Master zur Simulatorkopplung via FMI

Christoph Clauß, Fraunhofer IIS EAS, Zeunerstr. 38, 01069 Dresden,
christoph.clauss@eas.iis.fraunhofer.de
Martin Arnold, Tom Schierz, Martin-Luther-Universität Halle-Wittenberg,
06099 Halle (Saale),
{martin.arnold, tom.schierz}@mathematik.uni-halle.de
Jens Bastian, ITI GmbH, Webergasse 1, 01067 Dresden,
bastian@itisim.com

Zusammenfassung

Die Kopplung von Simulationswerkzeugen erfordert den Datenaustausch wie auch die Synchronisation, d.h. die geeignete Definition von Kommunikationszeitpunkten, an denen dieser Datenaustausch erfolgt. Die Kommunikation mit den Simulatoren, die Bereitstellung von deren Eingangsdaten und die Ansteuerung (Start, Stop, Fehlerverwaltung) unterliegen einem übergeordneten Master-Algorithmus. Mit dem Functional-Mock-Up-Interface (FMI)-Standard wurde eine Schnittstelle für die Kopplung von Simulatoren definiert, die Datenaustausch und Simulatoransteuerung vereinheitlicht. Für die Entwicklung und Erprobung von Master-Algorithmen wurde im ITEA2-Forschungsprojekt MODELISAR am Fraunhofer IIS EAS Dresden in enger Zusammenarbeit mit Projektpartnern ein prototypisches Tool für Entwicklung und Test von Master-Algorithmen („EAS-Master“) geschaffen. Weitgehend basierend auf FMI 1.0 gestattet dieser EAS-Master die Kopplung von zwei oder mehreren Simulatoren und bietet je nach Eigenschaften der beteiligten Simulatoren gegenwärtig folgende Verfahren zur Auswahl an: Mit konstanter Kommunikationsschrittweite eine einfache Abarbeitung ohne Iteration, mit einfacher Fixpunktiteration oder mit Newtonverfahren, sowie mit variabler Kommunikationsschrittweite die einfache Abarbeitung, wobei die Schrittweite durch numerische Fehlerabschätzungen gesteuert wird (Richardson-Extrapolation). Im Beitrag werden die Problematik der Simulatorkopplung und FMI erläutert, sowie der aktuelle Stand des EAS-Masters vorgestellt und an drei Beispielen die Funktionsweise demonstriert. Erweiterungs- und Einsatzmöglichkeiten des EAS-Masters werden diskutiert.

1 Einleitung

In Entwicklungsprozessen komplexer Produkte ist die Modellbildung und die Untersuchung dieser Modelle (Simulation) anstelle der Produkte selbst zum Zweck des Studiums der Funktionsweise sowie zur Optimierung oder Gewinnung von Robustheitsaussagen nicht mehr

wegzudenken. Komplexe, heterogene und multidisziplinäre Systeme z.B. im Automobilbau sind strukturiert. Dies impliziert eine Fülle hierarchisch strukturierter Modelle, die bedingt durch verschiedene im System wirkende physikalische Gebiete, durch verschiedene Untersuchungsziele und wegen der Produktionskette auch durch verschiedene Modellautoren uneinheitlich und oft auf verschiedene Zielsimulatoren ausgerichtet sind. Teilsysteme werden für bzw. mit Simulatoren/Tools modelliert, die besonders gut auf die physikalischen Eigenschaften der Teilsysteme zugeschnitten sind. Ist die simulative Untersuchung mehrerer Baugruppen oder die Gesamtsimulation das Ziel, wird es erforderlich, verschiedenartige Modelle zusammenzubringen. Dazu sind verschiedene Wege möglich:

a) Die Verwendung einer einheitlichen Modellierungssprache, in die alle Modelle zum Zweck der Simulation auf einen einzelnen Zielsimulator konvertiert werden,
b) der Austausch von simulationsfähigen Modellen zwischen den Simulatoren mit dem Ziel der Ausführung aller Modelle auf einem einzelnen Simulator,
c) die gekoppelte Simulation mehrerer Simulatoren (Co-Simulation).

Alle diese Wege sind sinnvoll und werden je nach Situation angewendet. Allgemeine Verhaltensbeschreibungssprachen wie VHDL-AMS oder Modelica gestatten theoretisch die Beschreibung einer großen Breite von Modellen, andererseits ist oft aus praktischen Gründen (Zeit, Kapazität, Toolverfügbarkeit) eine nachträgliche Modellkonvertierung, die meist nicht vollständig automatisiert werden kann, ausgeschlossen. Es ist dann entweder auf (standardisierten) Modellaustausch zwischen den Simulatoren, der nicht die Konvertierung auf Sprachebene erfordert, auszuweichen, oder auf Simulatorkopplung.

Dieser Beitrag stellt als ein Ergebnis des ITEA2-Forschungsprojektes MODELISAR (2008-2011) ein Werkzeug (Master) vor, das die Kopplung von dafür vorbereiteten Simulatoren (Slaves) ermöglicht und das der Simulatorkopplung übergeordnete Algorithmen enthält [1] [2]. Als Schnittstelle zwischen den Simulatoren wird das ebenfalls in diesem Projekt entwickelte Functional-Mockup-Interface (FMI) verwendet. Der Master wurde am Fraunhofer IIS EAS Dresden in enger Zusammenarbeit mit der Martin-Luther-Universität Halle und weiteren Projektpartnern entwickelt und befindet sich in einem prototypischen Zustand, der die Implementierung und Erforschung verschiedener Kopplungsalgorithmen gestattet und mehrere Algorithmen zur Anwendung enthält. Nachfolgend wird die seit langem untersuchte Problemstellung der Simulatorkopplung erläutert, auf die Schnittstelle FMI eingegangen sowie der aktuelle Stand des Masters dargestellt und in drei Beispielen angewendet.

