TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Karl Kegel
karl.kegel@mailbox.tu-dresden.de

Verteilte Mensch-Roboter-Systeme: Ein Modellgetriebener Entwicklungsansatz

Verteidigung Bachelorarbeit, 6. Semester Bachelor Inf.

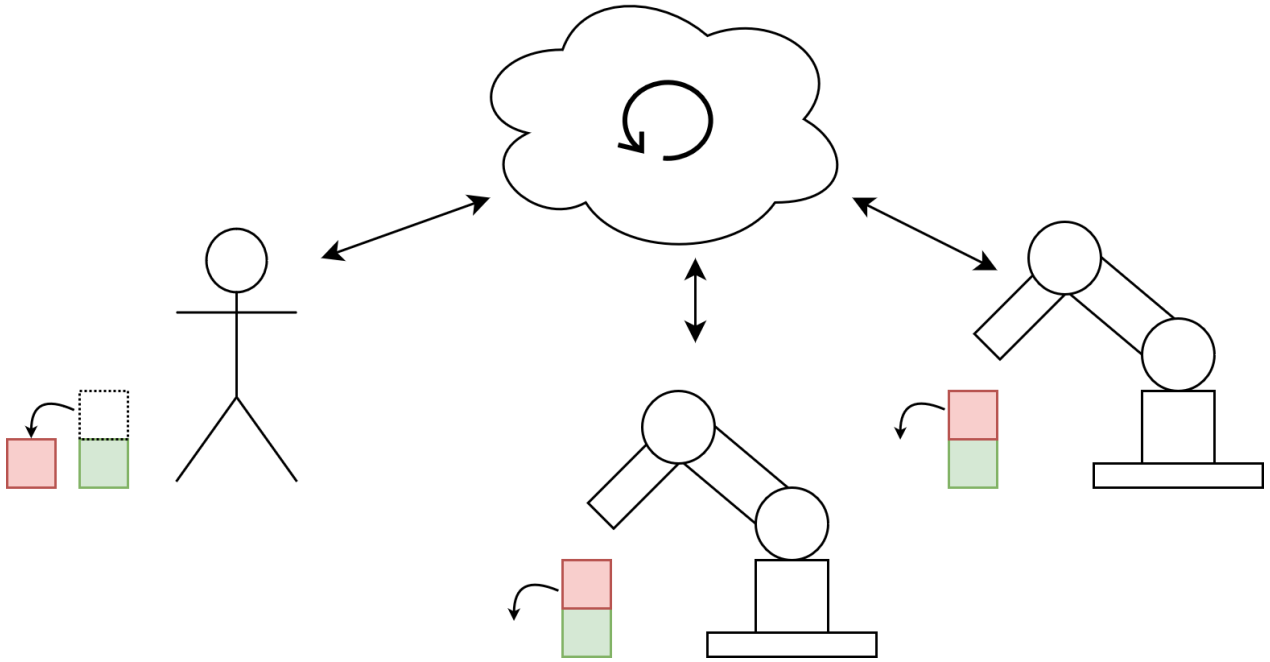
Dresden, Donnerstag, 15. August 2019

Motivation und Problemanalyse

Ein Verteiltes Mensch-Roboter-System

Merkmale

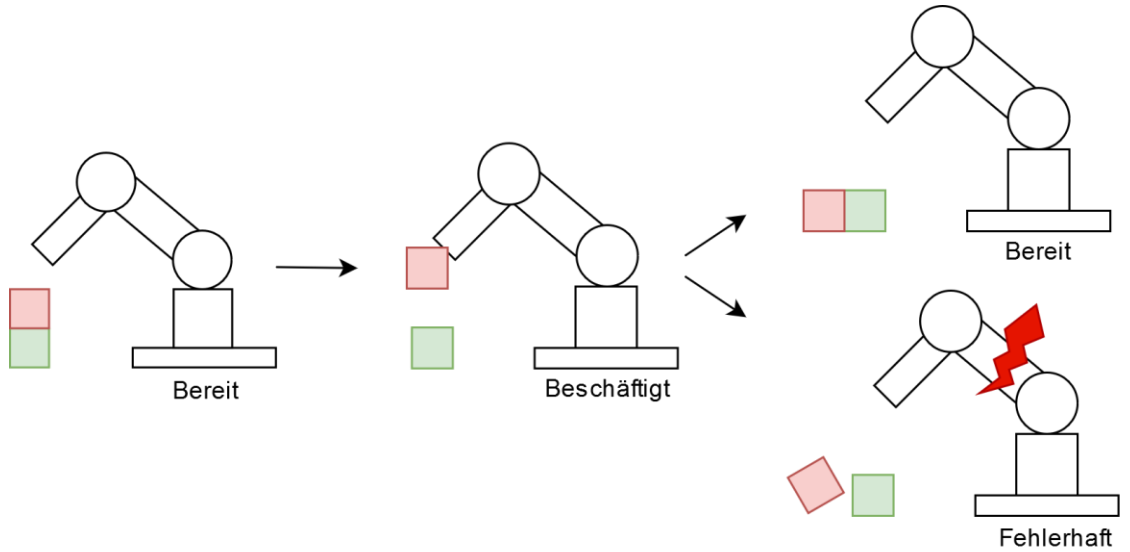
- Es gibt menschliche und robotische Arbeiter
- Arbeiter bzw. Arbeitsplätze sind vernetzt
- Menschen und Roboter kollaborieren direkt oder indirekt zum Erreichen eines gemeinsamen Ziels



Ein Verteiltes Mensch-Roboter-System

Merkmale

- Roboter können je Zeitabschnitt genau eine Tätigkeit ausüben
- Durchführung kann durch realweltliche Einwirkungen fehlerbehaftet sein
- Fehler eines Teilsystems kann zu Stillstand des gesamten Netzwerks führen



Problemanalyse

Probleme während der Entwicklung von Mensch-Roboter-Systemen

- Komplexes hardwaretechnisches Verhalten
- Ansteuerung über low-level APIs
- Viele verschiedenartige Teilsysteme
- Sehr viele mögliche Systemzustände

+ allgemeine Probleme der Softwareentwicklung

- Mangelhafte Expertise der Entwickler
- Lückhafte und veraltete Dokumentationen
- Vernachlässigtes Anforderungsmanagement

Problemanalyse

Daraus folgt

- Systemverhalten ist fehlerbehaftet
- Systeme sind nur schlecht wart- und erweiterbar

Die Entwicklung von verteilten Mensch-Roboter-Systemen ist komplex und fehleranfällig

Zielsetzung

Allgemeine Zielsetzung

Die Entwicklung von verteilten Mensch-Roboter-Systemen ist komplex und fehleranfällig

- Verringerung des nötigen Expertenwissens
- Vereinfachung von Architektur und Designentscheidungen
- Verbesserung der Interoperabilität

Die Entwicklung von verteilten Mensch-Roboter-Systemen erfolgt intuitiv und robust

Allgemeine Zielsetzung

Daraus folgt

- Das Systemverhalten ist fehlerarm
- Systeme sind gut wart- und erweiterbar

Aber

- ➢ Eine Lösung für alle verteilten Mensch-Roboter-Systeme zu finden ist nicht möglich
- ➢ Beschränkung auf die Lösung einer Teilklasse

Spezifikation der Zielsysteme

Menschzentrierte Arbeitsplätze

- Mensch dauerhaft anwesend
- Mensch führt Aktionen aus

- Sensoren erkennen Aktionen
- Aktoren geben Feedback
- Controller steuert Abläufe

- Aktionen lösen Steuerbefehle/Nachrichten aus

Roboterzentrierte Arbeitsplätze

- Mensch nicht dauerhaft anwesend
- Roboter führt Aktionen aus

- Sensoren überwachen Ausführung
- Controller steuert Abläufe

- Roboter befolgt Steuerbefehle
- Arbeitsplatz sendet Feedback

Gewählte Vorgehensweise

Modellgetriebene Softwareentwicklung (MDSD) mit UML Statecharts

- Vorteile der MDSD
  - Abstraktion
  - Einheitliche Architektur
  - Erhöhte Entwicklungsgeschwindigkeit
  - Wiederverwendbarkeit
  - Plattformunabhängigkeit [SVEH07]

- Vorteile von UML
  - Weite Verbreitung und Akzeptanz
  - Gute Unterstützung durch Tools

Eignung von Statecharts?

