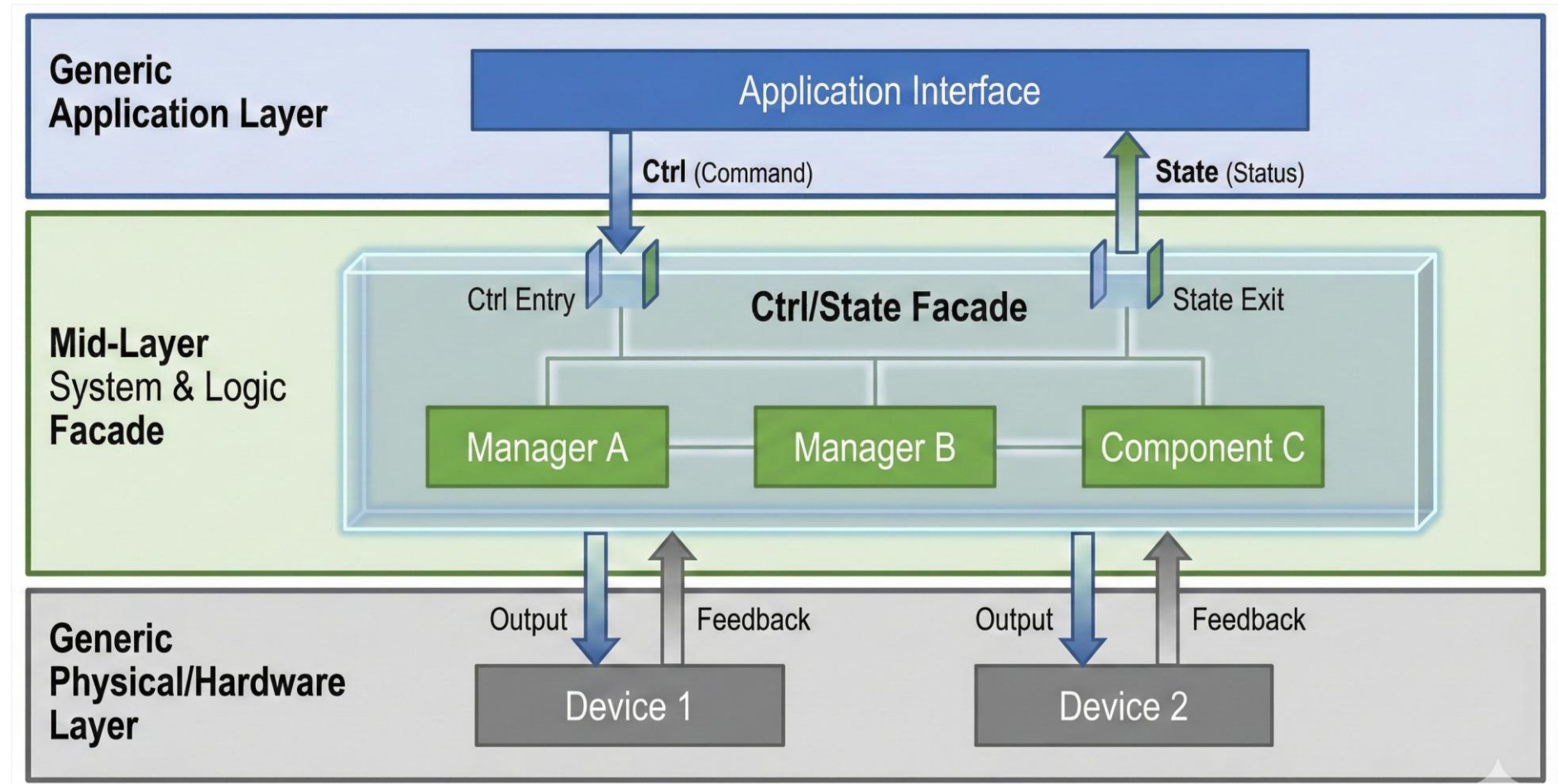


1. Motivation

2. Design



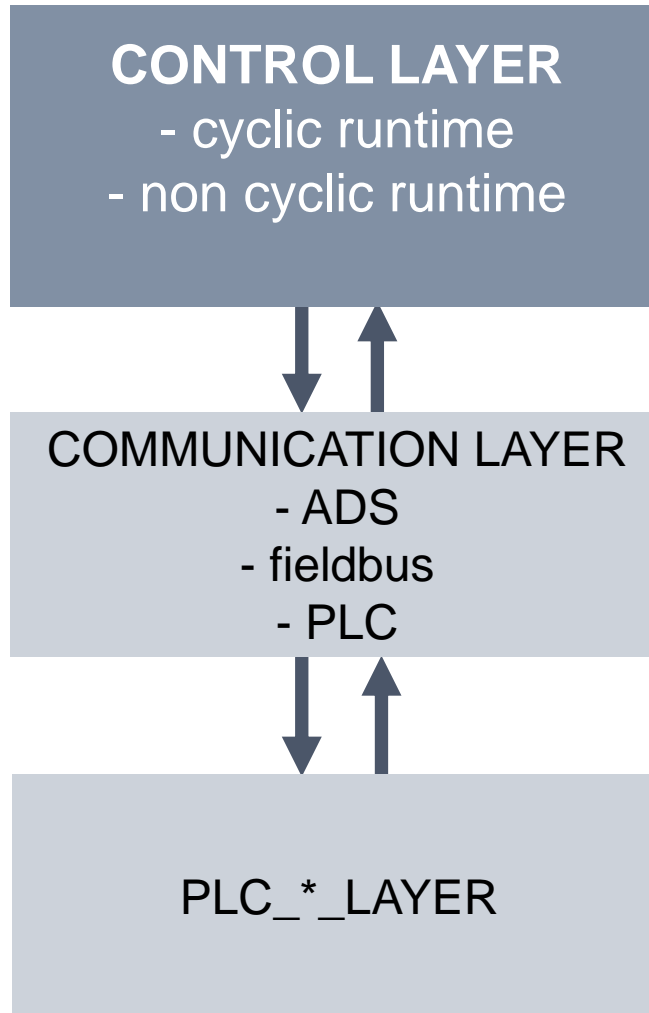
■ Motivation

- Einheitliche Kommunikation zwischen software Entitäten
 - Höhere Abstraktionsebenen sind geschützt und sicher vor Veränderungen in unteren Schichten.
- Plattformunabhängige Kommunikationsmedien
 - Datenfelder können zu jeglichem zyklischen oder nicht-zyklischen Kommunikationsmedium verbunden werden.
 - Harte Echtzeit, TCP/IP, ADS
- Asynchroner, atomarer Informationsfluß
 - Klasseninstanzen kommunizieren mittels verketteter Liste
- Reduzierung von Software Latenz
 - Sequentielle Abarbeitung mittels angepasster case syntax ermöglicht Zustandsänderungen ohne PLC Zyklus zu verwenden
 - LinkedList implementation ermöglicht atomaren Informationsfluß
 - Information kommt an Ziel an bevor die physikalischen Bedingungen erfüllt sein müssen, so dass Abläufe vorausschauend programmiert werden können
 - Software agiert, sie reagiert nicht mehr, was bei Optimierungen von bestehenden Abläufen zu deutlichen Zeiteinsparungen kommen kann
- Message-System hat sprechenden Inhalt
 - Die Verwendung von Enumerationen und Zeichenketten ermöglicht Loggingfunktionen zu bauen, die für as menschliche Auge lesbarer sind als Zahlenkolonnen.

