# Plant Sim-Modell

Erläuterung zu einigen Begriffen:

Episode Abarbeitung einer Lieferliste (umfasst mehrere BEs)

Durchlauf Abarbeitung mehrerer Episoden bis zur festgelegten Episodengrenze. Am Ende eines Durchlaufs werden die Daten exportiert und die Simulation resettet.

Gruppe Relevant bei Verwendung von Sortierern.

Information, wodurch die Methode aufgerufen wird

Simulationsrelevante Methoden

Methoden/Funktionen auf oberster Ebene

Ereignisverwalter

* Keine spezielle Methode
* Wird von Datenerfassung, Datenerfassung 1 oder Überprüfung\_Maschine(\_Herkunft) gestoppt
* Wird von Datenerfassung, Datenerfassung1, Steuerung, EndSim oder Überprüfung\_Maschine(\_Herkunft) gestartet

Init

* Wird bei Start jeder Episode aufgerufen
* Aktiviert Sockets
* Ändert einige Maschinenparameter nach Hälfte der Episodenanzahl
* Übermittelt Daten über die vorliegenden Maschinen an SocketOut (für Übermittlung an Jade)
  + Maschinennamen
  + Stammdaten
* Aktiviert Störungen bei den Maschinen
* Einstellung der Tabellen
  + Wenn Episode = 1, werden Durchlaufzeiten, Gesamtübersicht, BE\_Gruppen, Sankey-Tabelle, Visualisierungstabellen, Statistiken, Tabelle mit Bearbeitungsreihenfolgen etc. zurückgesetzt und ggf. neu initialisiert
* ZeitBisBearbeitung zurücksetzen
* Für das Beenden der Simulation relevanter Teil

EndSim

* Wird bei Ende jeder Episode aufgerufen
* Daten aus Tabellen auslesen, um Statistiken zu erstellen
* Maschinen aus Arbeitsgang-Tabelle in Attribut-Explorer überführen
* Nur wenn Reihenfolgebildung aktiv(!):
  + Schleife über alle Maschinen:
    - **Reward für jede Maschine berechnen!**
    - Übergabe des erzielten Rewards an SocketOut (für Übermittlung an Jade)
* Tabelle für Gesamtübersicht füllen
* Durchlaufzeit reseten
* Starte neue Episode, wenn letzte Episode noch nicht erreicht ist (Episode += 1)
* Wenn letzte Episode erreicht
  + Reset des Modells
  + Datenexport nach Excel
  + Erhöhe Durchlauf (Durchlauf += 1)

SocketOut

* Übermittelt Daten an Jade

SocketIn

* Erhält Daten von Jade
* Ruft Methode „Steuerung“ auf

Datenerfassung und Steuerungen

Steuerung-Methode

*Verarbeitet Eingabewerte von Jade*

* Wird von SocketIn aufgerufen
* Decodiert die über SocketIn eintreffenden Strings
* Öffnet ggf. die FehlermeldungVerfügbarkeit
* Führt Anweisung aus:
  + Lagert BE zu Lager/Sortierer um
    - Wenn Reihenfolgenbildung aktiv, wird Überprüfung\_Maschine und (falls Herkunft != Quelle) Überprüfung\_Maschine\_Herkunft aufgerufen
    - Startet Simulation wieder
  + Setzt Ergebnis der Reihenfolgebildung um:
    - Falls Lager\_Objekt[1,BE\_Lagerplatz] = void
      * **Lager\_Objekt.reward\_1 = -10**
      * Ruft Datenerfassung\_Lager auf
    - sonst
      * **Lager\_Objekt.reward\_1 = 10**
      * Stellt Zeiten der Zielmaschine ein
      * Lagert BE aus Lager zu Maschine um
      * Startet Simulation wieder