2 Simulatorkopplung

Mit „Co-Simulation“ wird eine Vielfalt von Lösungsansätzen zur Simulation von heterogenen Systemen bezeichnet, aber auch zur Kopplung von Tools oder von numerischen Lösern, und zur Einbeziehung real existierender Systemkomponenten in die Simulation. Im Fokus dieses Beitrages steht die Kopplung zweier oder mehrerer Simulationswerkzeuge zur Simulation von gekoppelten, modularen Systemen.

Unter einem Simulator wird hier nicht nur ein leistungsfähiges kommerzielles oder freies Simulationswerkzeug verstanden, es kann sich auch z.B. um ein einfaches Programm zur Lösung von Differentialgleichungen handeln. Der Simulator sollte im Zeitbereich arbeiten, d.h. die vom Simulator berechneten Variablen sollen zeitabhängige Größen sein.
Um einen Simulator koppeln zu können, muss er bestimmte Eigenschaften erfüllen:

- Er muss zeitinkrementell arbeiten.
- Die Simulation muss unterbrechbar und fortsetzbar sein.
- Während der Unterbrechung muss er berechnete Werte ausgeben und für die Fortsetzung der Simulation Werte annehmen können.
- Er sollte vorgegebene Intervalle mehrfach mit unterschiedlichen vorgegebenen Werten abarbeiten können.

Aus Sicht der Kopplung ist ein Simulator S eine Vorschrift (Funktion), die auf vorgegebene Werte $u(t)$ angewendet wird und Ergebnisse $y(t)$ berechnet (Abbildung 1).

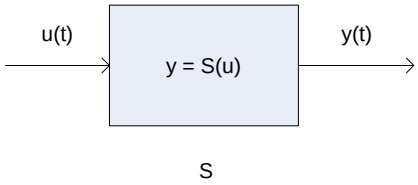

Abbildung 1: Blockdarstellung eines koppelbaren Simulators

Bei der Simulatorkopplung zur Lösung eines partitionierten Problems werden die für die Partitionen gewählten N Simulatoren „verschaltet“, wobei auch möglich ist, gleiche Simulatoren mehrmals zu verwenden. Es entsteht ein gerichteter Graph mit den Simulatoren als Knoten und den Verteilungsvorschriften der berechneten Werten $y(t)$ eines jeden Simulators als Eingänge $u(t)$ anderer Simulatoren als Kanten. Fasst man die Ausgänge $y_i(t)$ aller Simulatoren zum Vektor $Y(t)$ zusammen und alle Simulatoren S_i zu $\overline{S}$, und bezeichne G eine dem Graph zugeordnete Matrix, dann kann die Verkopplung wie folgt kompakt geschrieben werden: $Y(t) = \overline{S}(GY(t))$.

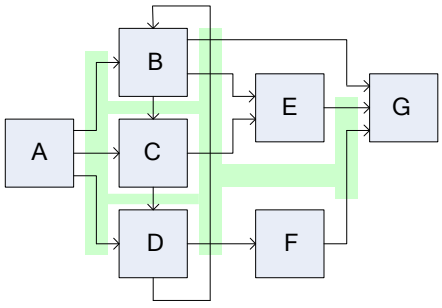

Abbildung 2: Graph mit sieben gekoppelten Simulatoren

Abbildung 2 zeigt als Beispiel eine Verkopplung von mehreren Simulatoren. Statt die Simulatoren direkt zu verkoppeln, wird ein Block zwischen allen Simulatoren angeordnet, mit dem die einzelnen Simulatoren kommunizieren (in Abbildung 2 angedeutet als Fläche

zwischen den Simulatoren). Dieser Block soll **Master** heißen, und die beteiligten Simulatoren **Slaves**.

Der Master hat folgende Aufgaben:

- Der Master arbeitet das vorgegebene, allen Simulatoren gemeinsame Zeitintervall $[t_{start}, t_{stop}]$ ab, indem er Kommunikationszeitpunkte tc_i mit $t_{start} \le tc_i \le tc_{i+1} \le t_{stop}$ definiert und die Slaves in Intervallen $[tc_i, tc_{i+1}]$ zur Simulation auffordert.
- Der Master nimmt an den Kommunikationszeitpunkten tc_i die von den Slaves berechneten Ausgangswerte $y(tc_i)$ auf, verarbeitet diese und stellt entsprechend des Verbindungsgraphen G Eingangsgrößen $u(tc_i)$ an die Slaves bereit.
- Der Master verfolgt für die vorgenannten Aufgaben einen ihm vorgegebenen oder von ihm selbst gewählten Algorithmus, den Master-Algorithmus.

Dieser punktweise Austausch von Werten stellt eine Einschränkung der Allgemeinheit dar, die z.B. den Austausch von kompletten Zeitfunktionen einer bestimmten Klasse wie Polynomen nicht abdeckt. Die Einschränkung wird wenigstens teilweise wieder aufgehoben, indem neben Werten $y(tc_i)$ und $u(tc_i)$ auch Ableitungen nach der Zeit oder Jacobimatrizen übertragen werden können.

