M-Q-M Werkzeugkette

Um die Nachhaltigkeit der Ergebnisse aus dem BMBF geförderten F&E-Projekt Lfd. Nr.: 189-720-722/2021-1/1, Kurzbezeichnung; „Dynamische Steuerung hochkomplexer Systeme mit Hilfe eines IT-gestützten Mehr-Quadranten-Modells“ zu gewährleisten, ist eine systematische Überführung des entwickelten Mehr-Quadranten-Modells „M-Q-M“ über Methoden und Techniken in tägliches Engineering nötig.

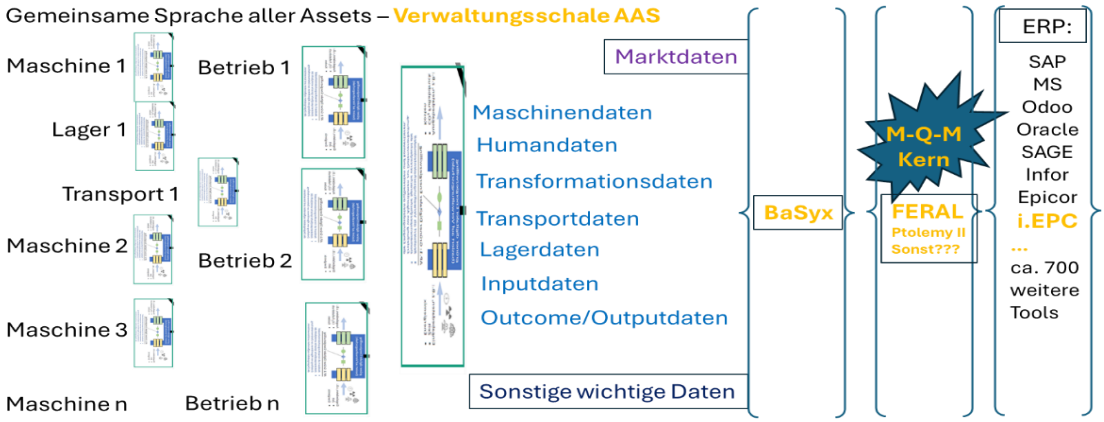
- Um einen zielsicheren Transfer zu ermöglichen, bedürfen insbesondere die entwickelten Konzepte einer durchgängigen modellbasierten Entwurfsmethodik für eine geeignete werkzeugtechnische Umsetzung.
- Um eine nachhaltige Ergebnisverwendung auch jenseits der Laufzeit des Forschungsprojekts sicherzustellen, soll in dem aktuellen Arbeitspaket (Validierung) daher die Definition und Realisierung einer integrierten und pragmatischen Toolunterstützung durch FERAL hier zunächst auf konzeptueller und später auf konkreter technischer Ebene bearbeitet werden.

Zur Erreichung des Vorhabenziels werden im aktuellen Arbeitspaket zwei unterschiedliche Ansätze verfolgt:

1) Die konkrete Realisierung von M-Q-M Methoden und Techniken in einem industriellen Kontext. Dies umfasst die Erweiterung existierender Werkzeuge um konkrete, noch zu entwickelnde Methoden und Techniken. Dies mit dem Ziel einer breiten Dissemination von M-Q-M Ergebnissen im industriellen Engineering.

2) Die Bereitstellung einer offenen Werkzeugkette, die als pragmatische Toolunterstützung die M-Q-M Methoden und Techniken auf konzeptueller und konkreter technischer Ebene darstellt. Diese offene Werkzeugkette soll später als (freies) Erprobungswerkzeug der M-Q-M Methodik zur Verfügung gestellt werden und anschließend als konkretes Demonstrationswerkzeug der entwickelten Schulungs- und Lehrmaterialien dienen.