Gewählte Vorgehensweise

Modellgetriebene Softwareentwicklung (MDSD) mit UML Statecharts

1. Analyse der Zielsysteme
2. Definition einer DSL mittels UML-Profilen
3. Spezifikation der Zielarchitektur und Modell-Code-Abbildung
4. Entwicklung eines Programmgenerators

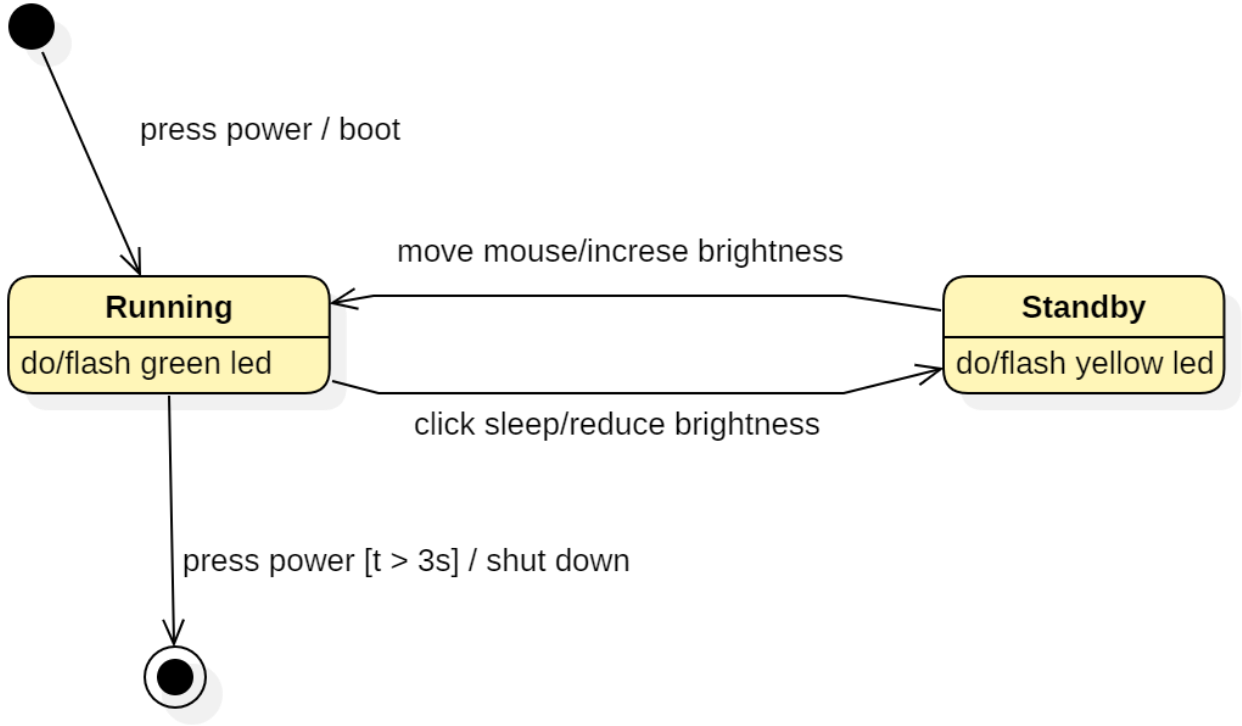
5. Einsatz der Techniken zur Erstellung eines Systems 

Grundlagen

Statecharts

Merkmale

- Zustände als Boxen
- Transitionen als Pfeile
- Definiertes Verhalten durch Trigger-Events, Guards und Effect-Behaviour

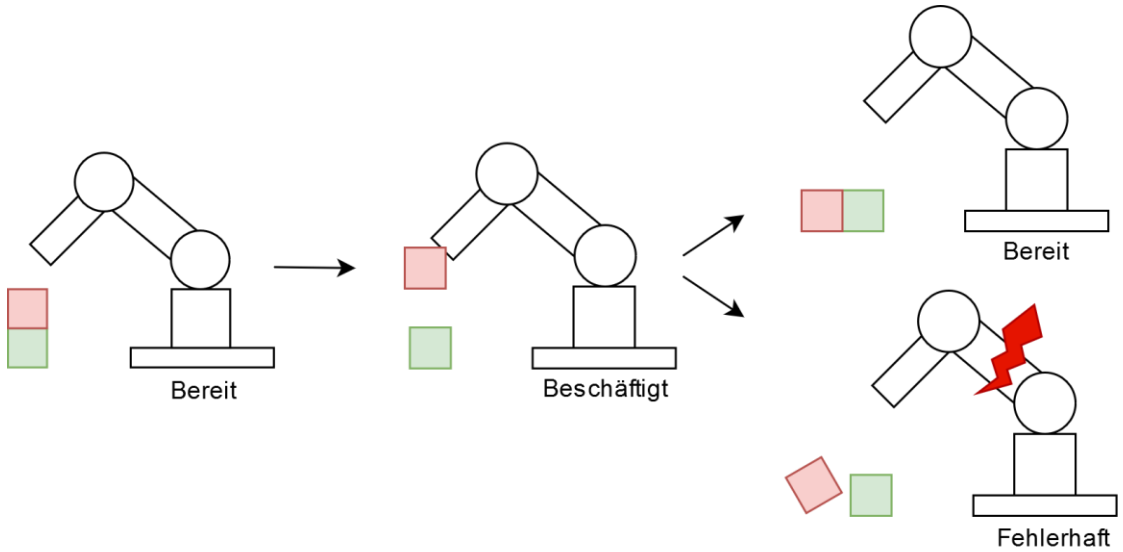


UML Statecharts entsprechen weitestgehend Der Definition nach Harel [HA87]

Statecharts

Semantik

- Zustände der Statechart entsprechen direkt Zustände der Hardware
- Transitionen der Statechart entsprechen Geschäftsregeln des Systems



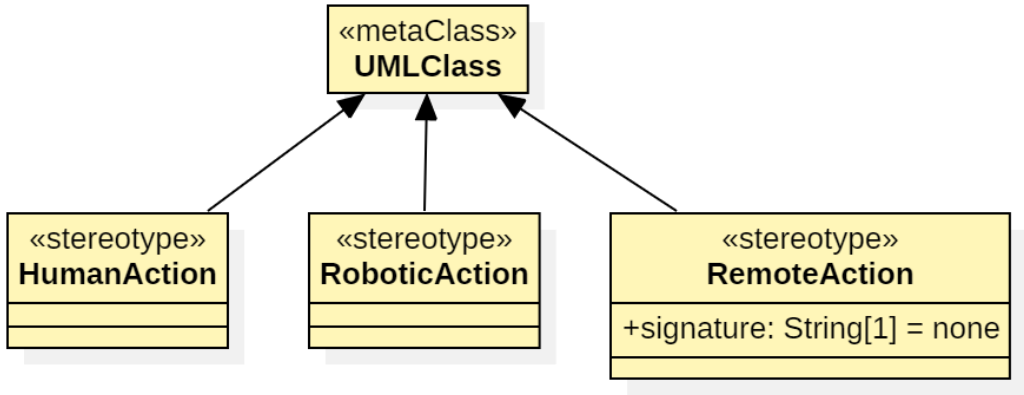
UML als Basis von DSLs

Ziel

- Nutzung der positiven Eigenschaften von UML
- Erhalten einer aussagekräftigen DSL

Vorgehen

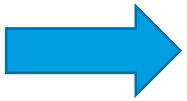
1. Aufstellen domänenspezifischer Klassen/Objekte etc.
2. Spezifikation eines UML-Profils
3. Nutzung der DSL innerhalb einer UML Umgebung



Elaboration der modellgetriebenen Methode

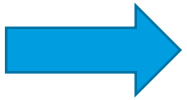
Die plattformunabhängige Architektur

- Erteilen von Steuerbefehlen
- Reagieren auf Feedback
- Einhaltung von Geschäftsregeln



