Ein typisiertes, rückgekoppeltes Simulationsmodell für das M-Q-M-Modell

Hierarchische Zustände, Kopplungsoperatoren und UML-Architektur

Entwurfsartikel auf Basis der vorliegenden MQM-Projektunterlagen

30. April 2026

Zusammenfassung

Der Beitrag entwickelt ein softwaretechnisch umsetzbares und mathematisch präzisiertes Simulationsmodell für das M-Q-M-Modell. Ausgangspunkt ist die Unterscheidung zwischen System, Quadranten und Zuständen: Die Quadranten $Q_1(S), \ldots, Q_4(S)$ werden nicht selbst als Zustände behandelt, sondern als Teilstrukturen eines Systems S, deren jeweilige Zustände wiederum aus Achsen- und Ebenenzuständen komponiert werden. Der Systemzustand entsteht folglich als Funktion der Quadrantenzustände. Zentral ist die Erweiterung auf heterogene Zustandswerte: numerische, ordinale, nominale und relationale Werte werden in einem gemeinsamen typisierten Zustandsraum geführt. Für die Ebenen xy, xz und yz werden keine starken Faktorisierungsannahmen, etwa ein allgemeines Tensorprodukt, vorausgesetzt. Stattdessen werden kontextabhängige Kopplungsoperatoren eingeführt. Am Beispiel einer virtuellen Produktionszelle werden die Quadranten Planung (Q_3), Steuerung (Q_1), Prozess (Q_2) und Ergebnis (Q_4) mit Rückkopplungsschleifen formalisiert. Der Artikel schließt mit einer UML-nahen Architektur und einer Diskussion der mathematischen Reichweite, insbesondere der Rolle von Tensor-, Faser- und kategorialen Lesarten.

Schlüsselwörter: M-Q-M, UML, Simulation, heterogene Zustände, Kopplungsoperatoren, Rückkopplung, Produktionszelle, typisierte Modellierung.

Inhaltsverzeichnis

1 Einleitung

Das M-Q-M-Modell wird in den zugrunde liegenden Projektunterlagen als eine strukturierte Systembeschreibung verstanden, in der ein System über vier Quadranten beschrieben wird: Transformation, Prozess, Plan und Ergebnis. Die UML-nahe Vorarbeit präzisiert, dass diese Quadranten nicht mit Zuständen gleichgesetzt werden dürfen, sondern Teilstrukturen eines Systems sind [1]. Der Zustand eines Systems ist vielmehr eine Funktion der Zustände seiner Quadranten. Diese Unterscheidung ist für eine simulationsfähige Softwarearchitektur entscheidend.

Der vorliegende Artikel verfolgt drei Ziele. Erstens wird eine formale Zustandsstruktur entwickelt, die Achsen- und Ebenenzustände berücksichtigt. Zweitens wird eine Operatorbibliothek vorgeschlagen, die numerische, ordinale, nominale und relationale Zustände kombinieren kann. Drittens wird ein konkretes Simulationsschema anhand einer virtuellen Produktionszelle ausgearbeitet, einschließlich Rückkopplungsschleifen zwischen Planung, Steuerung, Prozess und Ergebnis.

Die mathematische Motivation berührt auch weiterführende geometrische Deutungen. Die ontologisch-mathematische Projektnotiz diskutiert eine mögliche Verbindung des MQM-Modells mit Quaternionen und der Hopf-Fibration [2]. Diese Interpretation ist als strukturelle Hypothese aufzufassen, nicht als notwendige Grundlage des hier entwickelten Simulationsmodells.

2 Begriffliche Präzisierung: System, Quadrant und Zustand

Ein häufiger Modellierungsfehler besteht darin, die Quadranten selbst als Zustandsraum zu interpretieren. Für das hier entwickelte Modell gilt stattdessen:

$$Q_i = Q_i(S), \qquad i \in \{1, 2, 3, 4\}. \tag{1}$$

Die Quadranten sind also Teilstrukturen oder Projektionen eines Systems S. Der Zustand des Systems ist eine Funktion der Quadrantenzustände:

$$Z(S) = F\big(Z(Q_1(S)), Z(Q_2(S)), Z(Q_3(S)), Z(Q_4(S))\big). \tag{2}$$

Die vier Quadranten werden im Simulationsbeispiel wie folgt interpretiert:

Tabelle 1: Quadranten des Simulationsmodells.

