MasterSim Benutzerhandbuch

Andreas Nicolai <andreas.nicolai@gmx.net>

Version 0.9.5 (22.12.2022)

Inhaltsverzeichnis

Dieses Handbuch ist verfügbar als:

- PDF Version (Deutsch)
- PDF Version (Englisch)
- Online Handbuch (Deutsch)
- Online Handbuch (Englisch)

Das Programm ist verfügbar unter: https://bauklimatik-dresden.de/mastersim

1. Einführung und Grundbegriffe

MasterSim ist ein Co-Simulations-Masterprogramm, welches die FMI-Co-Simulation unterstützt. Wenn die Co-Simulation für Sie etwas gänzlich Neues ist oder Sie mit dem funktionalen Mock-Up-Interface (FMI) noch nicht vertraut sind, empfehle ich Ihnen, zunächst ein wenig über die Grundlagen zu lesen, z. B. auf der fmi-standard.org-Web-Seite.

Grundsätzlich verbindet *MasterSim* verschiedene Simulationsmodelle und tauscht Daten zwischen Simulation-Slaves zur *Laufzeit* aus. Die folgende Grafik illustriert die einzelnen Komponenten des Programms und den Datenaustausch zwischen diesen.

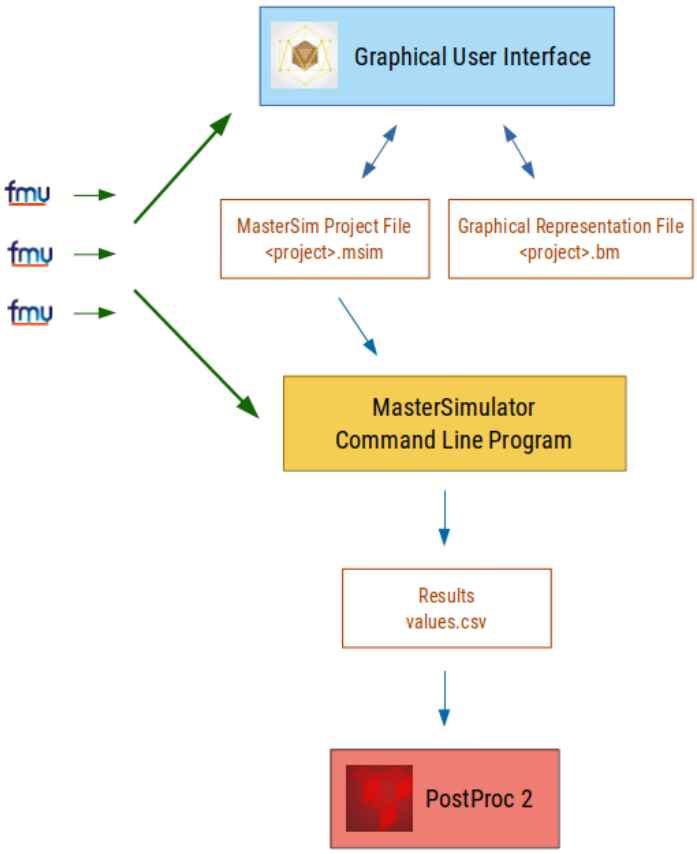

Abbildung 1. Diagramm über den Datenfluss (Dateien) und beteiligten Programme

1.1. Die Teile des Programms

MasterSim besteht aus zwei Teilen:

1. einer grafischen Benutzeroberfläche (*graphical user interface* - GUI) und
2. dem Simulationsprogramm *MasterSimulator* für die Befehlszeile

Die Oberfläche macht es sehr einfach, Simulations-Projekte zu erzeugen, anzupassen und abzuändern. Ein Simulations-Projekt wird in zwei Dateien gespeichert, dem *MasterSim*-Projekt und der grafischen Darstellung (Verknüpfungsschematik). Letzteres ist optional und nicht notwendig für die Simulation.

Die Simulation wird durch das Befehlszeilen-Programm *MasterSimulator* ausgeführt, welches Projekt-Dateien liest, referenzierte FMUs importiert und die Simulation durchführt. Die erzeugten Ergebnisse, sowohl von *MasterSimulator* selbst als auch diejenigen der Slaves werden dann von weiterverarbeitenden Werkzeugen genutzt, um die Ergebnisse zu visualisieren und zu analysieren (das kostenfreie Analysetool mit PostProc 2 ist dafür sehr gut geeignet und meine Empfehlung für die *MasterSim* Ergebnisauswertung).

Die Trennung zwischen der Benutzeroberfläche und dem eigentlichen Simulator macht es sehr einfach, *MasterSim* in einer geskripteten Umgebung oder für eine systematische Variantenuntersuchung zu nutzen, wie sie weiter unten im Abschnitt Arbeitsabläufe beschrieben wird.

1.2. Unterstützte FMU - Varianten

- FMI für die Co-Simulation in der Version 1
- FMI für die Co-Simulation in der Version 2, inklusive der Unterstützung für Zustandssicherung/-Rücksetzung (Serialisierungsfunktionen)

Es werden keine asynchronen FMU-Varianten unterstützt.

MasterSim ist für Linux und MacOS als 64-Bit-Anwendung verfügbar. Für Windows ist *MasterSim* sowohl als 32-Bit als auch 64-Bit-Anwendung vorhanden.

Nutzen Sie für 32-Bit-FMUs eine 32-Bit-Version, für 64-Bit-FMUs eine mit 64 Bit. Gemischte FMU-Plattform-Typen (32 Bit und 64 Bit) werden nicht unterstützt.

1.3. Entwicklungsziele/Schwerpunkte aus Entwicklersicht

- Betriebssystemübergreifend: Windows, MacOS, Linux
- Projektdateien sind im ASCII-Format abgelegt und dadurch leicht editierbar oder durch Skripte zu modifizieren
- da Projektdateien und Netzwerkbeschreibungsdateien ASCII-Format haben, sind sie sehr gut für die Verwendung in Versionskontrollsystemen geeignet
- enthält Instrumentierung und stellt Zähler und Zeitmessungen für den Leistungsvergleich von Master-Algorithmen und FMU bereit

- keine Abhängigkeit von extern installierten Bibliotheken, alle Quelltexte sind im Repository enthalten (Ausnahme: Standard C++ Laufzeitbibliotheken und die Qt 5-Bibliothek), insbesondere gibt es keine Abhängigkeit von FMI-unterstützenden Bibliotheken; den Quelltext vom Repository zu holen und zu erstellen sollte leicht sein (ebenso das Verpacken und Veröffentlichen für andere Plattformen)
- vollständige, in die *MasterSim*-Bibliothek eingebettete Master-Funktionalität, nutzbar über das Kommandozeilen-Werkzeug und die Oberfläche
- Meldungen/Ausgaben sind gekapselt, um GUI-Einbindung und Log-Dateien zu unterstützen, keine direkten `printf()`- oder `std::cout` -Ausgaben
- unterstützt FMU-Fehlerdiagnose: kann das Entpacken deaktivieren (dauerhafte dll/so/dylib-Dateien), der Quelltextzugang erlaubt die Fehlersuche während des Ladens von gemeinsam genutzten Bibliotheken und angefügten Debuggern
- High-Level- C++-Code (lesbar und wartungsfreundlich)
- der Code ist angepasst für die Fehleranalyse des Master-Algorhitmus - alle Variablen sind typspezifisch für eine *einfache Analyse im Debugger* in Datenfeldern gruppiert

Für Details über Funktionen, die insbesondere für FMU-Entwickler und bei Problemen der Fehlerbeseitigung von Co-Simulationen wichtig sind, lesen Sie bitte das Kapitel Assistenzfunktionen für FMU-Entwicklung und Fehlerbeseitigung.

1.3.1. Besondere Funktionen von *MasterSim*

Einige Funktionen in *MasterSim* sind (aus meiner Sicht) sehr praktisch und gibt es so in anderen Co-Simulations-Masterprogrammen nicht.

Es gibt eine besondere Funktion in *MasterSim*, welche hilfreich für die Verwendung von FMUs ist, welche eigene Ausgabedaten schreiben. *MasterSim* erstellt für jede FMU Instanz/jeden Slave ein beschreibbares, *slave-spezifisches* Ausgabeverzeichnis. Der Pfad zu diesem Verzeichnis setzt *MasterSim* im Parameter `ResultsRootDir`. Falls ein Slave einen solchen Parameter definiert, erhält die FMU über den Parameter ein verlässliches, gültiges Verzeichnis zum Hineinschreiben seiner Daten (siehe auch Abschnitt Verzeichnis *Slaves*).

Zyklen bei der Definition von verknüpften Slaves können explizit definiert werden und reduzieren damit dem Aufwand für iterative Masteralgorithmen (siehe auch Abschnitt Zyklen).

MasterSim beinhaltet eine Zeitmessung und Erfassung der Ausführungshäufigkeiten für einzelne Funktionen. Dies ist hilfreich, zeitkritische Simulationen zu optimieren oder Fehler zu finden (siehe Abschnitt Simulations-Statistik/Zusammenfassung).

Verknüpfungen zwischen Ausgangs- und Eingangsvariablen können in *MasterSim* mit Umrechnungsfaktoren und Verschiebungen definiert werden, was besonders für Einheitenumrechnungen und Flussverknüpfungen mit Vorzeichenumkehr praktisch ist (siehe Abschnitt Verbindungsgraph).

1.4. Terminologie

Die folgenden Begriffe werden sowohl im Handbuch als auch in der Benennung von Klassen/Variablen genutzt:

FMU	beschreibt das FMU-Archiv inklusive der Modellbeschreibung und der Laufzeit-Bibliotheken
Slave/Simulator	beschreibt ein Simulationsmodell, dass aus einer FMU erstellt(instanziiert) wird; dabei können mehrere Slaves durch eine einzige FMU erstellt werden, falls die Fähigkeit/Eigenschaft **canBeInstantiatedOnlyOncePerProcess** deaktiviert ist
Master	beschreibt den gesamten Rahmen der Simulationskontrolle, der die generelle Verwaltungsarbeit übernimmt
Simulationsszenario	definiert eine Reihe von Slaves und deren Verbindungen (Datenaustausch) ebenso wie weiteren Eigenschaften, wie z. B. Start- und Endzeit, algorithmische Optionen, Ausgabeeinstellungen; alternativ auch **System** genannt
Netzwerk-Graph	eine andere Beschreibung für die räumliche Struktur miteinander verbundener Slaves
Masteralgorithmus	beschreibt die Implementierung eines mathematischen Algorithmus, der die zeitliche gekoppelte Simulation beschreibt; kann ein Iterationsverfahren enthalten
Fehlerkontrolle	bezeichnet die Prüfung eines lokale Fehlers (Schritt-basiert), genutzt für die Anpassung des Kommunikationsschritts
Masterzeit	Zeitpunkt der Mastersimulation, startet mit 0; die Zeiteinheit ist nicht strikt definiert (es muss eine gemeinsame Festlegung zwischen FMUs geben, normalerweise werden Sekunden verwendet; Ausnahmen sind Datei-Lese-Slaves, siehe Abschnitt CSV-FileReader-Slaves).
Gegenwärtige (Master-)Zeit	Zeitpunkt des aktuellen Master-Zustands; ändert sich **ausschließlich** am Ende eines erfolgreichen `doStep()`- oder `restoreState()`- Aufrufs.

1.5. Arbeitsabläufe

Wie bei anderen Simulationsmodellen beinhalten die meisten Arbeitsabläufe eine Variantenanlyse. Im Kontext der Co-Simulation werden solche Varianten häufig durch die Modifizierung von FMUs und ihrer Parameter erzeugt. *MasterSim* enthält Funktionen, um diesen Arbeitsprozess zu optimieren.

Viele Arbeitsabläufe beinhalten mehrfache Ausführungen von *MasterSim* mit nur

> kleinen oder gar keinen Modifikationen in der Projektdatei. Manchmal ist ist es sehr komfortabel, die selbe Projektdatei zu nutzen und zu verändern, aber ein anderes Arbeitsverzeichnis (für Ergebnisse) zu bestimmen, damit das Resultat verschiedener Varianten verglichen werden kann (siehe auch das `--working-dir` -Befehlszeilenargument, beschrieben in Abschnitt Arbeits-und Ausgangsverzeichnis).

Nachfolgend sind einige typische Arbeitsabläufe/Verwendungsvarianten skizziert:

1.5.1. Ersteinrichtung eines Simulationsszenarios

Das ist eine recht direkte Vorgehensweise:

1. Importieren Sie alle FMUs und weisen Sie Slave-ID-Namen zu
2. (optional) Legen Sie Parametermeterwerte für die Slaves fest
3. (optional) Definieren Sie die grafische Darstellung der Slaves
4. Verbinden Sie die Ausgangs- und Eingangsgrößen
5. Legen Sie die Simulationsparameter fest
6. Führen Sie eine Simulation durch
7. Prüfen und Bewerten Sie die Ergebnisse

1.5.2. Variantenanalyse: Nur veröffentlichte FMU-Parameter sind modifiziert

Dies ist ein sehr einfacher Fall und, wenn von FMUs unterstützt, durchaus eine praktikable Methode. In *MasterSim* müssen nur die den veröffentlichten Parametern zugewiesenen Werte geändert werden (dies kann auch direkt in der Projekt-Datei getan werden, z. B. auch mit Skripten) und die Simulation kann wiederholt werden.

1.5.3. Variantenanalyse: FMUs ändern das interne Verhalten, aber nicht die Schnittstelle

Dies ist einer der häufigsten Fälle. Hier bleiben die Namen der Eingangs- und Ausgangsgrößen unverändert (d.h. die FMU-Schnittstelle bleibt unverändert). Auch die publizierten Parameter bleiben gleich. Jedoch ändert sich das interne Verhalten aufgrund der Anpassung des internen Modellverhaltens, wonach die FMU nochmals exportiert wurde. Da *MasterSim* selbst die FMU-Archive nur über einen Dateipfad referenziert, können FMU-Dateien in solchen Fällen einfach ersetzt und der Simulator ohne weitere Anpassungen gestartet werden.

1.5.4. Variantenanalyse: FMUs ändern Parameter aber nicht die Ein- und Ausgangsgrößen

In dieser Situation, in der ein Parameter in *MasterSim* konfiguriert worden ist, der nicht länger existiert (oder dessen Name geändert wurde), muss die entsprechende Definition in der Projekt-Datei geändert oder von der Benutzeroberfläche entfernt werden.

1.5.5. Variantenanalyse: FMUs ändern die Schnittstelle

Wenn eine importierte FMU einen Teil ihrer Schnittstelle ändert (z. B. Ein- oder Ausgangsgrößen wurden

modifiziert), dann wird dies in der Benutzeroberfläche durch Hervorhebung der falschen/nun fehlenden Verbindungen angezeigt. Wenn nur Variablennamen verändert wurden, editieren Sie am besten die Projekt-Datei und benennen dort die Größenbezeichnung um. Ansonsten sollte man einfach die Verbindung entfernen und eine neue erzeugen.

Wenn sich der Variablentyp einer Eingangs-/Ausgangsgröße ändert, sodass eine ungültige Verbindung entsteht (oder die Kausalität geändert wird), dann zeigt die Benutzeroberfläche die ungültige Verbindung nicht unbedingt direkt an. Allerdings wird das Befehlszeilenprogramm des *MasterSimulator* den Fehler während der Initialisierung anzeigen und abbrechen. Auch hier ist empfehlenswert, die fehlerhafte Verbindung zu löschen und neu zu erstellen.

1.6. Ein Überblick über den Simulations-Algorithmus

MasterSim hat folgende zentrale Bausteine:

- Initialisierung (Lesen der Projekt-Datei, Extraktion von FMUs, Überprüfung ...)
- Anfangsbedingungen
- Korrekturschleife während der Laufzeit
- Master-Algorithmus (d.h. er versucht Schritte durchzuführen)
- Fehleranalyse
- Ausgaben zu angeforderten Zeitpunkten schreiben

Diese Bausteine werden nachfolgend näher erläutert.

1.7. Initialisierung

Zu Beginn der aktuellen Simulation (das Befehlszeilenprogramm *MasterSimulator*, siehe Abschnitt Befehlszeilen-Argumente zu Details zum Simulationsstart) wird die Struktur des Arbeitsverzeichnisses erzeugt und das Schreiben der Log-Datei gestartet.