Tool-Kombination für einen angestrebten Prototypen



Das lizensierte Tool **FERAL (Fraunhofer IESE)** wird im Folgenden zunächst mit seinem Open Source Basis-Tool **Ptolemy II (UC Berkeley)** in Bezug auf **Zeit-/Time-Domains** durch ChatGPT „Kurt“ verglichen:

Kerndifferenz FERAL vs. Ptolemy II in einem Satz

Ptolemy II hat ein **sehr formalisiertes, reiches Zeitmodell** (u. a. *superdense time* mit Microsteps, mehrere „gleichzeitig gültige“ Zeitbegriffe in hierarchischen/Modal-Modellen), während **FERAL** Zeit vor allem als **praktische Simulations-/Co-Simulationszeit** zur **Kopplung von Simulatoren** über MOCCs (z. B. DE/DT, plus Realzeit-Synchronisation) behandelt – mit zusätzlichen „Zeitdomänen“ wie **historisch** eher als **anwendungs-/methodenspezifische Erweiterung** (z. B. im MQM-Kontext). (feral-docs.iese.de)

Gegenüberstellung nach Zeit-Aspekten

Zeit-Aspekt	FERAL (Fraunhofer)	Ptolemy II (UC Berkeley)
Dauer (Duration)	Dauer ergibt sich i. d. R. aus **Simulationszeit-Differenzen** (z. B. Schrittweite bei DT, Ereignisabstände bei DE). FERAL kann außerdem **Simulationszeit vs. Wall-Clock** koppeln. (feral-docs.iese.de)	Dauer ist Teil des **modellierten Zeitfortschritts** je Domäne (CT/DE/DT/SR …). In CT werden Zeitinkremente ggf. **adaptiv verfeinert**. (ptolemy.berkeley.edu)
Stetiger/kontinuierlicher Zeitfluss	In den FERAL-Konzepten wird „continuous time“ als gängige MOCC-Familie genannt, aber in den aktuellen MOCC-Detailseiten stehen sichtbar **DE/DT/SRT** im Vordergrund. (CT ist damit eher als „unterstützt/angedacht“ dokumentiert als formal ausgeführt.) (feral-docs.iese.de)	**Continuous Time (CT)** ist eine Kern-Domäne: Signale sind kontinuierlich, Zustände ändern sich mit Zeitfortschritt; ein **numerischer Solver** bestimmt Signalwerte über Zeitinkremente, ggf. adaptiv. (ptolemy.berkeley.edu)
Diskrete Zeit (Fixed-step, Ticks)	**DT-MOCC**: Zeit schreitet in **festen Inkrementen** fort; synchrone Updates pro Schritt, gut für periodisches Sampling. (feral-docs.iese.de)	**DT-Domain** existiert explizit und definiert u. a. **global time + local time** (pro Actor) und nutzt ein **period**-Parameter zur Zeitfortschreitung. (ptolemy.berkeley.edu)
Periode / Sampling	Periodik ist typisch über **DT-Schrittweite** bzw. in DE über wiederkehrende Timer/Events modelliert (praktisch in vielen FERAL-Komponenten „event-/timer-getrieben“). (feral-docs.iese.de)	In Ptolemy ist Periodik zentral in zeitgetriebenen Domänen (z. B. SR/DT); in hierarchischen Mischungen wird **period** teils benötigt, damit Zeit „vorankommt“. (ptolemy.berkeley.edu)