Symbol	Kurzname	Rolle	Simulationstechnische Bedeutung
$Q_1(S)$	Transformation	Steuerung	Transformation von Planung in Prozessparameter
$Q_2(S)$	Prozess	Ausführung	Laufender Material-, Energie- und Informationsprozess
$Q_3(S)$	Plan	Planung	Sollstruktur, Auftragsmenge, Priorität und Sequenzierung
$Q_4(S)$	Ergebnis	Bewertung	Output, Qualität, Bestand, Effizienz und Rückstand

Diese Zuordnung ist nicht als starres Dogma zu verstehen, sondern als simulationsfähige Lesart des Modells. Je nach Anwendungsdomäne können die semantischen Inhalte der Quadranten verfeinert werden.

3 Hierarchisches Zustandsmodell

3.1 Achsen- und Ebenenstruktur

Jeder Quadrant besitzt drei Achsen und drei Ebenen. Für einen Quadranten $Q_i(S)$ werden folgende Komponenten unterschieden:

$$x,\ y,\ z,\ xy,\ xz,\ yz. \tag{3}$$

Die Achsen stehen für Basisdimensionen, die Ebenen für Kopplungs- oder Interaktionszustände. Im MQM-Kontext werden sie in der hier verwendeten Lesart wie folgt zugeordnet:

Tabelle 2: Achsen und Ebenen innerhalb eines Quadranten.

Komponente	Ebene	Kurzinterpretation	Bemerkung
x	Achse	Zeitstruktur	Dauer, Takt, Planungshorizont
y	Achse	Raum-/Mengenstruktur	Menge, Kapazität, Bestand
z	Achse	Modusstruktur	Betriebsart, Priorität, Status
xy	Ebene	Energie / Aufwand	Kopplung von Zeit und Menge
xz	Ebene	Information / Steuerung	Kopplung von Zeit und Modus
yz	Ebene	Form / Struktur	Kopplung von Menge und Modus

Der Zustand eines Quadranten ist dann nicht ein einzelner Wert, sondern eine Komposition:

$$Z(Q_i(S)) = f_i\big(Z(Q_{ix}), Z(Q_{iy}), Z(Q_{iz}), Z(Q_{ixy}), Z(Q_{ixz}), Z(Q_{iyz})\big). \tag{4}$$

Damit entsteht eine mehrstufige Zustandsstruktur:

$$\text{Achsen und Ebenen} \longrightarrow \text{Quadrant} \longrightarrow \text{System}. \tag{5}$$

Die in den Vorarbeiten formulierte Kernaussage, dass Quadranten Strukturträger und nicht Zustände sind, ist damit direkt in die mathematische Modellierung übernommen [1].

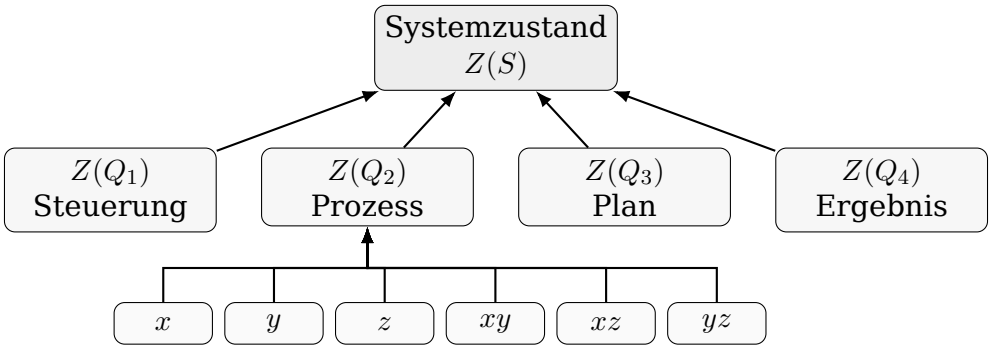


Abbildung 1: Hierarchische Zustandsstruktur: Achsen und Ebenen komponieren Quadrantenzustände; Quadrantenzustände komponieren den Systemzustand.

3.2 Typisierte Zustandswerte

Für eine realistische Simulation genügt ein rein numerischer Zustandsraum nicht. Zustände können numerisch, ordinal, nominal oder relational sein. Ein Zustandswert wird daher als typisiertes Objekt modelliert:

$$Z = (v, \tau, u, m), \tag{6}$$

mit Wert v, Typ τ, optionaler Einheit u und optionalen Metadaten m. Die Typmenge ist:

$$\mathcal{T} = \{\text{Num}, \text{Ord}, \text{Nom}, \text{Rel}\}. \tag{7}$$

Tabelle 3: Zustandstypen im MQM-Simulationsmodell.