Danach wird die Projekt-Datei gelesen und alle referenzierten FMUs werden entpackt. Wenn Verweise auf CSV-Dateien auftauchen (siehe Abschnitt CSV-FileReader-Slaves), werden diese Dateien eingelesen und für die Berechnung ausgewertet/vorbereitet.

Das Entpacken der FMU-Archive kann mit der Befehlszeilen-Option `--skip-unzip` übersprungen werden (siehe Abschnitt Modifikation/Fixierung des FMU-Inhalts).

Als erster Schritt der aktuellen Co-Sim-Initialisierung werden alle FMU-Slaves erzeugt (dynamische Bibliotheken werden geladen und Symbole importiert, danach wird `fmiInstantiateSlave()` oder `fmi2Instantiate()` aufgerufen (entsprechend für FMI 1.0 bzw. FMI 2.0-Slaves). Es folgt eine Zusammenstellung aller Austauschvariablen und das Erstellen einer Variablenzuordnung.

Treten Fehler während der Initialisierung auf, führt dies zu einem Abbruch des Simulators mit einer entsprechenden Fehlermeldung.

1.7.1. Anfangsbedingungen

Die erste Aufgabe des Simulators ist es, für alle Slaves konsistente Anfangswerte zu erhalten. Das ist bereits eine nicht-triviale Aufgabe und nicht in allen Fällen überhaupt möglich. Die einzige Prozedur, die sowohl für FMI 1 und FMI 2-Slaves zum Einsatz kommen kann, ist das schrittweise Lesen und Setzen von Eingangs- und Ausgangsgrößen in allen Slaves. Dieses wird wiederholt, bis keine Änderungen mehr beobachtet werden.

Der Algorithmus in *MasterSim* ist:

```
Schleife über alle Slaves:
  - setupExperiment() für den aktuellen Slave aufrufen
  - setzen aller Variablen der Kausalitäten INPUT oder PARAMETER auf ihre Standardwerte, wie sie in der
modelDescription.xml gegeben sind
  - setzen aller Parameter auf die in der Projektdatei angegebenen Werte (falls Werte zugewiesen wurden)

nur für FMI 2: in allen Slaves enterInitializationMode() aufrufen

Schreife mit max. 3 Wiederholungen:
  Schleife über alle Slaves:
    alle Ausgangsvariablen des aktuellen Slave abfragen und in der globalen Variablenzuordnung speichern
  Schleife über alle Slaves:
    setzen aller Eingangsvariablen auf Werte der globalen Variablenzuordnung

nur für FMI 2: in allen Slaves exitInitializationMode() aufrufen
```

Der Berechnungsalgorithmus für die Anfangsbedingungen ist derzeit ein Gauss-Jacobi-Algorithmus und als solcher nicht übermäßig stabil oder effizient.

Wenn Sie mehr als 3 Slaves in einer Sequenz mit direktem Durchgang von Ein- zu Ausgangsvariablen verbunden haben, z. B. wenn die Ausgangsvariablen mit den Eingangsvariablen via algebraischer Verbindungen verknüpft sind, werden die 3 Wiederholungen des Gauss-Jacobi-Algorithmus eventuell nicht genügen, um alle Slaves korrekt zu initialisieren.

Durch eine Uneindeutigkeit im aktuellen FMI-Standard wird von Co-Simulations-Slaves nicht gefordert, die Ergebnisvariablen immer dann zu aktualisieren, wenn sich Eingangsvariablen ändern. Die meisten FMUs aktualisieren ihre Ausgangswerte tatsächlich erst nach der Aufforderung `doStep()`. Daher ist es mit dem gegenwärtigen Standard nicht möglich, zwischen den direkten mathematischen Beziehungen von Aus- und Eingängen zu unterscheiden. Dies heißt eine Änderung der Ergebnisvariablen **ohne Aufruf** von `doStep()` und **nach einem Aufruf** von `doStep()`.

MasterSim wählt hier die Funktionalität von FMI 1.0, d.h. keine Schleifen innerhalb einer Iteration nur um Ein- und Ausgänge zu synchronisieren. Dies erfolgt unter der Annahme, dass die Ausgangsgrößen sich nicht direkt ändern, sobald neue Eingangsvariablen gesetzt wurden (dies gilt für die meisten FMUs). Unter dieser Bedingung sind 3 Wiederholungen immer ausreichend.

1.7.2. Start- und Endzeit der Simulation

MasterSim behandelt die Simulationszeit in der Programmoberfläche als gegeben in *Sekunden*.

Wenn die gekoppelten FMUs eine unterschiedliche Zeiteinheit verwenden (d. h. Jahre), benutzen Sie einfach Sekunden auf der Benutzeroberfläche und der Projektdatei und interpretieren die Werte als Jahre.

Die Simulationszeit wird in der Benutzeroberfläche und der Projektdatei in Sekunden eingetragen (oder irgend einer anderen unterstützten Einheit, die in Sekunden umgewandelt werden kann). Während der Simulation werden alle erfassten Zeiten (Start- und Endzeit und die Zeitstufengrößen und Größenbegrenzung) zuerst in Sekunden umgewandelt und danach ohne irgend eine weitere Einheitenumrechnung benutzt.

Beispiel: Wenn Sie einen Endzeitpunkt auf `1 h` festlegen, wird der Master bis zur Simulationszeit `3600` (s) laufen, welche dann als *Endzeitpunkt des Kommunikationsintervals* im letzten `doStep()`-Aufruf gesendet wird.

Das gesamte Simulationszeit-Intervall wird an die Slaves im `setupExperiment()`-Aufruf weitergegeben. Wenn Sie die Startzeit anders als mit 0 festlegen, wird der Master-Simulator sein erstes Kommunikationsintervall zu diesem Zeitpunkt starten (der Slave braucht dies, um den `setupExperiment()`-Aufruf korrekt zu verarbeiten und den Slave zum Startzeitpunkt zu initialisieren).

Der korrekte Umgang mit der Startzeit ist wichtig für alle FMUs, die eine Form der Bilanzierung oder Integration durchführen.

Die Endzeit der Simulation wird zur FMU auch per `setupExperiment()`-Aufruf übermittelt (das Argument `stopTimeDefined` ist durch *MasterSim* immer auf `fmiTrue` gesetzt).

1.8. Anpassung der Kommunikationsschrittlängen

Sobald das Kommunikationsintervall abgeschlossen ist, geht der Simulator in die Zeitschrittsanpassungsschleife über. Wenn die Anpassung der Zeitschritte über die Eigenschaft **adjustStepSize** deaktiviert ist (siehe Abschnitt Einstellungen für die Simulation), wird der Schleifeninhalt nur einmal ausgeführt. Für FMI 1.0 Slaves oder FMI 2.0 Slaves ohne die Fähigkeit zur Speicherung/Wiederherstellung des Slave-Status ist eine Wiederholung eines Schritts ebenfalls nicht möglich (tatsächlich löst das Abfragen eines Wiederholungs-Algorithmus für solche Slaves einen Fehler während der Initialisierung aus).

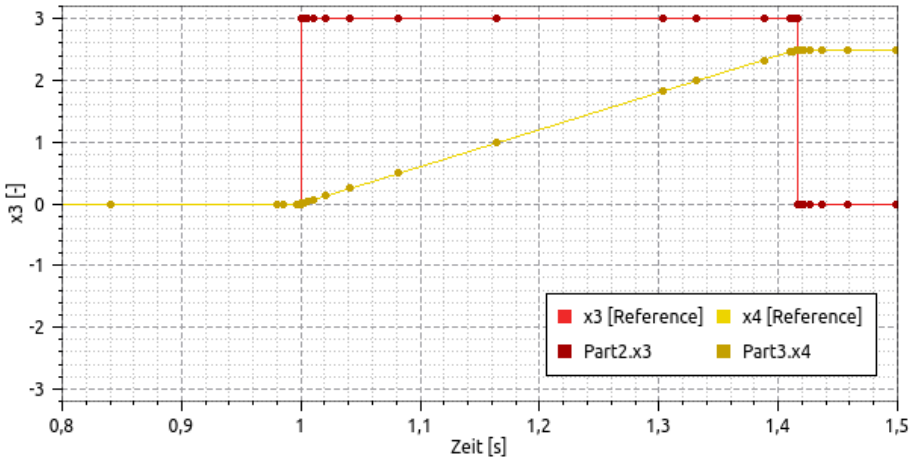

Abbildung 2. Simulationsbeispiel bei dem sowohl Fehlerschätzerüberschreitungen als auch Konvergenzfehler eine drastische Veränderung des Kommunikationszeitschritts bedingen

Innerhalb der Schleife versucht der ausgewählte *Master-Algorithmus* einen einzelnen Schritt mit der gegenwärtig vorgeschlagenen Zeitschrittgröße zu machen (für eine Methode mit konstanter Schrittweiter wird der **hStart**-Parameter genutzt). Dabei kann der *Master-Algorithmus* möglicherweise eine iterative Auswertung der Slaves beinhalten (siehe unten).

Für einen sich wiederholenden Master-Algorithmus ist es dabei möglich, dass die Methode nicht innerhalb der gegebenen Grenzen konvergiert (siehe Parameter **maxIterations** in Abschnitt Einstellungen für die Simulation).

1.8.1. Zeitschritt-Verringerung, wenn der Algorithmus nicht konvergiert

Wenn der Algorithmus nicht innerhalb des vorgegebenen Wiederholungslimits konvergiert, wird die Kommunikationsschrittlänge um den Faktor 5 reduziert:

```
h_new = h/5
```

Der Faktor 5 ist so ausgewählt, dass die Zeitschrittgröße schnell reduziert werden kann. Falls zum Beispiel eine Unstetigkeit auftritt, z. B. ausgelöst durch eine stufenweise Änderung diskreter Signale, muss der Simulator die Zeitschritte schnell auf einen niedrigen Wert reduzieren, um die Unstetigkeit zu passieren.

Die Schrittgröße wird dann mit der unteren Schrittlängengrenze verglichen (Parameter **hMin**). Dies ist notwendig, um zu verhindern, dass die Simulation in extrem langsamen Zeitschritten stecken bleibt. Falls der Fehlerkontrollalgorithmus die Schrittgröße unter den Wert von **hMin** reduziert würde, **wird die Simulation abgebrochen**.

In manchen Fällen kann die Interaktion zwischen zwei Slaves das Konvergieren jedweder Master-Algorithmen verhindern (sogar beim Newton-Algorithmus). Dennoch kann in diesen Fällen der verbleibende Fehler unerheblich sein und die Simulation kann in winzigen Schritten langsam über die problematische Zeit hinweggehen und danach die Schritte wieder vergrößern. In diesen Fällen können Sie den Parameter **hFallBackLimit** festlegen, welcher größer sein muss als **hMin**. Wird h auf einen Wert unter diese *zulässige* Kommunikationsschrittlänge reduziert, wird der Master-Algorithmus nach

Durchlaufen alle Iterationen den Schritt als erfolgreich akzeptieren. Der Schritt wird dadurch als *sich angenähert* behandelt und die Simulation geht zum nächsten Intervall weiter.

Die Publikation

Nicolai, A.: *Co-Simulations-Masteralgorithmen - Analyse und Details der Implementierung am Beispiel des Masterprogramms MASTERSIM*, http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa2-319735

illustriert das Verhalten der Simulation beim Benutzen dieser Parameter.

1.8.2. Fehlerkontrolle und Zeitschritt-Anpassung

Wenn eine Fehlertestmethode (**ErrorControlMode**) festgelegt ist, folgt nach einem konvergiertem Schritt eine lokale Fehlerschätzung. Derzeit basiert diese Fehlerprüfung auf der Schritt-Verdopplungs-Technik und kann als solche nur eingesetzt werden, wenn die Slaves die FMI 2.0 Funktionalität zum Speichern/-Rücksetzen des Zustands unterstützen.

Grundsätzlich läuft der Test folgendermaßen ab:

```
- der Slave-Zustand wird zurückgesetzt, um das aktuelle Kommunikationsintervall zu starten
- es werden zwei Schritte (jeweils mit einem kompletten Master-Algorithmus pro Schritt) nacheinander
durchgeführt
- die Fehlerkriterien 1 und 2 werden ausgewertet
- der Zustand der Slaves wird auf den Zustand nach dem ursprünglichen Master-Algorithmus zurückgesetzt
```

Der Fehlertest benötigt demnach zwei weitere Durchgänge des *Master-Algorithmus* pro Kommunikationsschritt. Für wiederholende Master-Algorithmen oder den Newton-Algorithmus kann dadurch der zusätzliche Aufwand für den Fehlertest erheblich sein.

Die mathematischen Formeln und detaillierte Beschreibungen der Fehlertests sind in der folgenden Publikation dokumentiert:

Nicolai, A.: *Co-Simulation-Test Case: Predator-Prey (Lotka-Volterra) System* (siehe MasterSim Dokumentations-Webpage).

Die Fehlersuche nutzt die Parameter `relTol` und `absTol` um die akzeptable Differenz zwischen Voll- und Halbschritt einzugrenzen (oder deren Anstiege). Abhängig von der lokalen Fehlerschätzung existieren zwei Optionen:

- die lokale Fehlerschätzung ist klein genug und der Zeitschritt wird vergrößert, oder
- die Fehlersuche scheitert; die Schrittgröße wird entfernt und der gesamte Kommunikationsschritt wird wiederholt.

Wenn Sie einen Fehlersuche-Algorithmus in *MasterSim* benutzen, sollten Sie ein Rückfall-Zeitschrittlimit setzen (**hFallBackLimit**). Andernfalls könnte *MasterSim* versuchen, die eventuell große Dynamik der Veränderungen von Variablen von Schritt zu Schritt dadurch zu verfolgen, dass die Zeitschritte auf extrem niedrige Werte

reduziert werden (und damit die Simulation sehr langsam wird).

1.9. Master-Algorithmen

Ein *Master-Algorithmus* bezeichnet grundsätzlich die mathematische Prozedur, um die gekoppelte Simulation einen Schritt voran zu bringen. Solch ein Co-Simulations-Master-Algorithmus verfügt über einen charakteristischen Satz an Regeln, wie Werte von einer FMU abgerufen werden, wann und wie diese Werte an andere FMUs übergeben werden und die Kriterien für das Konvergieren von Iterationsverfahren.

MasterSim enthält mehrere Standard-Algorithmen. Eine detaillierte Diskussion der unterschiedlichen Algorithmen und wie die Wahl von Algorithmen und Parametern Ergebnisse beeinflusst, kann in der folgenden Publikation nachgelesen werden:

Nicolai, A.: *Co-Simulations-Masteralgorithmen - Analyse und Details der Implementierung am Beispiel des Masterprogramms MASTERSIM*, http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa2-319735

1.9.1. Gauss-Jacobi

Basis-Algorithmus:

```
Schleife über alle Slaves:
  Holen aller Ausgangswerte

Schleife über alle Slaves:
  setzen aller Eingangswerte
  den Slave einen Schritt durchführen lassen (`doStep()`)
```

Gauss-Jacobi wird stets ohne Iteration ausgeführt. Wie in der Publikation gezeigt (siehe oben), ergibt es wirklich keinen Sinn, eine Iteration zu nutzen.

Anstatt einen Schritt zur Datenübertragung für 10 Sekunden zu nutzen und Gauss-Jacobi mit 2 Iterationen zu nutzen, ist es effizienter die Iterationen zu deaktivieren (festlegen von **maxIterations=1**) und die Größe der Datenübertragungsschritte auf 5 Sekunden zu begrenzen. Der Aufwand für die Simultion ist exakt der Gleiche (2 FMU-Auswertungen je 10 Sekunden Laufzeit), jedoch läuft die Simulation mit dem 5-sekündigem Kommunikationsintervall genauer ab (und stabiler).

1.9.2. Gauss-Seidel

Basis-Algorithmus:

```
Iterationsschleife:
  Schleife über alle Slaves:
    setzen aller Eingangswerte aus globaler Variablenzuordnungsliste
    den Slave einen Schritt durchführen lassen (`doStep()`)
    Ergebnisgrößen vom Slave abrufen
```

```
    globale Variablenzuordnungsliste aktualisieren
  Konvergenztest durchführen
```

Durch die Aktualisierung der Variablenliste nach jedem Slave erhalten die nachfolgenden Slaves bereits aktualisierte Größen für den Kommunikationsschritt, welches das Gauss-Seidel-Verfahren auszeichnet.