Eine Simulatorkopplung nach dem Master-Slave-Prinzip lässt sich grundsätzlich in folgende drei Phasen einteilen:

1. **Initialisierung**: Die beteiligten Simulatoren werden vorbereitet, automatisch oder manuell gestartet, die Datenübertragung wird eingerichtet, und notwendige Initialisierungen erfolgen. Der Master erhält den Verbindungsgraphen G sowie Informationen über die Eigenschaften der Slaves und über den anzuwendenden Master-Algorithmus.
2. **Simulation**: Der Master lässt die Slaves in Intervallen $[tc_i, tc_{i+1}]$ das gesamte Zeitintervall simulieren. Je Kommunikationspunkt übernimmt er die errechneten Werte jedes Slaves, berechnet für diesen neue Eingabewerte und übergibt diese zusammen mit dem nächsten Kommunikationszeitpunkt. Danach wird die Simulation für jeden Slave fortgesetzt. Weiterhin ist je Kommunikationszeitpunkt des Status des Slaves (Fehler, Erfolg, …) zu kommunizieren.
3. **Abschluß**: Der Master beendet die komplette Simulation einschließlich der Beendigung der Slaves.

Die skizzierten Aufgaben des Masters erfordern eine generelle Lösung der Kommunikation zwischen Master und Slave (Daten und Ablaufkommandos) und die Bereitstellung von Masteralgorithmen. Für die Kommunikation wurde das Functional-Mockup-Interface (FMI) geschaffen, Masteralgorithmen werden z.B. im hier beschriebenen Master behandelt.

Bei vielen Kopplungen vor allem zweier Simulatoren wird kein separater Master bemüht, sondern Daten werden nach einem bestimmten Zeitschema ausgetauscht, oft mit fester Kommunikationsschrittweite und ohne Überprüfung der an der Schnittstelle entstehenden Fehler. Derartige Kopplungen lassen sich in den meisten Fällen adäquat auch mit einem einfachen Master-Algorithmus beschreiben, so dass die hier beschriebene Vorgehensweise keine Einschränkung darstellt. „Implizit“ existiert stets ein Master-Algorithmus.

3 Functional-Mockup-Interface

Das Functional-Mockup-Interface [3] [4] ist ein in Entwicklung begriffener Standard für Kopplung und Austausch von Modellen verschiedener Domänen und für Simulatorkopplung. Eine sogenannte Functional-Mockup-Unit (FMU) enthält C-Funktionen entweder im Quelltext, meist aber vorverarbeitet in Form dynamisch linkbarer Bibliotheken („Binary“ wie dll, oder sharable object), und ein Beschreibungsfile im XML-Format. Die C-Funktionen können gerufen werden, um Werte an die FMU zu schicken (z.B. `fmiSetReal`) und Werte abzuholen (z.B. `fmiGetReal`). Wenn die FMU für Simulatorkopplung gedacht ist, enthält sie einen kompletten Simulator (Slave) einschließlich des Modelles, das dieser simuliert, oder sie stellt die Verbindung zu einem separaten installierten Simulationstool her. Dieser Slave kann gesteuert werden, z.B. fordert `fmiDoStep` auf, ein Kommunikationsintervall abzuarbeiten. Um den im vorigen Abschnitt angegebenen grundsätzlichen Ablauf der Simulatorkopplung einzuhalten, können die C-Funktionen nur in einer bestimmten Reihenfolge gerufen werden, die in der FMI-Definition als Ablaufdiagramm festgelegt ist. Im XML-Beschreibungsfile stehen alle „statischen“ Informationen, die zum Aufruf der FMU erforderlich sind, wie Namen, Anzahlen und Typen von Variablen sowie Referenzen für den Zugriff zur Anwendung in den C-Funktionen. Zur Charakterisierung der Möglichkeiten des Slaves in der Simulatorkopplung enthält das XML-File Attribute. Zum Beispiel bedeutet das auf `true` gesetzte Attribut `canHandleVariableCommunicationStepSize`, dass der Slave variable Kommunikationsschrittweite akzeptiert.

In der Definition des FMI sind keine Algorithmen zur Simulatorkopplung festgelegt. Die Schnittstelle ist aber sehr allgemein gehalten, um sehr viele Algorithmen zu ermöglichen.

4 Master

Zur Entwicklung und zum Test von Master-Algorithmen wurde ein Master prototypisch implementiert, der grundlegende Funktionen des FMI 1.0 unterstützt und bereits verschiedene Master-Algorithmen zur Auswahl anbietet. Das am EAS verfügbare Package enthält den ANSI-C-Code des Masters, einen einfachen Slave (C-Funktion) und eine Beispielsammlung.

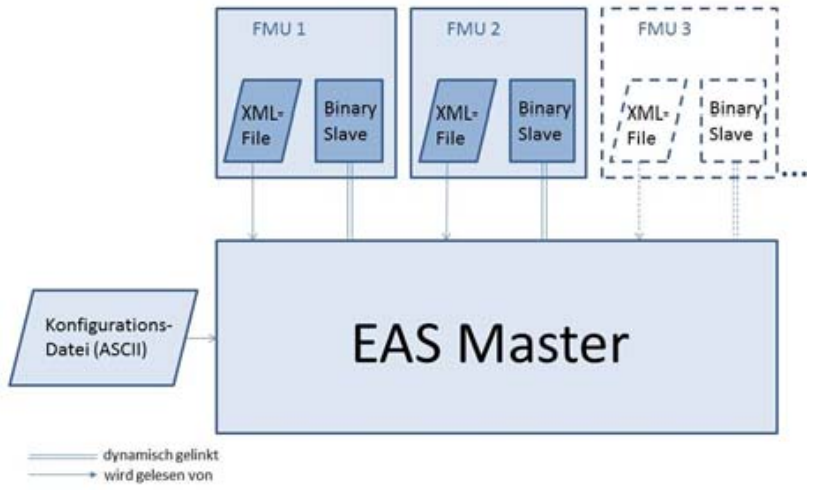

Abbildung 3: Anwendungsprinzip des Masters

Um den EAS-Master mit gegebenen FMUs anzuwenden (siehe Abbildung 3), sind folgende Schritte erforderlich:

- Die Binaries der Slaves sind mit dem übersetzten C-Code des Masters zu verlinken. Dies ist je nach verwendeter oder durch die FMUs vorgegebener Plattform individuell durchzuführen. Gegebenenfalls besitzt die FMU anstelle des Binary auch C-Code, der in die Übersetzung einbezogen werden kann. Ergebnis dieses Schrittes ist ein ablauffähiges Programm, das Master und Slaves enthält. Es ist auch möglich, dass das Binary der FMU eine weitere Schnittstelle (nicht FMI-Schnittstelle) zu einem in einem separaten Prozess z.B. per Hand gestarteten Simulator enthält, der ggf. über Internet angekoppelt ist.
- Die `modelDescription.xml` XML-Files der FMUs werden dem Master durch Angabe des entsprechenden Pfades in der Konfigurationsdatei zur Auswertung bereitgestellt.
- Der Master wird gestartet und erhält dabei als Parameter eine Konfigurationsdatei.

4.1 Aufbau der Konfigurationsdatei

Die Konfigurationsdatei ist eine einfache ASCII-Textdatei mit durch Zeilenwechsel getrennten Angaben. Sie enthält in fest vorgegebener Reihenfolge die Anzahl der auszutauschenden (Einzel-)Werte (`nval`) und die Anzahl der beteiligten Simulatoren (`nsim`). Es folgen Start- und Endzeitpunkt der gekoppelten Simulation (`tstart, tend`) und Angaben zur Schrittweite (`tstepmax, tstepstart`), deren Bedeutung durch den gewählten Algorithmus festgelegt wird. Weiterhin wird der Algorithmus des Masters festgelegt (`MasterMode`), der Debug-Level (`MasterDebug`) zur Überprüfung der Arbeitsweise des Masters und die Ausgabe einer Textdatei zur Anzeige der errechneten Koppelvariablen mit Gnuplot [6] (`OutputGnuplot`). Es folgen Angaben zur Steuerung der Numerik (`it_max_steps, it_tol_abs, it_tol_rel, sz_tol_abs, sz_tol_rel),` die für die einzelnen Master-Algorithmen notwendig sind. Anschließend wird den beteiligten Slaves beginnend ab Null eine Nummer zugewiesen und der Pfad zur FMU angegeben, die dem Slave entspricht. Z.B. bedeutet die Zeile

```
simulator   1   QC_WheelSystem_fmu
```

dass für den Slave 1 im Unterverzeichnis `QC_WheelSystem_fmu` die Datei `modelDescription.xml` für die mit QC_WheelSystem bezeichnete FMU liegt.
Anschließend wird der Verbindungsgraph angegeben, wobei die zu übertragenden Variablen ebenfalls mit Null beginnend nummeriert werden. Z.B. bedeutet folgende Zeile

```
2   0   -1      r 335544320
```

dass die Variable 2 (erste Zahl) vom Slave 0 (zweite Zahl in der Zeile) ausgegeben wird (-1, eine Eingabe würde durch 1 gekennzeichnet), dass es sich um eine reelle Zahl handelt (r), und dass sie unter der `fmiValueReference`-Nummer 335544320 mit der FMI-Funktion `fmiGetReal` vom Master beim Slave 2 abgefragt werden kann. Auf diese Weise werden sämtliche Eingabe- und Ausgabegrößen aller Slaves dem Master mitgeteilt. Jede Variable muss dabei genau einmal als Ausgabegröße eines Slaves vorkommen.

Die folgenden Angaben sind vorläufig noch erforderlich und sollen in einer weiteren Ausbaustufe automatisch durchgeführt werden. Es werden zunächst die im Graph vorhandenen Zyklen festgestellt und je Zyklus zu einem „Superslave“ zusammengefasst. Der Graph ist dann frei von Zyklen und es kann eine Berechnungsreihenfolge angegeben werden. Die höchste Priorität (null) erhalten alle Slaves und Superslaves, die keine Eingänge besitzen. Werden diese und deren Ausgangsvariablen aus dem Graph entfernt, entstehen neue Slaves und Superslaves, die keine Eingänge besitzen. Diese erhalten die nächste Priorität (eins) usw. Im Graph der Abbildung 2 gibt es z.B. einen Zyklus (B, C, D). Slave A hat die Priorität 0, während B, C und D die Priorität 1 erhalten, E und F die Priorität 2 und G die Priorität 3, siehe Abbildung 4.

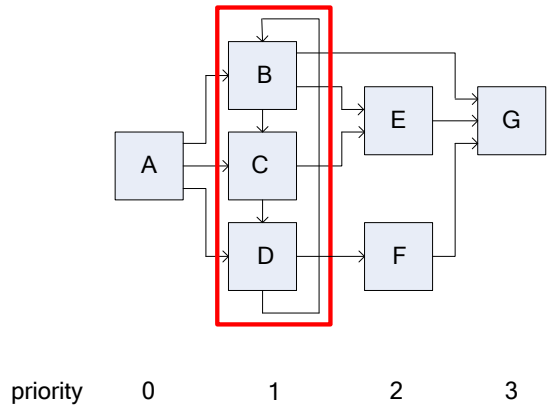

Abbildung 4: Prioritäten im Graph aus Abbildung 2

In der Konfigurationsdatei werden in einer Tabelle die Prioritäten aller Slaves angegeben, und in einer weiteren Tabelle die Anzahl der Zyklen in jeder Priorität. Beispiele von Konfigurationsdateien sind im Punkt 5 angegeben.