- Event-Condition-Action (ECA) Systemarchitektur

- Einzelne Arbeitsplätze
- Verschiedene Standorte
- Austausch von Nachrichten



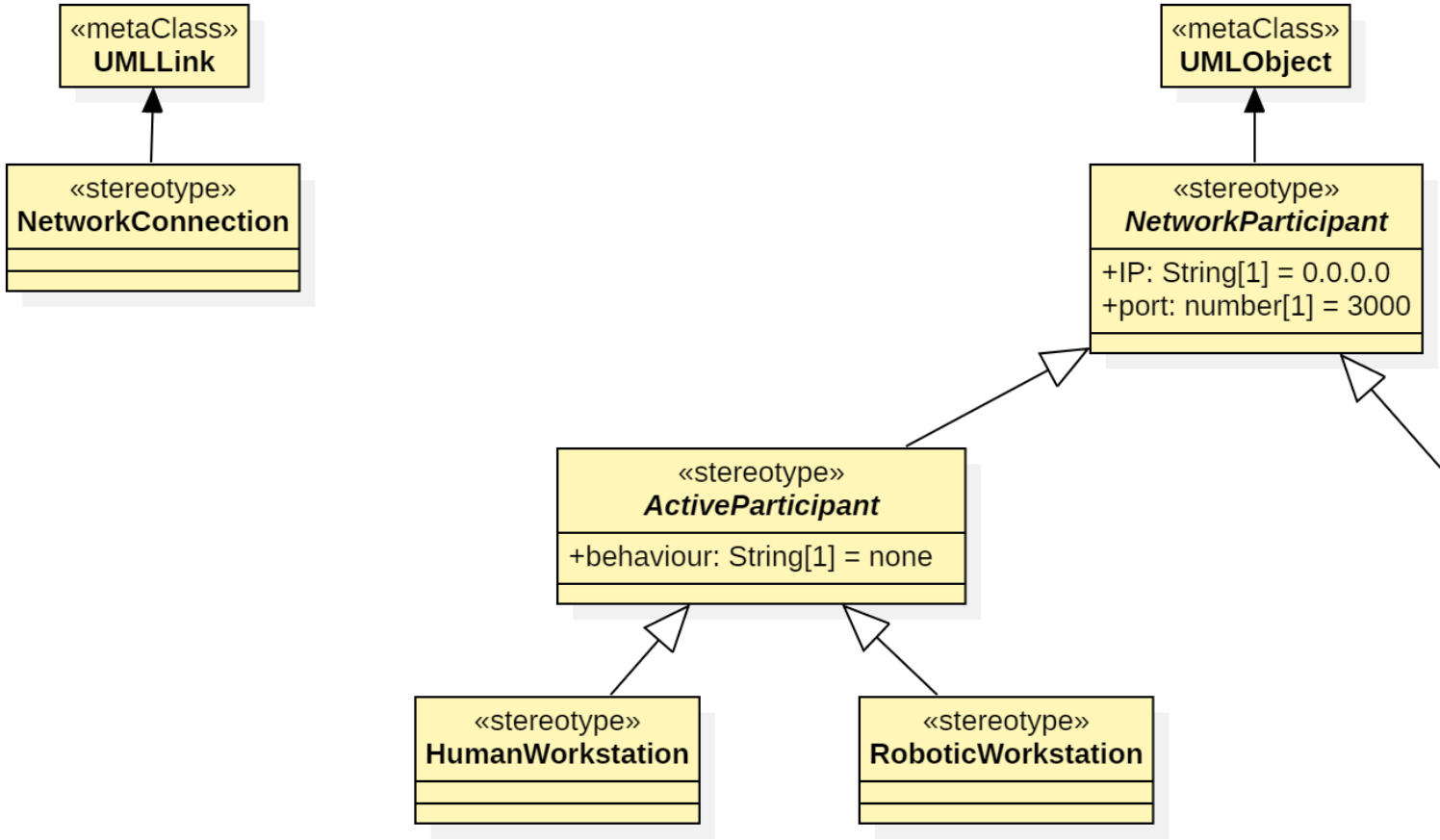
- Client-Server Netzwerkarchitektur
- Server als Broker