Typ	Symbol	Beispiele	Zulässige Grundoperationen
Numerisch	Num	Energie, Zeit, Menge, Effizienz	Addition, Skalierung, Vergleich, Optimierung
Ordinal	Ord	niedrig, normal, hoch, kritisch	Ordnung, Minimum, Maximum, Eskalation
Nominal	Nom	automatic, rush, stabilize	Regelzuordnung, Klassifikation, Fallunterscheidung
Relational	Rel	Graphen, Abhängigkeiten, Konfigurationen	Merge, Einschränkung, Pfade, Annotation

Der typisierte Zustandsraum eines Quadranten ist folglich kein Vektorraum im klassischen Sinn, sondern ein Produkt heterogener Zustandsräume:

$$\mathcal{Z}_{Q_i} = \mathcal{Z}_{ix} \times \mathcal{Z}_{iy} \times \mathcal{Z}_{iz} \times \mathcal{Z}_{ixy} \times \mathcal{Z}_{ixz} \times \mathcal{Z}_{iyz}. \tag{8}$$

Der Systemzustandsraum ist entsprechend:

$$\mathcal{Z}_S = \prod_{i=1}^{4} \mathcal{Z}_{Q_i}. \tag{9}$$

4 Kopplungsoperatoren statt starker Tensorannahme

Die Aussage

$$Z_{xy} \sim Z_x \otimes Z_y \tag{10}$$

ist als allgemeine MQM-Annahme zu stark. Sie würde nahelegen, dass der Ebenenzustand Z_{xy} vollständig aus den Achsenzuständen Z_x und Z_y erzeugt werden kann. Für numerische Spezialfälle mag dies sinnvoll sein, etwa bei Energie als Produkt aus Leistung und Zeit. Im allgemeinen MQM-Modell besitzen die Ebenenzustände jedoch eigenständige semantische und datenbezogene Qualität.

Stattdessen wird ein Kopplungsoperator eingeführt:

$$Z(Q_{iab}) = C_{ab}^{(i)}\big(Z(Q_{ia}), Z(Q_{ib}); \theta_{ab}^{(i)}, \kappa\big), \qquad ab \in \{xy, xz, yz\}. \tag{11}$$

Hier bezeichnet $\theta_{ab}^{(i)}$ Parameter des Operators und κ den Kontext, etwa Domäne, Quadrant, Regelwerk oder Simulationsmodus. Gleichung (11) erlaubt die Faktorisierung als Spezialfall, erzwingt sie aber nicht.

4.1 Operatorfamilien

Die Operatoren werden nach ihrer Herkunft und Begründungsform unterschieden:

Tabelle 4: Operatorfamilien für Ebenenzustände.

Familie	Form	Beispiel	Charakter
Physikalisch-deterministisch	$C(x,y) = f(x,y)$	Energie $= \eta xy$	quantitativ, parameterisierbar
Regelbasiert	$C(x,y) = R[x,y]$	Priorität + Status $\mapsto$ Alarm	nominal/ordi auditierbar
Datengetrieben	$C(x,y) = \hat{f}(x,y)$	Verbrauchsmodell aus Historie	empirisch, lernbar
Relational	$C(x,y) = G'$	Graphannotation oder Subgraph	strukturbilde nicht skalar
Hybrid	Kombination	Steuerlogik mit Schätzwert	praktisch häufigster Fall

5 Operatorbibliothek für heterogene Zustände

Eine simulationsfähige Software benötigt eine definierte Bibliothek für die Kombination heterogener Typen. Allgemein sei ein Operator eine partielle, typisierte Abbildung:

$$\mathcal{C}^{\gamma}_{\alpha\beta} : \mathcal{Z}_\alpha \times \mathcal{Z}_\beta \rightharpoonup \mathcal{Z}_\gamma, \qquad \alpha, \beta, \gamma \in \mathcal{T}. \tag{12}$$

Die Abbildung ist partiell, weil nicht jede Typkombination in jedem Kontext sinnvoll ist. Eine robuste Implementierung muss deshalb nicht nur Werte, sondern auch Typ, Kontext und Validitätsbedingungen prüfen.

Tabelle 5: Vorschlag einer Operatorbibliothek für alle elementaren Typkombinationen.

Typkombination	Operatoridee	Möglicher Zieltyp	Beispiel
Num $\times$ Num	additiv, multiplikativ, saturierend, verlustbehaftet	Num	Energie $= \eta \cdot x \cdot y$