4.2 Master-Algorithmen

Die angebotenen Master-Algorithmen rufen die Slaves entsprechend ihrer Priorität auf, beginnend mit der Priorität Null. Sofern sie nicht in Zyklen eingebunden sind, können die Slaves gleicher Priorität parallel abgearbeitet werden. Dies ist vorbereitet, indem bei der Implementierung OpenMP [7] verwendet wurde. Im einzelnen werden folgende Algorithmen angeboten:

- **Einfache Abarbeitung ohne Iteration mit fester Kommunikationsschrittweite:** Die Slaves werden entsprechend ihrer Priorität abgearbeitet. Falls Zyklen vorhanden sind, werden diese genau einmal durchlaufen. Bei diesem Verfahren lässt sich die Genauigkeit durch Reduktion der Schrittweite erhöhen, falls keine Zyklen vorhanden sind. Es treten dann nur Fehler durch die auf die Kommunikationszeitpunkte eingeschränkte Datenübertragung auf. Ableitungen werden bei der Übertragung nicht berücksichtigt. Falls Zyklen vorhanden sind, sind Annahmen für zumindest einzelne Eingangsgrößen erforderlich, die nach Abarbeitung des Zyklus nicht in jedem Fall bestätigt werden und zu zusätzlichen Fehlern führen. Die einfache Abarbeitung

entspricht der einmaligen Auswertung der im Abschnitt 2 angegebenen kompakten Formel $Y(t) = \overline{S}(GY(t))$ in geeigneter Reihenfolge je Kommunikationszeitpunkt.

- **Fixpunktiteration aller Zyklen bei fester Kommunikationsschrittweite:** Die Abarbeitung erfolgt wie bei einfacher Abarbeitung ohne Iteration, jedoch werden alle Zyklen mehrfach durchlaufen, bis sich alle Ein- und Ausgangsgrößen der einzelnen, am Zyklus beteiligten Slaves im Rahmen einer vorgegebenen Toleranz nicht mehr ändern, das heißt, bis die Iteration $Y^{j+1}(t) = \overline{S}(GY^{j}(t))$ konvergiert. Dabei werden die Ausgangsgrößen eines gerade abgearbeiteten Slaves für die folgenden bereits wirksam (Gauss-Seidel-Iteration). Die Iterationszahl wird begrenzt, die Grenze wird im Konfigurationsfile festgelegt. Auch hierbei kann die Genauigkeit durch Verkleinerung der Kommunikationsschrittweite verbessert werden, weil damit der Approximationsfehler bei der Datenübertragung verringert werden kann. Voraussetzung ist die Konvergenz der Iteration bei sinnvollen Fehlerschranken. Nichtkonvergenz mit unbrauchbaren Ergebnissen ist möglich. Um dieses Verfahren anwenden zu können, ist es erforderlich, dass alle an den Zyklen beteiligten Slaves Kommunikationsintervalle mehrfach abarbeiten können.
- **Newton-Verfahren für alle Zyklen bei fester Kommunikationsschrittweite:** Die Abarbeitung erfolgt wie im vorhergehenden Algorithmus, allerdings werden die Zyklen mit einem Newtonverfahren angewendet auf $0 = \overline{S}(GY(t)) - Y(t)$ anstelle der einfachen Fixpunktiteration gelöst. Die Genauigkeit kann über die Kommunikationsschrittweite und die Fehlertoleranzen gesteuert werden. Konvergenz tritt ein bei hinreichend genauer Startlösung für das Newtonverfahren, die durch Übernahme der errechneten Werte des letzten Zeitpunktes und kleiner Kommunikationsschrittweite gewonnen werden kann. Schwierigkeiten können beim Start auftreten, weil eine gute Startlösung fehlt. Auch hier müssen die Slaves Kommunikationsintervalle mehrfach abarbeiten können, auch um die für das Newtonverfahren benötigte Jacobimatrix berechnen zu können.
- **Variable Kommunikationsschrittweite, gesteuert über Richardson-Extrapolation [5]:** Ein Kommunikationsintervall wird zunächst in einem Schritt abgearbeitet. Anschließend werden die Slaves zurückgesetzt, und das gleiche Intervall wird in zwei Zeitschritten mit halber Kommunikationsschrittweite abgearbeitet. Weichen die Ergebnisse zu sehr voneinander ab, wird die Kommunikationsschrittweite verringert, andernfalls wird diese beibehalten oder erhöht. Diese Schrittweitensteuerung kann mit allen drei vorgenannten Algorithmen kombiniert werden. Im aktuellen Master wird die Richardson-Extrapolation mit einfacher Abarbeitung aller Slaves angeboten.

Alle Verfahren berücksichtigen nur kontinuierliche, reelle Ein- und Ausgangsgrößen der Slaves.

5 Anwendungsbeispiele

Zur Erprobung der Master-Algorithmen wurden mehrere Testbeispiele einfacher Art bereitgestellt, bei denen die Slaves in C geschriebene Formelauswertungen sind. Weiterhin wurden Kopplungen mit den Simulatoren SimulationX [8] und Dymola [9] durchgeführt. In diesem Abschnitt wird für jede dieser Kopplungen ein Beispiel angegeben.