Zeit-Aspekt	FERAL (Fraunhofer)	Ptolemy II (UC Berkeley)
Zeitpunkte / Time stamps	In DE werden Zustandsänderungen durch **Events** ausgelöst; Events können priorisiert werden. (Dokumentiert als „state changes at discrete points in time“.) (feral-docs.iese.de)	Im **DE-MoC** ist ein Event ein Wert **mit globaler Zeit**; Reaktionen folgen chronologischer Verarbeitung und **weak causality**. (ptolemy.berkeley.edu)
Gleichzeitigkeit & Reihenfolge bei gleichem Zeitpunkt	In FERAL-DE ist Priorisierung erwähnt, aber ein **formalisiertes Microstep-Konzept** (gleiches t, verschiedene „n“) ist in den sichtbaren Konzeptseiten nicht der zentrale Mechanismus. (feral-docs.iese.de)	Ptolemy II nutzt **superdense time**: Zeitstempel sind **(t, n)** mit **Microstep n**, um Ereignisse **bei gleicher Modellzeit** deterministisch zu sequenzieren. (ptolemy.berkeley.edu)
Historische Zeit	In den MQM/FERAL-Unterlagen wird neben „stetig“ und „diskret“ zusätzlich eine **historische Zeitdomäne (HS)** sowie „elementar (SubS)“ erwähnt. Das wirkt wie eine **fachliche Zeitdomänen-Erweiterung** (z. B. für MQM-Wissens-/Historienbezug), nicht zwingend wie ein „klassischer“ Simulations-MoC-Kern à la DE/CT.	Ptolemy II führt „historische Zeit“ i. d. R. nicht als eigene MoC-Zeitdomäne; Historien/Traces werden üblicherweise als **Ereignis-/Signalfolgen mit Zeitstempeln** innerhalb existierender Domänen modelliert (DE/CT/DT …). (ptolemy.berkeley.edu)
Reale/Uhrzeit (Wall-clock) vs. Simulationszeit	FERAL dokumentiert explizit **Time Synchronization** zwischen Simulationsuhr und Realzeit (Wall-Clock) sowie Betriebsarten, wo Simulation „unter Wall Clock“ läuft. (feral-docs.iese.de)	Ptolemy unterscheidet konzeptionell (insb. in Modal Models) **local, environment, global, real time (wall-clock)**; außerdem wird „model time“ bewusst von real time getrennt. (ptolemy.berkeley.edu)
Event-/Event-triggered	FERAL ist stark auf **event-getriebene** Simulatio-triggered worker“). (feral-docs.iese.de)	In Ptolemy ist DE der klassische event-getriebene MoC; zusätzlich werden hybride Kombinationen (CT + DE + Modal) sehr formal unterstützt. (ptolemy.berkeley.edu)

Was das praktisch bedeutet (kurz)

- Wenn du **sehr präzise Semantik** brauchst für „gleichzeitige“ Ereignisse, **deterministische Reihenfolge bei gleichem Zeitpunkt**, oder **mehrere Zeitbegriffe in Hierarchien/Moden** → **Ptolemy II** ist hier konzeptionell „tiefer“ (superdense time, lokale/Umgebungs-/Global-/Realzeit in Modal Models). (ptolemy.berkeley.edu)
- Wenn du **heterogene Simulatoren koppeln** und **virtuelle Engineering-/vHiL-Szenarien** aufbauen willst, inkl. **Simulationszeit ↔ Wall-Clock-Synchronisation** und MOCC-basierter Ausführung (DE/DT, plus Realzeit-Aspekte) → **FERAL** zielt stärker auf diese praktische Integrationsrolle. (feral-docs.iese.de)
- Die in den Unterlagen erwähnte **historische Zeitdomäne (HS)** ist ein Punkt, der bei FERAL im **MQM-Kontext** explizit auftaucht, aber bei Ptolemy eher als Modellierungsfall *innerhalb* bestehender Domänen behandelt wird.

Was ist FERAL (online auffindbare Kernbeschreibung)

- Fraunhofer beschreibt FERAL als Framework/Plattform zur **Kopplung heterogener Simulatoren und Simulationsmodelle**, die unterschiedliche **Models of Computation and Communication (MOCC)** nutzen.
- In der öffentlichen FERAL-Doku wird es als Plattform für die **schnelle Erstellung von Simulations- und virtuellen Evaluations-Tools** beschrieben; Kern ist die **MOCC-basierte Kopplung** (hierarchisch).
- FERAL ist **Java-basiert**, modular als **Komponentenbibliothek** aufgebaut und wird über **Simulationsskripte** konfiguriert/verdrahtet (vergleichbar mit SystemC als „scripted setup“).