Zyklen

MasterSim enthält eine Funktion zur Reduktion des Rechenaufwands, wenn viele FMUs involviert sind und nicht alle direkt miteinander verbunden sind. Abb. Abbildung 3 zeigt ein Simulationsszenario, in dem die Berechnung in drei Stufen ausgeführt werden kann.

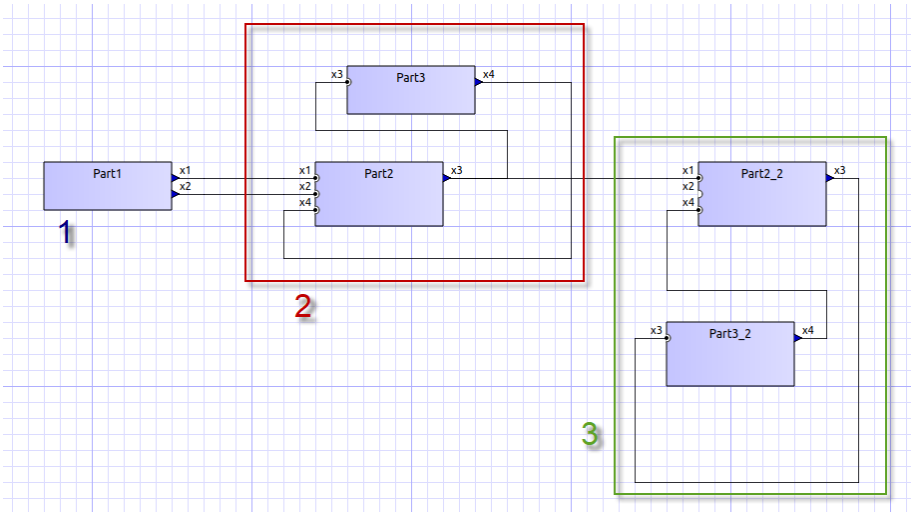

Abbildung 3. Zyklen in sich wiederholenden Algorithmen

(1) Diese FMU erzeugt nur Ergebnisse und wird als Erstes und nur ein einziges Mal im Gauss-Seidel-Algorithmus ausgewertet werden

(2) Diese zwei FMUs tauschen Werte aus, sie sind in einem *Zyklus* verbunden. Wenn der Gauss-Seidel-Algorithmus mit aktivierter Iteration ausgeführt wird, brauchen nur diese beiden FMUs aktualisiert werden und Werte austauschen, denn sie erfordern keine weiteren Ergebnisgrößen der anderen FMUs (abgesehen von der ersten FMU, deren Ausgangsvariablen bereits bekannt sind)

(3) Die letzten beiden FMUs sind auch in einem Zyklus gekoppelt, aber wiederum nur miteinander. Sie werden in der letzten Phase ausgewertet. Da die Ergebnisse der anderen drei FMUs bereits berechnet wurden und bekannt sind, müssen wieder nur zwei FMUs im Zyklus ausiteriert werden.

Die Anzahl an FMUs in einem Zyklus zu begrenzen reduziert nicht nur den gesamten Aufwand, sondern berücksichtigt auch die Steifigkeit der Kopplung. In einem Zyklus könnten die FMUs nur lose miteinander verbunden sein und die Konvergenz ist mit 2 oder 3 Wiederholungen erreicht. In anderen Zyklen können die FMUs in einer nichtlinearen Beziehung gekoppelt sein oder sensibler auf Änderungen der Eingangswerte reagieren (= starre Kopplung) wodurch zehn oder mehr Wiederholungen benötigt würden. Das Trennen der Zyklen kann daher *den Rechenaufwand* bei der Gauss-Seidel signifikant *verkleinern*.

Jede FMU kann einem Zyklus zugewiesen werden. Die Zyklen sind durchnummeriert (beginnend bei 0) und werden in der Reihenfolge der Zyklusnummer berechnet (siehe Simulatordefinition im Abschnitt Simulator-/Slave-Definitionen).

1.9.3. Newton-Verfahren

Basis-Algorithmus:

```
Iterationsschleife:
  In der ersten Iteration berechne Newton/Jacobi-Matrix mittels Differenzquotienten

  Schleife über alle Slaves:
    setzen aller Eingangswerte aus globaler Variablenzuordnungsliste
    den Slave einen Schritt durchführen lassen (doStep())

  Schleife über alle Slaves:
    Ergebnisgrößen vom Slave abrufen

  Newton-Gleichungssystem lösen
  Berechnen der Korrekturgrößen für die Variablen und Aktualisieren der
  Variablen

  Konvergenz-Test durchführen
```

Zyklen werden genauso behandelt wie beim Gauss-Seidel-Algorithmus.

Für den Fall, dass nur eine einzige FMU innerhalb des Zyklus ist, wird der Newton-Master-Algorithmus diese FMU nur einmal auswerten und die Ergebnisse als bereits konvergiert behandeln. Natürlich wird in diesem Fall keine Newton-Matrix benötigt und erstellt. Allerdings wird dies in dem (seltenen) Fall, dass eine solche FMU seine Eingangswerte mit *seinen eigenen Ausgängen* verbindet, zu Problemen führen, da potentiell ungültige/unpassende FMU-Bedingungen vom Newton-Schritt akzeptiert werden.

1.10. Schreiben von Ergebnisgrößen/Ausgangsvariablen

Ergebnisgrößen werden nach jedem vollendeten Schritt geschrieben, aber nur, wenn die Zeitspanne seit dem letzten Schreiben mindestens so lang ist wie im Parameter **hOutputMin** festgelegt.

Wenn Sie Ausgänge wirklich nach jedem einzelnen Kommunikationsinterval/Berechnungsschritt haben wollen, setzen Sie **hOutputMin** auf 0.

2. Grafische Benutzeroberfläche

MasterSim verfügt über eine komfortable grafische Benutzeroberfläche, um *Simulations-Szenarios* festzulegen und anzupassen. Mit *Simulations-Szenario* meine ich die Festlegung, welche FMUs zu

importieren sind, welche Slave (oder Slaves) erstellt werden sollen, wie Eingangs-und Ausgangsvariablen verbunden sind und die Definition aller Eigenschaften im Zusammenhang mit den Berechnungsalgorithmen. Grundsätzlich behaltet ein *Szenario* alles, das gebraucht wird, um eine Co-Simulation durchzuführen.

2.1. Startseite

Die Software startet mit einer Startseite, welche eine Liste kürzlich verwendeter Projekte und einiger webbasierter Neuigkeiten enthält (diese werden von der Datei https://bauklimatik-dresden.de/downloads/mastersim/news.html bezogen, welche aktualisiert wird, sobald eine neue Veröffentlichung oder Funktion zur Verfügung steht).

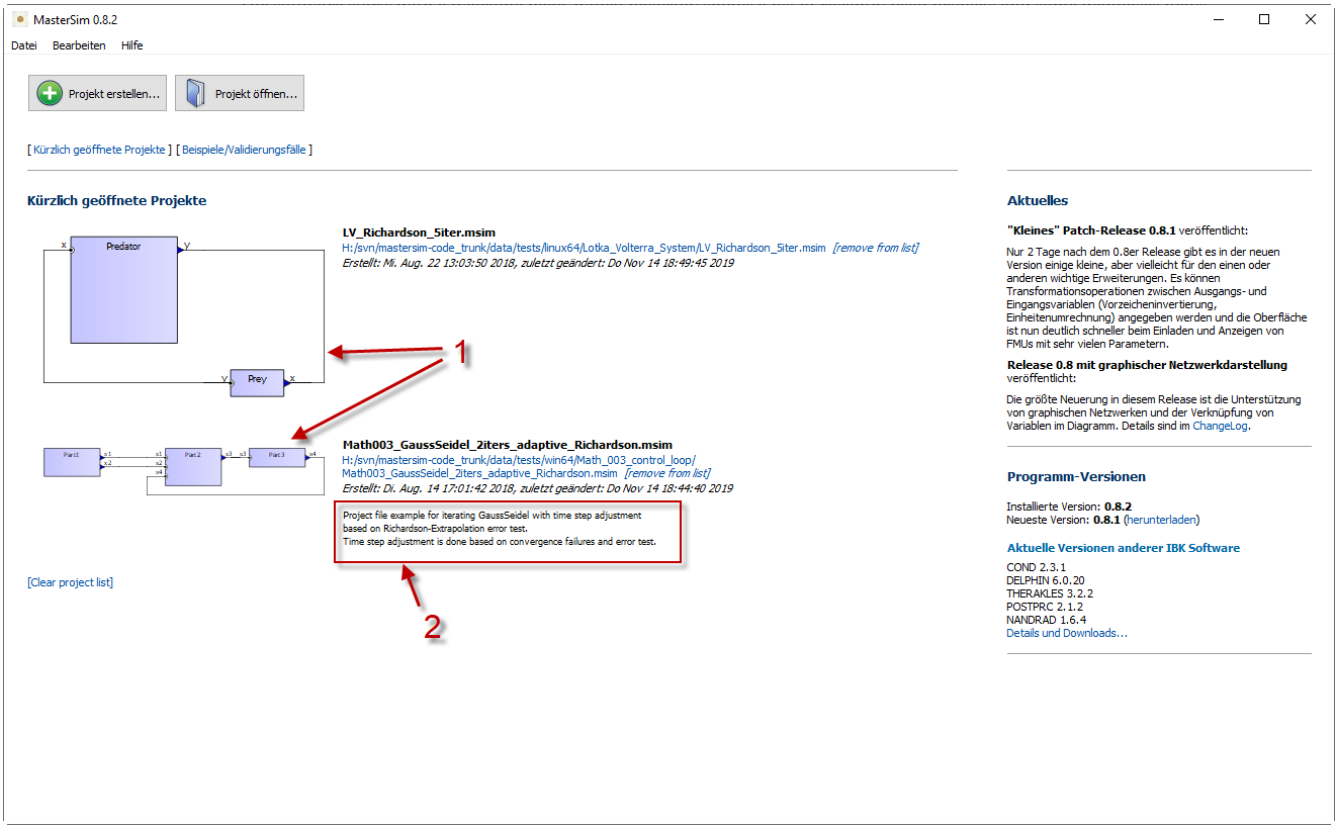

Abbildung 4. Startseite mit kürzlich verwendeten Projekten und webbasierten Neuigkeiten

(1) Miniaturansicht mit der Vorschau eines Simulations-Szenarios

(2) Kurze Beschreibung des Projekts. Die Beschreibung entstammt der Kommentarzeile der Projektzeilenüberschriften (siehe Abschnitt Format der Projekt-Datei).

Die auf der Startseite angezeigten Miniaturansichten werden immer dann erzeugt/aktualisiert, wenn das Projekt gespeichert wird. Die Dateien werden innerhalb des Programm-Benutzerverzeichnisses platziert:

- bei Windows in `%APPDATA\Roaming\MasterSim\thumbs` und
- bei Linux/MacOS in `~/.local/share/MasterSim`

> und die Bilddatei ist genauso benannt wie die Projektdatei mit angehängter Erweiterung `png`.

2.1.1. Beispiele

Beim Öffnen eines Beispiels von der Startseite/Beispielseite werden Sie aufgefordert, das Projekt zunächst in einer benutzerdefinierten Stelle zu sichern (Beispiele sind im Installationsverzeichnis abgelegt, welches gewöhnlich schreibgeschützt ist).

2.2. Symbolleiste und nützliche Tastenkombinationen

Sobald ein Projekt erstellt/geöffnet ist, wird eine der Projektinhaltsansichten gezeigt und eine Symbolleiste an der linken Seite des Programms angezeigt. Die Symbole der Symbolleiste haben die folgenden Funktionen (sie werden auch durch einen Tooltip angezeigt, wenn man mit dem Mauszeiger über das Symbol fährt):

Programminformation	Zeigt Programminformationsfenster
Neues Projekt	Erzeugt ein neues Projekt (Kürzel **Strg** + **N**)
Projekt öffnen	Öffnet eine `*.msim`-Projektdatei (Kürzel **Strg** + **O**)
Projekt speichern	Sichert das aktuelle Projekt (Kürzel **Strg** + **S**) (sichert außerdem die graphische Netzwerk-Darstellung)
Öffnet Post-Prozessing	Öffnet das Postprozessing-Werkzeug, angegeben im Einstellungsdialog. Ich würde grundsätzlich empfehlen, PostProc 2 zu nutzen. Sie können hier jede andere Software starten oder sogar ein automatisches Analyseskript. Setzen Sie einfach die entsprechende Kommandozeile im Einstellungsdialog.
FMU Analyse	*MasterSim* wird alle referenzierten FMUs entpacken und deren Modellbeschreibungs-Dateien lesen. Es aktualisiert außerdem die grafischen Schemata und Verbindungsansichten, wenn die FMU-Oberflächen sich geändert haben. Ebenso wird die Eigenschaftentabelle auf den neuesten Stand gebracht. Nutzen Sie diese Funktion, wenn Sie eine FMU im Dateiensystem aktualisiert haben und diese Änderungen in der *MasterSim*-Benutzeroberfläche anwenden/ansehen wollen (alternativ laden sie das Projekt einfach neu).
Slave-Definitionsansicht	Wechselt zur Ansicht zum Definieren der FMU-Slaves. Hier legen Sie fest, welche FMUs verwendet werden und setzen Parameter der FMU-Slaves. Außerdem können Sie eine grafische Darstellung des Netzwerkes erstellen.

Verknüpfungsansicht	Wechselt zur Verknüpfungsansicht. Hier können Sie die Verbindungen zwischen den Slave Ein- und Ausgangsvariablen verwalten und spezielle Attribute (Transformationen) zwischen den Verbindungen zuweisen.
Simulationseinstellungen	Wechselt zur Simulationsansicht. Alle Simulationsparameter und numerischen Algorithmusoptionen sind hier spezialisiert. Die Start der eigentlichen Simulation erfolgt ebenso von dieser Ansicht.
Undo/Redo	Die nächsten zwei Buttons steuern die Funktionen *Rückgängig* und *Wiederholen* der Benutzeroberfläche. Alle im Projekt gemachten Änderungen können zurückgenommen und noch einmal gemacht (Kürzel sind **Strg** + **Z** für *Rückgängig machen* und **Strg** + **Shift** + **Z** für *Wiederholen*).
Sprache umschalten	Diese Schaltfläche öffnet ein Kontextmenü mit einer Sprachauswahl. Sie müssen die Anwendung neu starten, um die neue Sprache zu aktivieren.
Beenden	Schließt die Software. Wenn das Projekt verändert worden ist, wird der Benutzer gefragt, ob er die Änderungen speichern oder verwerfen möchte.

2.2.1. Nützliche Tastenkürzel

Hier ist eine Liste der nützlichen programmweiten Tastaturkürzel:

Tabelle 1. Programmweite Tastenkombinationen

Windows/Linux	MacOS	Command
Strg + **N**	**CMD** + **N**	erstellt ein neues Projekt
Strg + **O**	**CMD** + **O**	lädt ein Projekt
Strg + **S**	**CMD** + **S**	speichert ein Projekt
Strg + **Shift** + **S**	**CMD** + **Shift** + **S**	speichert das Projekt mit einem neuen Dateinamen
Strg + **Z**	**CMD** + **Z**	rückgängig machen
Strg + **Shift** + **Z**	**CMD** + **Shift** + **Z**	erneut versuchen
F2	**F2**	öffnet die Projektdatei im Texteditor
F8	**F8**	öffnet das Verzeichnis der Projektdatei in der Dateiverwaltung
F9	**F9**	startet die Simulation (kann von jeder Ansicht genutzt werden, es ist nicht notwendig, zunächst zur Ansicht der Simulationsumgebung zu wechseln!)

Windows/Linux	MacOS	Command
	CMD + .	öffnet den Einstellungs-Dialog

2.3. Ansicht zum Definieren der FMU-Slaves

Die Eingabe eines Simulations-Szenarios ist in drei Ansichten aufgeteilt. Das Erstellen einer Simulation startet mit dem Importieren von FMUs/Slaves. Somit ist die Ansicht zum Festlegen der Slaves die Erste (und Wichtigste).