Typkombination	Operatoridee	Möglicher Zieltyp	Beispiel
Num × Ord	Gewichtung oder Schwellenlogik	Num oder Ord	Menge × Priorität ↦ gewichtete Last
Num × Nom	kontextuelle Transformation	Num oder Nom	Verbrauch abhängig vom Maschinenmodus
Num × Rel	Verteilung über Graph oder Annotation	Rel oder Num	Energie auf Maschinenknoten verteilen
Ord × Num	Gewichtung oder Eskalation	Num oder Ord	hoher Rückstand eskaliert Priorität
Ord × Ord	Maximum, Minimum, Dominanz, Kompromiss	Ord	max(hoch, kritisch) = kritisch
Ord × Nom	Regelmatrix	Nom oder Ord	Priorität + Modus ↦ Steuerstrategie
Ord × Rel	Graphfilter oder Constraint	Rel	nur kritische Aufträge im Graph aktivieren
Nom × Num	parametrische Fallunterscheidung	Num oder Nom	Modus rush erhöht Kapazitätsfaktor
Nom × Ord	Klassifikation	Nom oder Ord	Modus + Qualitätsgrad ↦ Maßnahme
Nom × Nom	Lookup, endlicher Automat	Nom	machine_critical ↦ alarm
Nom × Rel	Annotation oder Auswahl	Rel	Modus energy_save wählt Teilgraph
Rel × Num	Bewertung oder Kapazitätsfluss	Rel oder Num	Graph mit Kantenlasten versehen
Rel × Ord	Constraint Propagation	Rel	Priorität beschränkt zulässige Pfade
Rel × Nom	Labeling oder Modusfilter	Rel	alle Knoten mit Modus stabilize markieren
Rel × Rel	Merge, Pullback, Join, Differenz	Rel	Auftragsgraph und Maschinengraph verbinden

Die Tabelle ist nicht als vollständige Semantik zu verstehen, sondern als Software-Schnittstelle. Für jede konkrete Domäne müssen die Operatoren parametrisiert, validiert und getestet werden.

6 Anwendungsfall: virtuelle Produktionszelle

6.1 Systembeschreibung

Als Demonstrator dient eine virtuelle Produktionszelle. Das System S umfasst Maschinen, Aufträge, Steuerlogik, Energieverbrauch, Qualitätsbewertung und Rückmeldungen. Die Quadranten werden wie folgt instanziiert:

- $Q_3(S)$: Planung, z. B. Planungshorizont, geplante Menge, Priorität, Sequenzierungsregel.
- $Q_1(S)$: Steuerung, z. B. Taktzeit, Kapazität, Steuerungsmodus, Maschinenkonfiguration.
- $Q_2(S)$: Prozess, z. B. tatsächliche Prozesszeit, verarbeitete Menge, Energieverbrauch, Stabilität.
- $Q_4(S)$: Ergebnis, z. B. Output, Qualität, Durchsatz, Rückstand und Traceability.

Die Reihenfolge der zyklischen Wirkung lautet:

$$Q_3 \longrightarrow Q_1 \longrightarrow Q_2 \longrightarrow Q_4 \longrightarrow Q_3. \tag{13}$$

Die Rückkopplung ist zeitversetzt: Ergebnisse aus Zeitpunkt t beeinflussen Planung und Steuerung zum Zeitpunkt $t+1$.

6.2 Planung: Q_3

Für Q_3 seien definiert:

$$H_t := Z(Q_{3x}) \qquad \text{Planungshorizont}, \tag{14}$$

$$M_t^{plan} := Z(Q_{3y}) \qquad \text{geplante Menge}, \tag{15}$$

$$P_t := Z(Q_{3z}) \qquad \text{Priorität}. \tag{16}$$

Der Ebenenzustand Q_{3xy} wird als Planlast modelliert:

$$L_t = C_{xy}^{(3)}(H_t, M_t^{plan}; K_t) = \frac{M_t^{plan}}{K_t\, H_t}, \tag{17}$$

sofern $K_t H_t > 0$. K_t bezeichnet die real verfügbare Kapazität aus Q_1. Für Q_{3xz} wird eine Sequenzierungsregel gewählt:

$$R_t^{seq} = C_{xz}^{(3)}(H_t, P_t; \kappa). \tag{18}$$

Für Q_{3yz} wird ein relationales Auftragsportfolio erzeugt:

$$G_t^{portfolio} = C_{yz}^{(3)}(M_t^{plan}, P_t; \kappa). \tag{19}$$

6.3 Steuerung: Q_1

Für Q_1 seien definiert:

$$\tau_t := Z(Q_{1x}) \qquad \text{Taktzeit,} \tag{20}$$
$$K_t := Z(Q_{1y}) \qquad \text{Kapazität,} \tag{21}$$
$$m_t := Z(Q_{1z}) \qquad \text{Steuerungsmodus.} \tag{22}$$

Der Steueraufwand wird modelliert als:

$$A_t^{ctrl} = C_{xy}^{(1)}(\tau_t, K_t; m_t) = \beta(m_t)\frac{K_t}{\tau_t}. \tag{23}$$

Der Modus m_t ist nominal, beispielsweise:

$$m_t \in \{\texttt{automatic}, \texttt{rush}, \texttt{energy\_save}, \texttt{stabilize}, \texttt{maintenance}\}. \tag{24}$$

