Validierung des Co-Simulations-Masterprogramms MASTERSIM

Andreas Nicolai
Institut für Bauklimatik, TU Dresden
andreas.nicolai@tu-dresden.de

12. Oktober 2020

Zusammenfassung

Die im Functional Mock-Up Interface (FMI) Standard definierte Co-Simulations-Funktionalität ist vielfältig und enthält viele Implementierungsregeln. Die korrekte Umsetzung dieser Regeln ist zwingend notwendig, um Simulationsmodelle verschiedener Tools/Hersteller gemeinsam gekoppelt einsetzen zu können. In diesem Artikel wird die Methodik und Vorgehensweise bei der Prüfung der Implementierung des Co-Simulations-Programms MASTERSIM beschrieben. Es werden einige der dabei erkannten und behobenen Probleme beim Test der Implementierung mit anderen Simulationsmodellen diskutiert, sowie die dafür notwendigen Anpassungen und Erweiterungen am MASTERSIM Programm. Weiterhin werden die Ansätze zur Automatisierung des Testprozederes beschrieben und die verschiedenen entwickelten Skripte vorgestellt.

Inhaltsverzeichnis

1 Hintergrund

Der *Functional Mock-Up Interface* (FMI) Standard definiert eine Programmierschnittstelle für eine Bibliothek, welche eine spezifische Modell-/Berechnungsfunktionalität kapselt. Dadurch können Simulationsmodelle zusammen mit anderen Modellen zur Laufzeit gekoppelt werden. Der Standard defiiert weiterhin den Inhalt und die Struktur von modellbeleitenden Metadaten, welche die Eigenschaften des Modells und der Schnittstelle definieren (z.B. die veröffentlichten Variablennamen und -typen, etc.).

Programme, welche diese Bibliotheken und dazugehörige Metadaten erstellen werden als Exporttools bezeichnet. Die exportierte Programmbibliothek und Metadaten werden in einer Verzeichnisstruktur abgelegt und in einem zip-Archiv verpackt (welches die Endung `.fmu` trägt) und als *Functional Mock-Up Unit* (FMU) bezeichnet wird. Programme, welche derartige FMUs verwenden und damit Simulationen durchführen sind Importtools.

Der Standard definiert zwei Kopplungsmodi: *FMI for Model Exchange* und *FMI for Co-Simulation*. Bei Letzterem enthält jede FMU einen eigenen Zeitintegrator, sofern dynamische Gleichungen zu lösen sind. Bei Ersterem muss das importierende Tool einen solchen Integrationsalgorithmus zentral für alle importierten FMUs bereitstellen. Das Programm MASTERSIM ist ein Importtool für Co-Simulation FMUs, daher auch *Co-Simulations-Masterprogram* genannt.

2 Validierungsmethodik

Bei der Entwicklung des Programms MASTERSIM wurden nach Erreichen einer Mindestfunktionalität, entwicklungsbegleitende Modultests entwickelt. Diese testen (und demonstrieren) die korrekte Implementierung der verschiedenen Masteralgorithmen und Zeitschrittsadaptionstechniken. Im Detail sind diese Tests in [1] und [2] beschrieben. Eine kurze Zusammenfassung der Algorithmentest ist in Abschnitt 3 beschrieben.

Anschließend erfolgte eine Validierung durch Querschnittstests entsprechend der FMI Cross-Check-Regeln. Diese Validierung ist Voraussetzung für die Auflistung des Programms in der offiziellen Tool-Seite des FMI-Standards und dient der Information über mögliche Interoperabilität zwischen Simulationsprogrammen. Für einen Co-Simulations-Master ist die Fähigkeit, möglichst viele unterschiedliche FMUs aus verschiedenen Tools verwenden zu können, eine der wichtigsten Eigenschaften. Die Details dieser Validierungsprozedur und der benötigten Anpassungen am MASTERSIM-Programm sind in Abschnitt 4 beschrieben.

Eine allgemeine Einführung zu Validierungsmethodiken und Verfahrensweisen zur Qualitätssicherung bei Simulationsprogrammen einschließlich der Beschreibung von Modul- und Querschnittstests ist [3] in gegeben.

3 Modultests

3.1 Konzept

MASTERSIM kann verschiedene Kopplungsalgorithmen verwenden. Diese unterscheiden sich hinsichtlich der Funktionalitäten der beteiligten FMUs. FMUs mit Schnittstelle 1.0 können nicht zurückgesetzt werden, wodurch eine Iteration über den gleichen Kommunikationsschritt nicht möglich ist. FMUs mit Schnittstelle 2.0 können zurückgesetzt werden, wodurch auch iterierende Algorithmen, d.h. die Wiederholung eines Kommunikationsschritts, möglich ist.

Weiterhin unterscheiden sich die Algorithmen bei der Zeitschrittweitenanpassung:

- feste Kommunikationsschrittweite,
- Schrittweite reduzieren bei Konvergenzfehler (nur iterierende Algorithmen),
- Schrittweite entsprechend Fehlerschätzer anpassen.

Natürlich haben auch numerische Konstanten, wie die Schrittweitenbegrenzung nach oben und unten, die geforderten Toleranzen bei fehlerkontrollierten Verfahren, und die Ausgabehäufigkeit einen Einfluss auf das Ergebnis.

Alle Kombinationsmöglichkeiten indivuell zu testen, wäre extrem zeitaufwändig und nur bedingt zielführend. Es wurden daher einzelne Tests entwickelt, die immer eine sinnvolle Kombination der Optionen vereinen und charakteristische Ausgaben erzeugen.

Der Testfall *Math003* (einer, aus einer Reihe von speziell definierten Testfällen für Co-Simulation) ist dabei besonders geeignet, da drei indivuell unterschiedliche FMUs mit unstetigem Verhalten gekoppelt werden. Dabei lassen sich die theoretisch erwarteten Ergebnisse genau ablesen. Abbildung 1 zeigt beispielsweise, wie bei einem einfachen Gauss-Jacobi-Algorithmus ohne Iteration die Information über Variablenänderungen von einem Kommunikationsschritt zum

nächsten von einer FMU (hier Variable) in die nächste propagiert wird. Dies ist für diesen Algorithmus das erwartete Ergebnis und muss bei korrekter Implementierung auch genau so erhalten werden.

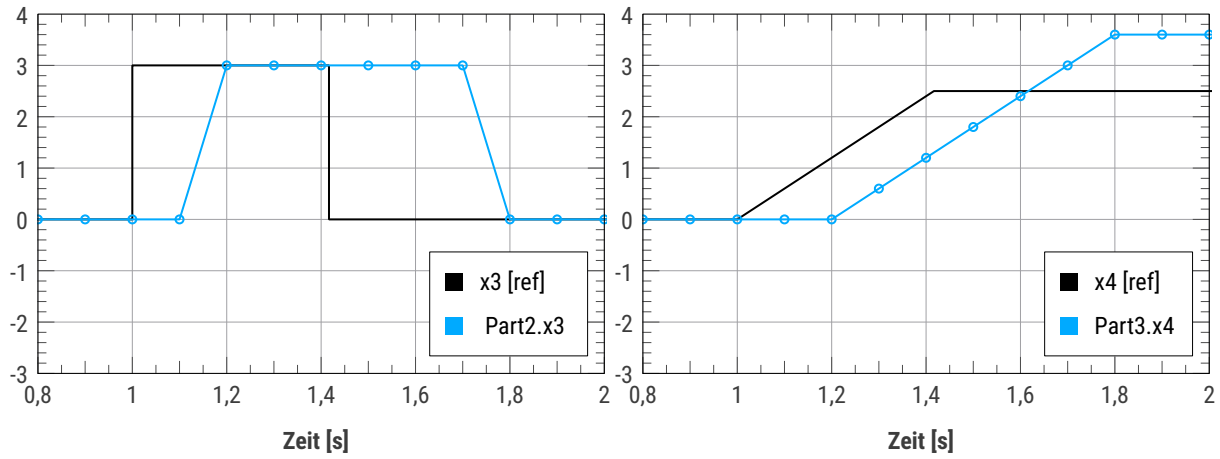

Abbildung 1: Beispielergebnisse der Modultest-Validierung, Vergleich exakte Referenzlösung mit numerischer Lösung: einfacher Gauss-Jacobi-Algorithmus ohne Iteration mit relativ großer Zeitschrittweite; um einen Schritt verzögerte Signalweiterleitung in Variable x_3 und wiederum um einen weiteren Schritt verzögerte Reaktion von Variable x_4 sind *korrekte* Ergebnisse für diese Algorithmus.