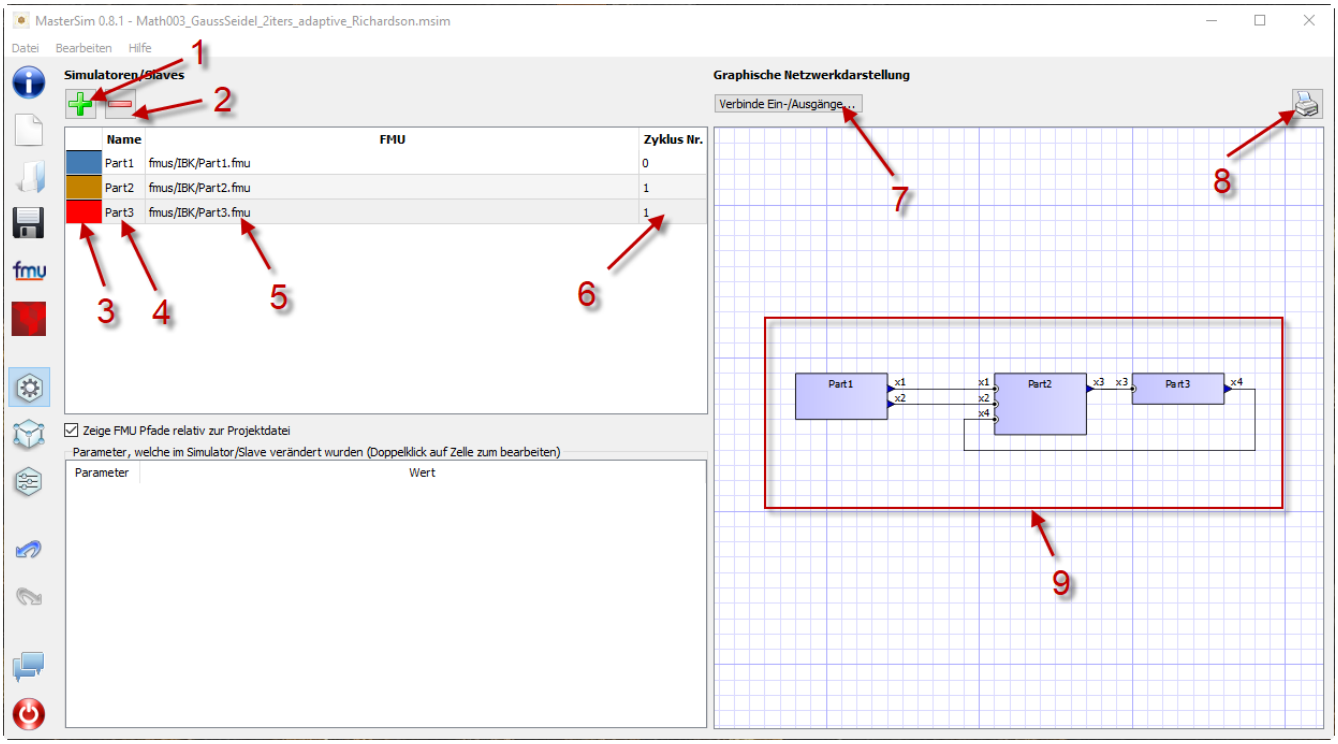

Abbildung 5. Die Slave-Ansicht zeigt eine Liste importierter FMUs, zugewiesene Slave-ID-Namen und eine optionale graphische Darstellung.

Elemente der Ansicht:

(1) Fügt einen neuen Slave durch das Auswählen einer FMU-Datei (`*.fmu`) oder eines Datei-lese-Slaves zu (`csv` or `tsv` file, siehe Abschnitt CSV FileReader Slaves)

(2) Entfernt den gegenwärtig ausgewählten Slave (und alle zu ihm gemachten Verbindungen)

(3) Durch Doppelklicken wird die Farbe des Slaves geändert (die Farbe wird genutzt, um den Slave in der Verbindungsansicht identifizieren zu können)

(4) Der ID-Name des Slaves. Standardmäßig weist *MasterSim* den Basisdateinamen der FMU-Datei zu. Durch Doppelklicken dieser Zelle kann dies geändert werden. Beachten Sie: Slave-ID-namen müssen innerhalb des Simulations-Szenarios einzigartig sein.

(5) Pfad zu einer FMU-Datei, entweder der absolute Pfad oder relativ zur aktuellen *MasterSim* -Projekt-Datei, abhängig vom Kontrollkästchen *"Zeige FMU Pfade relativ zur Projektdatei"*. Außerdem muss die Projektdatei gespeichert worden sein, bevor relative Pfade angezeigt werden können.

(6) Definiert, in welchem Zyklus der FMU-Slave berechnet werden soll (standardmäßig sind alle Slaves im Zyklus 0 verknüpft und damit werden alle gekoppelt berechnet. Siehe Zyklen-Beschreibung im Abschnitt Master-Algorithmus).

(7) Aktivieren des grafischen Verbindungsmodus (siehe Diskussion unten). Wenn dieser Modus aktiv ist, können Sie eine neue Verbindung von einem Ausgang zu einem Eingang im Netzwerk ziehen.

(8) Druckt Netzwerkschemata oder exportiert eine PDF-Datei.

(9) Dies ist ein grafisches Netzwerkschema - rein optional, aber es hilft, das Co-Simulations-Szenario zu verstehen.

2.3.1. Bearbeiten der Eigenschaften von Projekt, ausgewähltem Slave oder Verbindung

Im unteren linken Teil der Ansicht befindet sich eine Kontext-abhängige Eingabe für Projektkommentare (falls nichts gewählt ist), Slave-Eigenschaften (falls ein Slave markiert ist) oder Verbindungseigenschaften (falls eine Verbindung markiert ist).

2.3.2. Slaves hinzufügen

Neue Slaves werden durch das Auswählen von `fmu`- oder `csv`- oder `tsv`-Dateien zugefügt. *MasterSim* nutzt automatisch den Basisnamen der ausgewählten Datei als ID-Namen für den Slave. Falls bereits ein solcher ID-Name existiert, fügt *MasterSim* eine Nummer zum Basisnamen hinzu. In jedem Fall müssen die Slave-ID-Namen einzigartig innerhalb des Projekts sein.

Sie können die gleiche FMU mehrere Male importieren. In diesem Fall werden die Slaves unterschiedliche ID-Namen haben, referenzieren aber trotzdem die gleiche FMU-Datei. Parameter und das visuelle Auftreten können für einen Slave der selben FMU unterschiedlich gesetzt sein. Es ist zu beachten, dass eine FMU die Fähigkeit **canBeInstantiatedOnlyOncePerProcess** auf *false* gesetzt haben muss, wenn man sie mehrfach in einem Simulationsszenario verwenden möchte.

2.3.3. Eigenschaften/Parameterwerte der Slaves

Unterhalb der Tabelle mit den verwendeten Slaves ist eine Liste der von den FMUs publizierten Parameter. Die Liste gilt für den *gegenwärtig ausgewählten* Slave. Ein Simulations-Slave kann in der Slave-Tabelle oder durch Anklicken eines Blocks in der Netzwerkansicht ausgewählt werden.

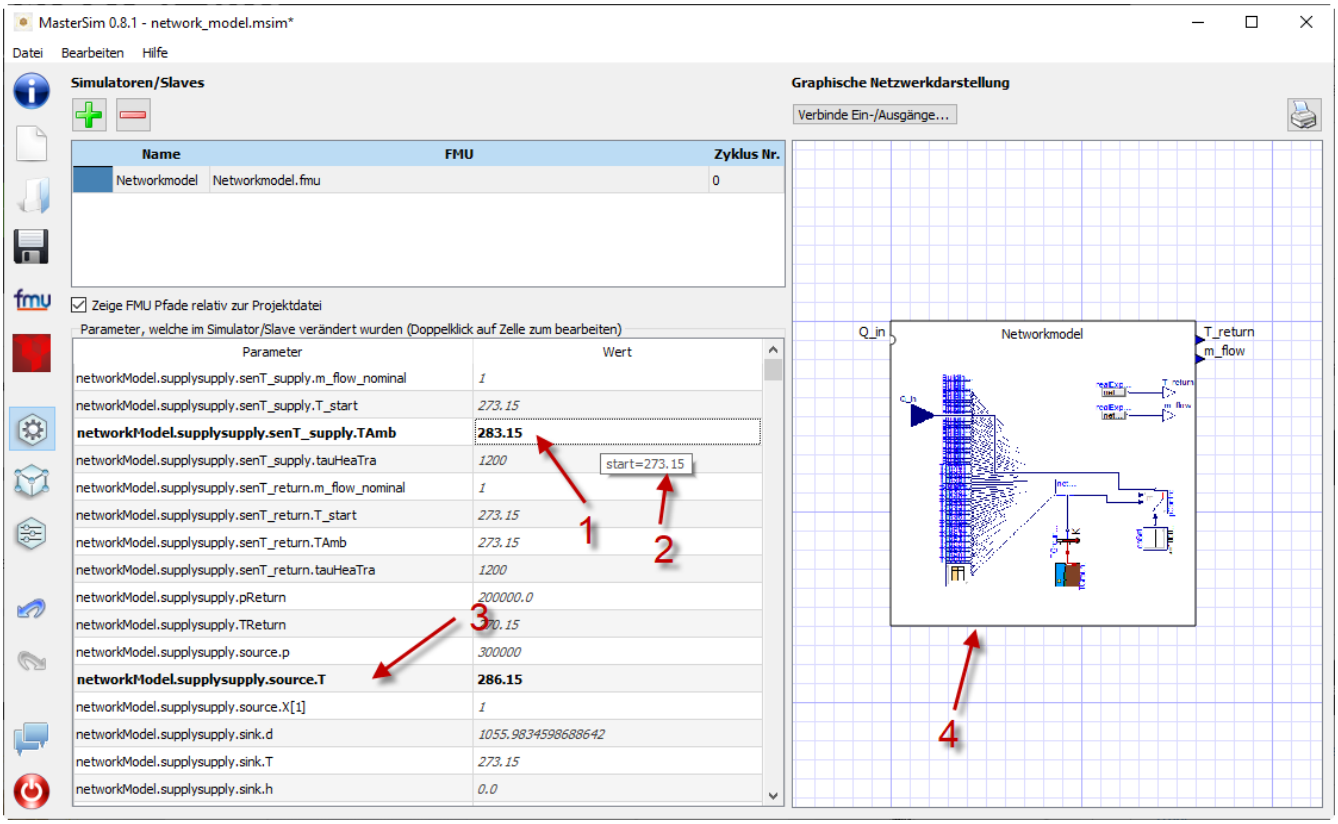

Abbildung 6. Tabelle mit Slave-spezifischen Parameterwerten

(1) Schwarze und fette Schrift gibt an, dass dieser Parameter modifiziert oder auf einen bestimmten Wert gesetzt worden ist. Grauer, kursiver Text zeigt einen standardmäßigen, unveränderten Wert.

(2) Fährt man mit der Maus über einen Parameterwert, zeigt sich ein Tooltip mit den Standardparametern. Dies kann genutzt werden, um den Standardwert zu sehen, falls der Parameter bereits verändert wurde.

(3) Parameter, die in schwarzer Fettschrift geschrieben sind, wurden von *MasterSim* gesetzt (während der Initialisierung).

Parameter können durch **Doppelklicken** der Wertezelle editiert werden. Das Löschen des Inhalts der Zelle setzt die Parameter auf ihre standardmäßigen Werte zurück.

2.3.4. Netzwerkansicht

Die Netzwerkansicht **(9)** zeigt ein simples Schema aller FMU-Slaves und ihrer Verbindungen. Diese Netzwerkansicht ist optional und wird für die Simulation nicht wirklich gebraucht. Dennoch ist die visuelle Darstellung des Simulations-Szenarios wichtig für die Kommunikation und die Prüfung der Eingabe.

Mit dem Scrollrad der Maus können Sie in der Netzwerkansicht heraus- und hineinzoomen. Es wird zu der Stelle gezoomt, an der sich der Mauszeiger befindet.

Das Netzwerk zeigt ***Blöcke*** (je importiertem Simulator/Slave), und in jedem Block eine oder mehrere

Anschlussstellen/Sockel (engl. *socket*). Diese ***Sockel*** zeigen die Eingangs-/Ausgangsvariablen eines jeden Simuations-Slaves an. Die Blöcke werden in unterschiedlichen Farben angezeigt, welche individuelle Blockzustände anzeigen.

Blöcke können markiert und dann mit der Maus verschoben werden. Ebenso können Verbindungslinien verschoben werden.

Falls Sie mehrere Blöcke gleichzeitig verschieben wollen, können sie mehrere Blöcke mittels **Strg** + **Click** auswählen. Wenn Sie nun einen der Blöcke verschieben, werden sich die anderen ausgewählten Blöcke ebenso bewegen.

Verbindungen in der Netzwerkansicht herstellen

Sie können eine neue Verbindung zwischen Ein- und Ausgängen von Slaves herstellen, indem Sie einfach von einem Variablenausgang (Dreieck) mit der Maus eine Verbindungslinie ziehen. Solange die Maustaste gedrückt bleibt, ist die Ansicht im *Verknüpfungsmodus*. Wenn der Verbindungsmodus aktiv ist, wandelt sich der Zeiger innerhalb des Netzwerkansicht-Fensters zu einem Kreuz. Sie können die Maus dann über einen *freien* Variableneingang (leerer Halbkreis) ziehen. Wenn die Verbindung hergestellt wurde, wird eine Verbindungslinie dauerhaft gezeigt und der Variableneingang leicht ausgefüllt.

Verbindungen zwischen Slaves können in der Verknüpfungsansicht bequemer festgelegt werden (welche ebenso effizienter ist, wenn mehr Verbindungen hergestellt werden, vergleichbar zum manuellen Ziehen der Verbindung mit der Maus).

Block-Zustandsanzeige

Da *MasterSim* die verwendeten FMUs selbst nur referenziert, erhält die Programmoberfläche nur dann Kenntnis vom eigentlichen Inhalt (z. B. Anschlusseigenschaften aus der `modelDescription.xml` Datei), wenn die FMU analysisert wird. Der FMU-Analyseschritt wird automatisch vorgenommen, wenn ein Projekt geöffnet ist und ein neuer FMU-Slave hinzugefügt wird.

Beim Analysieren einer FMU versucht die Benutzeroberfläche, das FMU-Archiv zu entpacken und dessen Inhalt zu analysieren. Wenn die `modelDescription.xml`-Datei korrekt gelesen werden konnte, bietet *MasterSim* an, den Block-Bearbeitungsdialog zu öffnen. Innerhalb dieses Dialogs können Sie die grundlegende Geometrie des Blocks (SLave-Darstellung) und die Gestaltung der Anschlüsse (die Position der Eingangs-/Ausgangsvariablen) festlegen. Sie können diesen Schritt aber auch überspringen. Grundsätzlich kann eine Block eines FMU-Slaves drei Zustände haben, die in der UI unterschiedlich dargestellt sind:

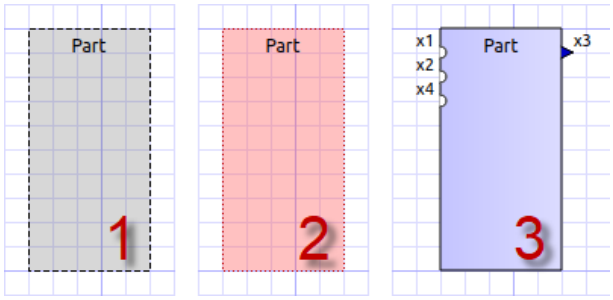

Abbildung 7. Unterschiedliche Zustände von Blöcken und deren visuelle Darstellung

(1) Die entsprechende `fmu`-Datei existiert nicht oder kann nicht gelesen werden (kein Archiv, kann nicht extrahiert werden, beinhaltet keine `modelDescription.xml`-Datei, die XML-Datei ist ungültig/fehlerhaft, oder … viele Dinge können hier schief gehen)

(2) Die Modellbeschreibung wurde für diese FMU erfolgreich analysiert, aber die Blockdefinition stimmt nicht mit dem der Modellbeschreibung überein, oder es wurde noch gar keine Blockdefinition erstellt. Typischerweise, wenn eine FMU zum ersten Mal importiert wird, gibt es noch keine Definition der graphischen Darstellung dieser FMU. Es wird dann einfache eine rote Box angezeigt. Sie können diese Box **doppelt anklicken**, um den Block-Bearbeitungsdialog zu öffnen.