Aufstellen der DSL - Topologie

Modellieren von

- HumanWorkstations
- RoboticWorkstations
- Nodes
- NetworkConnections

In Form eines Objektdiagramms

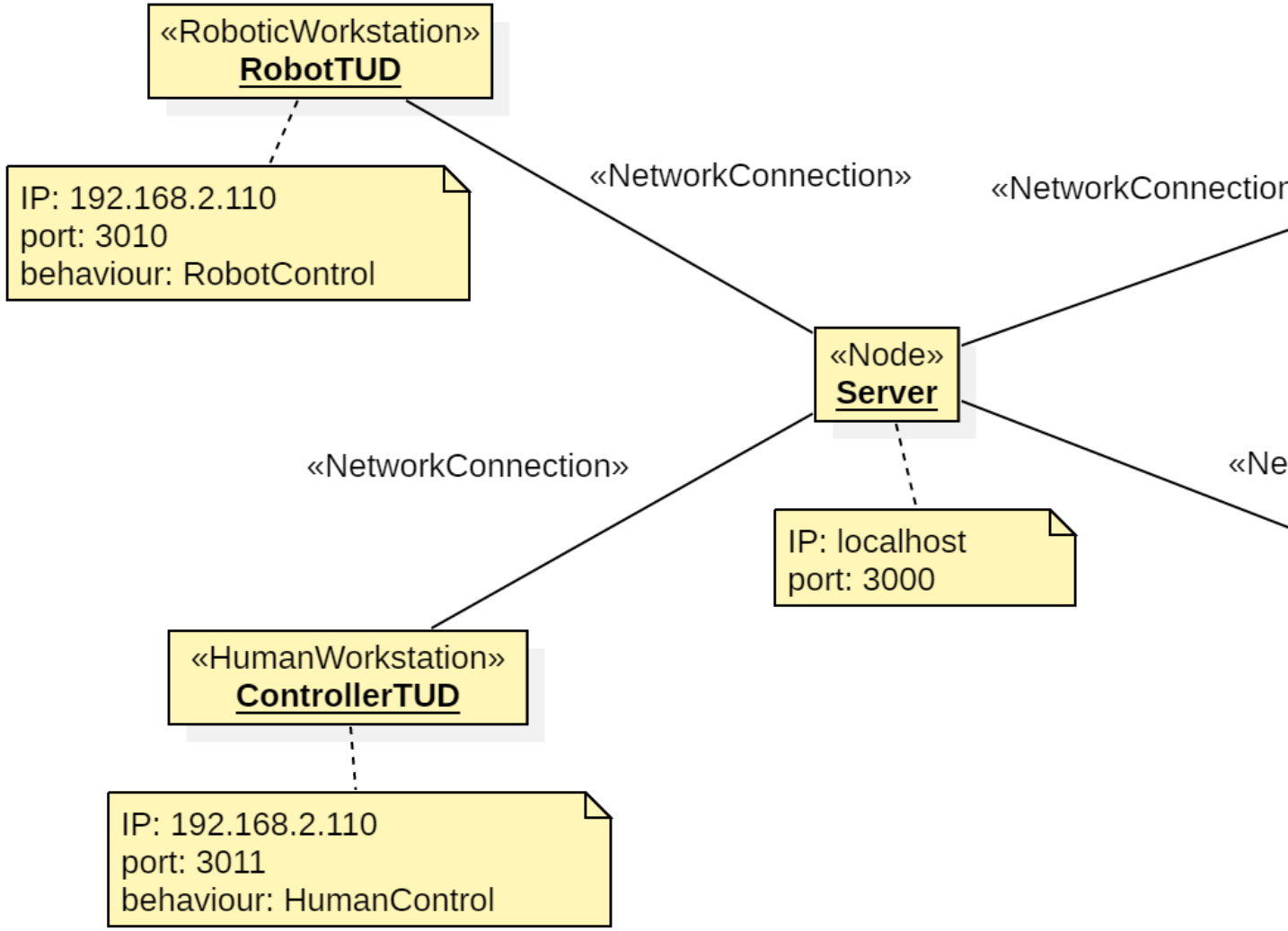


Aufstellen der DSL - Topologie

Modellieren von

- HumanWorkstations
- RoboticWorkstations
- Nodes
- NetworkConnections

In Form eines Objektdiagramms

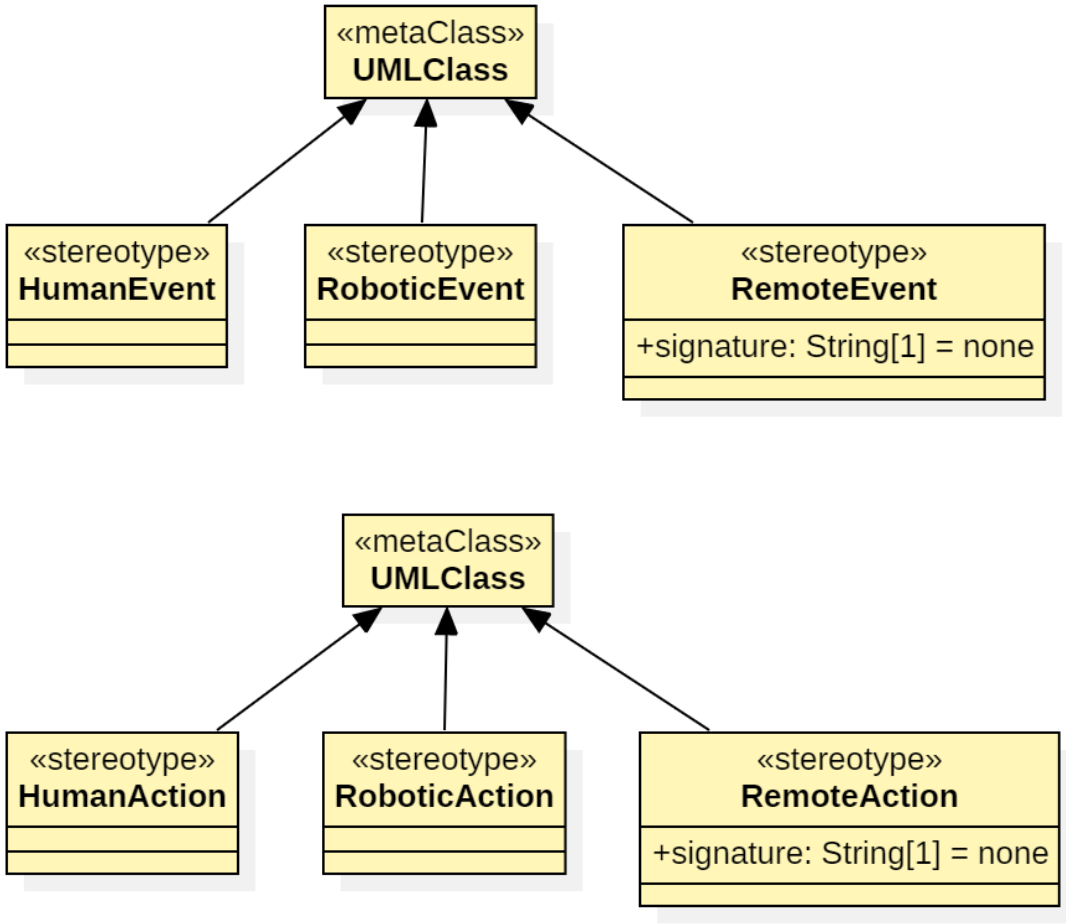


Aufstellen der DSL - Nachrichten

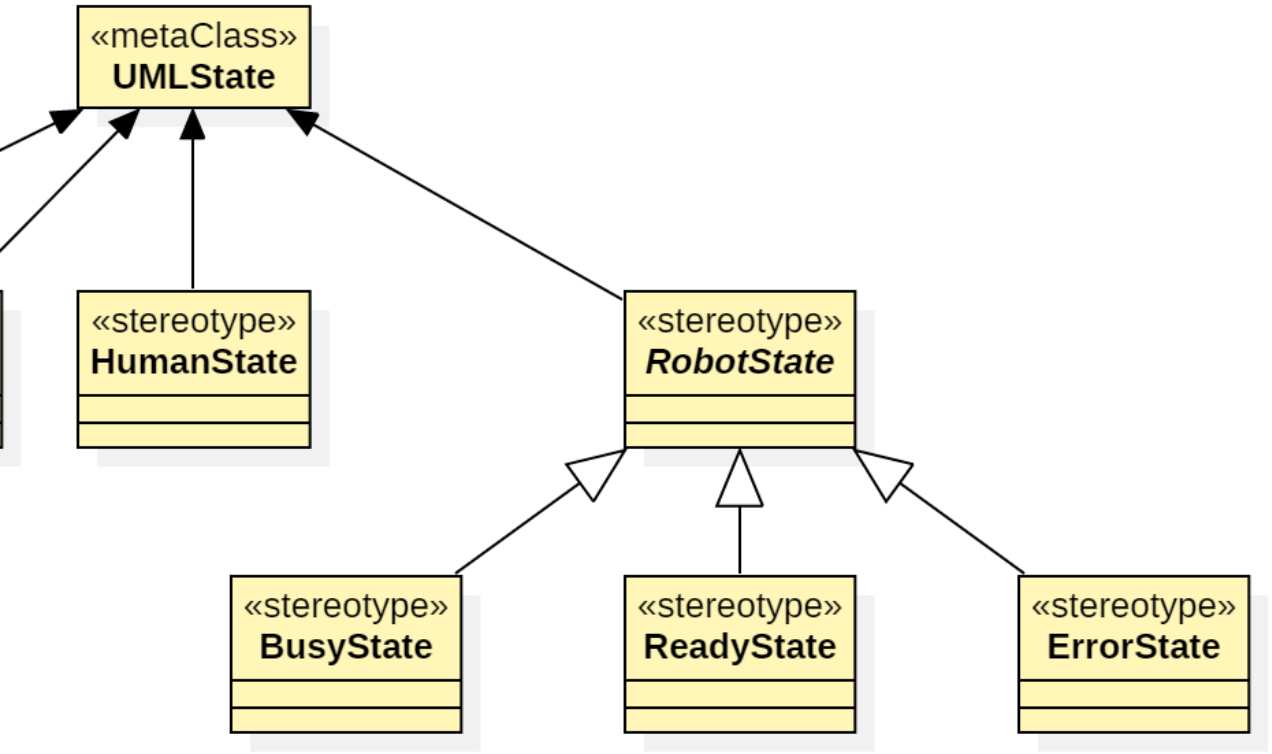
Modellieren von

- Events
- Actions

In Form von Klassendiagrammen



Aufstellen der DSL - Verhalten



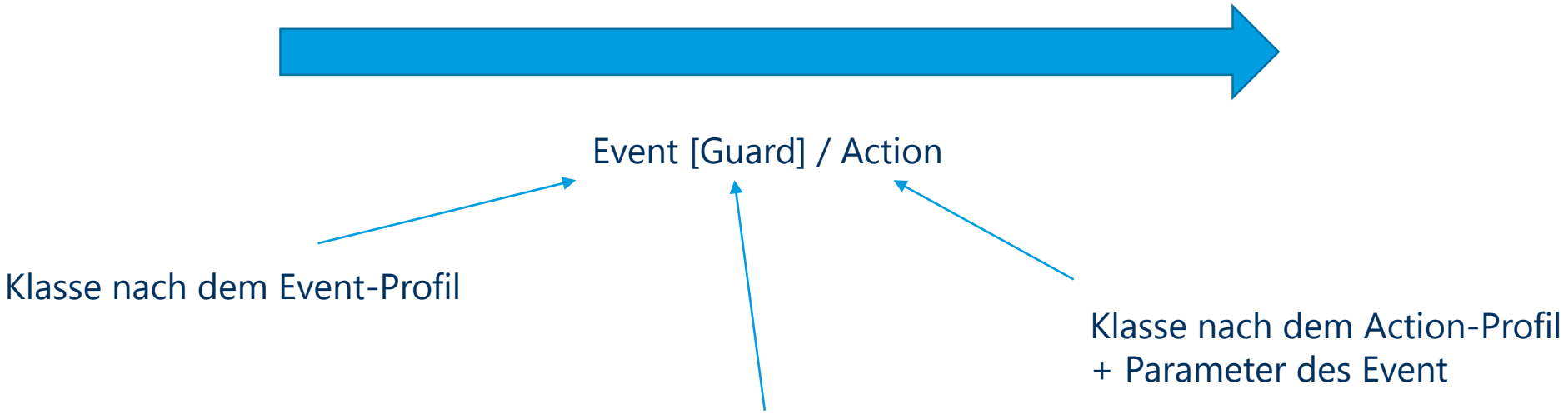
Modellieren von

- Zustände der HumanWorkstations
- Zustände der RoboticWorkstation
- Regelbasierte Transitionen