Datenerfassung

* Wird von den Maschinen und der Quelle beim Ausgang oder der BE\_Ausgang-Methode aufgerufen
* Stoppt die Simulation
* Falls Station != Quelle
  + Notiert Reihenfolge der BEs auf der Maschine
  + Aktualisiert Sankey-Diagramm
  + Hinterlegt Maschine und Aufenthaltsdauer in Tabelle BE\_Zeiten
* Wenn BE alle Arbeitsschritte durchlaufen hat
  + lager BE in Senke um
  + starte Simulation wieder
  + Falls Reihenfolgenbildung: Rufe Methode Überprüfung\_Maschine\_Herkunft auf
* Wenn Reihenfolgebildung aktiv
  + Frage Maschinen eines Arbeitsgangs ab, ob sie verfügbar sind
  + Frage ab, wie viele und welche BEs im Lager für die Maschine sind
  + Ermittle Zeit bis zur möglichen Bearbeitung des Auftrags
  + Sende ermittelte Infos an Jade
  + Ermittle Bauteilinformationen des umzulagernden BEs (Schlupfzeit, Fertigungsfortschritt)
  + Sende ermittelte Infos an Jade
* Wenn Reihenfolgebildung inaktiv
  + Frage Maschinen eines Arbeitsgangs ab, ob sie verfügbar sind
  + Frage ab, wie viele und welche BEs im Sortierer für die Maschine sind
  + Ermittle Zeit bis zur möglichen Bearbeitung des Auftrags
  + Infos über Maschine an Jade
  + Ermittle Informationen zu BE\_Gruppe
  + Infos über BE\_Gruppe an Jade
  + Ermittle Bauteilinformationen des umzulagernden BEs (Schlupfzeit, Fertigungsfortschritt)
  + Infos über BE an Jade

Umlagern von Sortierer

* Wird von den Sortierern beim Ausgang aufgerufen
* Bestimmt Ziel-Maschine
* Übergibt ihr die Bearbeitungsdauer
* Stellt Rüstzeit ein
* Notiert Sortierer und Aufenthaltsdauer in Tabelle BE\_Zeiten
* Lagert BE um
* Stellt Zeiten und vorheriges BE ein

Überprüfung\_Maschine

*Überprüft, ob die Zielmaschine frei ist und ruft dann ggf. die Datenerfassung\_Lager auf, um die Umlagerung zu unterstützen*

* Wird von der Steuerung aufgerufen, wenn Umlagerung zu Lager/Sortierer und Reihenfolgebildung = true
* Stoppt die Simulation
* Definiert Maschine
* Ruft Datenerfassung\_Lager auf, falls Maschine frei
* Startet die Simulation, falls Maschine belegt

Überprüfung\_Maschine\_Herkunft

*Überprüft, ob die Herkunftsmaschine frei ist und ruft dann ggf. die Datenerfassung\_Lager auf, um nachfolgenden BEs ihre Umlagerung zu ermöglichen*

* Wird von Datenerfassung oder Steuerung aufgerufen, falls Reihenfolgenbildung = true
* Stoppt Simulation
* Definiert Maschine
* Ruft Datenerfassung\_Lager auf, falls Maschine frei
* Startet Simulation, falls Maschine belegt

Datenerfassung1

*Zum Großteil wie die Methode Datenerfassung*

* Wird von der Steuerung aufgerufen, wenn ein Auftragsagent initialisiert wurde
* Stoppt Simulation
* Verschickt Stammdaten einmalig an Jade, sobald Auftragsagent initialisiert
* Reihenfolgebildung = false
  + Ermittle Informationen über Bauteil\_Gruppe
  + Sende ermittelte Infos an Jade
  + Ermittle Bauteilinformationen des umzulagernden BEs (Schlupfzeit, Fertigungsfortschritt)
  + Sende ermittelte Infos an Jade
* Reihenfolgebildung = true
  + Ermittle Bauteilinformationen des umzulagernden BEs (Schlupfzeit, Fertigungsfortschritt)
  + Sende ermittelte Infos an Jade