5.1 Lineares Gleichungssystem mit zeitabhängiger Matrix

Zeitabhängige lineare Gleichungen werden in folgender Weise je einem Slave zugeordnet:

Slave 0: kein Eingang,	Gleichung: $r_1 = 1,\ r_2 = 0, r_3 = 1$	Ausgang: r_1, r_2, r_3
Slave 1: Eingang r_1, x_2, x_3	Gleichung: $3x_1 + (0.1+t)x_2 + 0.2x_3 = r_1$	Ausgang: x_1
Slave 2: Eingang r_2, x_1, x_3	Gleichung: $0.1x_1 + 3x_2 + (0.1+t)x_3 = r_2$	Ausgang: x_2
Slave 3: Eingang r_3, x_1, x_2	Gleichung: $(0.1+t)x_1 + 0.2x_2 + 4x_3 = r_3$	Ausgang: x_3
Slave 4: Eingang x_1, x_2, x_3	Gleichung: $x_1 + x_2 + x_3 = y$	Ausgang: y

Slave 0 hat Priorität 0, die Slaves 1, 2 und 3 bilden einen Zyklus und haben die Priorität 1, Slave 4 hat Priorität 2 und wird deshalb als Letzter aufgerufen.

Nachfolgend sind die Bestandteile des Konfigurationsfiles für den Master angegeben (allgemeine Angaben, Graph, Prioritätentabelle und Zyklen):

```
nval            7
nsim            5
tstart          0
tend            4
tstepmax        1
tstepstart      1
MasterMode      9
MasterDebug     1
OutputGnuplot   1
it_max_steps    1000
it_tol_abs      1e-6
it_tol_rel      1e-4
simulator   0   bspEmodel0_fmu
simulator   1   bspEmodel1_fmu
simulator   2   bspEmodel2_fmu
simulator   3   bspEmodel3_fmu
simulator   4   bspEmodel4_fmu
```

```
graph #val #sim -1(out)/1(in)
r|i|b|s valueref
3   0   -1      r 0
4   0   -1      r 2
5   0   -1      r 1
0   1    1      r 2
1   1    1      r 0
2   1    1      r 1
6   1   -1      r 3
0   2    1      r 1
1   2    1      r 0
2   2   -1      r 3
5   2    1      r 2
0   3   -1      r 3
1   3    1      r 0
2   3    1      r 1
4   3    1      r 2
0   4    1      r 1
1   4   -1      r 3
2   4    1      r 0
3   4    1      r 2
end
```

```
priority #sim priority
0   0
1   2
2   1
3   1
4   1
end
```

```
cycles  #prior    0(no)/1(yes)
0   0
1   1
2   0
end
```

Bei diesem Beispiel liefert unter den Verfahren mit fester Schrittweite nur das Newtonverfahren richtige Ergebnisse (siehe Abbildung 5).

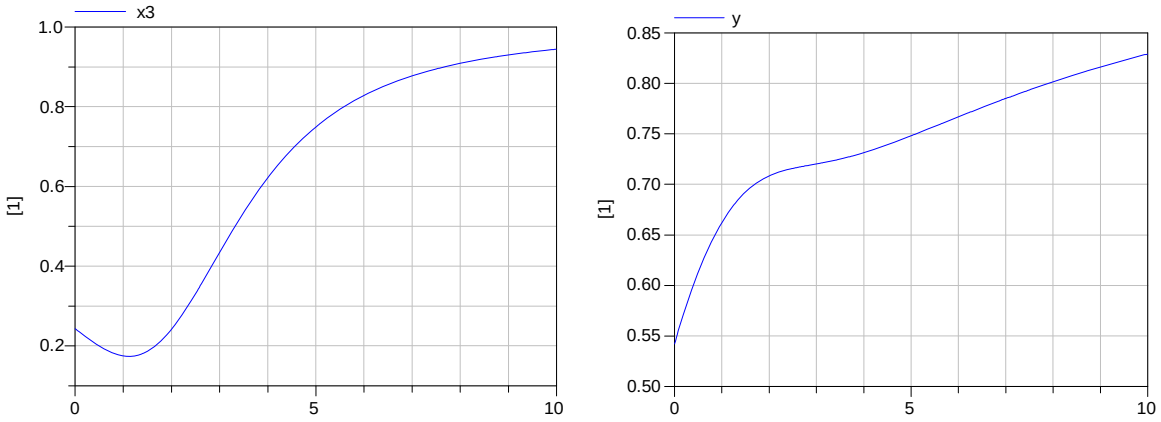

Abbildung 5: Lösungen y und x3 von Beispiel 5.1 (Darstellung über der Zeit in s)

5.2 Geregeltes rotatorisches System

Es werden zwei SimulationX-Slaves (rotatorisches System und Steuerung) und eine einfache C-Funktion mit dem EAS-Master gekoppelt. Das nicht zerlegte Gesamtsystem, dargestellt in Abbildung 6 als SimulationX Modell, ist ein einfaches geregeltes rotatorisches System, dessen Regler durch folgende „Geschwindigkeitsfunktion“ angesteuert wird:

$$\text{}(\text{}) = \text{}^{100rpm} \quad 0.2s$$

$$< t < 1s$$
$$0 \qquad sonst$$

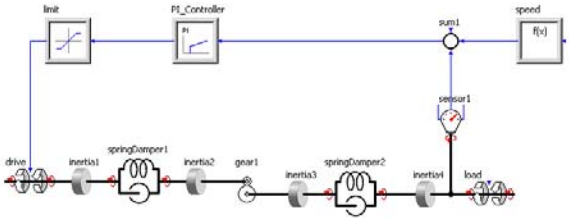

Abbildung 6: Komplettes SimulationX Modell

Dieses Modell wird in drei FMUs unterteilt (Abbildung 7), zwei SimulationX-FMUs für Regler und System und eine C-Funktion-FMU für die Eingabefunktion f(t) (Speed).