In Form von Statecharts

➢ Regeln für Transitionen außerhalb des Profils

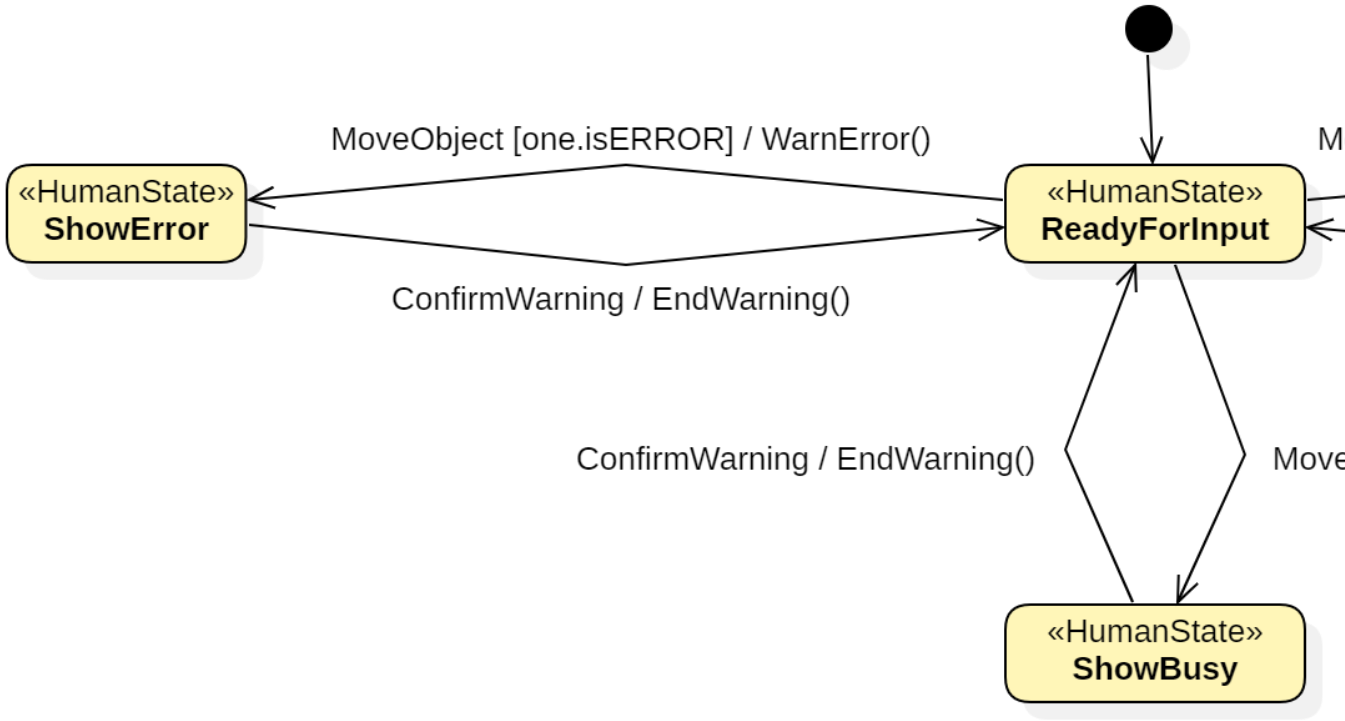
Aufstellen der DSL - Verhalten



Aufstellen der DSL - Verhalten

Transitionen modellieren Geschäftsregeln

1. Wenn Event eintritt
2. [und Bedingung erfüllt ist]
3. führe Aktion aus.



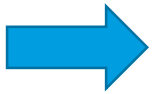
Modell zu Zielplattform Abbildung

- Event-Condition-Action (ECA) Systemarchitektur
- Client-Server Netzwerkarchitektur
- Server als Broker

+

- Hohe Plattformunabhängigkeit
- Einfache Inbetriebnahme
- Verfügbarkeit von Frameworks und Treibern
- Einfache Einbindung bestehender Systeme

- Vernachlässigung von Hardware/Software-Aufwand



Zielplattform: NodeJS mit TypeScript

- Verfügbar für alle Betriebssysteme
- Fokus auf Netzwerkanwendungen
- Einfache Nutzung von Sockets
- Vielzahl von Frameworks
- Auslieferung als Skript

Client-Server System

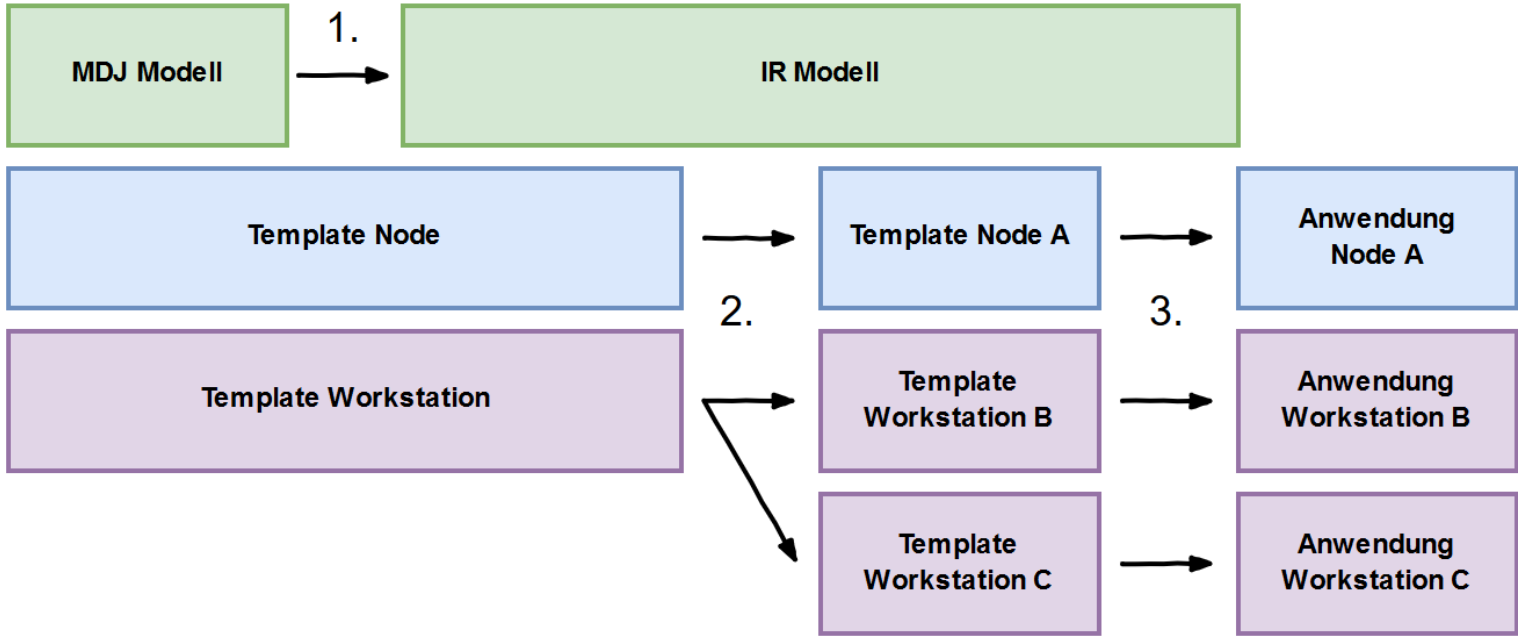
Kommunikation via TCP Sockets

Modell zu Code Abbildung

Topologie	→	Eigenständige Anwendung für jede Workstation/Node	
Statecharts	→	Verwendung des State-Patterns	[GA05]
Events/Actions (lokal)	→	Lösen Methoden des State-Patterns aus	
Events/Actions (remote)	→	Lösen Nachrichten aus, werden vom Broker zugestellt	
Globaler Zustand	→	Synchronisation durch Broker	