Datenerfassung\_Lager

* Wird von der Steuerung aufgerufen, wenn Reihenfolgebildung umgesetzt werden soll und Lagerplatz leer ist
* Wird von Überprüfung\_Maschine und Überprüfung\_Maschine\_Herkunft aufgerufen, falls Maschine frei
* Stoppt Simulation
* Falls nur ein BE im Lager
  + Lagert BE vom Lager an Maschine um und passt Rüstzeiten und Daten an
  + Startet Simulation wieder
* Falls Lager und Maschine Leer
  + Starte Simulation
* Falls mehrere BEs im Lager („Vorbereitung der Reihenfolgebildung“)
  + Berechnet Zeiten für jedes BE (maximale Bearbeitungszeit, Wartezeit, etc.)
  + Bündelt Informationen und schickt sie an Jade

Maschinen - Netzwerk

Keine für den Simulationsablauf relevanten Methoden

BE - Netzwerk

BE\_Eingang

* Wird von den Sortierern beim Eingang aufgerufen
* Anpassung der Wartezeit

BE\_Ausgang

* Wird scheinbar nirgendwo aufgerugen
* Anpassung der Wartezeit
* Aktiviert Datenerfassung

BE\_Start

* Wird scheinbar nirgendwo aufgerufen
* Weist dem BE die aktuelle absolute Zeit zu

Start\_Zeit

* Wird von Quelle beim Eingang aufgerufen
* Definiert Daten für das eingegangene BE
  + Produktionsbeginn
  + Herkunft
  + Ermittelt Liefertermin
* Eintragung des BE in die Tabelle BE\_Daten mit seinen Anfangsdaten
  + In bestehende Gruppe eintragen oder
  + Neue Gruppe anlegen
* Für das Beenden der Simulation relevanter Code

StartBearbeitung

* Wird von den Maschinen beim Eingang aufgerufen
* Notiert die Startzeit der Bearbeitung (relevant für Datenerfassung)
* Notiert aktuelles BE, um ggf. Rüstzeit für Nachfolger zu bestimmen
* Notiert aktuelle Zeit in @.BE\_Zeit

Ende\_Zeit

* Wird von der Senke beim Eingang aufgerugen
* Definition des Produktionsendes
* Eintragung der Daten in BE\_Daten-Tabelle
* Berechnung der Daten/Zeiten
* Test, ob Auftragsgruppe vollständig bearbeitet wurde. Wenn ja:
  + **Globalen Reward für Gruppe berechnen**
  + Nachricht an SocketOut (für Übermittlung an Jade)
  + Gruppe in Tabelle BE\_Gruppe\_beendet aufnehmen
* Berechnung der mittleren Durchlaufzeit
* Für das Beenden der Simulation relevanter Code

# Java/Jade Multi-Agenten-System

Order\_agents

*Der hier liegende Order\_agent führt die Belegungsplanung mithilfe des neuronalen Netzes durch.*