Die Regelstrategie und Maschinenkonfiguration entstehen über:

$$R_t^{ctrl} = C_{xz}^{(1)}(\tau_t, m_t), \tag{25}$$
$$G_t^{machine} = C_{yz}^{(1)}(K_t, m_t). \tag{26}$$

6.4 Prozess: Q_2

Der Prozess verarbeitet die geplante Menge unter den von Q_1 bestimmten Steuerbedingungen. Die tatsächlich verarbeitbare Menge ist:

$$M_{t+1}^{proc} = \min\big(M_{t+1}^{plan}, K_{t+1}H_{t+1}\big). \tag{27}$$

Der Energieverbrauch wird durch einen hybriden Operator beschrieben:

$$E_{t+1} = \alpha(m_{t+1})\, M_{t+1}^{proc}\, T_{t+1}^{proc}\big(1 + \lambda \max(0, L_{t+1} - 1)\big). \tag{28}$$

Damit werden Menge, Prozesszeit, Steuerungsmodus und Überlast gemeinsam berücksichtigt. Die Prozessstabilität ist ein ordinaler Zustand:

$$S_{t+1}^{stab} \in \{\text{stable}, \text{unstable}, \text{critical}\}. \tag{29}$$

6.5 Ergebnis: Q_4

Das Ergebnis bewertet die Prozessausführung. Der Output sei:

$$O_{t+1} = M_{t+1}^{proc}\, Y_{t+1}^{quality}, \tag{30}$$

wobei $Y_{t+1}^{quality}$ die Qualitätsausbeute bezeichnet. Der Rückstand ist:

$$B_{t+1} = \max(0, M_{t+1}^{plan} - O_{t+1}). \tag{31}$$

Eine einfache Effizienzgröße ist:

$$\eta_{t+1} = \frac{O_{t+1}}{E_{t+1}}, \qquad E_{t+1} > 0. \tag{32}$$

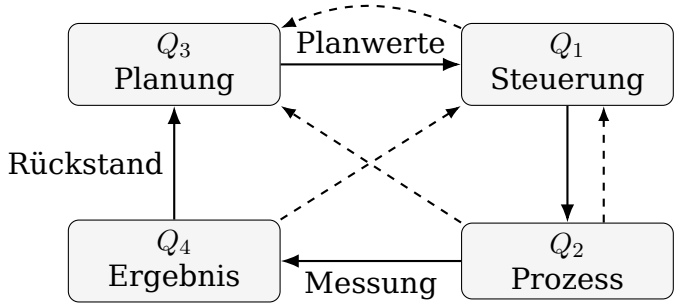
Die Traceability wird relational modelliert, z. B. als Graph aus Auftrag, Maschine, Modus, Energie und Qualität.

7 Rückkopplungsschleifen

Das Modell besitzt mehrere Rückkopplungen. Die Hauptiteration lautet:

$$Z(Q_3)_{t+1} = A_3\big(Z(Q_3)_t, Z(Q_4)_t, Z(Q_2)_t, Z(Q_1)_t\big), \tag{33}$$
$$Z(Q_1)_{t+1} = A_1\big(Z(Q_1)_t, Z(Q_3)_{t+1}, Z(Q_4)_t, Z(Q_2)_t\big), \tag{34}$$
$$Z(Q_2)_{t+1} = A_2\big(Z(Q_2)_t, Z(Q_1)_{t+1}, Z(Q_3)_{t+1}\big), \tag{35}$$
$$Z(Q_4)_{t+1} = A_4\big(Z(Q_4)_t, Z(Q_2)_{t+1}, Z(Q_3)_{t+1}\big). \tag{36}$$

Damit ist die Kausalität in einem Simulationsschritt eindeutig: altes Ergebnis und alter Prozess beeinflussen neue Planung und neue Steuerung; daraus entstehen neuer Prozess und neues Ergebnis.



Gestrichelte Rückkopplungen: Qualität $Q_4 \to Q_1$, Energie $Q_2 \to Q_3$, Stabilität $Q_2 \to Q_1$ und Kapazität $Q_1 \to Q_3$.

Abbildung 2: Rückgekoppelte MQM-Simulation: Planung, Steuerung, Prozess und Ergebnis bilden eine zeitverzögerte Schleifenstruktur.

7.1 Beispiele für Rückkopplungsregeln

Output-Rückstand nach Planung. Wenn $B_t > 0$, wird die nächste Planmenge erhöht:

$$M_{t+1}^{plan} = D_{t+1} + B_t, \tag{37}$$

wobei D_{t+1} die neue Nachfrage bezeichnet. Bei hohem Rückstand wird die Priorität ordinal eskaliert.