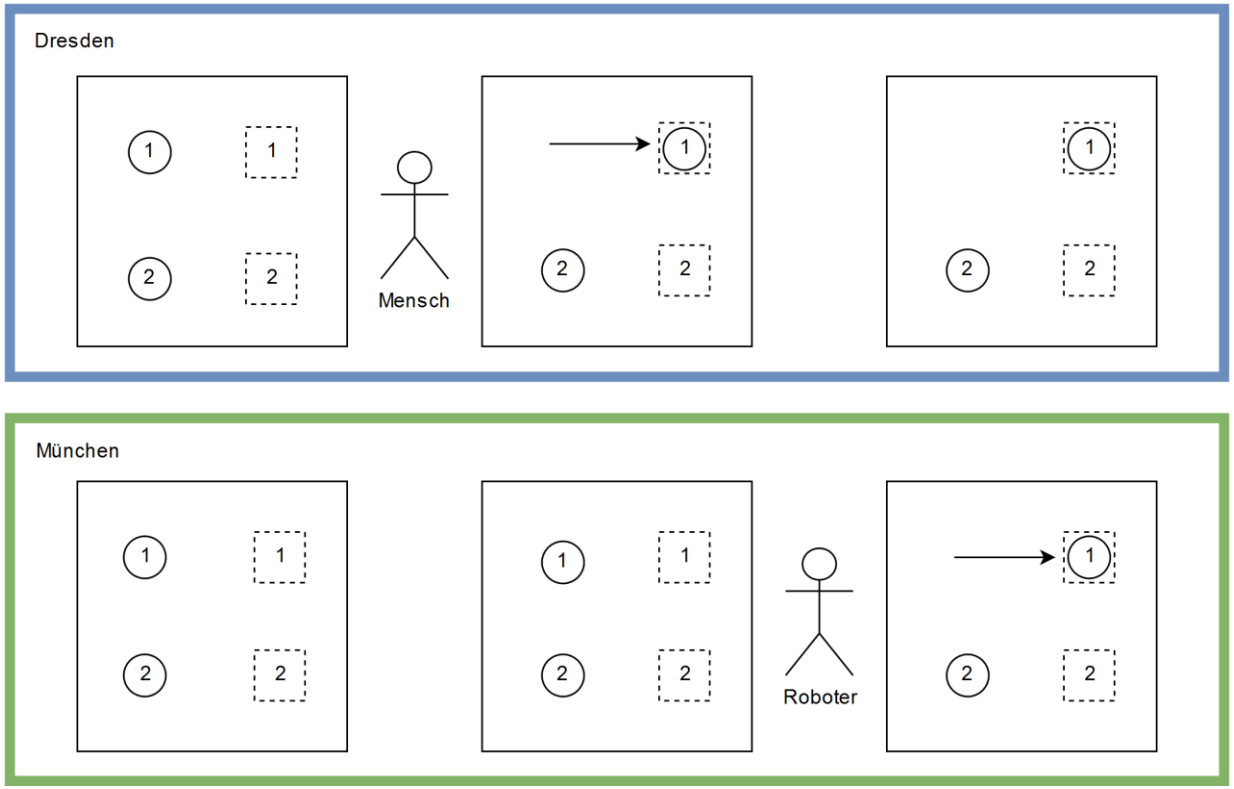
Codegenerierung

1. **Modell einlesen**
2. **Topologie umsetzen**
3. **Templates ausfüllen**



Fallstudie

Fallstudie



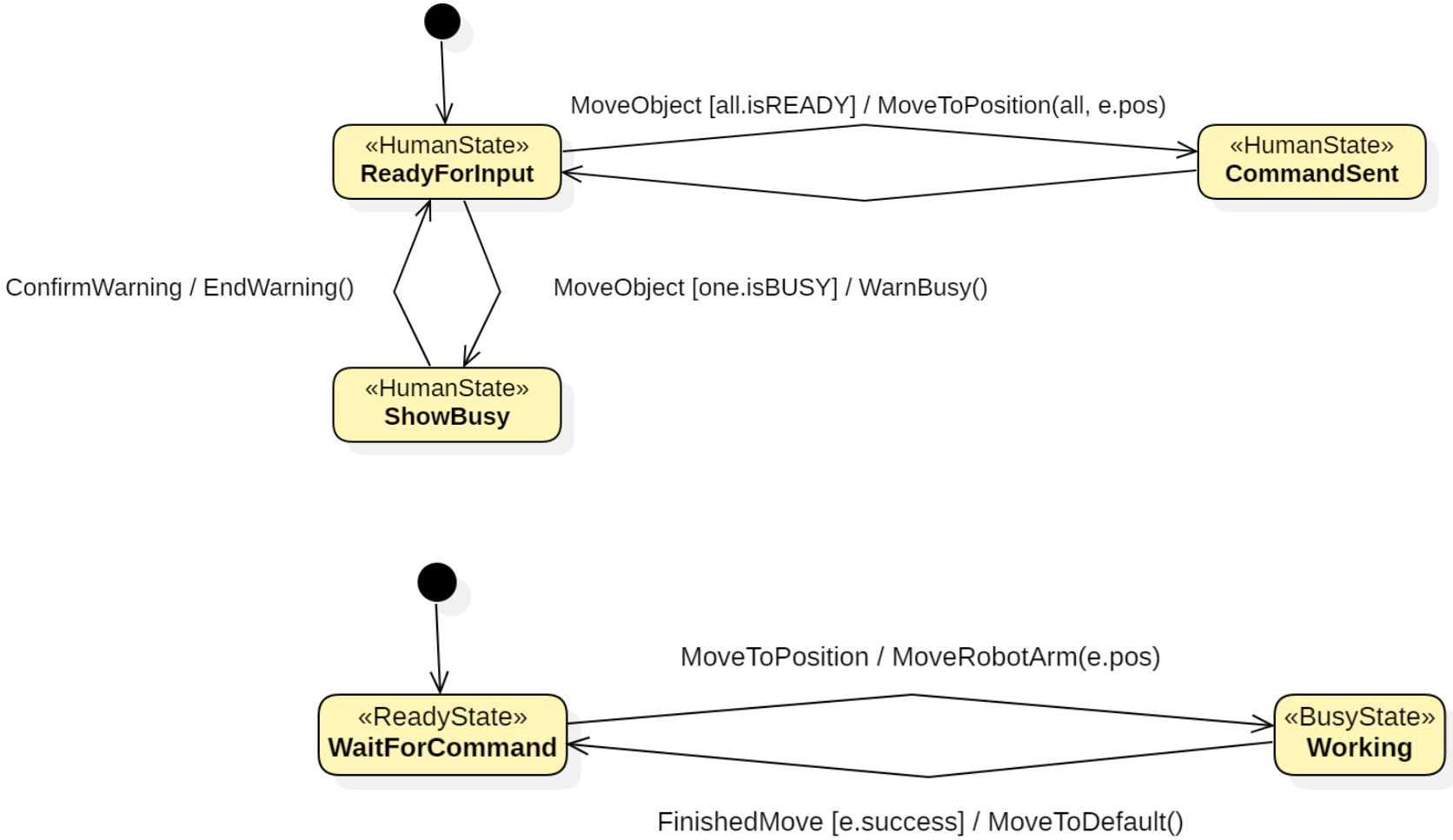
Angelehnt an:

„Deutschlanddemo 50 Jahre Informatik“

1. Mensch bewegt Werkstück
2. Sensoren erkennen neue Position
3. Roboter erhält Befehl
4. Roboter führt gleiche Bewegung aus
5. Arbeitsplätze sind synchron