Für diesen Testfall und zwei weitere Testfälle wurden nun verschiedene Projektdateien mit unterschiedlichen Algorithmen- und Parameterkombinationen erstellt, ausgeführt und die Ergebnisse kontrolliert. Danach wurden die erhaltenen Ergebnisse und Solverstatistiken als Referenz für zukünftige Kontrollrechnungen abgelegt (im Quelltextrepository im Unterverzeichnis `<repo-root>/data/tests`).

3.2 Verwendete FMUs

Eine Schwierigkeit bei der Validierung von Co-Simulations-Programmen liegt in der Verwendung von FMUs als Black-Box-Simulationsmodule. Die Korrektheit der Masterprogramm-Funktionalität hängt direkt von der Voraussetzung ab, dass die Berechnungsfunktionalität in den FMUs korrekt funktioniert.

In den ersten Versuchen wurden die Testgleichungen in Modelica implementiert und dann aus einer Modelica-Entwicklungsumgebung (hier SimulationX und Dymola) FMUs exportiert. Es zeigte sich jedoch, dass aufgrund der bei diesem Export verwendeten numerischen Zeitintegrationsverfahren und der individuellen Umsetzung des Modelica-Modells in Machinencode Abweichungen vom idealen vorgegeben Verhalten auftraten. Diese wiederum verfälschten die Ergebnisse der gekoppelten Simulation und machten eine direkte Prüfung der MASTERSIM-Implementierung schwierig.

Daher wurden für die Tests eigene, spezifische C++ Implementierung mit exakt den geforderten Gleichungen und analytischen Integrationsgleichungen implementiert. Diese Test-FMUs sind strukturell sehr ähnlich, sodass für den Zweck des Test/Referenz-FMU-Entwicklung ein eigenes Projekt entwickelt wurde. Dieser FMICodeGenerator[1] kann dazu verwendet werden, sehr schnell ein fertig compilierfähiges Grundgerüst für eine FMU zu erstellen, worin nur noch an wenigen Stellen im Quelltext die eigentliche Mathematik implementiert werden muss (siehe auch Beispiel auf den Wiki-Seiten des Projekts[2]). Mithilfe dieses Code-Generators wurden nun alle für die MASTERSIM-Modultests benötigten FMUs erstellt.

3.3 Automatisierte Modultests während der Entwicklung

Die Erstellung des Programms MASTERSIM erfolgt scriptgesteuert. Nachgelagert kann ein Python-Script (entwickelt am IBK, TU Dresden) gestartet werden, welches eine Verzeichnisstruktur `<repo-root>/data/tests` durchläuft, und nach Testfällen sucht (`*.msim`-Dateien) . Existiert für den Testfall ein Unterverzeichnis mit Referenzergebnissen für die aktuelle Plattform und Compiler, wird der Testfall simuliert und die Simulationsstatistiken und Ergebnisse verglichen. Listing 1 zeigt exemplarisch einen Auszug aus dem Validierungsreport:

[1] `FMICodeGenerator` Quelltext-Repository: https://github.com/ghorwin/FMICodeGenerator

[2] https://github.com/ghorwin/FMICodeGenerator/wiki/Test-scenario:-Stateless-P-controller, Wiki-Seite mit komplettem Ablauf der Script-gestützten FMU-Generierung

```
...

../../data/tests/linux64/Math_003_control_loop/Math003_GaussSeidel_2iters_adaptive_Richardson.msim
                                  Reference             New
  ConvergenceFails                       41 ==            41
  ConvergenceIterLimitExceeded           41 ==            41
  ErrorTestFails                         85 ==            85
  MasterAlgorithmSteps                  324 ==           324
  --
  ErrorTestTime                        0.00 ~~          0.00
  FrameworkTimeWriteOutputs            0.01 ~~          0.07
  MasterAlgorithmTime                  0.00 ~~          0.00
  Slave[1]Time                         0.00 ~~          0.00
  Slave[2]Time                         0.00 ~~          0.00
  Slave[3]Time                         0.00 ~~          0.00
  WallClockTime                        0.01 ~~          0.07


Checked: values.csv
../../data/tests/linux64/Math_003_control_loop/Math003_GaussSeidel_2iters_fixedStep.msim
                                  Reference             New
  ConvergenceFails                       0 ==             0
  ConvergenceIterLimitExceeded          95 ==            95
  ErrorTestFails                         0 ==             0
  MasterAlgorithmSteps                1000 ==          1000
  --
  ErrorTestTime                        0.00 ~~          0.00
  FrameworkTimeWriteOutputs            0.02 ~~          0.02
  MasterAlgorithmTime                  0.00 ~~          0.00
  Slave[1]Time                         0.00 ~~          0.00
  Slave[2]Time                         0.00 ~~          0.00
  Slave[3]Time                         0.00 ~~          0.00
  WallClockTime                        0.02 ~~          0.02


Checked: values.csv

Successful projects:

Project path                                                  Wall clock time [s]
FileReaderSlave/LV_Richardson_PredatorFromFile_noiter                      0.318
Lotka_Volterra_System/LV_Richardson_5iter                                  0.244
Lotka_Volterra_System/LV_Richardson_noiter                                 0.224
Math_003_control_loop/Math003_GaussJacobi_1iters_fixedStep                 0.040
Math_003_control_loop/Math003_GaussSeidel_1iters_fixedStep                 0.024
Math_003_control_loop/Math003_GaussSeidel_2iters_adaptive                  0.006
Math_003_control_loop/Math003_GaussSeidel_2iters_adaptive_Richardson       0.073
Math_003_control_loop/Math003_GaussSeidel_2iters_fixedStep                 0.023
```

Listing 1: Auszug aus dem Validierungsreport für MASTERSIM (automatische Modultests)

4 FMI Cross-Check Validierung

4.1 Hintergrund

Wie im Abschnitt 3 über Modultests beschrieben, können verschiedene Algorithmen und Optionen zu zum Teil sehr unterschiedlichen Ergebnissen führen. Man könnte den Ansatz dennoch für die Validierung verschiedener Co-Simulations-Master verallgemeinern, d.h. man benennt für jeden Testfall kontret die zu verwendenden Algorithmen und Parameter. Dann könnte man entsprechend strenge Übereinstimmung mit entsprechend passenden Referenzwerten fordern.

Zum Zeitpunkt der Einführung des FMI Standards wurde die FMU-Importfunktionalität in vielen bereits vorhandenen Tools nachgerüstet. Dabei wurden viele Kompromisse und Workarounds verwendet, um relativ schnell eine gewisse Basiskompatibiltät herzustellen. Es zeigte sich jedoch, dass nur wenige Importtools (d.h. Co-Simulations-Masterprogramme) mit nur wenigen exportieren FMUs zusammenarbeiten konnten. Dies lag an zum Teil unspezifischen Formulierungen im Standard, unterschiedlichen Interpretationen bzw. Interpretationsfehlern und natürlich in Programmier- und Umsetzungsfehlern. Hätte man nun den strengen Ansatz aus der Modulprüfung zum Test verwendet, würden noch mehr Tools durch das Validierungsraster fallen. Das würde aber dem Entwicklungsziel der maximalen Interoperabilität des FMI-Standards zuwiederlaufen und im Wiederspruch zu dem Wunsch des FMI Konsortiums stehen, möglichst schnell den FMI Standard zu etablieren und zu verbreiten. Deshalb ist ein wesentlich einfacheres Testprozedere formuliert worden.