Order\_agent

* Setup
  + Registrierung im df-Agent
  + Setzt Agententyp auf „Auftrag“
  + Initialisierung der Auftragsdaten mit null
  + Lädt Neuronales Netz (als Graph g)
  + Initialisiert lokales Replay Memory
  + Initialisiert Behaviours
* TakeDown
  + Austragung aus dem df-Agenten
* Behaviour „Stammdatenverwaltung“
  + Wird über CyclicBehaviour ständig aufgerufen
  + Liest eingehende Nachricht ein, wenn der Betreff = S ist
  + Speichert eingegangene Stammdaten in den „Auftragsdaten“
* Behaviour „Bewegungsdatenverwaltung“
  + Wird über CyclicBehaviour ständig aufgerufen
  + Liest eingehende Nachricht ein, wenn der Betreff = B ist
  + Falls Infos über Bauteil: Speichert eingegangene Daten in den „Auftragsdaten“
  + Falls Infos über Auftragsgruppe: Speichert eingegangene Daten in „Aufträge\_in\_Gruppe“
  + Fügt Behaviour **Belegungsplanung** hinzu (führt also die Belegungsplanung durch)
* Behaviour „Auftragsgruppenverwaltung“
  + Wird über CyclicBehaviour ständig aufgerufen
  + Liest eingehende Nachricht ein, wenn der Betreff = G ist
  + Falls Zweck der Nachricht = INFORM: Speicher Daten in „Auftragsdaten\_Gruppe“
  + Falls Zweck der Nachricht = REQUEST: Versende Rückmeldung über „Auftragsdaten\_Gruppe“ mit Betreff = GB
* Behaviour „Auftragsabschluss“
  + Wird über CyclicBehaviour ständig aufgerufen
  + Liest eingehende Nachricht ein, wenn der Betreff = L ist
  + Speichert erhaltenen Reward im lokalen Replay Memory
  + Löscht den Auftragsagenten (also sich selbst)
* Behaviour „**Belegungsplanung**“
  + Swith-Case-Struktur, die wie ein Automat folgende Schritte abarbeitet:
  + Case 0:
    - Falls Aufträge in Aufträge\_in\_Gruppe vorhanden sind: Frage Informationen von anderen Aufträgen an
    - Wenn nicht: gehe zu Case 2
  + Case 1:
    - Lese Rückmeldungen der angefragten Aufträge ein und berechne daraus die Schlupfzeiten und Bearbeitungszeiten
  + Case 2: Suche Maschinen mit anstehendem Arbeitsgang
    - Suche Maschinenagenten, die anstehenden Arbeitsgang durchführen können
    - Anfrage an passende Maschinenagenten für die aktuellen Bewegungsdaten
  + Case 3: Bestimmung der Umlagerungsaktion
    - Lese Rückmeldungen der angefragten Maschinenagenten ein
    - Bewerte jede Maschine über die voraussichtliche Zeit bis zur Fertigstellung
    - Berechne daraus die Bearbeitungszeit für den aktuellen Auftrag für jede Maschine (im Zustandsvektor „state“)
    - Ermittlung der minimalen und maximalen Bearbeitungszeit (ist der Code hier Fehlerhaft in Zeile 476, 482 und 484??)
    - Setze state[0][i] beim Maximum auf 1, Minimum auf -1 und sonst auf 0
    - Erweitere Zustandsvektor um Schlupfzeit und Fertigungsfortschritt
    - Gebe den Zustandsvektor „state“ in der Konsole aus. Ergebnis z.B.: [[-1.0, 0.0, 1.0, -286.81897, 0.5]]
    - Zeile 500 **int** action\_Wert = Tensorflow.*action*(g, state); berechnet den Ausgabewert aus dem Neuronalen Netz. action\_Wert ist z.B. = 0 oder = 2
    - Bestimmung der Zielstation bestStation aus dem action\_Wert
    - Ermittlung der minimalen und maximalen Bearbeitungszeit
    - Berechne **Reward** für getroffene Entscheidung
      * Reward = 1, falls Station mit minimaler Bearbeitungszeit ausgewählt wurde
      * Reward = -1, falls Station mit maximaler Bearbeitungszeit ausgewählt wurde
      * Reward = 0, sonst
    - Falls aktuelle Station = Puffer, schreibe Zustandsvektor state in den String state\_t
    - Falls aktuelle Station != Puffer, schreibe Zustandsvektor in den String state\_t\_
    - Falls bestStation != null
      * Falls aktuelle Station != Puffer, schreibe Ergebnisse (State, Reward, Action) in Local Memory und setze state\_t = state\_t\_
      * Nachricht an Plantsim über Steuerungsentscheidung schicken