Fallbeispiel

Produktive Events

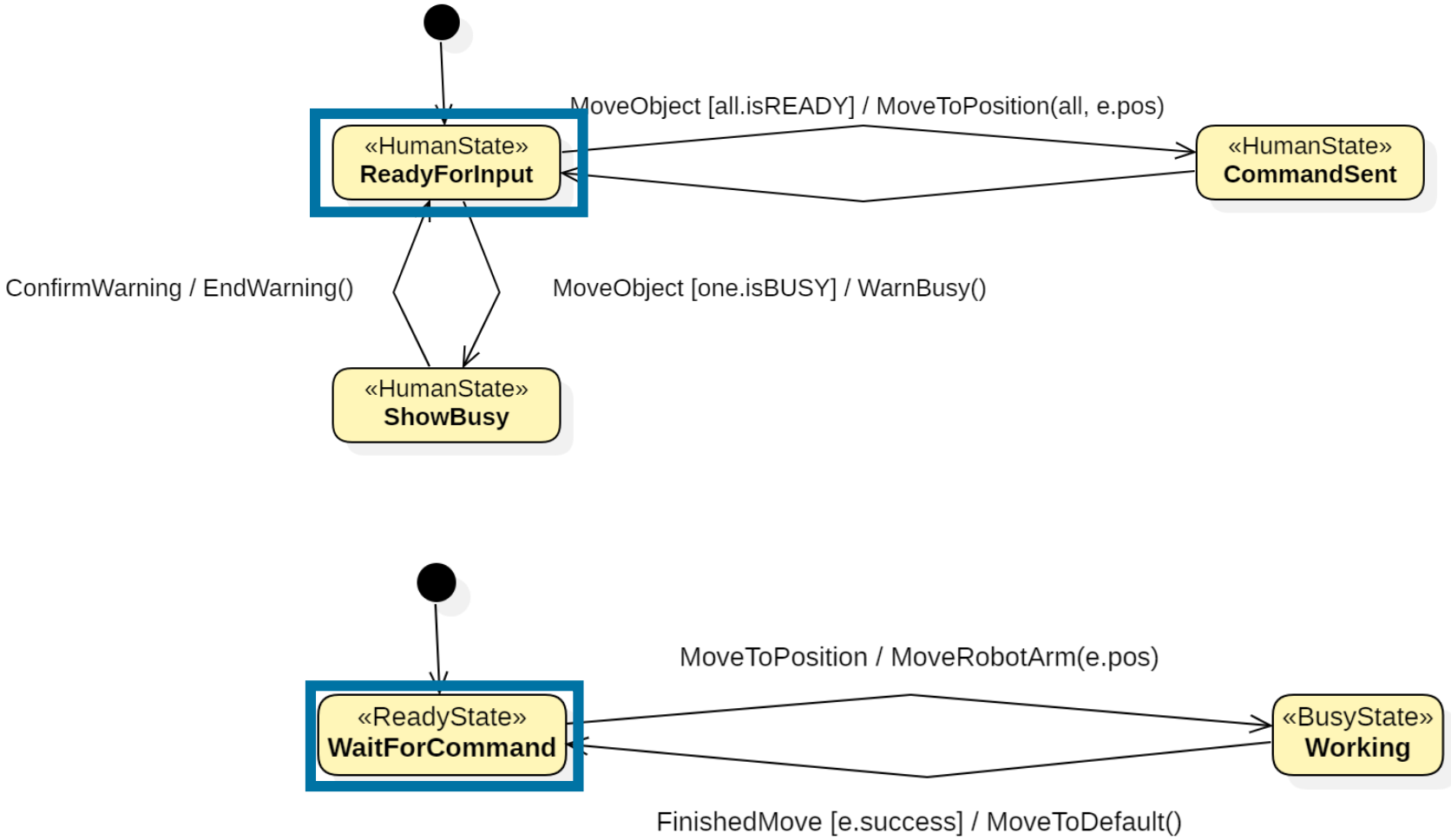


Weiter

Fallbeispiel

Produktive Events

- MoveObject

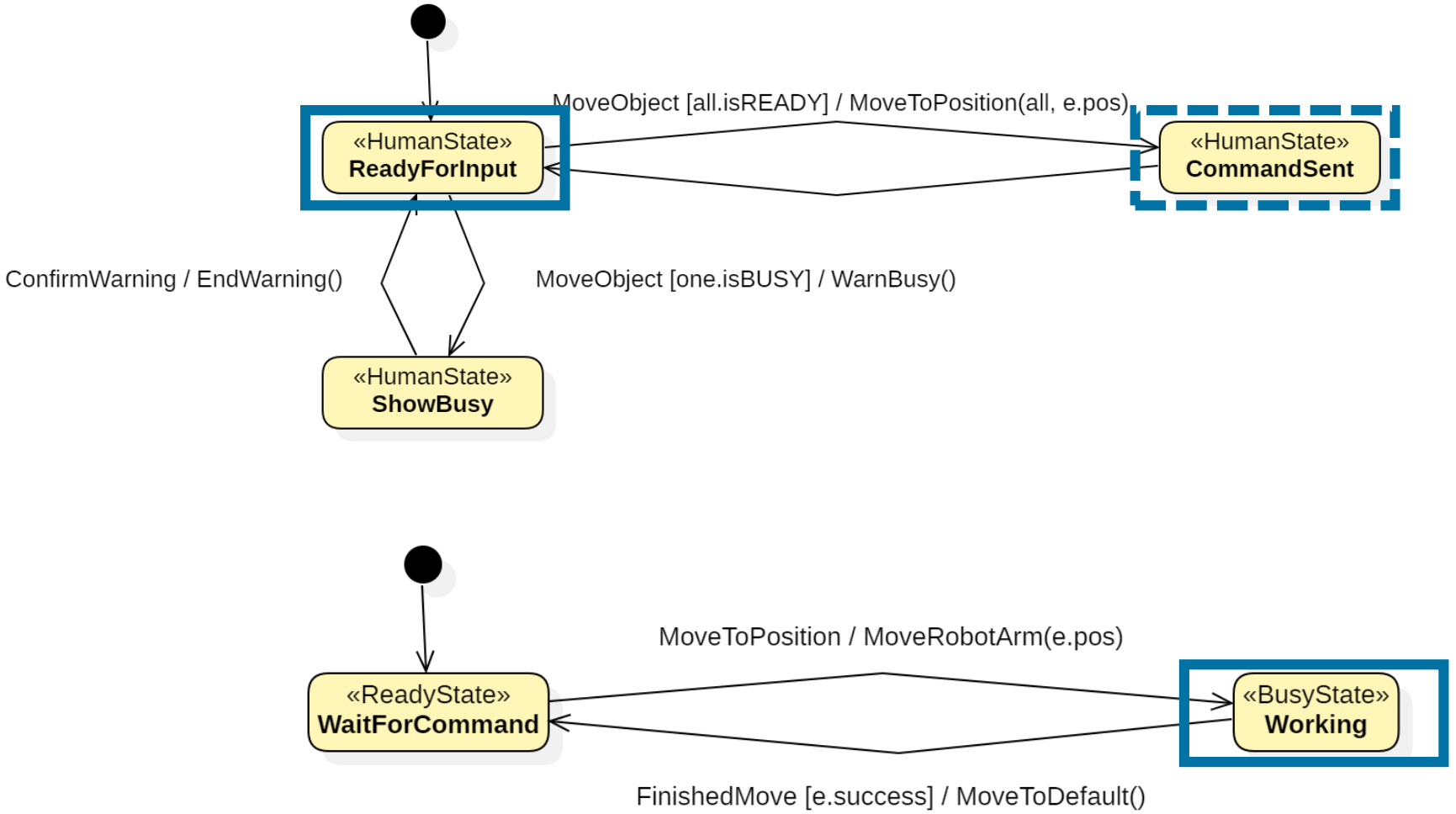


Weiter

Fallbeispiel

Produktive Events

- MoveObject
- FinishedMove

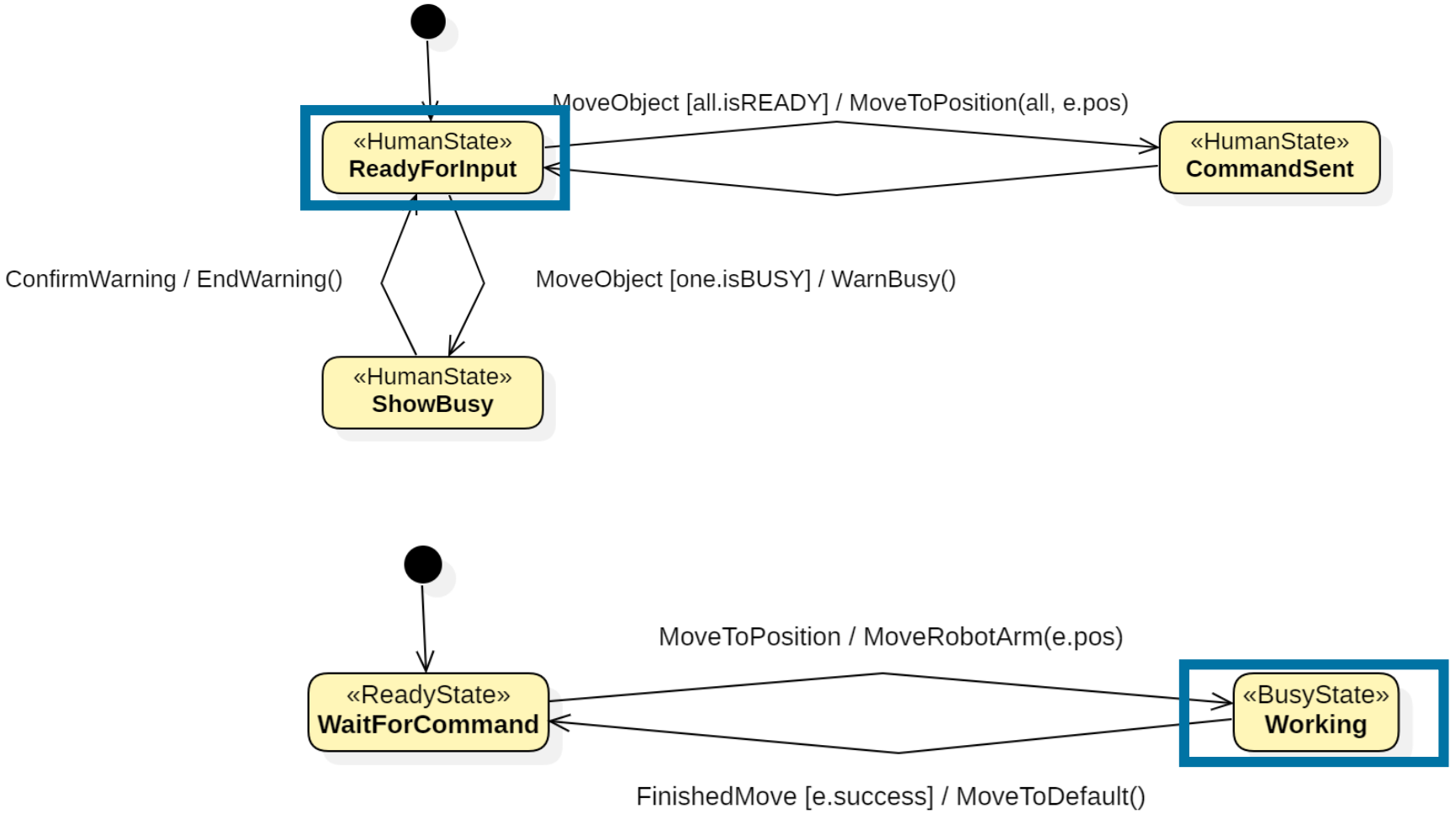


Weiter

Fallbeispiel

Produktive Events

- MoveObject
- FinishedMove

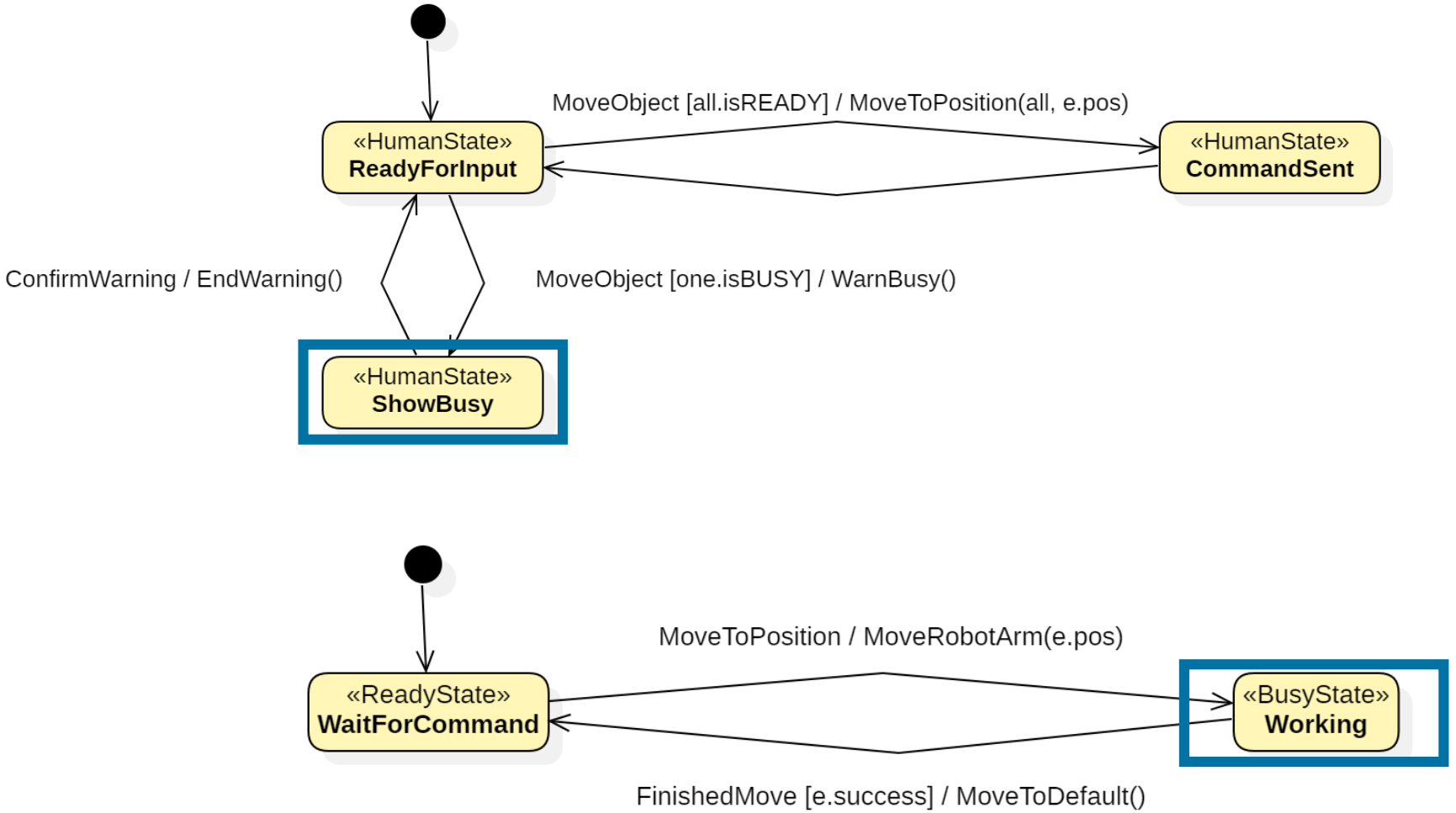


Weiter

Fallbeispiel

Produktive Events

- ConfirmWarning
- FinishedMove

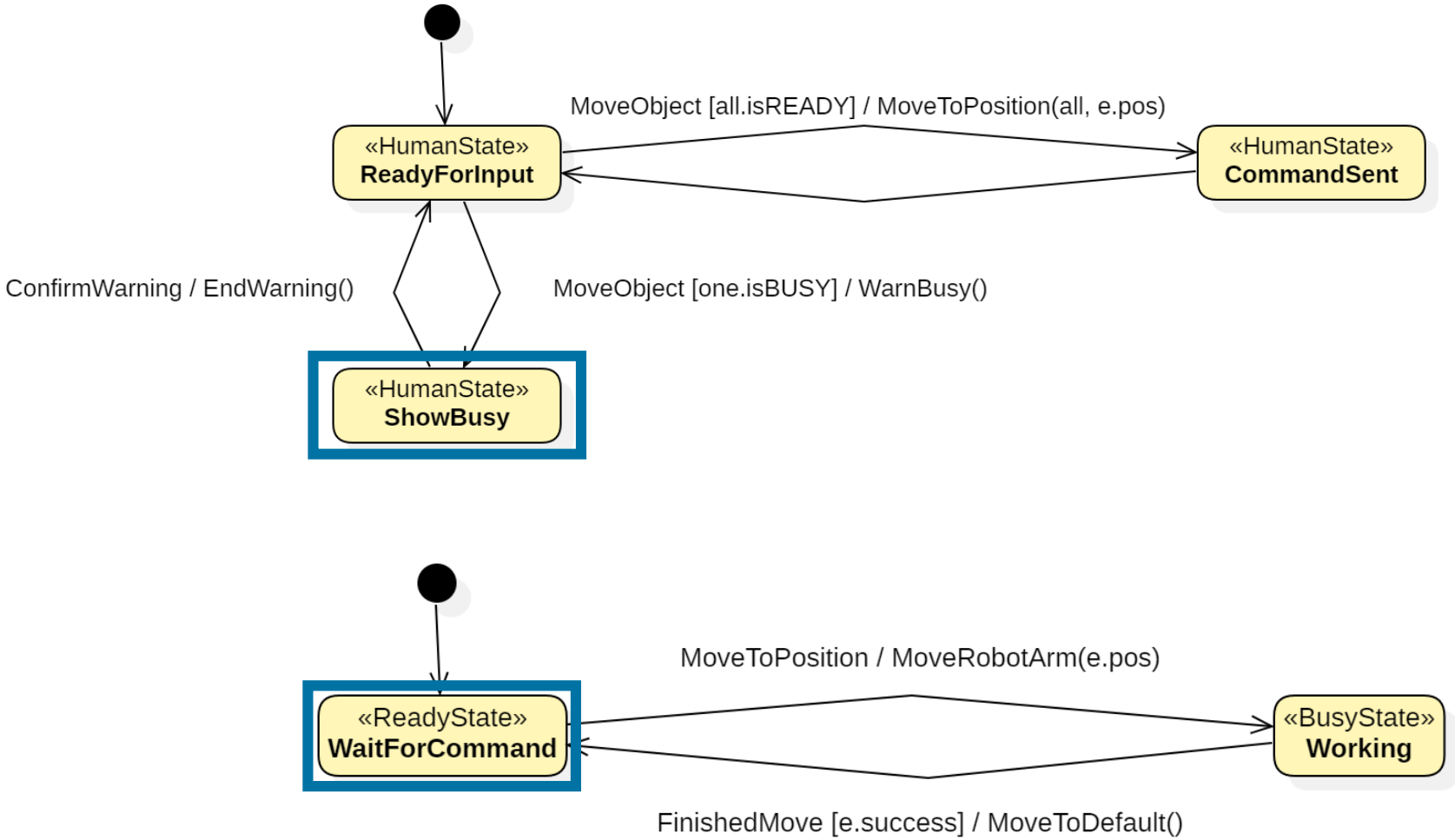


Weiter

Fallbeispiel

Produktive Events

- ConfirmWarning



Weiter

Evaluation

Evaluation

Vorteile durch…

Elaboration des MDSD

- Durchdachte Architektur
- Wart- u. Erweiterbarkeit per Design
- Vorgaben zur Modellierung
- Nachvollziehbare Entscheidungen
- Sauberer Programmcode durch Generierung

Nutzung des MDSD

- Schnelle Entwicklung
- Wenig Wissen zur Technologie nötig
- Kompatibilität/Interoperabilität
- Einfache Veränderbarkeit
- Vermeidung von Programmierfehlern

Evaluation

Herausforderungen

- Gute Technologiekenntnisse für Architektur erforderlich
- Breites Wissen zu Frameworks/Plattformen nötig
- Hohe Anforderungen an Programmgenerator

Es war ein iteratives Vorgehen unabdingbar!

Weiterführende Arbeiten

- Generierung von Tests bzw. Testfällen