▪ Design

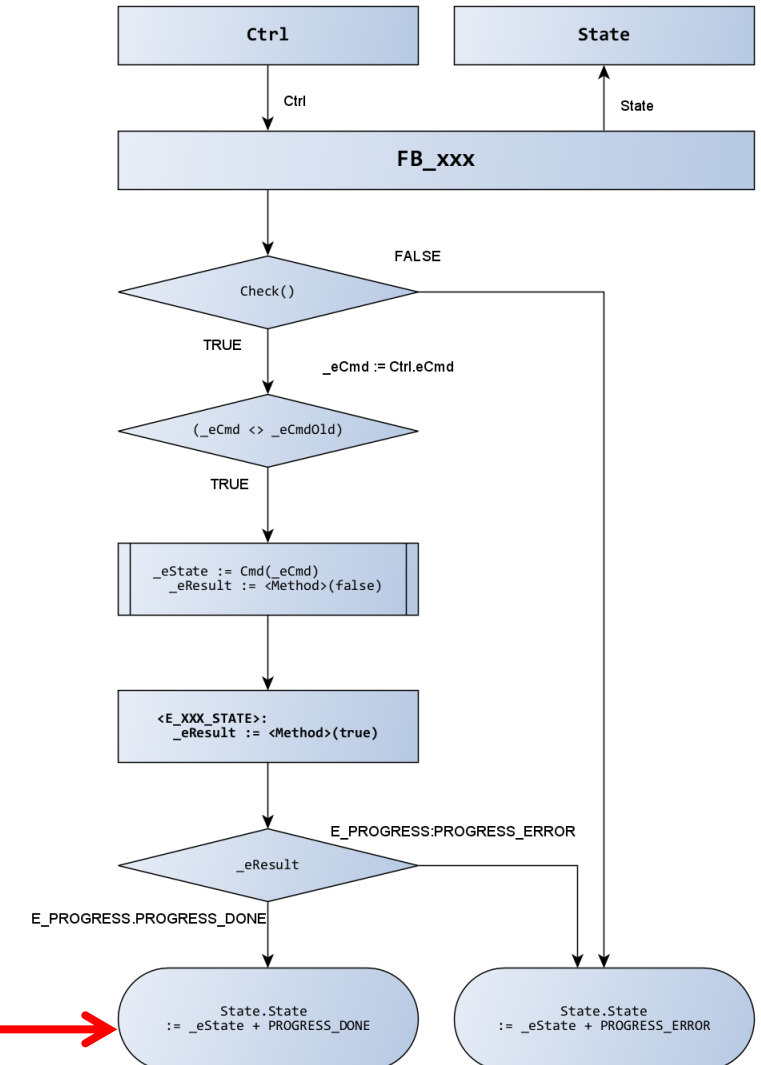
- Konform mit Entwurfsprinzipien SOLID und ACID
 - Selbstähnlichkeit des Entwurfes ermöglicht eine fraktale Architektur, die es ermöglicht auch aus den Zwischenschichten Funktionen zu verwenden, die die gleiche Sprache sprechen wie die oberste oder unterste Ebene.
 - Durch die rigide Anwendung der Trennung der Aufgaben, ist es möglich Module in harten Echtzeitbedingungen nahtlos einzubinden
- Starke Kapselung der finite Zustandsmaschinen anhand einer generischen Designvorlage
 - Diese generische Designvorlage ist für bestehende code segmente aufnahmefähig.
 - Code segmente können wiederverwendet werden.
 - Diese generische Designvorlage ermöglicht rückwirkungsfreie Systematik für komplexe Abläufe
 - Isolierung von Zuständen verhindert “race conditions” strukturell
- Verwendung von Basisklassen, die durch zyklische Wrapper erweitert werden (Ctrl/state)
- Querkommunikation mittels Interface Zeigern (pointer).

- **Design:**
- Kommunikationsschicht ist transparent
- Quellcodebasis der oberen Schichten ist unabhängig und isoliert.
- Konfiguration der Kommunikationsebene kann aus den verfügbaren Bibliotheken nahtlos eingebunden werden.
- Rechenlast kann einfach ausbalanciert werden.
- Gleichmäßiger Verbrauch an Rechenzeit unterstützt XFC Anwendungen

