via `UIMode.LEVEL`)[cite: 422].
- **Sicherheitskonzept**: Implementierung eines “Deadman-Switch” (Totmannknopf). Steuerbefehle werden nur an andere Geräte weitergeleitet, wenn eine definierte Taste am Gamepad gehalten wird.

4. LogicService: Die Automation Bridge

[cite_start]Zentraler asynchroner Dienst in der `SystemEngine`[cite: 81], der als Vermittler zwischen dem Bus und den Geräte-Kommandos fungiert.

4.1 Die Rule-Engine

[cite_start]Der Dienst abonniert den `EventBus` [cite: 85, 427] und prozessiert Regeln aus einer `rules.json`.

- [cite_start]**Trigger**: Ein Sample von Gerät A (z.B. Gamepad-Achse oder BMS-Temperatur)[cite: 424].
- **Transformation (Scaling)**: Mathematische Umwandlung von Eingangswerten (z.B. Gamepad-Stick -1.0…+1.0) in Zielwerte (z.B. Netzteil 0.0…32.0 V).
- [cite_start]**Action**: Ausführung von `engine.execute_command(target_dev, key, transformed_val)`[cite: 84, 433].

4.2 Cross-Device Szenarien (Beispiele)

- **Synchronisation**: Die elektronische Last (Senke) folgt automatisch der Spannung des Netzteils (Quelle), um eine konstante Leistung (CP-Mode) über das Framework zu simulieren.
- **Master-Slave**: Zwei Netzteile werden so gekoppelt, dass Kanal 2 immer exakt dem Wert von Kanal 1 folgt.

5. Erweiterter Entitäten-Katalog

[cite_start]Zusätzliche spezialisierte Typen für professionelle Laboranforderungen[cite: 421, 422, 423]:

Typ	UI-Repräsentation	Funktionalität
LogEntity	Scrollende Konsole	Lokaler Ereignis-Speicher für gerätespezifische Fehler (z.B. SCPI-Fehlermeldungen).
StatusIndicator	Virtuelle LED	Farb-Mapping für Zustände (z.B. 0=Off, 1=OK/Grün, 2=Warnung/Gelb, 3=Alarm/Rot-Blinkend).
XYGraphEntity	Kennlinien-Plot	Darstellung von X-Y-Beziehungen (z.B. Batterie-Entladekurve: Spannung über Kapazität).
FileEntity	Upload/Download	Schnittstelle für Firmware-Dateien (z.B. ESPHome .bin) oder Konfigurations-Exports.
RangeEntity	Multi-Slider/Input	Gruppiert logisch zusammengehörige Werte für Sweeps (Start, Stop, Step, Intervall).

Typ	UI-Repräsentation	Funktionalität
ScheduleEntity	Zeitplan-Editor	Verwaltung von Zeitereignissen (z.B. “Schalte Ausgang an Wochentagen um 08:00 Uhr an”).

6. Implementierungs-Leitfaden für KI-Entwicklung

- [cite_start]**Concurrency**: Alle Logik-Operationen müssen asynchron (`async/await`) ausgeführt werden, um den Hardware-Poll-Loop nicht zu blockieren[cite: 1, 9].
- [cite_start]**Caching**: Die `SystemEngine` nutzt ihren `state_cache` als “Single Source of Truth” für die Logic-Regeln[cite: 81, 84].
- **Modularität**: Neue Entitäten müssen in `structures/entities.py` definiert und in der `settings.html` mit einem entsprechenden UI-Generator verknüpft werden.