- Erweiterung der DSL
- Behandlung von Echtzeitproblemen und Anforderungen

Fazit

Fazit

Interoperabilität durch…

- Einheitliche Architektur
- Einheitliches Modell

- Systemteile interoperieren fehlerfrei
- Architektur ist einheitlich
- Ein Modell für ein gesamtes System

Reduktion von Expertenwissen durch…

- Einfach verwendbare DSL
- Wiederverwendbare Modelle
- Hoher Grad an Abstraktion

- Die DSL lässt sich einfach und schnell nutzen
- Modellteile können problemlos wiederverwendet werden
- Netzwerktechnologie wird durch Modell verborgen

Durch den Einsatz der MDSD und einer DSL basierend auf Statecharts, wurden die gesetzten Ziele erreicht.

Literatur

[GA05] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley, Boston, München, 32. Auflage, 2005 ISBN 978-0-201-63361-0 (siehe S. 305-310)

[SVEH07] Thomas Stahl, Markus Völter, Sven Eftinge, Arno Haase Modellgetriebene Softwareentwicklung Techniken, Engineering, Management, dpunkt.verlag, 2.Auflage, Heidelberg 2007, ISBN 978-3-89864-448-8 (siehe S. 1 ff., Definition Kapitel 2.1)

[HA87] David Harel Statecharts: A Visual Formalism for Complex Systems, (S. 231-274), Science of Computer Programming, Elsevier, June 1987, https://www.sciencedirect. com/science/article/pii/0167642387900359 (siehe S. 1 ff.)