In diesem Testprozess, formuliert in den Cross-Check-Regeln, wurde der Schwerpunkt auf Interoperabilität gelegt.

4.2 Vorgehensweise und Validierungsprinzip

Das Prinzip des Cross-Checks kann wie folgt zusammengefasst werden:

1. verschiedenen Simulationsprogramme exportieren FMUs und führen für diese zunächst einen Standard-Übereinstimmungs-Check (engl. *Compliance check*) durch. Dieser prüft grundlegende Eigenschaften, wie z.B. korrekt gesetzte Metadaten, und grundlegend korrekte Funktion der Programmierschnittstelle. Es wird hierbei Testweise eine einfache alleinstehende Simulation mit Standardwerten durchgeführt (konstante Kommunikationsschrittweite, keine Iteration und kein Datenaustausch mit anderen FMUs). Ist der Test erfolgreich abgeschlossen, wird die FMU in eine Verzeichnisstruktur überführt. Diese wird in einem fmi-cross-check-Repository öffentlich sichtbar vorgehalten[3].
2. Zusätzlich zu der FMU selbst muss man Referenzergebnisse ablegen. *Hierbei sind die Cross-Check-Regeln sehr knapp und unpräzise formuliert. Es ist z.B. nicht definiert, ob die Referenzergebnisse mit der FMU selbst erzeugt werden, oder mit einem anderen Berechnungsverfahren/Modell erstellt wurden. Es ist daher möglich, dass selbst bei korrekter Funktionalität des FMU-Importtools, die Referenzergebnisse nicht wiedergegeben werden können.*
3. Wurden bei der Berechnung der Referenzergebnisse Eingangsvariablen im Modell verwendet, sind diese auch in einer csv-Datei anzugeben. *Die Cross-Check-Regeln definieren hier jedoch nicht, wie verschiedenen Datentypen in der csv-Datei identifiziert werden. Es muss daher aktuell angenommen werden, dass der Datentyp dem der verknüpften Eingangsvariable der FMU entspricht.*
4. Jedes FMU-Importtool (z.B. MASTERSIM als Co-Simulations-Masterprogramm) kann nun diese FMUs einbinden und simulieren. Hierbei obliegt es dem importierenden Tool zu entscheiden, ob Zeitschrittanpassung oder konstante Zeitschritte verwendet werden. Auch alle weiteren (numerischen) Parameter, können beliebig gewählt werden.
5. Die simulierten Ergebnisse werden nun offiziell dokumentiert. Hierfür muss eine Verzeichnisstruktur im oben genannten fmi-cross-check-Repository entsprechend folgendem Verzeichnisschema befüllt werden:

```
results/
  {FMI_Version}/
    {FMI_Type}/
      {Platform}/
        {Importing_Tool_Name}/
          {Importing_Tool_Version}/
            {Exporting_Tool_Name}/
              {Exporting_Tool_Version}/
                {Model_Name}
```

6. In dieses Verzeichnis kopiert man die erstellten Simulationsergebnisse und bei Erfolg die Datei `passed`. Diese Änderungen werden in einem persönlichen Fork[4] vom offiziellen Repository eingecheckt und dann via Pull-Request[5] an das offizielle Repository übertragen.
7. Beim Erstellen des Pull-Requests werden die eigenen Simulationsergebnisse mit den Referenzergebnissen verglichen (siehe Abschnitt 4.2.1). Stimmen die Ergebnisse hinreichend gut überein, so gilt der Test als Bestanden und der Betreuer des offiziellen fmi-cross-check-Repositories übernimmt die Daten in die offizielle Struktur. Damit wird auch automatisch die Webseite mit dem FMI-Kompatibilitätslisten aktualisiert.

4.2.1 Vergleichskriterium

Für den Vergleich zwischen Referenz- und Simulationsdaten wurde seitens des FMI-Konsortiums bewusst ein Test mit sehr großen Toleranzen gewählt. Fokus lag zunächst darauf, sicher zu stellen, dass die Tools nicht kompletten Unsinn ausrechnen und recht bald viele miteinander (im Prinzip) koppelbare Simulationsprogramme entstehen. Folgendes Vergleichskriterium wird verwendet:

> *„The provided import results are filtered against the reference solution provided by the exporter using the following epsilon band method:*
> *Each reference signal is re-sampled (using linear interpolation) into an array of 1000 equally spaced samples between the first and last element of the reference time. The upper limit y_max of the epsilon band is calculated by taking the maximum of 21 neighboring values (10 to either side) at each sample point and adding 10% of the absolute maximum of all values in the reference signal (or 0.5 if all values are equal 0). The lower y_min is calculated respectively. The validation passes if at least 90% of the samples in the provided output are inside the epsilon band defined by y_min and y_max (again linearly interpolated at the respective sample time of the result file). Samples before and after the first and last sample time of the reference result will be ignored.“*[6]

[3] Github-Repository mit Cross-Check Daten: https://github.com/modelica/fmi-cross-check
[4] Ein *Fork* ist eine Abspaltung von einem Repository, d.h. eine Kopie eines bestimmten Bearbeitungsstandes.
[5] Ein *Pull-Request* ist eine Anfrage an die Verwalter/Entwickler des ursprünglichen Repositories, von dem ein Fork erstellt wurde, die in der Anfrage enthaltenen Änderungen zu übernehmen.
[6] `FMI-CROSS-CHECK-RULES.md`, https://github.com/modelica/fmi-cross-check, vom 02.08.2019

Dieses Akzeptanzband ist recht grob, funktioniert aber praktisch recht gut. Abschnitt 4.8 zeigt einen Testfall, wobei Referenzergebnisse und MASTERSIM-Simulationsergebnisse (unter Verwendung unterschiedlicher Parameter und Algorithmen) einschließlich des Toleranzbandes dargestellt werden.

4.2.2 Nicht geprüfte Einflussfaktoren

Bei der Prüfung einzelner FMUs in einem Importprogramm können jedoch einige (wichtige) Aspekte nicht geprüft werden:

- Funktioniert der Masteralgorithmus, d.h. die *Koordination des Datenaustauschs zwischen mehreren FMUs* korrekt?
- Werden die Algorithmus-Parameter und/oder eingebaute Konstanten und Verhaltensregeln korrekt umgesetzt?
- Funktionieren fortgeschrittene FMU-Funktionen, wie das Rücksetzen oder Speichern/Wiederherstellen des FMU-Zustands?
- Wie performant ist die jeweilige Master-Implementierung?

Für solche Tests müssen noch einige Fragestellungen geklärt werden:

- Wie ist das zu testende Co-Simulationsszenario definiert? Spielt die Reihenfolge der Auswertung bei mehreren FMUs eine Rolle?
- Wie ist der Vergleich mit den Ergebnisgrößen definiert? Wie wird mit Rundungsfehlern umgegangen? Woran erkennt man ein Fehlverhalten einer Implementierung?

Die Entwicklung solcher erweiterter Algorithmentests ist aktuell noch in Arbeit. Die Modultests vom MASTERSIM könnten hierbei als Vorlage dienen.

4.3 Ausgangspunkt bei der Cross-Check Prüfung

Die MASTERSIM Version 0.5.3 vom September 2018 war Ausgangspunkt der Cross-Check-Prüfung. In dieser Version waren die Modultests bereits für alle implementierten Algorithmen und die selbst erstellten FMUs erfolgreich absolviert worden. Eine erste Analyse ergab:

- FMI 1.0
  - Win32 - 25 erfolgreich
  - Win64 - 16 erfolgreich
- FMI 2.0
  - Win32 - 22 erfolgreich
  - Win64 - 13 erfolgreich

Anbetracht von derzeit 557 Test-FMUs alleine für die Win32 und Win64 Plattformen, war die Kompatibilität des MASTERSIM-Tools in diesem Entwicklungsstand definitiv noch nicht zufriedenstellend.