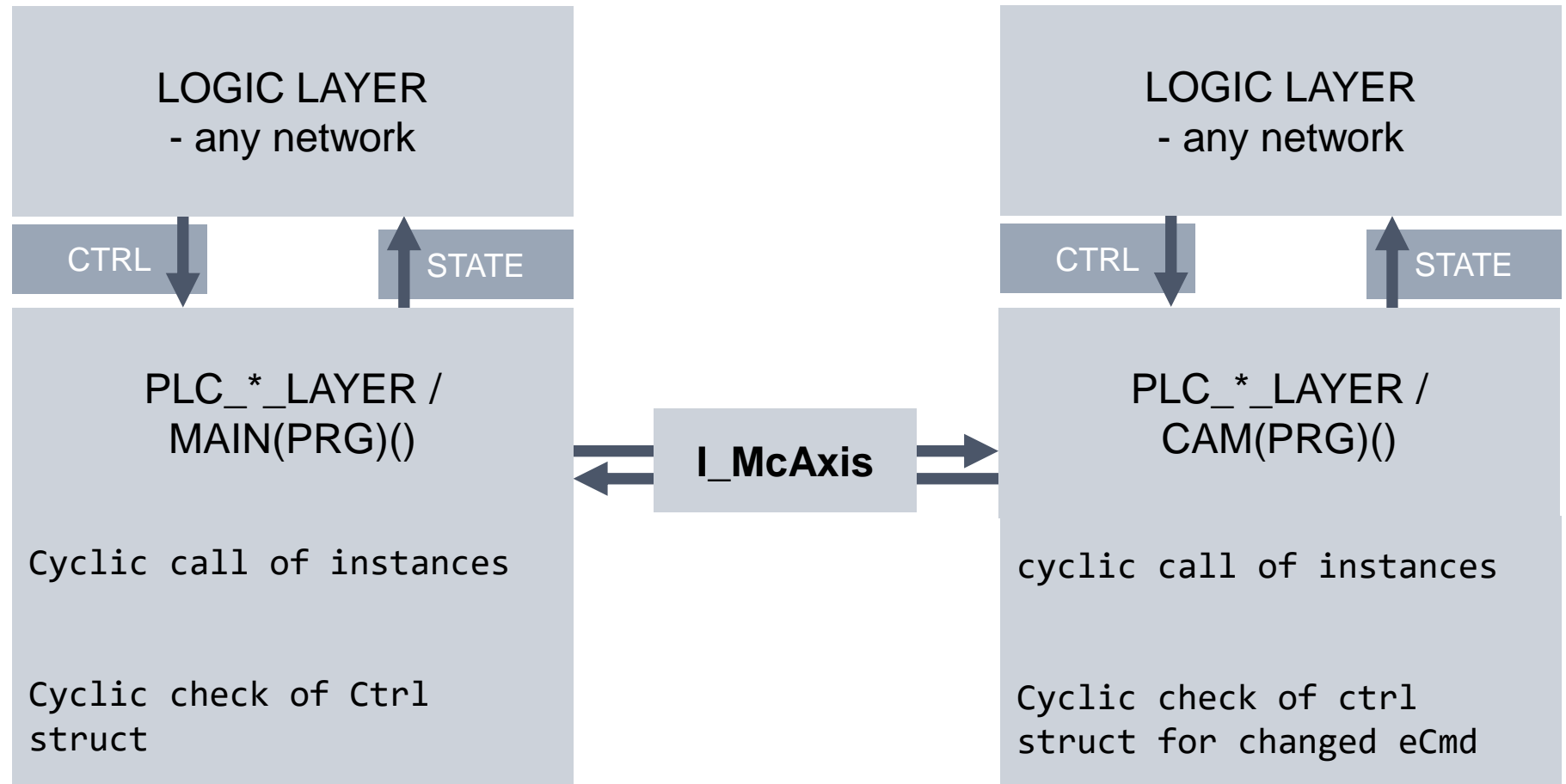


■ Design:

- State / Ctrl Strukturen
 - Ermöglicht einheitliche Kommunikation
 - Kommandos können im Ctrl Datenfach “abgeladen” werden
 - asynchrone Kommunikation mit PLC_*_LAYER
(z.B. C# via ADS, C/C++ vis ADS, ADS over MQTT, ...)
 - Zyklische Kommunikation mit PLC_*_LAYER
 - Strukturen können sehr einfach an alle zyklischen Protokolle, welche TwinCAT unterstützt, angebunden werden. (EtherCAT, Profinet, CanOpen, EAP, ...)
 - State wird zyklisch durch PLC_*_LAYER aktualisiert, so dass nach start einer Funktion die Abfrage des Zustandes asynchron erfolgen kann
- Zustandsrückmeldung
 - **IMMER** kombiniert mit E_PROGRESS



▪ Software Design Example: Separation of Concerns



▪ SOLID DESIGN

- **S — Single Responsibility Principle**
 - a class should have one, and only one, reason to change.
- **O — Open/Closed Principle**
 - software entities should be open for extension but closed for modification.
- **L — Liskov Substitution Principle**
 - Objects of a superclass shall be replaceable with objects of its subclasses without breaking the application.
- **I — Interface Segregation Principle**
 - users should not be forced to depend upon interfaces that they do not use.
 - a software entity only sees the methods it actually needs to the work.
- **D — Dependency Inversion Principle**
 - Depend upon abstractions, not concretions.

▪ ACID DESIGN

- **A — Atomicity**
 - A transaction is an "all-or-nothing" unit of work.
- **C — Consistency**
 - A transaction must bring the system from one valid state to another valid state.
- **I — Isolation**
 - Concurrent transactions should not interfere with each other.
- **D — Durability**
 - Once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.