Wofür wird FERAL eingesetzt (typische Use Cases)

- **Virtuelle Prototypen / Digital Twins**: Kopplung von Modellen, existierendem Code und virtuellen Hardware-Plattformen, um Architektur- und Systementscheidungen früh abzusichern.
- **MiL / SiL / vHiL (Virtual HiL)**:
  - MiL z. B. mit UML-Zustandsautomaten/Activity-Diagrammen oder Kopplung mit **Matlab/Simulink**
  - SiL (Software-in-the-Loop) zur Kombination erster Software-Realisierungen mit Modellen
  - vHiL durch **virtuelle Hardwareplattformen** (Prozessor-/Netzwerkmodelle), auf denen Software „deployt“ werden kann; Ziel ist u. a. weniger teure klassische HiL-Tests
- **Automotive/ADAS & autonomes Fahren**: Fraunhofer nennt z. B. die Notwendigkeit, das komplette Fahrzeug als Digital Twin (Sensoren, Steuergeräte, Aktuatoren + Netzwerktechnik) zu simulieren; zudem wird ein Driving-Simulator beschrieben, inkl. Integration diverser Modelle/Tools und **Fault Injection**.
- **Bus-/Netzwerksimulation** (inkl. Protokoll/Topologie): CAN, Ethernet, FlexRay, LIN, MOST werden explizit genannt.

Technische Konzepte, die man online (und in den zur Verfügung gestellten Dateien) wiederfindet

1) Hierarchie aus Directors (MOCC) und Workers (Simulationskomponenten)

FERAL baut eine **hierarchische Komponentenstruktur** (Baum) auf: Directors definieren MOCC-Semantik, Leaf-Komponenten enthalten Ports; Ausführung wird vom jeweils nächsten Director gesteuert.

Das deckt sich mit akademischen Beschreibungen: Simulation wird in **semantische Domänen** zerlegt, jeweils von einem Director (MOCC) verwaltet; Directors können verschachtelt werden; Workers sind die „Leaf"-Komponenten (z. B. FMUs, Simulink, Netzwerksimulatoren) und kommunizieren über Ports/Links.

2) MOCC-Kopplung (Ptolemy-inspiriert)
Die FERAL-Doku nennt explizit, dass der MOCC-Kopplungsansatz aus dem **Ptolemy-Ansatz** abgeleitet ist, um heterogene Ausführungsmodelle hierarchisch zu koppeln.

3) Integration/Adapter zu anderen Tools
Fraunhofer nennt u. a. **Simulink**, **FMI**, C/C++, Java und externe Komponenten (z. B. HTTP-REST-Services).

Die FERAL-Doku listet Adapter (u. a. Simulink, FMI2/3, SystemC) als Integrationsbausteine.

4) Verteilte Kopplung / FCAPI
Die öffentliche Doku beschreibt eine **FERAL Co-Simulation API (FCAPI)** mit Gateway/Client, mehrere Backends (**UDP/TCP/Shared Memory**) und Client-Libraries z. B. für **C++**, **Python**, **Java** (inkl. Sim-Zeit-Synchronisation).

Konkretes Referenzbeispiel: Bosch (aus Datei + Fraunhofer-Webseite)

- Fraunhofer IESE + Robert Bosch: virtuelle Integration/Validierung von ECUs; Einflüsse von Bus und Plattform werden mitgetestet; Integration u. a. von realem Steuergerätecode (AUTOSAR-Treiber), Fahrdynamik-Modellen und Busmodellen.

- In dem Setup wurden Simulink- und FMI-Schnittstellen genutzt; **CAN & FlexRay-Modelle** waren in FERAL bereits enthalten; Ergebnisse wurden u. a. als Paper („Towards virtual validation of distributed functions") publiziert und Bosch hat FERAL lizenziert.

- Dasselbe Referenzthema findet sich auch auf der Fraunhofer-Website („Virtual validation of ECU networks").

Verfügbarkeit (Stand aus Fraunhofer-Quellen)

- Fraunhofer IESE bietet online eine **kostenfreie 14-Tage-Testversion** an und verweist auf die **öffentliche Dokumentation** (technische Hintergründe).

- Im IESE-Jahresbericht/Interview wird zudem ein **Lizenzmodell** beschrieben (inkl. Entwicklungsunterstützung); alternativ kann Fraunhofer auch „als Dienstleistung“ Ergebnisse liefern, ohne direkten Kundenzugang zum Tool.