(3) Das Block-Erscheinungsbild ist bereits definiert worden und die Anschlüsse passen zu der Modellbeschreibung (Name und Eingangs-/Ausgangstypen stimmen überein).

2.3.5. Block-Bearbeitungsdialog

Der Block-Bearbeitungsdialog erlaubt Ihnen, die grundlegende, rechteckige Gestalt der FMU festzulegen und die Abschlüsse zu auszurichten. Der Block-Bearbeitungsdialog wird entweder direkt nach dem Import einer FMU geöffnet oder indem Sie auf einen Block in der Netzwerkansicht **doppelklicken**.

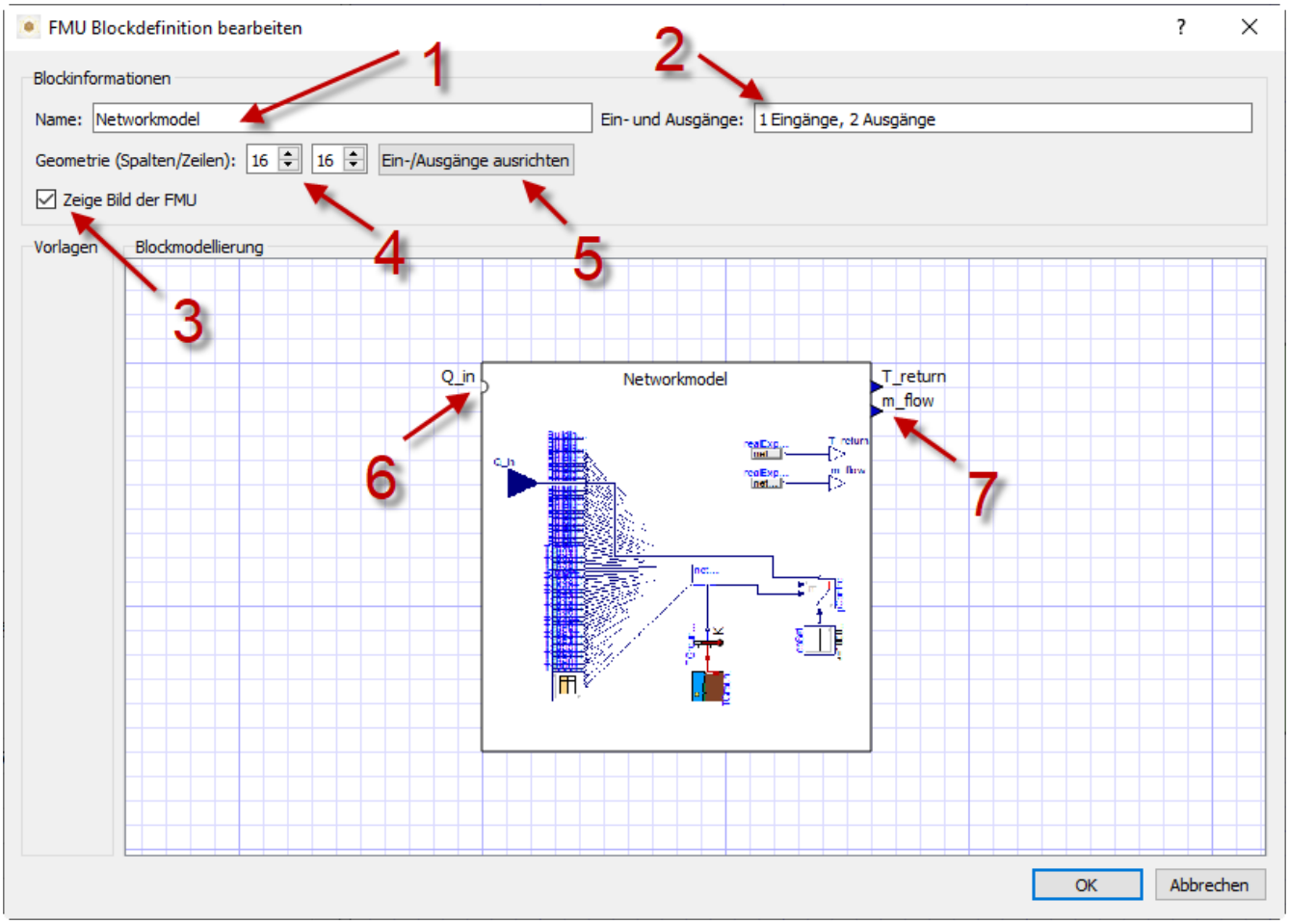

Abbildung 8. Bearbeitungsdialog für die Blockgeometrie und die Gestaltung des Sockels

(1) Slave-ID-Name

(2) Zeigt die Anzahl der veröffentlichten Eingangs- und Ausgangsvariablen

(3) Wenn angeschaltet, wird das FMU-Archiv nach der Image-Datei `model.png` durchsucht (die sollte neben der `modelDescription.xml`-Datei im Hauptverzeichnis des FMU-Archivs liegen). Wenn vorhanden, wird das Bild skaliert entsprechend der Blockgröße angezeigt.

(4) Hier können Sie die Weite und Höhe des Blocks in Rasterlinien festlegen.

(5) Dieser Knopf richtet die Anschlüsse aus. Eingänge sind an der linken, oberen Seite ausgerichtet, Ausgänge an der rechten, unteren Seite. Falls es nicht genügend Platz für alle Anschlüsse gibt, werden die verbleibenden Anschlüsse übereinander platziert.

(6) Markierung eines Variableneingangs (Eingangsvariable)

(7) Markierung eines Variablenausgangs (Ausgangsvariable)

In einer der nächsten Programmversionen wird es möglich sein, das Erscheinungsbild eines Blocks als Vorlage für die Nutzung bei ähnlichen oder gleichen FMUs zu speichern. Gegenwärtig müssen Sie den Block jedes Mal konfigurieren, wenn Sie eine FMU importieren. Ebenso ist die verbesserte Anwendung und der benutzerdefinierte

Sockel-Speicherort noch nicht umgesetzt.

Wenn dieses Feature benötigt wird, bitte ein Ticket anlegen.

2.4. Verknüpfungsansicht

In dieser Ansicht können Sie Slaves verknüpfen, indem Sie Ausgangs- und Eingangsvariablen verbinden.

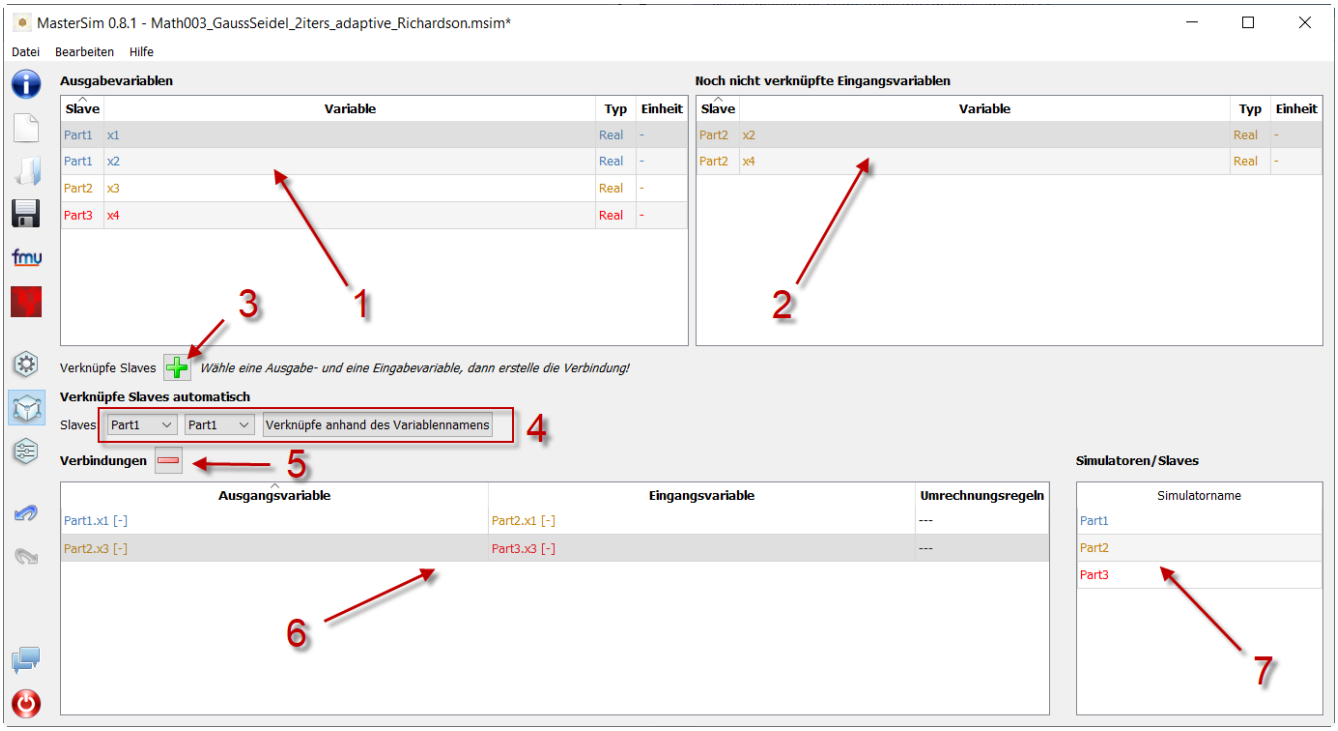

Abbildung 9. Verknüpfungsansicht mit Anzeige der Eingangs- und Ausgangsvariablen für alle Slaves und der bereits festgelegten Verbindungen

(1) Zeigt alle veröffentlichten Ausgangs- und Eingangsvariablen aller Slaves.

(2) Zeigt die Eingangsvariablen aller Slaves, welche noch **nicht** verbunden worden sind.

(3) Wählen Sie zunächst eine Ausgangsvariable und eine Eingangsvariable aus, welche verbunden werden sollen und drücken dann diesen Knopf, um eine Verbindung herzustellen.

(4) Hier können Sie gleich mehrere Verknüpfungen zwischen zwei Slaves erstellen (siehe Erklärung unten)

(5) Dies entfernt die aktuell ausgewählte Verbindung in der Tabelle (6)

(6) Zeigt alle bisher erzeugten Verknüpfungen. Durch einen **Doppelklick** auf die letzte Spalte kann eine Umrechenoperation (z.B. für Einheitenumrechnung oder Vorzeichenwechsel) definiert werden.

(7) Eine Tabelle mit allen Slaves und ihren zugeordneten Farben (erleichtert die Identifikation der Variablen nach Slave-Farbe)

2.4.1. Die Besonderheiten automatischer Verbindungen

Diese Funktion ist sehr hilfreich, wenn FMUs miteinander verbunden werden sollen, deren Ausgangs- und Eingangsvariablen den gleichen Namen haben. Dies ist insbesondere hilfreich, wenn Sie viele Eingangs- und Ausgangsvariablen zwischen zwei Slaves verbinden müssen. Falls Sie FMUs mit passender Namensgebung der Variablen erzeugen, können Sie den folgenden Ablauf nutzen:

1. wählen sie in den Auswahllistenboxen die zu verbindenden Slaves aus, und
2. drücken Sie den Verknüpfungsknopf.

Eine Verbindung wird erstellt, wenn:

- der Variablenname übereinstimmt
- der Datentyp der Variable passt
- eine Variable eine Eingangsvariable ist (Causality = INPUT) und die andere eine Ausgangsvariable ist (Causality = OUTPUT)

Beispiel 1 verdeutlicht eine solche automatische Verknüpfung.

Beispiel 1. Automatische Verknüpfung zweier Slaves

1. Slave1 publiziert:
   - `Raum1.Temperatur` (real, Ausgang)
   - `Raum1.Heizleistung` (real, Eingang)
   - `Raum1.Betriebstemperatur` (real, Ausgang)
2. Slave2 publiziert:
   - `Raum1.Temperatur` (real, Eingang)
   - `Raum1.Heizleistung` (real, Ausgang)
   - `Raum2.OperativeTemperatur` (real, Eingang)

Die automatische Verbindung erstellt:

- `Slave1.Raum1.Temperatur -> Slave2.Raum1.Temperatur`
- `Slave1.Raum1.Heizleistung -> Slave2.Raum1.Heizleistung`

Die dritte Verbindung wird nicht hergestellt, da *Raum1.Betriebstemperatur* namentlich nicht zu *Raum2.OperativeTemperatur* passt.

2.4.2. Eine Umrechnungsoperation zwischen Variablen definieren

Falls Sie die Umwandlung einer Einheit oder andere Änderungen (Zeichenumkehrung, Skalierung) zwischen Ausgangs- und Eingangsvariablen vornehmen wollen, können Sie in der dritten Spalte der Tabelle **(6) doppelklicken**, um einen Dialog für das Bearbeiten der Umrechungsoperation zu öffnen

(siehe Abschnitt Verbindungsgraph für eine detaillierte Beschreibung).

Es ist mitunter einfacher und übersichtlicher, Umrechungsoperationen in der graphischen Netzwerksicht direkt festzulegen. Dazu wird in der Slave-Ansicht im Netzwerk eine Verbindung markiert und in dem Eigenschaftsfenster unten links kann man direkt Skalierungsfaktor und Abstand eingeben.

2.5. Simulationsansicht

In dieser Ansicht werden alle Einstellungen zu den Co-Simulations-Algorithmen festgelegt. Eine detaillierte Beschreibung der Einstellungen und ihrer Anwendung findet man im Abschnitt Master-Algorithmen.

Abschnitt Projektdatei-Referenz - Simulationseinstellungen beschreibt die zugehörigen Einträge in der *MasterSim*-Projekt-Datei.

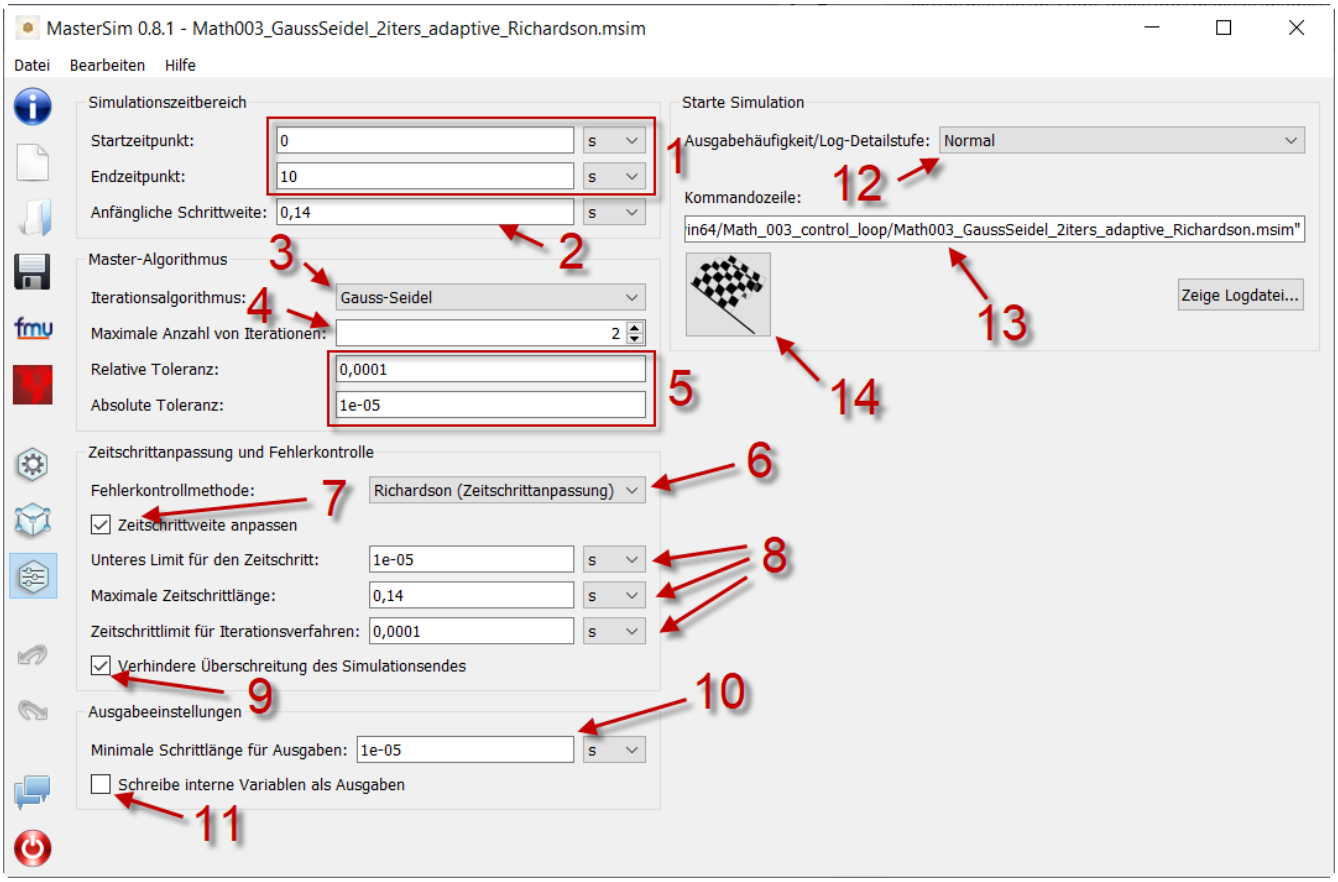

Abbildung 10. Simulationseinstellungen und die Startansicht der Simulation

(1) Hier können sie den Start- und Endzeitpunkt ihrer Simulation festlegen.