Supervisor Agents

Initialization\_agent

* Behaviour „Agenteninitialisierung“
  + Wird über CyclicBehaviour ständig aufgerufen
  + Deserialisiert Nachricht über Msg\_serialization
  + Erstellt neuen Agenten mit Name, Inhalt und Typ
  + Sendet Informationsnachricht mit Daten an neuen Agenten
  + onEnd-Methode

Socket\_in

* Behaviour „Datenweiterleitung“
  + Wird über CyclicBehaviour ständig aufgerufen
  + Empfängt Nachrichten von PlantSim im Objekt Object\_Msg
  + Packt Infos in Msg\_serialization-Objekt
  + Leitet das Objekt als Nachricht an den adressierten Agenten weiter

Socket\_out

* Registriert sich im df-agent
* Behaviour „Datenübermittlung“
  + Wird über CyclicBehaviour ständig aufgerufen
  + Sendet Steuerungsentscheidungen an PlantSim

Machine\_agents

Drehen\_agent

* Setup
  + Registrierung im df-Agent
* Behaviour „Maschinensteuerung“
  + Wird über CyclicBehaviour ständig aufgerufen
  + Erhält Nachrichten von PlantSim: Datenerfassung, Datenerfassung1 und Ende\_Zeit
  + Aktualisiert Stammdaten der Maschine basierend auf Eingangsnachricht „S“
  + Aktualisiert Bewegungsdaten basierend auf Eingangsnachricht „M“
  + Gibt eine Rückmeldung mit Daten über eine mögliche Bearbeitung, falls Eingangsnachricht „B“ (Bearbeitung angefragt)

Fräsen\_agent

* Genau wie Drehen\_agent, nur Name des individuellen Prozesses ist anders

Verpackung\_agent

* Genau wie Drehen\_agent, nur Name des individuellen Prozesses ist anders

Dqn\_functions

Clear\_replay\_memory

* Löscht replay memory

Initialization\_replay\_memory

* Erstellung einer neuen Datenbank
* Erstellung einer neuen Tabelle innerhalb der Datenbank

Load\_graph

* Lädt Graph aus Datenbank

Tensorflow

* Methode „action“
  + Nutzt Graph und berechnet daraus eine Aktion für den Auftragsagenten
* Methode „predict“
* Methode „get\_warm\_up\_steps“

Transfer\_replay\_memory

* Methode „Übertragung“
  + Schreibt Informationen in die SQL-Verbindung
* Methode „getAnzahl\_steps“

Write\_local\_memory

* Methode „insert“
  + Baut verbindung mit SQL auf
  + Schreibt state\_t, state\_t\_, a\_t, r\_t und done\_flag in die Datenbank

Communication\_functions

Msg\_serialization

*Datenformat für den Austausch zwischen den Agenten*

* Wird vom initialization\_agent aufgerufen
* Enthält Definition etlicher Variablen
* Implementiert Serializable (Objekte können also als String serialisiert werden)
* Stellt Setter/Getter für die Variablen zur Verfügung

Object\_msg

*Objekt, in das Socket\_in die Nachrichten einliest*

* Klasse für Object\_msg-Objekte
* Enthält diverse Konstruktoren und Setter/Getter
* Methode „recordMsg“
  + Liest Nachrichten über BufferedReader ein

# Aufbau der Nachrichten zwischen PlantSim und Jade

*(Hinweis: bisher nur für Reihenfolgebildung = false untersucht)*

## Von PlantSim an Jade verschickte Nachrichten

Grundsätzliches

Grundsätzlicher Aufbau einer Nachricht:

Maschinenname/Auftragsname : Verfahren(Maschine)/“Auftrag“ : Betreff : individueller Inhalt abhängig vom Betreff;

Bedeutung des Betreffs:

* S = Übermittlung der Maschinen- oder Bauteilstammdaten
* M = Übermittlung der aktuellen Maschinenbelegungen und –zeiten
* B = Übermittlung des aktuellen Bauteilstatus
* G = Übermittlung von Informationen über BE\_Gruppen (wird scheinbar nicht genutzt)
* E = Übermittlung eines Rewards für eine Maschine (nur bei Reihenfolgebildung)
* L = Reward für ein Bauteil