4.4 Ausschluss von FMUs

Wärend der Tests stellte sich heraus, dass einige FMUs nicht den Cross-Check-Regeln entsprachen. Insbesondere betraf dies diejenigen FMUs, welche ein lizensiertes Simulationsprogramm benötigten. Da diese Lizenz nicht bereitgestellt wurde, konnten diese FMUs nicht validiert werden und wurden aus der Liste der Test-FMUs entfernt (in der Übersicht in der Spalte *rejected* aufgeführt). Ebenso ausgeschlossen wurden FMUs, die im Repository mit der Datei `notCompliantWithLatestRules` markiert wurden (d.h. nicht den aktuellen Cross-Check Regeln entsprechen).

Weiterhin gab es einige FMUs, die bezüglich des Simulationsendzeitpunkts eine strikte Prüfung implementiert haben. Diese ist nach aktueller Ansicht[7] nicht standardkonform, jedoch der eventuell unterschiedlich auslegbaren Standardformulierung geschuldet. Hierfür konnte jedoch eine Speziallbehandlung in MASTERSIM integriert werden (siehe Abschnitt 4.6.14).

[7] siehe Diskussion in Ticket `https://github.com/modelica/fmi-standard/issues/575`.

4.5 Automatisierung des Validierungsprozederes

4.5.1 Schritte bei der Validierung ohne Skriptunterstützung

Zum Verständnis der nachfolgend erläuterten Automatisierung sollen kurz die Arbeitsschritte beim Validieren einer einzelnen FMU mittels MASTERSIM vorgestellt werden. Es wird hierzu als Beispiel der Testfall *CoupledClutches* ausgewählt.

1. Erstellen eines neuen MASTERSIM-Projekts `CoupledClutches_xc.msim`.
2. Bei diesem Testfall ist eine `CoupledClutches_in.csv`-Datei vorhanden, welche als erstes unter dem Namen *CoupledClutches_in* importiert wird.
3. Import der zu testenden FMU `CoupledClutches.fmu` in das Projekt als *CoupledClutches*. Die MASTERSIM Programmoberfläche zeigt nun die importierten FMUs (Abbildung 2).

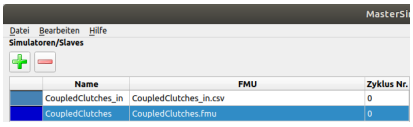

Abbildung 2: Ansicht der importierten FMUs (Simulationsslaves)

4. Verknüpfung der Variablen der von der `csv`-Datei bereitgestellten Variablen mit den Eingangsvariablen in der Verknüpfungsansicht. Bei sehr vielen Variablen kann das semi-automatisch durch die „Verknüpfe anhand gleicher Namen“-Funktionalität erfolgen. Bei diesem Beispiel ist nur die Variable u zu verknüpfen (Abbildung 3).

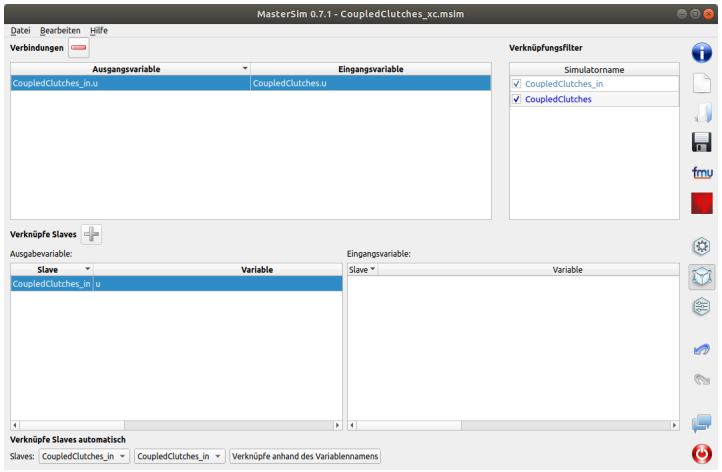

Abbildung 3: Ansicht der verknüpften Variablen

5. In der Simulationsansicht (Solveroptionen) werden nun die Simulationzeiträume eingetragen. Diese sind in der Datei `CoupledClutches_ref.opt` aufgeführt, welche den Inhalt hat:

```
StartTime, 0.000000
StopTime, 1.500000
StepSize, 0.010000
RelTol, 0.000100
```

   Die Zeitangaben sind dabei in Sekunden.

6. Es wird als Master-Algorithmus *Gauss-Seidel, 1 Iteration*, ausgewählt (nicht-iterierender Algorithmus)[8]. Beim Gauss-Seidel-Algorithmus wird zuerst Variable der `csv`-Datei am Ende des Kommunikationsintervals ausgewertet und dann die Variable $u\,(t_{comm,end})$ an die FMU übergeben.
7. Die Option zur Zeitschrittweitenanpassung wird deaktiviert, da mit konstanten Kommunikationsschritten gerechnet werden soll.
8. Die in der Datei `CoupledClutches_ref.csv` bereitgestellten Ergebnisgrößen sind interne bzw. lokale Variablen. Da MASTERSIM standardmäßig nur Ausgabevariablen (causality=output) in Ausgabedateien schreibt, muss die Option „Schreibe interne Variablen als Ausgaben“ eingeschaltet werden.

[8] Bei der automatisierten Validierung wurde alternativ der Gauss-Jacobi Algorithmus ohne Iteration verwenden, da viele der anderen Tools offensichtlich diesen auch verwendet haben und so die Übereinstimmung größer war.

9. Als Ausgabeinterval (`hOutputMin`) wird 0,01 s gewählt, also eine Ausgabe je Kommunikationsinterval. Bei Co-Simulationsanwendungen ist dieses Ausgabeinterval entscheidend, da eine Ausgabe der Variablen nach jedem Kommunikationsschritt zu unnötig großen Ausgabedateien führen würde. Daher wird diese Interval als Raster für Ausgaben definiert, d.h. es wird nach jeder erfolgten Ausgabe der nächste Ausgaberaster-Zeitpunkt bestimmt und erst nach Erreichen/Überschreiten dieses Ausgabezeitpunkts wird eine weitere Ausgabe erlaubt. Beispielsweise kann so die Zeitintegration mit Schrittlängen 0,001 s durchgeführt werden, wohingegen Ausgaben nur alle 0,1 s erfolgen. Die Umsetzung dieser Regel birgt jedoch zahlreiche Fehlerquellen, wie in den Abschnitten 4.6.9 bis 4.6.12 erläutert ist.
10. Nachdem alle Simulationseinstellungen eingetragen wurden (siehe Abbildung 4) kann die Berechnung gestartet werden.

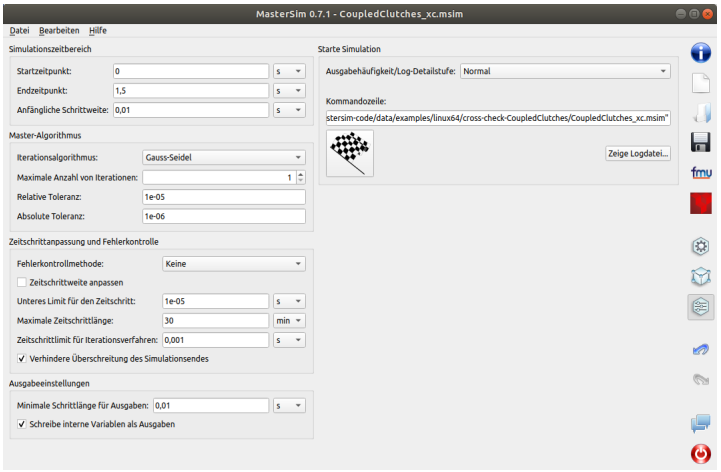

Abbildung 4: Angepasste Simulationsoptionen für den *CoupledClutches* Testfall

11. Die Analyse der Übereinstimmung kann zunächst visuell im Post-Processing erfolgen. Dazu wird das Post-Proc gestartet, und Beispielsweise die Referenzergebnisse *clutch1.psi_rel* und *clutch1.w_rel* zusammen mit den berechneten Werten dargestellt (siehe Abbildung 5).