Wissenschaftliche/öffentliche Referenzen (gut zum Weiterrecherchieren)

- MEMOCODE 2013 Paper: „Feral – framework for simulator coupling on requirements and architecture level“ (Kuhn et al.) wird als zentrale Quelle genannt (z. B. im VALU3S-Repo).

- TU Kaiserslautern (AG Vernetzte Systeme) beschreibt FERAL als model-driven Simulationsframework und nennt u. a. Simulink + CAN-Integration samt Visualisierung/Compliance-Checks.

Was sind MOCCs?

„Model of Computation and Communication (MOCC)“ ist ein formales Rahmenwerk, das verwendet wird, um das Verhalten und die Interaktion von Rechen- und Kommunikationsprozessen innerhalb eines Systems zu beschreiben. MOCCs sind unerlässlich, um Systeme zu verstehen und zu entwerfen, die komplexe Interaktionen zwischen verschiedenen Komponenten beinhalten, insbesondere in eingebetteten Systemen und Echtzeitanwendungen.

Diskretes Ereignismodell der Berechnung (DE-MOCC)¶

Im Diskreten Ereignis-MOCC ändert sich der Systemzustand zu diskreten Zeitpunkten, ausgelöst durch Ereignisse. Jedes Ereignis tritt augenblicklich auf und kann priorisiert werden. Das Modell ist besonders nützlich für Systeme, bei denen der Zeitpunkt von Ereignissen unregelmäßig ist, wie etwa Netzwerksimulationen, digitale Schaltungssimulationen und bestimmte Regelungssysteme. Zu den wichtigsten Merkmalen gehören:

- Ereignisbasierte Ausführung: Das System reagiert auf Ereignisse, sobald sie auftreten.
- Diskrete Zustandsübergänge: Veränderungen treten nur zu Ereigniszeiten auf.
- Simulation: Wird häufig zur Simulation von Systemen verwendet, bei denen Ereignisse die Haupttreiber für Zustandsänderungen sind.

Diskretes Zeitmodell der Berechnung (DT-MOCC)¶

Im Discrete Time MOCC schreitet die Zeit in festen Schritten voran und der Systemzustand wird bei jedem Zeitabschnitt aktualisiert. Dieses Modell eignet sich für Anwendungen, bei denen Aktionen in regelmäßigen Abständen stattfinden, wie digitale Signalverarbeitung, Regelungssysteme und synchrone Schaltungen.

Zu den wichtigsten Merkmalen gehören:

- Zeitliche Ausführung: Das System schreitet in einheitlichen Zeitschritten voran.
- Synchrone Zustandsaktualisierungen: Zustandsänderungen finden an jedem Zeitschritt gleichzeitig statt.
- Regelmäßiges Timing: Ideal für Systeme mit periodischem Verhalten oder Sampling.

Weiches Echtzeit-Berechnungsmodell (SRT-MOCC)¶

Das Soft Real-Time MOCC ist für Systeme konzipiert, bei denen eine rechtzeitige Bearbeitung von Aufgaben wichtig ist, aber gelegentliche Frist-Verpassungen erträglich sind. Sie steht im Gegensatz zu harten Echtzeitsystemen, bei denen das Verpassen einer Frist zu katastrophalen Fehlern führen kann. Dieses Modell ist in Multimedia-Anwendungen, Telekommunikation und anderen Bereichen anwendbar, in denen Leistungsverschlechterungen akzeptabel sind, aber minimiert werden sollten. Zu den wichtigsten Merkmalen gehören:

- Pünktlichkeit mit Flexibilität: Betont das Einhalten von Fristen, toleriert aber gelegentliche Fehltritte.
- Leistungsoptimierung: Konzentriert sich auf die Minimierung von Latenz und maximale Durchsatzleistung.
- Sanfte Verschlechterung: Die Systemleistung verschlechtert sich unter Last anmutig, anstatt abrupt auszubrechen.

Jedes MOCC bietet eine strukturierte Möglichkeit, Zeit- und Interaktionseigenschaften eines Systems zu modellieren und zu analysieren, was es erleichtert, komplexe Verhaltensweisen in rechnergestützten und kommunikativen Prozessen zu entwerfen und zu verifizieren.