Folgende Nachrichten werden versendet, wenn Reihenfolgebildung aus ist (Beispiele in Grau):

* Nachricht über Maschineninitialisierung:

Maschinenname : Verfahren ;

Drehen\_1:Drehen;

* Übermittlung der Maschinenstammdaten:

Maschinenname : Verfahren : Betreff=S : Verfahren : Länge : Durchmesser : Tiefe : Achsenanzahl : Materialien : Maschinenfaktor\_Zeit : Aluminium\_Zeitfaktor : Stahl\_Zeitfaktor : Titan\_Zeitfaktor ;

Drehen\_1:Drehen:S:Drehen:50:50:50:5:Stahl:2.1:1:1.02:1.02;

Verpackung\_1:Verpackung:S:Verpackung:50:50:50:5:Stahl:1.1:1:0:0;

* Übermittlung des Maschinenstatus:

Maschinenname : Verfahren : Betreff=M : Verfügbarkeit : Zeit bis Bearbeitung : Status : MTTR : BE\_Anzahl : BE\_Typen ;

Fräsen\_3:Fräsen:M:100:0:Frei:0:0:0;

Drehen\_3:Drehen:M:100:1.35802906697791e-008:Arbeitend:0:1:0;

* Initialisierung eines Bauteils (BE):

Auftragsname + BE\_Auftrag.id : „Auftrag“ : Betreff=S : Länge : Durchmesser : Tiefe : benötigte Achsenanzahl : Material ;

Auftrag\_Typ\_2.2:Auftrag:S:15:10:10:3:Stahl;

* Übermittlung des Bauteilstatus:

Auftragsname + BE\_Auftrag.id : „Auftrag“ : Betreff=N : BE\_Typ : Abgeschlossener Arbeitsgang : Bearbeitungszeit : Schlupfzeit : Fertigungsfortschritt : Alle\_BE\_Gruppe ;

Auftrag\_Typ\_2.2:Auftrag:B:Auftrag\_Typ\_2:Fräsen:58:494:0.5:;

* Übermittlung des Rewards für ein fertiges Bauteil:

Auftragsname + BE\_ID : „Auftrag“ : Betreff=L : Reward;

Auftrag\_Typ\_2.2:Auftrag:L:-10;

Einsatz der Nachrichten

Init-Methode:

* Nachricht über Maschineninitialisierung
* Übermittlung der Maschinenstammdaten

Datenerfassung 1:

* Initialisierung eines Bauteils (BE)
* Nachricht über BE\_Gruppe (wird nicht genutzt)
* Übermittlung des Bauteilstatus

Datenerfassung:

* Übermittlung des Maschinenstatus:
* Nachricht über BE\_Gruppe (wird nicht genutzt)
* Übermittlung des Bauteilstatus

EndSim

* Nachricht über Reward (nur bei Reihenfolgebildung)

Ende\_Zeit

* Übermittlung des Rewards für ein fertiges Bauteil

## Von Jade an PlantSim verschickte Nachrichten

Die Methode „Steuerung“ empfängt die Nachrichten in PlantSim und setzt sie um

* Information über initialisierten Auftragsagenten

AuftragsAgent\_initialisiert

* Umlagerungsbefehl für ein Bauteil

B : Auftragsname + ID : Name der Zielmaschine :

Von Jade komm noch „: Bearbeitungszeit : aktuelle Station ;“ an. Das wird von Plantsim aber abgeschnitten

B:Auftrag\_Typ\_2.1:Fräsen\_1:

# Jade-interne Kommunikation

Die genaue Kommunikation zwischen den Agenten wurde nicht untersucht. Es stehen aber folgende Sequenzdiagramme für einen Überblick zur Verfügung, falls noch ein Untersuchung erforderlich wird.