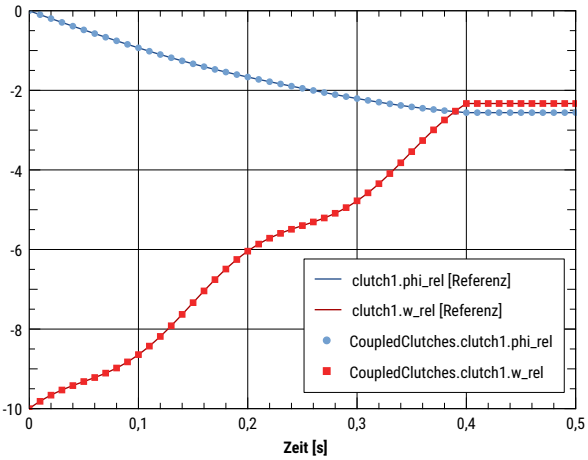

Abbildung 5: Vergleich der Referenzergebnisse (durchgezogene Linien) und Simulationsergebnisse (Symbole). Die Abweichungen sind bei den gewählten Simulationseinstellungen sehr klein.

12. Nun muss man eine `csv`-Datei erstellen, die die gleichen Variablen wie die Datei `CoupledClutches_ref.csv` enthält. Dazu muss man die Ergebnisdatei `values.csv` von MASTERSIM in eine Tabellenkalkulation (LibreOffice, Excel, ...) einlesen. Falls Referenzvariablen nicht in den Berechnungsergebnissen enthalten sind, handelt es sich um synonyme Variablen, d.h. Variablen mit gleichem Wert. Die jeweils entsprechende Ausgabespalte kann man mithilfe der von MASTERSIM generierten Datei `synonymous_variables.txt` suchen und entsprechend umbenennen. Dann enfernt man alle Spalten, welche nicht Teil der geforderten Referenzergebnisse sind. Die Reihenfolge der einzelnen Spalten ist ohne Bedeutung, sodass man die nun bereinigte Ergebnisdatei unter dem Namen `CoupledClutches_out.csv` speichern kann.
13. In der lokalen Arbeitskopie des eigenen Forks des fmi-cross-check-Repositories legt man nun das Verzeichnis `results/1.0/cs/linux64/MasterSim/0.7.0/JModelica.org/1.15/CoupledClutches` an (bzw. jeweils angepasst and Plattform und FMI Version) und kopiert die Datei `CoupledClutches_out.csv` hinein. Zudem erstellt

man noch eine Datei `passed` (z.B. mit der Versionsnummer von MASTERSIM) und `README.md`, wobei man in letztere sinnvollerweise die MASTERSIM Projektdatei kopiert, sodass man später die Werte reproduzieren kann.

14. Nun führt man das offizielle Prüfskript des fmi-cross-check-Repositories aus, d.h. ein Python-Programm, welches die neuen Ergebnisdateien mit den Referenzergebnissen unter Verwendung des Toleranzbandes vergleicht. Ggfs. entfernt das Skript die `passed`-Datei bei zu großen Abweichungen wieder und gibt dabei Fehlermeldungen aus.
15. Ist die Prüfung erfolgreich abgeschlossen, werden die 3 neuen Dateien in das Repository übertragen (via git commit und push) und ein pull-Request ausgelöst.

4.5.2 Skript-Automatisierung

Es ist offensichtlich, dass das manuell Durchführen obiger Arbeitsschritte für jeden der über 800 Testfälle nicht praktikabel ist. Glücklicherweise lassen sich viele Schritte sinnvoll automatisieren. Dies erfolgt im wesentlichen durch 2 selbst entwickelte Python-Skripte:

1. Generieren der MASTERSIM-Projektdateien
2. Ausführen der Simulation, Prüfung der Abweichungen zwischen Referenzwerten und Simulationsergebnissen, Anlegen der Veröffentlichungsdateien für alle bestandenen Testfälle.

Nach Ausführen der beiden Skripte muss man nur noch das offizielle Prüfskript starten, eventuell aufgezeigte Fehler beheben und die Ergebnisse veröffentlichen. Auf diese Art und Weise lassen sich in relativ kurzer Zeit sehr viele FMUs testen.

4.5.3 Skript zur Generierung von MASTERSIM-Projektdateien

Die Generierung der Projektdateien wird mittels Python-Skripten wie folgt automatisiert:

- Einlesen der Verzeichnisstruktur und bestimmen einer Liste von FMUs zum Testen,
- Filtern der Liste nach Co-Simulations-FMUs, ausgewähltem Tool/Hersteller, Plattform (Filterkriterien werden dem Skript als Kommandozeilenargumente übergeben),

Für jeden verbleibenden Testfall werden nun MASTERSIM Projektdateien erstellt, dabei werden:

- Für jeden Testfall ein entsprechendes Testverzeichnis generiert,
- Pfade zur FMU-Datei und ggfs. zur `<fmu>_in.csv` eingetragen,
- Verknüpfungsgraph zwischen Variablen aus der `<fmu>_in.csv` und der FMU erstellt,
- Datei `<fmu>_ref.csv` mit Referenzergebnissen in das Testverzeichnis kopiert,
- `tStart = StartZeit` aus der Datei `<fmu>_ref.opt` gesetzt,
- `tStop = StopZeit` aus der Datei `<fmu>_ref.opt` gesetzt,
- `hStart = StepSize` aus der Datei `<fmu>_ref.opt` gesetzt, bzw. falls `StepSize` $= 0$, auf (`StopZeit-StartZeit`)/1000 gesetzt, und
- `hOutputMin` automatisch gesetzt auf (`StopZeit-StartZeit`)/100.

4.5.4 Skript zur Simulation und Auswertung

4.6 Erkannte Probleme und Lösungen

Nachfolgend sind die beim Testen aufgetretenen Probleme und Fehler in der chronologischen Reihenfolge beschrieben, wie sie aufgetreten sind und behoben wurden.

4.6.1 MASTERSIM übergibt das falsche „Resources“-Verzeichnis bei FMI 1.0

FMUs verweigern die Initialisierung, da das übergebene *Resources*-Verzeichnis nicht existiert. Hintergrund ist eine Änderung des Verzeichnisses zwischen FMI 1.0 und FMI 2.0. MASTERSIM unterstützte bislang nur die FMI 2.0 Konvention.

Lösung: MASTERSIM erhält eine FMI-versionsspezifische Regelung für die Generierung des *Resources*-Verzeichnisses.

4.6.2 MASTERSIM bricht mit Fehlermeldung „`variability` Attribut fehlt“ ab.

MASTERSIM bricht beim Einlesen der modelDescription.xml-Datei bestimmter FMUs ab, und verlangt das Attribut `variability` bei allen Variablen. Im Standard ist dieses Attribut jedoch nicht als notwendiges Attribut aufgeführt und ein Standardwert angegeben.

Lösung: MASTERSIM wird standardkonform angepasst und verwendet `continuous` als Standardwert für das Attribut `variability`.

4.6.3 MASTERSIM hängt beim Schreiben der Ausgaben

Ursache: `hOutputMin = 0 s` ist in der Projektdatei gesetzt. In der gegebenen Referenzergebnisdatei sind identische Zeitpunkte gegeben, entweder wegen Unstetigkeiten (2 Werte beim gleichen Zeitpunkt) oder durch Rundungsfehler beim Schreiben der Zeitpunkte. Ursache ist die automatische Generierung der MASTERSIM-Projektdateien (siehe Abschnitt 4.5.3), welches ursprünglich den minimalen zeitlichen Abstand zweier Referenzergebnisse für `hOutputMin` verwendet hat.