Qualitätsproblem nach Steuerung. Falls die Qualität in Q_4 niedrig ist, wird m_{t+1} nicht automatisch auf `rush`, sondern auf `stabilize` oder `maintenance` gesetzt. Damit wird eine rein durchsatzorientierte Fehlreaktion vermieden.

Energieüberschreitung nach Planung und Steuerung. Falls E_t das Energiebudget überschreitet, kann Q_1 auf `energy_save` wechseln und Q_3 eine energieoptimierte Sequenzierungsregel aktivieren.

Kapazitätsrückmeldung nach Planung. Die reale Kapazität K_t aus Q_1 begrenzt die Planbarkeit über Gleichung (17). Wenn $L_{t+1} > 1$, ist der Plan nicht vollständig erfüllbar und muss geglättet, verschoben, gesplittet oder mit Zusatzkapazität versehen werden.

8 Konfliktauflösung in der Steuerung

Rückkopplungen erzeugen Zielkonflikte. Ein System kann gleichzeitig hohe Rückstände, schlechte Qualität und Energieüberschreitungen melden. Daher benötigt Q_1 eine explizite Konfliktauflösung. Eine mögliche Prioritätsordnung lautet:

Tabelle 6: Beispielhafte Prioritätsordnung für die Steuerung.

Rang	Ziel	Steuerungsfolge
1	Sicherheit / Maschinenzustand	`maintenance`
2	Qualität	`stabilize`
3	Energiegrenze	`energy_save`
4	Liefertermin / Rückstand	`rush`
5	Normalbetrieb	`automatic`

Als Entscheidungsregel:

Listing 1: Konfliktauflösung in Q1 als Pseudocode.

```
if machine_state == "critical":
   mode = "maintenance"
elif quality == "low" or stability == "unstable":
   mode = "stabilize"
elif energy > energy_budget:
   mode = "energy_save"
elif load > 0.95 and priority in ["high", "critical"]:
   mode = "rush"
else:
   mode = "automatic"
```

Die Regel zeigt, dass Q1 nicht nur numerische Parameter berechnet, sondern nominale und ordinale Zustände in konkrete Steuerentscheidungen transformiert.

9 UML-nahe Softwarearchitektur

9.1 Kernklassen

Die Architektur trennt Systemzustände, Quadrantenzustände, Werte, Operatoren und Rückkopplungen. Eine Implementierung sollte mindestens folgende Klassen oder Komponenten enthalten:

Tabelle 7: Kernklassen des vorgeschlagenen UML-Modells.

Klasse	Rolle	Verantwortung
`MQMSystem`	Aggregat	Identität und Historie des simulierten Systems
`MQMState`	Systemzustand	Enthält vier `QuadrantState`-Objekte zu einem Zeitpunkt
`QuadrantState`	Quadrantenzustand	Enthält x, y, z, xy, xz, yz und optionale Summary-Daten
`StateValue`	typisierter Wert	Wert, Typ, Einheit und Metadaten
`ValueType`	Enumeration	`NUMERIC`, `ORDINAL`, `NOMINAL`, `RELATIONAL`
`CouplingOperator`	Operator	Typ- und kontextabhängige Kopplung zweier Zustände
`FeedbackLoop`	Rückkopplung	Quelle, Ziel, Verzögerung und Regel
`PlannerQ3`	Dienst	Aktualisiert Planung
`ControllerQ1`	Dienst	Aktualisiert Steuerung und löst Konflikte
`ProcessQ2`	Dienst	Simuliert Prozessausführung
`EvaluatorQ4`	Dienst	Bewertet Ergebnis und KPIs