(2) Die anfängliche Intervallgröße der Datenübertragung. Wenn die Zeitschritt-Anpassung **(7)** deaktiviert ist, wird diese Intervallgröße konstant verwendet, bis das Ende der Simulationszeit erreicht wurde.

(3) Auswahl des Master-Algorithmus

(4) Maximale Anzahl an Wiederholungen, 1 deaktiviert Wiederholung.

(5) Die relativen und absoluten Toleranzen werden für einen Konvergenztest in iterativen Algorithmen und, falls eingeschaltet, für die Prüfung des lokalen Fehlers und die Zeitschritt-Anpassung genutzt.

(6) Hier können Sie eine Fehlerkontroll-Methode auswählen, siehe Abschnitt Fehlerkontrolle und Zeitschritt-Anpassung.

(7) Wenn eingeschaltet, wird *MasterSim* den Zeitschritt adaptiv verändern. Dies verlangt, dass FMUs die Fähigkeit **canHandleVariableCommunicationStepSize** unterstützen.

(8) Diese drei Parameter legen fest, wie der Zeitschritt im Fall einer Anpassung/Scheitern des Fehlertests angepasst wird.

(9) Wenn eingeschaltet, wird *MasterSim* die Schrittgröße an das letzte Intervall so anpassen, dass es den Endzeitpunkt der Simulation als Ende des *letzten* Kommunikationsintervalls *exakt* erreicht, ohne dabei Rücksicht auf das Option **(7)** zu nehmen (siehe Diskussion in Abschnitt Anpassung der Kommunikationsschrittlängen).

(10) Legt das minimale Zeitintervall zwischen dem Schreiben von Ausgabegrößen fest. Dies hilft, die Anzahl an Ausgaben im Fall variabler Zeitschritte zu reduzieren, wenn diese Zeitschritte viel kleiner als ein aussagekräftiges Ausgangsraster werden können.

(11) Wenn eingeschaltet, schreibt *MasterSim* auch die Werte interner Variablen in die Ausgabedateien, ansonsten nur die Ausgabevariablen (Causality = OUTPUT). Dies ist hauptsächlich nützlich für Analysieren und die Fehlersuche in FMUs, oder um interne Werte zu erhalten, die nicht von der FMU selbst als Ausgaben bereitgestellt werden.

(12) Damit kann die Detailstufe der Anwendungsmeldungen angepasst werden (siehe Abschnitt Befehlszeilenargumente).

(13) Befehlszeile, welche für den Start des Simulators verwendet wird. Kann für die automatische Verarbeitung in einem Shell-Skript oder einer Batch-Datei kopiert werden.

(14) Der große, dicke Start-Button. **Auf die Plätze, Fertig, Los!**

Wenn Sie die Simulation startet, wird ein Konsolenfenster mit einer Fortschritts-/Warnungs-/Fehlermeldung für die laufende Simulation auftauchen. Da einige Simulationen sehr schnell sein können, und damit das Fenster auch sehr schnell wieder verschwindet, wird sich nach ungefähr 2 Sekunden das Ausgabenmeldungsfenster mit dem gegenwärtigen Inhalt des Bildschirmprotokolls angezeigt.

Beachten Sie, dass die Simulation vielleicht noch im Hintergrund laufen könnte, selbst wenn das Protokollfenster bereits gezeigt wurde. Wenn Sie die Simulation indes mehrere Male starten, werden mehrere Simulationsprozesse parallel laufen. Das wäre aber nur eine Rechenverschwendung, da die parallelen Simulationen ins selbe

Verzeichnis schreiben und sich gegenseitig die Dateien überschreiben würden.

2.6. Einstellungs-Dialog

Der Einstellungs-Dialog, geöffnet im Hauptmenü oder durch die Anwendung des Tastatur-Kürzels, bietet derzeit Konfigurationsoptionen für den externen Texteditor (wird genutzt, um die Projekt-Datei mittels Kürzel **F2** zu editieren) und das ausführbare Post-Prozessing-Programm an.

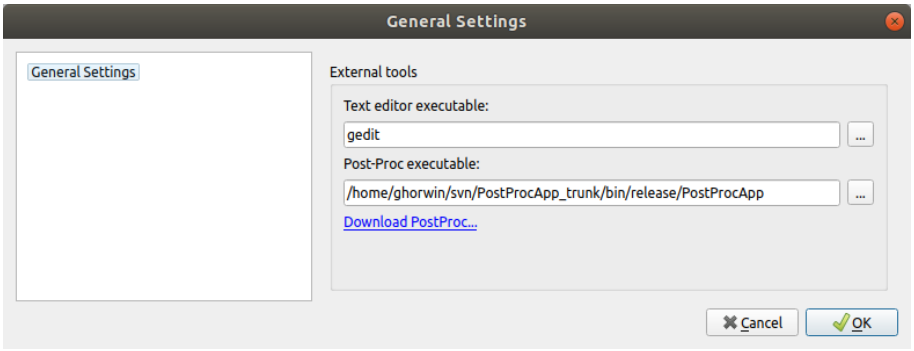

Abbildung 11. Einstellungsdialog mit Eingangsoptionen für den Texteditor und das ausführbare Nachbearbeitungsprogramm

Folgende (leichtgewichtige und open-source) Texteditoren bevorzuge ich:

- Linux: Geany
- Windows: Notepad++
- MacOS : Brackets (http://brackets.io)

Wenn Sie eine Textdatei im externen Texteditor bearbeiten und die Datei speichern, oder die Projektdatei anderweitig in einem externen Programm verändert wird, fragt die *MasterSim*-Benutzeroberfläche nach, wenn es das nächste Mal in den Vordergrund geholt wird, ob die Projektdatei neu geladen werden soll.

3. *MasterSimulator* - Das Befehlszeilen-Programm

Die eigentliche Simulation wird mit dem Befehlszeilen-Programm *MasterSimulator* durchgeführt. Es führt grundsätzlich die folgenden Schritte aus:

1. Es liest die msim-Projektdatei (die Netzwerkdarstellung wird ignoriert, da sie nur visuelle Informationen enthält).
2. Danach wird ein Arbeitsverzeichnis für die Simulationsdaten erzeugt (der Pfad kann angepasst werden, siehe: `--working-dir`-Befehlzeilen-Option unten).
3. Die FUMs werden extrahiert (dies kann übersprungen werden, falls die FMUs bereits entpackt vorliegen, siehe Befehlszeilen-Option `--skip-unzip` unten).
4. Die Simulation wird, wie in der Projekt-Datei konfiguriert, durchgeführt.

Benutzen von *MasterSim* in einer geskripteten Umgebung

Da die *MasterSim*-Projekt-Datei einfacher ASCII-Text ist und der Rechenkern durch die Befehlszeile gestartet werden kann, ist es möglich, *MasterSim* in einer geskripteten Umgebung und in automatisierten Prozessen zu nutzen (z. B. für Optimierungsrechnungen).

3.1. Befehlszeilenargumente

Allgemeine Syntax für das *MasterSimulator* Kommandozeilenprogramm:

```
Syntax: MasterSimulator [Schalter] [Optionen] <Projektdatei>

Schalter:
  --help            Zeigt diese Hilfsseite.
  --man-page        Generiert eine man-Hilfeseite (Quellversion)
  --cmd-line        Zeigt die Befehlszeile, wie sie vom Befehlszeilen-Syntaxanalysierer verstanden wurde.
  --options-left    Zeigt alle Schalter/Optionen, die dem Befehszeilen-Syntaxanalysierer unbekannt sind.
  -v, --version     Zeigt die Versionsinformation.
  -x, --close-on-exit  Schließt das Konsolenfenster nach dem Beenden der Simulation (nur Windows).
  -t, --test-init   Startet die Initialisierung und stoppt sie direkt danach.
  --skip-unzip      Überspringt den Schritt zum Entpacken der FMUs und erwartet bereits entpackte FMU-
Archive im Arbeitsverzeichnis.

Optionen:
  --verbosity-level=<0..4>
                    Detailstufe für Programmmeldungen (0-4).
  --working-dir=<working-directory>
                    Arbeitsverzeichnis für den Master.
```

3.1.1. Anpassung des Arbeits- und Ausgabeverzeichnis

Wenn kein Arbeitsverzeichnis vorgegeben wird, wird der Verzeichnis-Pfad zum Arbeitsverzeichnis aus dem Pfad zur Projektdatei ohne Dateierweiterung generiert. Zum Beispiel:

```
# Pfad zur Projektdatei
/simulations/myScenario.msim

# Basisverzeichnis für Simulationsdaten und -ergebnisse
/simulations/myScenario/...
```

Im Abschnitt Struktur und Inhalt des Arbeitsverzeichnis wird der Inhalt und die Struktur des Arbeitsverzeichnisses erklärt.

3.1.2. Anpassung der Ausgabedetailstufe

MasterSimulator schreibt Fortschrittsinformationen/Warnungen und Fehlermeldungen in das Konsolenfenster und in die Datei `<working-dir>/logs/screenlog.txt`. Die Menge an geschriebenem Text

und die Detailstufe wird durch den `--verbosity-level` Parameter kontrolliert, welcher die Menge der erzeugten Fehler-/Analyseinformationen festlegt.

Bei der Detailstufe 0 (`--verbosity-level=0`) wird so ziemlich je Ausgabe abgeschaltet. Die Detailstufe 1 ist der Standard mit normalen Fortschrittsmeldungen. Nutzen Sie eine höhere Ausgabedetailstufe zur Analyse der integrierten Algorithmen oder zur Eingrenzung von Fehlern.

Die Detailstufe der Protokolldatei ist während der Initialisierungsphase immer auf 3 gesetzt, um möglichst detaillierte FMU-Analyseausgaben in der Protokolldatei zu erhalten. Während der Simulation wird die Ausgabedetailstufe dann auf den Standardwert 1 zurückgesetzt, um eine Verlangsamung der Simulation aufgrund des exzessiven Ausgabeschreibens zu vermeiden. Falls der `--verbosity-level` -Befehlszeilenparameter angegeben ist, wird dieser Wert auch während der Simulation für die Protokolldatei verwendet.

3.1.3. Spezielle Optionen unter Windows

Normalerweise bleibt das Konsolenfenster (bei Windows) am Ende der Simulation offen. Man kann aber durch Angabe des `-x` Befehlszeilenschalters das Fenster nach Ende der Simulation automatisch schließen lassen.

3.2. Struktur und Inhalt des Arbeitsverzeichnis

MasterSim erstellt eine Verzeichnisstruktur nach folgendem Schema. Angenommen es gibt eine Projekt-Datei `simProject.msim`, welche im Simulationsszenario zwei FMUs `part1.fmu` und `part2.fmu` verwendet. Diese werden innerhalb des Projekts jeweils als Simulations-Slaves **P1** und **P2** bezeichnet. Nehmen wir an, diese Dateien sind in einem Unterverzeichnis abgelegt:

```
/sim_projects/pro1/simProject.msim
/sim_projects/pro1/fmus/part1.fmu
/sim_projects/pro1/fmus/part2.fmu
```

Bei Ausführung des Master-Simulators wird ein Arbeitsverzeichnis erstellt. Standardmäßig entspricht der Dateipfad zu diesem Arbeitsverzeichnis dem Projekt-Dateinamen ohne Dateierweiterung.

Die Verzeichnisstruktur sähe dann so aus:

```
/sim_projects/pro1/simProject/             - Arbeitsverzeichnis
/sim_projects/pro1/simProject/log/         - Logdateien und Statistiken
/sim_projects/pro1/simProject/fmus/        - Entpackte FMUs
/sim_projects/pro1/simProject/fmus/part1   - Entpackte part1.fmu
/sim_projects/pro1/simProject/fmus/part2   - Entpackte part2.fmu
/sim_projects/pro1/simProject/slaves/      - Ausgabe-/Arbeitsverzeichnis für FMU Slaves
/sim_projects/pro1/simProject/slaves/P1    - Ausgabe-/Arbeitsverzeichnis für Slave P1
/sim_projects/pro1/simProject/slaves/P2    - Ausgabe-/Arbeitsverzeichnis für Slave P2
/sim_projects/pro1/simProject/results/     - Ausgabeverzeichnis für Simulationsergebnisse
```

Der Basisname des Verzeichnisses, hier `simProject`, kann durch Angabe das Befehlszeilenarguments `--working-directory` verändert werden.

3.2.1. Verzeichnis `log`

Dieses Verzeichnis beinhaltet 3 Dateien:

`progress.txt`	beinhaltet den Simulations-Fortschritt (siehe Formatbeschreibung unten)
`screenlog.txt`	beinhaltet die Konsolenausgabe von *MasterSim*, welche in das Konsolenfenster mit der jeweils angeforderten Detailstufe geschrieben wurde (während der Initialisierung wird immer eine detaillierte Ausgabe an die Log-Datei geschrieben, selbst wenn die Bildschirmausgabe mit `--verbosity-level=0` deaktiviert wurde)
`summary.txt`	nachdem die Simulation *erfolgreich* vollendet ist, beinhaltet diese Datei eine Zusammenfassung der relevanten Lösungs- und Algorithmusstatistiken.

Das Format der Datei `progress.txt` ist recht simpel:

```
Simtime [s]     Realtime [s]  Percentage [%]
        600         0.000205          0.0019
       1200          0.00023          0.0038
       1800         0.000251          0.0057
       2400         0.000271          0.0076
        ...              ...             ...
```

Die Datei besitzt 3 Spalten, getrennt durch ein Tabulatorzeichen. Die Datei wird bei laufender Simulation geschrieben und aktualisiert und kann von anderen Werkzeugen genutzt werden, um den Gesamtfortschritt aufzugreifen und Fortschrittsdiagramme zu erzeugen. (Geschwindigkeit/Prozentsatz etc.)

Die Bedeutung der verschiedenen Werte im `summary.txt` werden im Abschnitt Simulations-Statistik/Zusammenfassung erklärt.

3.2.2. Verzeichnis `fmus`

Innerhalb dieses Verzeichnis werden die importierten FMUs extrahiert, jedes in ein Unterverzeichnis mit dem Basisdatei der FMU (`part1.fmu` → `part1`).

Wenn ein *MasterSim*-Projekt auf verschiedene FMUs desselben Basisnamen Bezug nimmt, welche zum Beispiel in verschiedenen Unterverzeichnissen stehen, wird es den Pfadnamen anpassen. Beispiel:

```
slave1 : /path/to/fmus/s1.fmu
slave2 : /path/to/fmus/s1.fmu               ①
slave3 : /path/other/project/fmus/s1.fmu    ②

# von _MasterSim_ erzeugte Verzeichnisse
.../fmus/s1
```

```
.../fmus/s1_2                    ③
```

① zweite Instanz der gleichen FMU (wird nur einmal entpackt)

② andere FMU mit gleichem Basisnamen

③ Suffix `_2` (bzw. `_3` etc.) wird durch *MasterSim* angehängt

Grundsätzlich wird jede FMU-Datei nur einmal entpackt.