Lösung: MASTERSIM wird angepasst und führt nun Fehlerprüfung durch. Generierungsskript wird angepasst und ignoriert Zeilen mit identischen Zeitpunkten bei der Ausgaberasterbestimmung. Später wird die Ausgabefrequenz ausschließlich in Abhängigkeit des Simulationszeitraums bestimmt.

4.6.4 MASTERSIM hängt bei der Simulation, wenn `hStart = 0 s` gegeben ist.

Ursache ist der Parameter `StepSize = 0` in der `<fmu>_ref.opt`-Datei.

Lösung: MASTERSIM wird erweitert und berechnet standardmäßig einen Zeitschritt von $h = (t_{End} - t_{Start})\,/1000$, wenn `hStart = 0 s` in der Projektdatei steht.

4.6.5 MASTERSIM bricht mit Fehlermeldung „Cannot import function“ ab

Bei der Instanzierung einer FMU bricht MASTERSIM mit der Fehlermeldung `Cannot import function 'van der Pol oscillator_fmiGetReal' from shared/dynamic library` ab.

Ursache: MASTERSIM liest das Attribut *ModelName* mit Wert *van der Pol* und generiert daraus den Symbolnamen`'van der Pol oscillator_fmiGetReal'`. Eine Analyse der von der FMU-shared library exportierten Symbole ergibt:

```
> nm VanDerPol.so

...
000000000000305f T VanDerPol_fmiGetIntegerStatus
00000000000027d1 T VanDerPol_fmiGetReal
0000000000002df5 T VanDerPol_fmiGetRealOutputDerivatives
0000000000003034 T VanDerPol_fmiGetRealStatus
...
```

Die Symbolnamen müssten aus dem *ModelIdentifier* mit Wert *VanDerPol* gebildet werden, welches so auch im Standard beschrieben ist (bei anderen FMUs ist meist ModelName = ModelIdentifier, weswegen das dort nicht aufgefallen ist).

Lösung: MASTERSIM wird entsprechend standardkonform angepasst.

4.6.6 MASTERSIM schreibt Ergebnisse nicht im CSV-Format

Das bisher von MASTERSIM verwendete DataIO Format ist für die automatische Analyse und Abgleich nicht geeignet.

Lösung: Die Ausgabedateien werden auf `csv`-Format umgestellt, wobei Zeichenketten in der Datei `string.csv` ausgegeben werden und alle anderen Werte (Integer, boolische Werte und Gleitkommazahlen) in die Datein `values.csv` geschrieben werden.

4.6.7 Referenzergebnisse nicht automatisierbar auswertbar

In den Referenzergebnissen wird die Zeitspalte ohne Einheit und mal als „time“ oder „Time“ beschriftet. Für die automatisierte Auswertung ist das nicht geeignet.

Lösung: Die Referenzergebnisdateien werden verlustfrei in das geforderte `csv`-Format konvertiert.

4.6.8 Ausgabefrequenz nicht hoch genug, bzw. Schrittweite zu groß

Beim Vergleich von Ergebniswerten mit Referenzergebnissen gibt es Abweichungen, welche durch Verwendung kleinerer Kommunikationsschrittweite reduziert werden können. Probleme gibt es jedoch beim Auftreten von Sprungstellen, da MASTERSIM je Simulationszeitpunkt stehts nur ein Ergebnis hat.

Lösung: Die Kommunikationsschrittweite wird reduziert, bis eine hinreichende Übereinstimmung der Ergebnisse erreicht wird (falls möglich).

4.6.9 Ausgabe enden vor Simulationsende (Rundungsproblematik 1)

Das offizielle fmi-cross-check-Verifizierungsskript beklagt, dass die Simulationsergebnisdatei vor dem Simulationsendzeitpunkt endet, teilweise mit der unsinnigen Fehlermeldung „Ausgaben enden zum Zeitpunkt 1,6 obwohl die Simulation bis 1,6 läuft“. Ursache ist eine Rundungsproblematik, die dazu führt, dass z.B. die Simulation bereits bei 1,599999 beendet wird, obwohl als Endzeitpunkt 1,6 verlangt ist. Dies liegt an Rundungseffekten, wenn z.B. Zeitschritte der Länge 0,001 vielfach aufeinanderaddiert werden. Um dieser Rundungsproblematik zu begegnen wurde bislang die Simulation beendet, wenn $t_{End} - t < 10^{-6}$ war. Zudem hat MASTERSIM bislang Ausgabezeitpunkte mit 10 Stellen Genauigkeit ausgegeben, welches in der Datei zu der Zahl 1,599999 geführt hat. Das Verifizierungsskript hat dann diese Zahlen verglichen und einen Fehler erkannt.

Lösung: Die Genauigkeit der Zeitpunktausgaben in den MASTERSIM Ausgabedateien wurde auf Standardgenauigkeiten (7 signifikante Stellen) reduziert. Damit hatte die letzte Ausgabe den Zeitpunkt 1,6 und wurden als korrekt akzeptiert.

4.6.10 Keine Ausgabe beim Endzeitpunkt (Rundungsproblematik 2)

In einzelnen Fällen ist die Situation eingetreten, dass der Simulationszeitraum nicht exakt ein Vielfaches der Kommunikationsschrittweite war, z.B. $t_{End} = 1,667$ und `hOutputMin` = 0,002. In diesem Fall war die letzte Ausgabe bei t=1,666. Bei Simulationsende t=1,667 war das Mindestinterval für Ausgaben von 0,002 noch nicht abgelaufen, weswegen keine Ausgaben geschrieben wurden.

Lösung: MASTERSIM wird erweitert und schreibt nun am Simulationsende in jedem Fall die Ergebnisse.

4.6.11 Letzte Ausgabe doppelt in Ausgabedatei (Rundungsproblematik 3)

Eine Sonderbehandlung war notwendig, dass nicht unbeabsichtigterweise nach dem letzten Rechenschritt eine planmäßige Ausgabe erstellt wird, gefolgt von der gleichen Ausgabe am Ende der Berechnung. Dies trat z.B. auf, wenn der letzte Integrationsschritt bei t=2,0 endete, der nächste Ausgabezeitpunkt ebenfalls bei t=2,0 gesetzt war. Wenn dann zusätzlich noch die Ausgabe am Simulationsende angehängt wurde, erschien die Ausgabe zum Zeitpunkt t=2,0 doppelt.

Lösung: Erweiterter Test integriert, um Zeitpunkt der letzten Ausgabe zusätzlich zu prüfen.

4.6.12 Ausgabe überspringt Zeitpunkte (Rundungsproblematik 4)

Ein weiteres Problem in Zusammenhang mit der Rundung von Zeitschritten zeigte sich darin, dass zum Teil Ausgabezeitpunkte übersprungen wurden. Dies lag daran, dass MASTERSIM intern den jeweils nächsten erlaubten Ausgabezeitpunkt dadurch berechnete, dass `hOutputMin` so oft addiert wurde, bis t < `hOutputMin` und dieser Zeitpunkt dann als nächster erlaubter Zeitpunkt festgelegt wurde. Allerdings tritt hier wieder die Situation ein, dass durch Rundungsfehler eine Bedingung wie bspw. 1,1000000001 <= 1,1 nicht gilt und deshalb eine Ausgabe übersprungen wird.

Lösung: MASTERSIM merkt sich nun den Zeitpunkt der letzten Ausgabe und bestimmt die Differenz zwischen aktuellem Zeitpunkt und Ausgabezeitpunkt zusätzlich zum Ausgaberaster.

4.6.13 Unterstützung für CSV-Eingabedaten (FileReader-Slaves)

Während die meisten FMUs ohne externe Eingabedateien erfolgreich getestet werden konnen, gibt es auch zahlreiche Testfälle (wie z.B. der oben beschriebene CoupledClutches-Test), welche Eingangsvariablen in `csv`-Dateien bereitstellen. Diese müssen gelesen werden und zur Laufzeit müssen die tabellierten Variablenwerte durch lineare Interpolation berechnet werden.