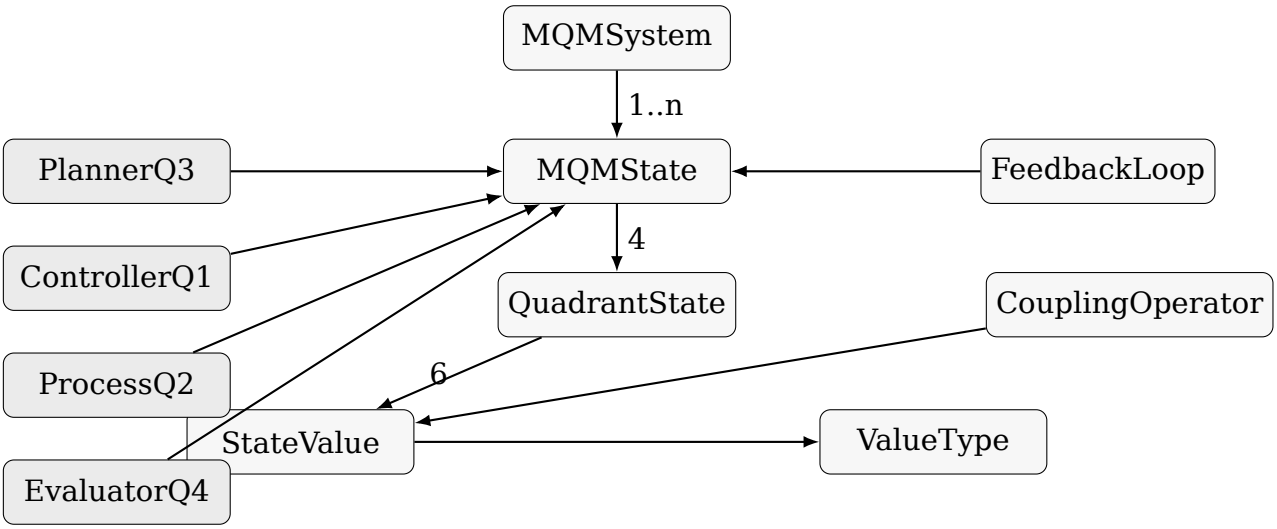


Abbildung 3: UML-nahe Architektur des MQM-Simulationskerns.

9.2 PlantUML-Skizze

Eine mögliche PlantUML-Repräsentation lautet:

Listing 2: UML-Skizze des MQM-Simulationskerns in PlantUML.

```
@startuml
class MQMSystem { id; name }
class MQMState { t }
class QuadrantState { name; x; y; z; xy; xz; yz }
class StateValue { value; type; unit; metadata }
enum ValueType { NUMERIC; ORDINAL; NOMINAL; RELATIONAL }
class CouplingOperator { name; context; apply() }
```

```
class FeedbackLoop { source; target; delay; rule }
class PlannerQ3 { updatePlan() }
class ControllerQ1 { updateControl(); resolveConflict() }
class ProcessQ2 { execute() }
class EvaluatorQ4 { evaluate() }

MQMSystem "1" --> "n" MQMState
MQMState "1" *-- "4" QuadrantState
QuadrantState "1" *-- "6" StateValue
StateValue --> ValueType
CouplingOperator --> StateValue
FeedbackLoop --> QuadrantState
PlannerQ3 --> ControllerQ1
ControllerQ1 --> ProcessQ2
ProcessQ2 --> EvaluatorQ4
EvaluatorQ4 --> PlannerQ3
@enduml
```

10 Mathematische Diskussion

10.1 Warum kein allgemeines Tensorprodukt?

Ein Tensorprodukt ist nur dann angemessen, wenn die beteiligten Zustandsräume genügend lineare Struktur besitzen und der Ebenenzustand tatsächlich aus den Achsenzuständen hervorgeht. Für heterogene MQM-Zustände gilt dies im Allgemeinen nicht. Insbesondere nominale und relationale Zustände besitzen keine natürliche bilineare Struktur. Daher wird der Ebenenzustand als eigenständiger Kopplungszustand modelliert:

$$Z_{xy} \in \mathcal{Z}_{xy}, \tag{38}$$

und nur im Spezialfall kann gelten:

$$Z_{xy} = Z_x \otimes Z_y. \tag{39}$$

Diese Einschränkung ist wesentlich, weil sie verhindert, dass das Modell eine nicht gerechtfertigte Reduktion komplexer Interaktionen erzwingt.

10.2 Monoidale und kategoriale Lesart

Der abstrakte Kombinationsgedanke kann dennoch kategorial gelesen werden. Zustände können als Objekte und Operatoren als Morphismen einer typisierten Struktur betrachtet werden. Ein monoidaler Tensor wäre dann nicht notwendig ein analytisches Tensorprodukt, sondern ein abstrakter Kombinationsoperator. Für das praktische Simulationsmodell genügt jedoch die schwächere Annahme einer typisierten Operatorfamilie. Die kategoriale Lesart bleibt eine mögliche Formalisierungsebene für spätere Arbeiten.

10.3 Faser- und Hopf-Interpretation als Hypothese

Die ontologisch-mathematische Projektnotiz deutet eine Verbindung von MQM, Quaternionen und Hopf-Fibration an [2]. Dort wird die Zuordnung der vier Qua-

dranten zu einer Quaternion diskutiert, etwa:

$$q = Q_3 + Q_2 i + Q_1 j + Q_4 k. \tag{40}$$

Unter Normierungsannahmen läge ein solcher Zustand auf S^3, und die Hopf-Fibration

$$S^1 \hookrightarrow S^3 \to S^2 \tag{41}$$

könnte Prozesszyklen, Systemorientierung und Gesamtzustand geometrisch interpretieren. Für den hier entwickelten Simulationskern ist diese Deutung jedoch optional. Sie wird erst dann mathematisch verbindlich, wenn gezeigt wird, dass die MQM-Zustände tatsächlich eine geeignete Mannigfaltigkeitsstruktur tragen und die Dynamik quaternionisch beschrieben werden kann.