Überspringen des FMU-Extraktions-Schritts

MasterSim unterstützt die Befehlszeilen-Option `--skip-unzip`, welche sehr nützlich ist, um FMUs durch Korrektur einer fehlerhaften `modelDescription.xml`-Datei oder fehlender Ressourcen zu reparieren. Wenn solch eine FMU auftaucht, können Sie *MasterSimulater* einmal durchlaufen lassen, um die FMUs in die Verzeichnisse zu extrahieren. Dann kann man die fehlerhaften Dateien im jeweiligen Entpack-Verzeichnis überarbeiten/anpassen und danach die Simulation noch einmal mit `--skip-unzip` durchlaufen lassen. *MasterSim* wird nun die (veränderten) Dateien direkt lesen und Sie können sich selbst die Mühe des Komprimierens und Umbenennens der FMUs sparen. Ebenso können Sie die `modelDescription.xml` im Editor geöffnet lassen und die wiederholte "Bearbeiten-und-Testsimulieren"-Prozedur schnell durchlaufen, bis alles funktioniert.

Siehe auch Erläuterungen im Abschnitt Einfache Veränderung/Reparatur von fehlerhaften FMUs.

3.2.3. Verzeichnis `Slaves`

Oft schreiben nicht-triviale Simulations-Slaves ihre eigenen Ausgabedateien, anstatt die gesamten Ausgabedaten per FMU-Ausgabevariablen zum Master zu verschieben. Bei FMUs, in denen partielle Differentialgleichungen gelöst werden, und Feldausgaben mit mehreren tausenden Datenpunkten je Zeitschritt erzeugt werden, wäre dies auch nicht sinnvoll (oder effizient). Für das Schreiben derartiger Ausgaben bräuchte die FMU einen Zielpfad, in den die FMU schreiben darf.

Da eine FMU mehrere Male instanziiert werden kann (d.h. mehrere Slaves werden aus der gleichen FMU-Datei erzeugt), ist die feste Programmierung eines Ausgabepfads innerhalb der FMU im Allgemeinen keine gute Idee (obgleich gegenwärtig noch immer häufige Praxis). Die Ausgaben der unterschiedlichen Slaves würden sich sonst gegenseitig überschreiben.

Ausgaben ins gegenwärtige Arbeitsverzeichnis zu schreiben ist ebenso ungeschickt, da das Arbeitsverzeichnis zwischen den Aufrufen der FMUs eventuell durch den Master geändert werden muss. Dadurch sollte der Zugriff auf das Arbeitsverzeichnis am besten vermieden werden.

Leider unterstützt der FMU-Standard keine Option, einem Slave ein gültiges Ergebnis-/Arbeitsverzeichnis zu übergeben. *MasterSim* behebt das Problem, indem es Slave-spezifische Verzeichnispfade in einem Parameter, genannt `ResultsRootDir`, angibt. Dazu braucht die FMU lediglich diesen Zeichenketten-Parameter zu deklarieren. Man kann in *MasterSim* den Wert dieses Parameters natürlich wie bei allen anderen Parametern manuell festlegen. Wenn allerdings kein Wert in der

Projekt-Datei für diesen Parameter festgelegt ist, wird *MasterSim* den für den Slave erzeugten Pfad im Arbeitsverzeichnis eintragen. Die FMU kann den von *MasterSim* erzeugten Slave-spezifischen Pfad nutzen und Ausgaben oder andere Daten dort hinein schreiben.

Das Analysetool *PostProc* wird bei Angabe des Projektbasisverzeichnis auch die Ausgaben in den Slave-Verzeichnissen finden und zur Analyse anzeigen.

3.3. Return-Code des *MasterSimulator* -Programms

MasterSimulator gibt folgende Return-Code zurück:

0 bei Erfolg

1 wenn ein Fehler aufgetreten ist (alle Ursachen, von schlechten oder fehlenden FMUs, oder Fehlern während der Berechnung,...), die Datei `screenlog.txt` beinhaltet Details.

3.4. Simulationsausgaben

3.4.1. Slave-Ausgabewerte

MasterSim erstellt zwei Ergebnisdateien innerhalb des `results`-Unterverzeichnis:

`values.csv` Ausgabe aller Ausgabevariablen vom Typ Zahl (einschließlich Booleans) von allen Slaves (egal, ob sie verbunden sind oder nicht).

`strings.csv` Werte aller Ausgabevariablen vom Typ Zeichenketter aller Slaves.

Abhängig davon, ob *synonyme Variablen* in der ModelDescription (siehe unten) definiert sind, die Datei `synonymous_variables.txt`.

Zeichenketten-Ausgabe-Dateien werden nur erzeugt, wenn tatsächlich Ausgaben dieses Datentyps von Slaves erzeugt werden.

3.4.2. Dateiformat der Ergebnisdateien

Die Ergebnisdateien haben die Erweiterung `csv`, nutzen aber Tabulatorzeichen als Trennzeichen. In der ersten Spalte steht immer der Zeitpunkt. Im Spaltenkopf jeder Spalte kann in [] die Zeiteinheit angegeben werden.

Beispiel `values.csv`-Datei:

```
Time [s]    slave1.h [-]    slave1.v [-]
0   1   0
0.001   0.999995099905  -0.0098100000000001
0.0019999999999999  0.99998038981   -0.019619999999999
0.0030000000000001  0.999955869715  -0.029430000000002
0.0040000000000002  0.99992153962   -0.039240000000001
```

Das Dateiformat entspricht dem der csv-Dateien, die als Datei-Lese-Slaves genutzt werden, siehe Abschnitt CSV-Datei-Lese-Slaves, mit:

- durch Tabulatoren getrennte Spalten,
- Nummern sind im englischen Nummernformat geschrieben, und
- eine einzelne Überschrift bestimmt die Variablen.

Den FMI- Variablennamen sind die entsprechenden Slave-Namen vorangestellt. Die Einheiten sind in Klammern angegeben und für einheitslose ganzzahlige und boolesche Datentypen, wird die Einheit [-] genutzt.

Vektorwertige Variablen, z.B. mit Namen "var[12]" werden immer einheitenlos geschrieben, sodass das *PostProc* Programm erkennt, dass es sich bei [12] nicht um eine Einheit handelt.

Synonyme Variablen

Einige FMUs, z.B. solche, die von Modelica Modellen erstellt wurden, können verschiedene (interne) Variblen aufweisen, welche den selben Wert referenzieren. Das passiert, wenn die symbolische Analyse des Modelica Modells diese Variablen als die selben erkennen konnte, z.B. bei einer Gleichung "a = b", wird eine FMU sicher intern nur eine Variable halten, aber gegebenenfalls bei Variablen getrennt als Ausgaben anbieten. Beide Variablen hätten in der `modelDescription.xml`-Datei die gleiche `valueReference` (Wertereferenz).

In diesem Fall schreibt *MasterSim* die Ausgabevariablen nicht doppelt (dies wäre eine Verschwendung von Festplattenkapazitäten und Simulationszeit, siehe Ticket #47), sondern erstellt eine Datei `synonymous_variables.txt` mit einer Tabelle synonymer Variablen.

Die Tabelle wird als einfache Textdatei geschrieben mit durch Tabulatoren getrennte Spalten:

1. FMU-Dateiname (gegenwärtig wird nur der Dateiname geschrieben - im Fall, dass der *gleiche Dateiname* mit *unterschiedlichen Dateipfaden* genutzt wird, muss dies geändert werden)
2. der Name der Variablen, erscheint in der `values.csv`-Datei
3. die synonyme Variable, die nicht in die Ausgabedatei geschrieben wird, da sie ohnehin den gleichen Wert hat.

Ein Beispiel für eine `synonymous_variables.txt`-Datei:

```
ControlledTemperature.fmu   heatCapacitor.T heatCapacitor.port.T
ControlledTemperature.fmu   heatCapacitor.T heatingResistor.T_heatPort
ControlledTemperature.fmu   heatCapacitor.T heatingResistor.heatPort.T
ControlledTemperature.fmu   heatCapacitor.T temperatureSensor.port.T
ControlledTemperature.fmu   heatCapacitor.T thermalConductor.port_a.T
ControlledTemperature.fmu   heatingResistor.p.v heatingResistor.v
ControlledTemperature.fmu   heatingResistor.p.v idealSwitch.n.v
ControlledTemperature.fmu   constantVoltage.i   constantVoltage.n.i
ControlledTemperature.fmu   constantVoltage.i   constantVoltage.p.i
```

```
ControlledTemperature.fmu   constantVoltage.i   heatingResistor.i
ControlledTemperature.fmu   constantVoltage.i   heatingResistor.n.i
ControlledTemperature.fmu   constantVoltage.i   heatingResistor.p.i
ControlledTemperature.fmu   constantVoltage.i   idealSwitch.i
ControlledTemperature.fmu   constantVoltage.i   idealSwitch.n.i
ControlledTemperature.fmu   constantVoltage.i   idealSwitch.p.i
ControlledTemperature.fmu   heatingResistor.LossPower   heatingResistor.heatPort.Q_flow
ControlledTemperature.fmu   fixedTemperature.port.Q_flow    thermalConductor.Q_flow
ControlledTemperature.fmu   fixedTemperature.port.Q_flow    thermalConductor.port_a.Q_flow
ControlledTemperature.fmu   fixedTemperature.port.Q_flow    thermalConductor.port_b.Q_flow
ControlledTemperature.fmu   onOffController.reference   ramp.y
ControlledTemperature.fmu   onOffController.u   temperatureSensor.T
ControlledTemperature.fmu   idealSwitch.control logicalNot.y
ControlledTemperature.fmu   logicalNot.u    onOffController.y
```

Aus dieser Datei wird ersichtlich, dass die Variablen `heatCapacitor.T`, `heatCapacitor.port.T`, `heatingResistor.T_heatPort`, `heatingResistor.heatPort.T`, `temperatureSensor.port.T` und `thermalConductor.port_a.T` alle den gleichen Wert haben und damit die Variablennamen Synonyme sind.

3.4.3. Simulations-Statistik/Zusammenfassung

MasterSim beinhaltet eine interne Zeitmessfunktion, welche die Ausführungszeiten der verschiedenen Teile der Software überwachen. Ebenso werden Ausführungshäufigkeiten für verschiedene wichtige Funktionen gezeigt. Die Statistik wird ins Konsolenfenster (bei `verbosity-level` > 0) und in die Log-Datei `screenlog.txt` geschrieben. Dabei wird folgendes Format verwendet (die unteren Zeilen beginnend ab "Part1" sind projektspezifisch).

```
Solver statistics
-------------------------------------------------------------------------------
Wall clock time                          =   78.044 ms
-------------------------------------------------------------------------------
Output writing                           =   76.767 ms
Master-Algorithm                         =    0.666 ms         324
Convergence failures                     =                      41
Convergence iteration limit exceeded     =                      41
Error test time and failure count        =    0.214 ms          85
-------------------------------------------------------------------------------
Part1                              doStep =    0.101 ms        1229
                                getState =    0.070 ms        1116
                                setState =    0.020 ms         509
Part2                              doStep =    0.079 ms        1496
                                getState =    0.039 ms        1116
                                setState =    0.024 ms         776
Part3                              doStep =    0.071 ms        1496
                                getState =    0.038 ms        1116
                                setState =    0.040 ms         776
-------------------------------------------------------------------------------
```

Die selben Statistikdaten werden in die `summary.txt`-Logsdatei kopiert, dabei aber in ein eher *maschinenfreundliches* Format (mit Zeitangaben immer in **Sekunden** bzw. der jeweils verwendeten Master-Zeiteinheit):

```
WallClockTime=0.078044
FrameworkTimeWriteOutputs=0.076767
MasterAlgorithmSteps=324
MasterAlgorithmTime=0.000666
ConvergenceFails=41
ConvergenceIterLimitExceeded=41
ErrorTestFails=85
ErrorTestTime=0.000214
Slave[1]Time=0.000191
Slave[2]Time=0.000142
Slave[3]Time=0.000149
```

Die Werte bedeuten im Einzelnen:

Wall clock time (`WallClockTime`)

gesamte Simulationszeit, die nach Abschluss der Initialisierung benötigt wurde. Die Dauer für Entpacken und Laden der Laufzeitbibliotheken wird nicht einbezogen

Output writing (`FrameworkTimeWriteOutputs`)

Zeit, die für das Schreiben von Ausgabedateien und das Berechnen damit zusammenhängender Werte gebraucht wurde.

Master-Algorithm

Zeit, die für den eigentlichen Master-Algorithmus (`MasterAlgorithmTime`) und die Anzahl der Aufrufe des Algorithmus und die gesamten genutzten Zeitschritte aufgewendet wurde (`MasterAlgorithmSteps`).

Convergence failures (`ConvergenceFails`)

Anzahl der Konvergenzfehler bei iterativen Master-Algorithmen.

Convergence iteration limit exceeded (`ConvergenceIterLimitExceeded`)

Anzahl der Überschreitungen der maximalen Iterationszahl in iterierenden Master-Algorithmen (diese Zahl sollte kleiner oder gleich der Anzahl der Konvergenzfehler sein).

Error test time and failure count

Anzahl der Fehlertest-Überschreitungen (`ErrorTestFails`) und die insgesamt genutzte Zeit, um die Fehlertests durchzuführen (`ErrorTestTime`). Dies schließt die Zeit ein, die für das Speichern und Zurücksetzen des FMU-Zustands und für die zusätzlichen Kommunikationsschritte benötigt wurde. Dies gilt nur für Master-Algorithmen mit aktivierter Fehlerkontrolle (Richardson-Varianten).

Die übrigen Zeilen zeigen Dauer und Aufrufhäufigkeiten individuell für jeden Slave. Die Zeilen zeigen die genutzte Zeit in den Funktionsaufrufen `doStep()`, `getState()` und `setState()` und die jeweilige Häufigkeit des Aufrufs. Die den Zustand betreffenden Funktionen (state) werden nur für iterierende Master-Algorithmen genutzt, falls die FMUs diese FMI 2 Funktionalität unterstützt. Es ist zu beachten, dass diese Funktionen sowohl vom Master-Algorithmus als auch vom Fehlertest aufgerufen werden (wenn eingeschaltet).

Das Schreiben von Ausgaben (**Output writing**) und Ausführen des Master-Algorithmus (**Master-Algorithm**) sind die beiden Hauptkomponenten des *MasterSimulator*-Pogramms, sodass ihre addierten

Zeiten nahe der Gesamtlaufzeit liegen sollten.

Die dritte Spalte in der Konsolen-Ausgabe-Statistik beinhaltet Zähler. Der Zähler für den Master-Algorithmus ist die Gesamtzahl der Ausführungen des Master-Algorithmus, also die Gesamtschrittzahl bzw. Gesamtzahl an *erfolgreich* absolvierten Kommunikationsintervallen. Neuversuche und Wiederholungen *innerhalb* des Master-Algorithmus werden hier nicht beachtet.

Sie sollten diese Profilierungswerte nutzen, um die Simulation abzustimmen und, im Fall einer sehr langsamen Simulation, herauszufinden, welche FMU die meiste Zeit benötigt. Diese kann man dann gezielt optimieren. Ebenso helfen sie zu erkennen, ob eine der eigentlich schnellen Funktionen, wie die zum Lesen und Rücksetzen des FMU-Zustands, ungewöhnlich lange braucht (beispielsweise, wenn FMU-Intern übermäßig lange Zeit für die Neuinitialisierung benötigt wird).

4. Format der Projekt-Datei

MasterSim benutzt eine einfache Projekt-Datei, welche das Simulations-Szenario beschreibt. Diese Projekt-Datei besitzt die Erweiterung `msim` und beinhaltet alle Daten, um eine Simulation durchzuführen.