Lösung: MASTERSIM wird erweitert, sodass neben FMU-Slaves auch FileReader-Slaves verwendet werden können. Beide Implementierungen teilen einen gemeinsame Schnittstelle, wodurch der Großteil der MASTERSIM-Implementierung unverändert bleibt. FileReader-Slaves lesen bei der Instanzierung die jeweilige `csv`-Datei ein und generieren eine virtuelle `modelDescription.xml` Datenstruktur und Zugriffsfunktionen auf die bereitgestellten Variablen (einschließlich der linearen Interpolation).

4.6.14 Überschreitung des Endzeitpunkts verhindern

Einige FMUs verhalten sich hinsichtlich des Überschreitens des Endzeitpunkts nicht standardkonform und brechen die Simulation im letzten Integrationsschritt ab. Dies tritt z.B. dann auf, wenn der letzte Kommunikationsschritt von 1,9 bis 2,0 geht, jedoch die Addition von 1,9 und 0,1 mit Maschinenzahlen einen Wert von 2,000000001 ergibt. In diesem Fall sollte die FMU keinen Fehler ausgeben. Einige FMUs prüfen den Endzeitpunkt jedoch sehr strikt und brechen die Simulation mit einem Fehler ab.

Lösung: MASTERSIM erhält eine zusätzliche Option, um notfalls den letzten Zeitschritt exakt so zu kürzen, dass tatsächlich der Simulationsendzeitpunkt *exakt* erreicht wird.

4.7 Zusammenfassung des Cross-Check-Validierungsstandes für MASTERSIM Version 0.7

Nach umfangreicher Anpassung und Korrektur von MASTERSIM war in der Version 0.7 nun eine recht umfängliche Validierung der Testfälle möglich. Abbildung 6 zeigt eine Übersicht der Validierungsergebnisse für die getesteten Plattformen Windows 64bit, Linux 64bit, und MacOS (Darwin) 64bit. Die Validierung der 32-bit Windows-Version ist für diese Programmversion ausgelassen worden, da dieses Betriebssystem einer immer geringere Verbreitung aufweist.

Platform/OS	**darwin64**			**linux64**			**win64**			**Summe**
Tool	failed	passed	rejected	failed	passed	rejected	failed	passed	rejected	
20sim-4.6.4.8004		1			1			1		**3**
Adams-2017.2						3			3	**6**
AMESim-13SL3					2		2			**4**
AMESim-14					2		2			**4**
AMESim-15					6		3	3		**12**
ASim-2019FD01							1	5		**6**
CATIA-R2015x							2	9		**11**
CATIA-R2016x							2	10		**12**
ControlBuild-2015_FD01					2		2	1		**5**
ControlBuild-2016							2	1		**3**
DS_FMU_Export_from_Simulink-2.1								6		**6**
DS_FMU_Export_from_Simulink-2.1.1								8		**8**
DS_FMU_Export_from_Simulink-2.1.2								8		**8**
DS_FMU_Export_from_Simulink-2.2.0								8		**8**
DS_FMU_Export_from_Simulink-2.3.0								8		**8**
dSPACE_TargetLink-Release_2018-B								3		**3**
Dymola-2015_FD01							1	6		**7**
Dymola-2015FD01							2	4		**6**
Dymola-2016							3	9		**12**
Dymola-2016FD01							4	11		**15**
Dymola-2017							4	12		**16**
Dymola-2019FD01							4	14		**18**
Easy5-2017.1								6		**6**
EDALab_HIFSuite-2017.05_antlia				4						**4**
FMIToolbox_MATLAB-2.1							2	8		**10**
FMIToolbox_MATLAB-2.3							1	7		**8**
FMUSDK-2.0.3								4		**4**
FMUSDK-2.0.4								4		**4**
JModelica.org-1.15					8					**8**
MapleSim-2015.1		3			3			3		**9**
MapleSim-2015.2		3			3			6		**12**
MapleSim-2016.1		3			3			6		**12**
MapleSim-2016.2		3			3			6		**12**
MapleSim-2018		3			3			6		**12**
MapleSim-7.01		1			1			1		**3**
MWorks-2016							2	13		**15**
Silver-3.3							1	4		**5**
Silver-3.5							4	2		**6**
SimulationX-3.6								5		**5**
SimulationX-3.7.41138								10		**10**
Test-FMUs-0.0.1	1	9	1	1	9	1	1	9	1	**33**
Test-FMUs-0.0.2	1	10		1	10		1	10		**33**
Summe Ergebnis	**2**	**36**	**1**	**6**	**56**	**4**	**46**	**237**	**4**	**392**

Abbildung 6: Stand der Cross-Check-Validierung von MASTERSIM Version 0.7

Die meisten fehlgeschlagenen Tests sind begründet mit einer noch verbleibenden Einschränkung im MASTERSIM: die durch `csv`-Eingabedateien bereitgestellten Variablen werden derzeit als vom Datentyp *Real* angenommen und stets linear interpoliert. Bei den Testfällen, wo diese Variablen mit Integervariablen oder boolischen Variablen verknüpft sind, scheitert die Berechnung. Aktuell könnten diese FMUs durch Einsatz von Adaptor-FMUs dennoch validiert werden, in denen eine entsprechende Typkonvertierung möglich ist.

4.8 Beispiel für Abgleich zwischen Simulationsergebnissen und Referenzwerten

Nachfolgend soll beispielhaft anhand von einzelnen Testfällen gezeigt werden, wie unterschiedlich die Berechnungsergebnisse verschiedener Tools im Vergleich zu den Referenzwerten ausfallen können, und wie breit das Toleranzband in diesem Fall gesetzt ist.

4.8.1 Beispiel-Testfall: ControlledTemperature

Abbildung 7 zeigt die Referenzergebnisse des Testfalls `2.0/cs/linux64/MapleSim/2016.1/ControlledTemperature`, die daraus abgeleiteten oberen und unteren Grenzen des Toleranzbandes und die Ergebnisse der Tools, welche diesen Test bestanden haben. Es schein eine recht gute Übereinstimmung der Berechnungsergebnisse zu geben. Man kann in der Abbildung aber auch sehen, dass das Toleranzband sehr breit ist.

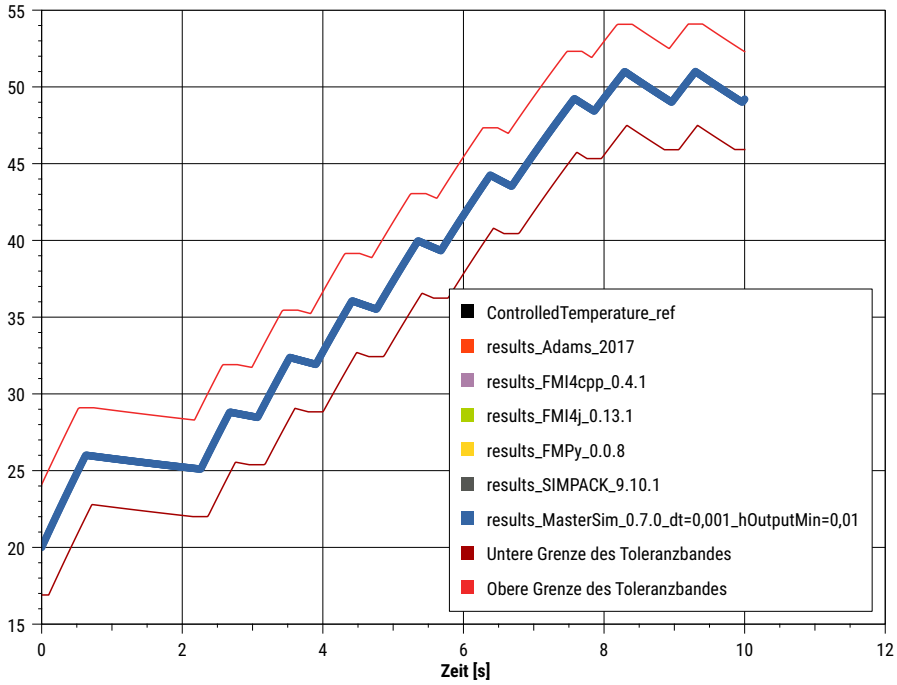

Abbildung 7: Vergleich Referenzergebnisse mit Simulationsergebnissen des Testfalls *ControlledTemperature* und Darstellung des Toleranzbandes.