11 Implementierungsperspektive

Eine robuste Implementierung sollte folgende Prinzipien beachten:

1. **Typprüfung vor Operatorausführung:** Jeder Operator prüft Eingangstypen, Kontext und Zieltyp.
2. **Explizite Kopplungsoperatoren:** Ebenenzustände werden nicht implizit berechnet, sondern durch benannte Operatoren erzeugt.
3. **Rückkopplung mit Verzögerung:** Feedbacks aus t wirken auf Entscheidungen in $t + 1$.
4. **Konfliktauflösung als eigenes Objekt:** Prioritätsregeln gehören nicht versteckt in numerische Formeln.
5. **Traceability:** Relationale Zustände dokumentieren, welche Aufträge, Maschinen, Regeln und Messwerte zu einem Ergebnis geführt haben.

Der Simulationskern kann als diskreter Zeitschritt implementiert werden:

Listing 3: Minimaler Simulationsschritt als Pseudocode.

```
def step(state):
   new_Q3 = update_Q3(state.Q3, state.Q4, state.Q2, state.Q1)
   new_Q1 = update_Q1(state.Q1, new_Q3, state.Q4, state.Q2)
   new_Q2 = execute_Q2(state.Q2, new_Q1, new_Q3)
   new_Q4 = evaluate_Q4(state.Q4, new_Q2, new_Q3)
   return MQMState(Q1=new_Q1, Q2=new_Q2, Q3=new_Q3, Q4=new_Q4)
```

12 Grenzen und Validierungsbedarf

Der Artikel formuliert ein Struktur- und Simulationsmodell, aber noch keine empirisch validierte Produktionssimulation. Für eine belastbare Anwendung sind mindestens vier Validierungsschritte nötig:

1. **Domänenkalibrierung:** Parameter wie $\alpha(m)$, $\beta(m)$, λ und Qualitätsausbeuten müssen aus Daten oder Expertenwissen bestimmt werden.

2. **Operatorvalidierung:** Für jeden Kopplungsoperator muss geprüft werden, ob Eingangs- und Ausgangstypen sowie Einheiten konsistent sind.
3. **Szenariotests:** Rückstand, Energieüberschreitung, Qualitätsabfall und Kapazitätsausfall sollten als Testszenarien simuliert werden.
4. **Vergleich mit realen Prozessdaten:** Output, Energie, Durchlaufzeit und Qualität müssen gegen historische oder experimentelle Daten geprüft werden.

Die wichtigste methodische Grenze besteht darin, dass die Operatorbibliothek kein universelles Naturgesetz darstellt. Sie ist eine formale Schnittstelle zur kontrollierten Definition domänenspezifischer Kopplungen.

13 Fazit

Das entwickelte Modell präzisiert das M-Q-M-Modell als hierarchische, typisierte und rückgekoppelte Simulationsstruktur. Die Quadranten $Q_1(S)$ bis $Q_4(S)$ sind Teilstrukturen eines Systems; ihre Zustände entstehen aus Achsen- und Ebenenzuständen. Da Zustände numerisch, ordinal, nominal und relational sein können, ist ein klassischer Vektorraum nicht ausreichend. Statt eines allgemeinen Tensorprodukts werden kontextabhängige Kopplungsoperatoren eingeführt.

Am Beispiel einer virtuellen Produktionszelle zeigt sich, wie Planung, Steuerung, Prozess und Ergebnis in einem diskreten Simulationszyklus zusammenwirken. Rückkopplungen von Output, Qualität, Energie und Stabilität machen das Modell dynamisch und entscheidungsfähig. Die UML-nahe Architektur bietet eine direkte Grundlage für eine objektorientierte Implementierung. Weiterführende geometrische Interpretationen, etwa über Quaternionen oder Hopf-Fibration, bleiben als Forschungsoption erhalten, sollten jedoch von der praktischen Simulationsarchitektur getrennt validiert werden.

Literatur

[1] Projektunterlage: *UML Modell für MQM*. Interne Arbeitsnotiz, bereitgestellt im Projektkontext, 2026.

[2] Projektunterlage: *Ontologisches und Mathematisches Modell*. Interne Arbeitsnotiz zur geometrischen und ontologischen Deutung des MQM-Modells, 2026.

[3] S. Mac Lane: *Categories for the Working Mathematician*. 2. Auflage, Springer, 1998.

[4] A. Hatcher: *Algebraic Topology*. Cambridge University Press, 2002.

[5] B. P. Zeigler, H. Praehofer, T. G. Kim: *Theory of Modeling and Simulation*. 2. Auflage, Academic Press, 2000.