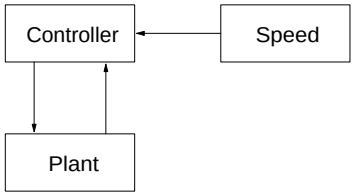

Abbildung 7: Kopplung von drei FMUs

Die FMUs werden mit dem EAS Master gekoppelt unter Verwendung des einfachen Algorithmus‘ ohne Iteration und mit fester Kommunikationsschrittweite von 10^{-3} s. Das Ergebnis der gekoppelten Simulation das und der Referenzsimulation des nicht zerlegten Modells sind in Abbildung 8 dargestellt. Es ist zu erkennen, dass beim originalen Modell eine

kleine, sehr schnell abklingende Oszillation auftritt, nachdem bei 1s die vorgegebene Geschwindigkeit auf Null gesetzt wird. Im Gegensatz dazu entsteht beim gekoppelten Modell eine größere Oszillation, die nicht abklingt. Wurde eine größere Kommunikationsschrittweite genutzt, erhöhte sich die Amplitude dieser Oszillation.

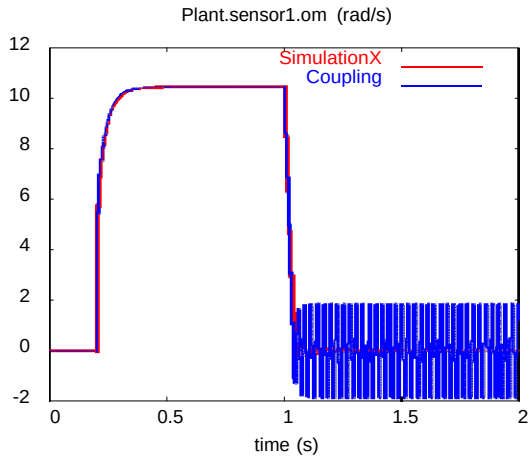

Abbildung 8: Drehzahl bei Simulatorkopplung (stark oszillierend) und ohne Kopplung

Die Konfigurationsdatei des Masters lautet:

```
nval             4
nsim             3
tstart           0
tend             2
tstepmax         0.001
tstepstart       0.001
MasterMode       1
MasterDebug      1
OutputGnuplot    1
it_max_steps     1000
it_tol_abs       1e-6
it_tol_rel       1e-4
sz_tol_abs       1e-4
sz_tol_rel       1e-4
si  0   S
si  1   C
si  2   P
mu      1
```

```
graph #val #sim -1(out)/1(in)
r|i|b|s valueref
0   0   -1  r   0
0   1    1  r   536870913
1   1   -1  r   805306368
1   2    1  r   536870912
2   1    1  r   536870912
2   2   -1  r   805306369
3   2   -1  r   805306368
end
```

```
priority #sim priority
0   0
1   1
2   1
end
```

```
cycles  #prior    0(no)/1(yes)
0   0
1   1
end
```

5.3 Feder-Masse-System

Ein gedämpftes Feder-Masse-System (siehe Abbildung 9) mit zwei Massen soll einen Teil eines Fahrzeuges (Rad und Chassis) beschreiben. Das System wird in Rad- und Chassis-Teil zerlegt. Beide Teile werden getrennt jeweils mit Dymola simuliert, und zwischen den Systemen werden Weg und Geschwindigkeit der Massen ausgetauscht.

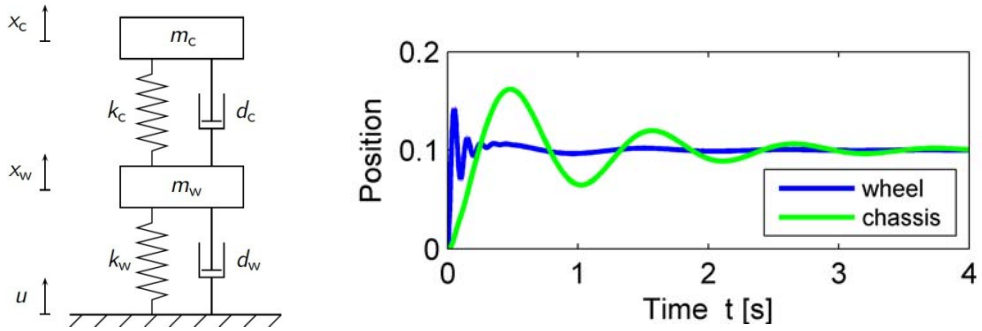

Abbildung 9: Rad-Chassis-System und Simulationsergebnis für die Auslenkungen von Rad und Chassis

An diesem Beispiel wurden bisher vor allem die Master-Algorithmen zur Kommunikationsschrittweitensteuerung basierend auf der Richardson-Extrapolation erprobt. Abbildung 10 zeigt in der unteren Kurve die Änderung der Kommunikationsschrittweite in Abhängigkeit von der Zeit, die mit abklingender Amplitude der Lösung (vergleiche Abbildung 9) tendenziell ansteigt. Übersteigt der geschätzte Fehler des aktuellen Kommunikationsschrittes die Genauigkeitsforderungen (d.h. EST/TOL > 1 in der oberen Kurve in Abbildung 10), so führt dies zum Verwerfen und Wiederholung des Kommunikationsschrittes mit reduzierter Kommunikationsschrittweite. Dies hat zur Folge, dass die Genauigkeitsforderung erfüllt werden kann.