Bei Detailbetrachtung (siehe Abbildung 8) fallen bereits kleinere Abweichungen in einem FMI-Importtool auf, welche auf ein nicht ganz korrektes Verhalten des Importtools zurückzuführen sind. Diese können aber aufgrund des breiten Toleranzbandes so nicht entdeckt werden.

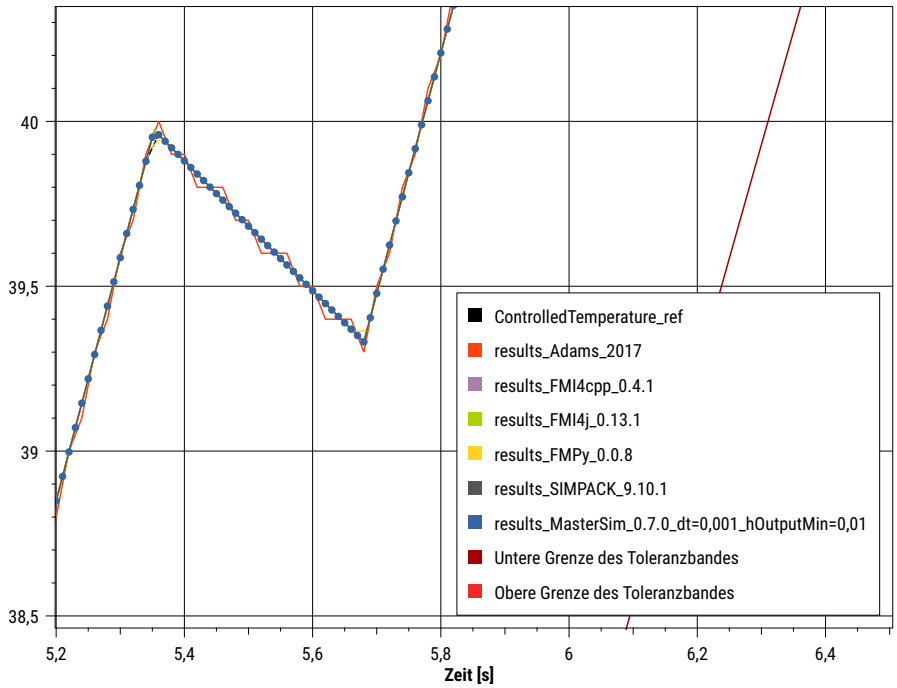

Abbildung 8: Detailbetrachtung der Berechnungsergebnisse mit Abweichungen im Ergebnis des Adams-Solvers, welche durch das breite Toleranzband nicht erkannt werden.

4.8.2 Beispiel-Testfall: Fuelrail

Die recht großzüge Wahl des Toleranzbandes kann jedoch auch dazu führen, dass die Details der Berechnungsergebnisse komplett irrelevant für die Bewertung werden. Abbildung 9 zeigt einen solchen Extremfall, im Testfall `2.0/cs/linux64/AMESim/15/fuelrail_cs`. Glücklicherweise liegen bei diesem Testfall alle Ergebnisse sehr gut übereinander - jedoch würde in diesem Fall auch ein komplett falsches Ergebnis, z.B. ein konstanter Wert von 200, als validiert gelten.

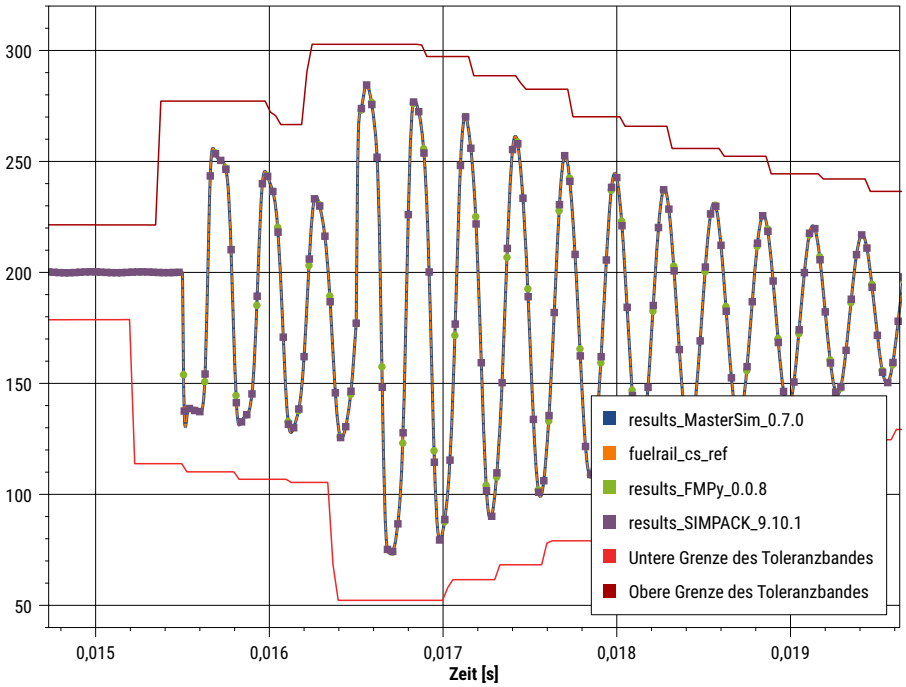

Abbildung 9: Extrembeispiel für ein zu großes Toleranzband. Der Verlauf der hochfrequenten Modellreaktion im Inneren der Toleranzhülle hat keinerlei Einfluss auf die Validierungsbewertung.

Aus diesem Testfall wird deutlich, dass das Verfahren mit dem gleitenden Mittel zur Bestimmung des Toleranzbandes nur bedingt für die automatische Prüfung geeignet ist. Wahrscheinlich wäre es zielführender, wenn der Ersteller eines

Testfalls die Genauigkeit des Toleranzbandes selbst kontrolliert, und gegebenefalls über die Ausgabefrequenz anpasst.

4.9 Eigene Bewertungskriterien für die MASTERSIM-Validierung

Wie oben dargestellt, sind die Kriterien für die Cross-Check-Validierung recht grob gesteckt, und erlauben daher zum Teil auch inkorrekte Ergebnisse die Tests zu bestehen.

Für die Validierung von MASTERSIM wurde deshalb zusätzlich ein eigenes Prüfkriterium eingebaut, welches auch dazu dient, die optimalen Kommunikationsschrittlängen und Ausgabefrequenzen zu bestimmen. Dafür wird eine gewichtete Vektornorm (WRMS-Norm, engl. *weighted-root-mean-square* Norm) zwischen Referenzergebnissen und Berechnungsergebnissen im MASTERSIM berechnet. Die erstellte README.md-Datei enthält dann für alle Referenzvariablen die selbst berechnete Norm in der Form:

```
WRMS(p1@general_hydraulic_chamber) = 2.88624153058
WRMS(p1@hydraulic_4) = 2.38231879683
WRMS(q@volumesensor) = 6.45486466337
```

wobei Werte < 100 eine gute Übereinstimmung signalisieren.

Literatur

[1] Nicolai, Andreas: Co-Simulations-Masteralgorithmen - Analyse und Details der Implementierung am Beispiel des Masterprogramms MASTERSIM. In: *Qucosa* (2018). https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-319735

[2] Nicolai, Andreas: Co-Simulation-Test Case: Predator-Prey (Lotka-Volterra) System. 2019. – Forschungsbericht

[3] Nicolai, Andreas: Validierung von Simulationsprogrammen / Institut für Bauklimatik, TU Dresden. 2019. – Forschungsbericht