Die Bestandteile des Konfigurationsfiles für den Master sind nachfolgend angegeben. Der Master-Mode 4 kennzeichnet das Verfahren der Richardson-Extrapolation.

```
nval             4
nsim             2
tstart           0
tend             4
tstepmax         0.001
tstepstart       0.001
MasterMode       4
MasterDebug      1
OutputGnuplot    1
it_max_steps     0
it_tol_abs       1e-4
it_tol_rel       1e-4

sz_tol_rel       1e-4
simulator   0    QC_ChassisSystem_fmu
simulator   1    QC_WheelSystem_fmu
```

```
graph #val #sim -1(out)/1(in)
r|i|b|s valueref
0   0    1      r 352321536
1   0    1      r 352321537
2   0   -1      r 335544320
3   0   -1      r 335544321
0   1   -1      r 335544320
1   1   -1      r 335544321
2   1    1      r 352321536
3   1    1      r 352321537
end
```

```
0   0
1   0
end
```

```
cycles  #prior  0(no)/1(yes)
0   1
end
```

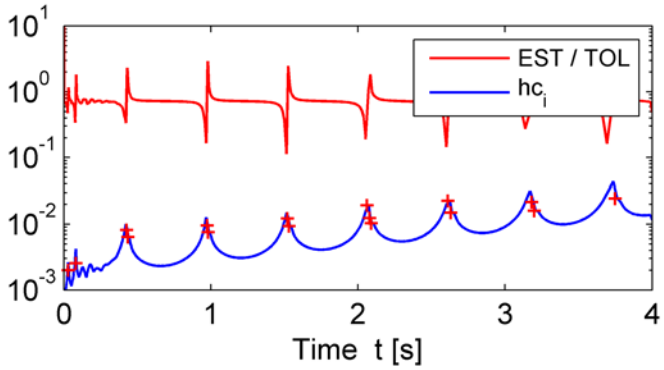

Abbildung 10: Rad-Chassis-System: Kommunikationsschrittweite, Fehlerschätzer und verworfene Schritte (rotes „x“) in Abhängigkeit von der Simulationszeit.

6 Zusammenfassung und Ausblick

Für die Simulatorkopplung zur Simulation von partitionierten Systemen wurde der Prototyp eines Masters vorgestellt, der mittels FMI mit Slaves kommuniziert. Die Slaves sind dabei Bestandteil von Functional-Mockup-Units (FMUs), die das zu simulierende Teilsystem, den zugehörigen Simulator und im FMI 1.0 standardisierte Zugriffs- und Steuerfunktionen enthalten.

Der Master ist zu Forschungszwecken eingerichtet, bietet mehrere Koppelalgorithmen an und kann zur Erprobung weiterer Algorithmen genutzt werden. Neben der Kopplung von Simulatoren kann er auch zum Test von FMUs eingesetzt werden.

Für eine erweiterte, ggf. kommerzielle Nutzung des Masters sind mehrere Verbesserungen notwendig:

- Unterstützung des vollständigen FMI 1.0 und bei Verfügbarkeit FMI 2.0
- Automatische Auswertung des Verbindungsgraphen
- Ausarbeitung weiterer Kopplungsalgorithmen in Abhängigkeit von den Eigenschaften der einbezogenen Simulatoren
- Unterstützung der einfachen Implementierung von nutzerspezifischen Algorithmen
- Erhöhung der Bedienbarkeit, Dokumentation des erreichten Standes
- Intensive Erprobung

Mit dem Master wurde ein Beitrag zur verbreiteten Anwendung des FMI geschaffen und Wege zu erweiterten Erprobung und Anwendung der Simulatorkopplung geleistet.

Dank

Diese Entwicklung wurde vom BMBF innerhalb des ITEA2-Projektes MODELISAR gefördert. Für die Mitwirkung an Teilaufgaben gebührt der Dank T. Blochwitz (ITI GmbH) sowie D. Henriksson und P. Nilsson (Dassault Systèmes).

Literatur

[1] Bastian, J.; Clauss, C.; Wolf, S.;Schneider, P.: Master for Co.Simulation Using FMI. 8th Modelica Conference, Dresden, 2011

[2] Wolf, S.; Blochwitz, T.: Master Slave Simulator Coupling. ITI-Symposium, Dresden, 24./25.11.2010

[3] Arnold, M.; Blochwitz, T.; Clauß, C.; Neidhold, T.; Schierz, T.; Wolf, S.: FMI-for-CoSimulation. 1st International Conference on Multiphysics Simulation, Bonn, 2010

[4] FMI. The Functional Mockup Interface. http://www.functional-mockup-interface.org/.

[5] Kübler, R.: Modulare Modellierung und Simulation mechatronischer Systeme. Fortschritt-Berichte VDI Reihe 20, Nr. 327. VDI-Verlag GmbH, Düsseldorf, 2000

[6] Gnuplot: http://www.gnuplot.info/

[7] OpenMP: http://www.openmp.org

[8] SimulationX: http://www.simulationx.com

[9] Dymola: http://www.dynasim.se