Eine zweite Datei mit demselben Namen und der Erweiterung `bm` wird im gleichen Verzeichnis wie die Projekt-Datei gespeichert. Diese beinhaltet die grafische Darstellung des Simulations-Szenarios. Da die grafische Netzwerkanzeige *rein optional* ist, kann die `bm`-Datei beliebig weggelassen/ignoriert/gelöscht werden.

SSP-Unterstützung

Derzeit ist die Forschung im Modelica Association Project SSP (System Structure and Parameterization of Components for Virtual System Design) in vollem Gang, um einen Standard für das Darstellen eines Simulations-Szenarios festzulegen. Wenn die Spezifizierungen rechtzeitig hinreichend abgeschlossen sind, kann der Master-Simulator diesen Datei-Standard vielleicht unterstützen, zumindest den Import und Export solcher Dateien. Tatsächlich fügt eine solche Datei die Beschreibung der räumlichen Struktur der FMU-Verbindung und ihrer (nach wie vor optionalen) grafischen Darstellung in einer Datei zusammen. Allerdings ist dieses Datei-Format, ähnlich den FMUs, eigentlich eine Zip-komprimierte Verzeichnisstruktur, wodurch SSP-Projekt-Dateien vielleicht nicht länger effektiv in Versionskontrollsystemen genutzt werden können. Hier ist das ASCII-Format der aktuellen `msim`- und `bm`-Dateien gut geeignet und nützlich.

Die Projektdatei wird im Klartext (ASCII,UTF8-kodiert) abgelegt und hat beispielsweise folgenden Inhalt:

Beispiel 2. MasterSim-Projektdatei

```
# Created:  Di. Aug. 14 17:02:20 2018
# LastModified: Di. Aug. 14 17:02:20 2018

# Project file example for iterating GaussSeidel with time step adjustment
```

```
#
# No error test included, time step adjustment based on convergence failures.
tStart                    0 s
tEnd                      12 s
hMax                      30 min
hMin                      1e-06 s
hFallBackLimit            0.001 s
hStart                    1e-07 s
hOutputMin                0.12 s
adjustStepSize            no
preventOversteppingOfEndTime yes
absTol                    1e-06
relTol                    0.01
MasterMode                GAUSS_JACOBI
ErrorControlMode          NONE
maxIterations             1
writeInternalVariables    yes

simulator 0 0 Part1 #ff447cb4 "fmus/simx/Part1.fmu"
simulator 1 1 Part2 #ffc38200 "fmus/simx/Part2.fmu"
simulator 2 1 Part3 #ffff0000 "fmus/simx/Part3.fmu"

graph Part1.x2 Part2.x2
graph Part1.x1 Part2.x1
graph Part2.x3 Part3.x3
graph Part3.x4 Part2.x4

parameters Part1.para1  14.3
```

Jede Zeile legt eine andere Eigenschaft fest. Die Wortbausteiner jeder Zeile sind durch Leerräume (Tabulatoren oder Leerzeichen) voneinander getrennt. Die Zeilen, die mit einem Rautezeichen # beginnen, werden als Kommentare interpretiert.

Auf allen Betriebssystemen werden Zeichenketten (auch Pfade) UTF-8-kodiert erwartet. Unter Windows und auf dem Mac muss das beim Bearbeiten/Erstellen von Projektdateien mit dem Texteditor berücksichtigt werden.

Alle Kommentarzeilen vor der ersten Nicht-Kommentarzeile werden als Kopfzeilen mit Projektinformationen interpretiert. Auf die Stichworte `Created:` und `LastModified:` sollte ein mehr oder weniger sinnvoller (aber nicht standardisierter) Datums-/Zeitstempel folgen, der in der Benutzeroberfläche gezeigt wird. Andere Kopfzeilen werden als Projektbeschreibung betrachtet, welche in der Projektzusammenfassung auf der Startseite der grafischen Benutzeroberfläche gezeigt werden (siehe Abbildung 12).

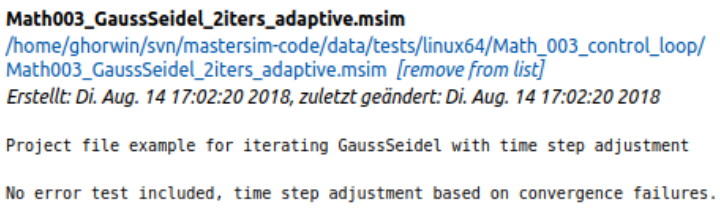

Abbildung 12. Auf der Startseite angezeigte Projekteigenschaften

Nach der Kopfzeile mit der Beschreibung ist die Ordnung der Einträge/Zeilen willkürlich. Es ist aber übersichtlich, die oben gezeigte Ordnung der Parametrisierung beizubehalten (welche von der *MasterSim* Programmoberfläche auch so geschrieben wird).

4.1. Einstellungen für die Simulation

Nachfolgend gibt es eine kurze Beschreibung der verschiedenen Parameter mit einer Formatsbeschreibung und den benötigten Werten. Für Details über ihren Gebrauch und welchen Einfluss sie haben, siehe Abschnitt Master-Algorithmen.

Parameter werden als mit einer Einheit versehenen Zahl angegeben, abgesehen von die Zählern (Maximum an Wiederholungen) oder relativen Fehlerschranken (welche sowieso ohne Einheit angegeben werden).

Optionen, die Kommunikationsschritte/Zeitschritte betreffend:

tStart (*default=0 s*) Startzeitpunkt der Simulation

tEnd (*default=1 a*) Endzeitpunkt der Simulation, muss > **tStart** sein

hMax (*default=30 min*) maximale Zeitschritt-Länge

hMin (*default=1e-5 s*) untere Grenze für Zeitschritte, wenn Zeitschritt-Anpassung eingeschaltet ist. Falls der Zeitschritt unter diese Grenze fällt, beendet der Master die Simulation mit einer Fehlermeldung

hFallBackLimit (*default=1e-3 s*) für einen Gauss-Seidel-Algorithmus mit Zeitschritt-Anpassung: wenn die Zeitschritte unter diese Grenze fallen, wird der nicht-iterierende Gauss-Seidel genutzt (um diskontinuierliche Variablenänderungen zu überspringen), sollte > **hMin** sein

hStart (*default=10 min*) anfänglicher Zeitschritt, wird auch als konstante Schrittgröße verwendet, wenn Zeitschritt-Anpassung abgeschaltet ist

Falls **hStart** in der Projekt-Datei auf 0 gesetzt ist, wird der Master automatisch **hStart** auf 1/1000 der Simulationsdauer festlegen, wobei die Simulationsdauer durch **tEnd - tStart** festgelegt ist.

hOutputMin

(*default=10 min*) minimale Zeitspanne die verstreichen muss, bevor das nächste Mal Ausgaben geschrieben werden. Wenn die Kommunikationsschrittlänge größer als **hOutputMin** wird, werden einige Ausgabezeitpunkte eventuell übersprungen, aber das reguläre Ausgaberaster bleibt erhalten.

outputTimeUnit

(*default=s*) Der Wert, der für die Zeitspalte (die erste Spalte) der Ausgabedateien genutzt wird.

adjustStepSize

(*default=false*) aktiviert/deaktiviert die Zeitschritt-Anpassung, wenn der Fehlerkontroll-Modus **ADAPT_STEP** ist, wird das Deaktivieren von **adjustStepSize** als Fehler gemeldet.

preventOversteppingOfEndTime

Diese Schalter wird für bestimmte FMUs gebraucht, welche einen Test gegen das Überschreiten der Simulationsendzeit enthalten. Dies ist in manchen Fällen mit Zeitreihen-Parameteren verbunden, die nur bis zum exakten Ende der Simulationszeit dauern. Ein anderes Problem ist, dass sich Rundungsfehler aufsummieren können und zu einer sehr kleinen Überschreitung des Endzeitpunktes führen können. Sich gut-verhaltende FMUs sollte zwar ein solches Überschreiten angemessen behandeln, es gibt jedoch auch FMUs, welche hier mit einer Fehlermeldung abbrechen. Um einen solchen FMU-Fehler und einen Simulationsabbruch zu vermeiden, kann *MasterSim* das letzte Kommunikationsintervall so anpassen, dass exakt die angegebene Endzeit der Simulation an das FMU übermittelt wird. Wenn dieser Schalter aktiviert ist, muss möglicherweise die letzte Intervallschritt-Größe verändert werden, selbst wenn eine Zeitschritt-Anpassung generell durch einen deaktivierten **adjustStepSize** Schalter verboten ist.

MasterMode (*default=GAUSS_SEIDEL*) ist einer von:

GAUSS_JACOBI	Gauss-Jacobi-Algorithmus (nicht iterierend)
GAUSS_SEIDEL	Gauss-Seidel-Algorithmus (iterierend oder nicht iterierend, abhängig von **maxIterations**)
NEWTON	Newton-Algorithmus mit einer Differenz-Quotient-Approximation der Jacobi-Matrix

Iterations- und Konvergenzparameter:

maxIterations	(*default=1=disabled*) max. Anzahl an Iterationen, wenn == 1 wird keine Iteration ausgeführt
absTol	(*default=1e-5*) absolute Toleranz für den Konvergenz-/Fehlertest
relTol	(*default=1e-6*) relative Toleranz für den Konvergenz-/Fehlertest

ErrorControlMode (*default=NONE=disabled*) ist einer von:

NONE	keine Fehlerprüfung und Anpassung
CHECK	nur Fehlerprüfung; Protokollzeit und Größenordnung von überschreitendem Fehlerlimit. Funktioniert auch mit FMI 1 (indem die Daten der letzten beiden Schritte genutzt werden). **WARNING:** Noch nicht implementiert. Nicht benutzen!
ADAPT_STEP	aktiviert automatisch die Zeitschritt-Anpassung und vergrößert/verkleinert die Kommunikationsschrittlänge entsprechend des Fehlerschätzers.

4.1.1. Fortgeschrittene Konfigurationen

Die folgenden Optionen werden zumeist für die Validierungs-Prozedur verwendet.

writeInternalVariables (*default=false*) Verfasst auch Variablen mit lokaler/interner Kausalität (wen es auf **no** gesetzt ist, werden nur Variablen mit der Kausalität *Ausgang* verfasst)

Abhängig von den gewählten Optionen, müssen einige Fähigkeiten durch die FMUs unterstützt werden, siehe dazu Erläuterungen in Abschnitt Master-Algorithmen. Grundsätzlich muss für die Verwendung eines iterierenden Master-Algorithmus oder für die Fehlerprüfung und Zeitschrittanpassung eine FMU den FMI 2 Standard und die Funktionalität für das Holen und Zurücksetzen des Zustands implementieren.

4.2. Simulator-/Slave-Definitionen

Jeder Slave wird festgelegt durch:

```
simulator  <priority> <cycle>  <slave-name> <html-color-code> <Pfad/zur/FMU-Datei>
```

Der **Zyklus** zeigt an, ob Slaves zu einem Zyklus mit anderen FMUs gehören. Der **Slave-Name** muss eine eindeutige Identifikation des Slaves zulassen (siehe Diskussion in Abschnitt Master-Algorithmen).

Die **Priorität** könnte genutzt werden, um die Reihenfolge der Auswertung innerhalb eines Zyklus auszuwählen (für Gauss-Jacobi/Gauss-Seidel). Die Funktion ist gegenwärtig aber nicht (mehr) implementiert und Slaves innerhalb des selben Zyklus werden in der Reihenfolge ausgewertet, in der sie festgelegt sind.

Der Slave-/Simulatorname muss eindeutig innerhalb des Simulationsprojekts sein.

Slave-Namen **dürfen keine** Leerzeichen oder Punkte enthalten. Wenn ein Slave-Name ein Leerzeichen oder einen Punkt enthält, wird der Parser der Projekt-Datei melden, dass die Definitionszeile der Simulation ungültig ist. Auch werden Slave-Namen für die Verzeichnisnamen genutzt (Zielverzeichnisse für Slave-spezifische Ergebnisse). Daher dürfen sie keine Zeichen beinhalten, die in Dateisystemnamen nicht erlaubt sind (wie z.B. *:* unter Windows).

Der **html-Farb-Code** st eine übliche html-basierte Farbdefinition, die mit einem Rautezeichen beginnt, auf welches entweder 8 oder 6 Zeichen folgen, zum Beispiel: `#ff00ff00` oder `#00ff00` für grün. Im 8-Zeichen-Format ist die erste Hexadezimalzahl der Alphawert/Transparenzwert. Gegenwärtig gibt es keinen Gebrauch für diesen Wert auf der Benutzeroberfläche, sodass die 6-Zeichen-Variante die gebräuchliche Wahl ist.

Das letzte Argument in der Zeile ist der Dateipfad zur eigentlichen FMU-Datei. Der Pfad zur FMU-Datei kann in Anführungszeichen angefügt werden, wenn der Pfad oder der Dateiname Leerzeichen enthält. Der Pfad kann ein absoluter Pfad oder relativ zur `msim`-Projektdatei sein. Einige Slaves können durch die

selbe FMU-Datei beschrieben werden (wenn die FMU diese Funktion unterstützt). In diesem Fall wird in einigen Simulatorzeilen der gleiche FMU-Dateipfad eingetragen.

4.2.1. CSV-Datei-Lese-Slaves

Bisweilen ist es sinnvoll/notwendig, vorgegebene Zeitreihen als Eingangsgrößen für FMUs bereitszustellen. Diese können in einer Datendatei bereitgestellt werden. Eine solche Datendatei (Erweiterung mit `tsv` oder `csv`) kann man wie eine FMU auswählen. *MasterSim* wird eine solche Datei wie eine FMU behandeln, und daraus CSV-Datei-Lese-Slaves erstellen, die lediglich Ausgabevariablen bereitstellen.

Effektiv unterstützt *MasterSim* zwei Varianten von CSV/TSV-Dateien (Dateierweiterung ist dabei egal). In beiden Varianten werden Zahlen immer im **englischen Zahlenformat** erwartet. Beim Einlesen wird zuerst versucht, eine durch Tabulatoren getrennte Spaltenstruktur zu erkennen, indem die ersten beiden Zeilen mit Tabulatorzeichen aufgeteilt werden. Wenn dies mehr als zwei Spalten ergibt und die Anzahl an Spalten in beiden Zeilen (in Kopf- und erster Datenzeile) übereinstimmt, wird eine Tabulator-getrennte CSV/TSV-Variante erkannt. Andernfalls wird eine CSV-Datei im "Excel"-Format erwartet (siehe unten).

Tabulator-getrennte Werte

Das Format einer solchen Datei folgt denselben Konventionen wie das Dateiformat, dass von PostProc 2 unterstützt wird (siehe auch Beschreibung dazu im *PostProc 2* Handbuch). *MasterSim* selbst schreibt Ergebnisse in diesem Dateiformat (siehe Simulationsausgaben).

Die Datei startet mit einer einzelnen Zeile (der Kopfzeile), in welcher Variablennamen und (optional) Einheiten angegeben werden. Es wird folgendes Format erwartet (wobei die Anzahl der Spalten nicht begrenzt ist).

```
Time [<time unit>] <tab> <var1 name> [<unit>] <tab> <var2 name> [<unit>]
```

`<tab>` ist ein Tabulatorzeichen. Beispiel:

```
Time [d]  <tab>  T_lab [C]  <tab>  T_sample [C]  <tab>  RH_lab [%]
```

Beispiel 3. Datei mit 3 Variablen

```
Time [h]     T_lab [C]     T_sample [C]     RH_lab [%]
0            20            20.2             46
0.5          20.1          20.3             43
1.0          22            25               40
3.0          19            15               65
```

Die Variablennamen entsprechen den Spaltenüberschriften in der Kopfzeile, ausgenommen der Einheiten (falls angegeben). Im Beispiel wird die erzeugte Datei-Lese-FMU Ausgangsvariablen mit

den Namen **T_lab**, **T_sample** und **TH_lab** anbieten.

Eine Datei mit diesem Format erhält man automatisch, wenn eine Tabelle mit solchen Daten aus LibreOffice/Calc oder Excel etc. in einen einfachen Text-Editor kopiert wird.

Kommatrennung mit Anführungszeichen

Das zweite unterstützte Format entspricht dem Format von Dateien, welche als CSV-Dateien gespeicherte Excel-Dateien haben. In solchen Dateien ist das Trennungszeichen das , (Komma) und Werte werden durch Anführungszeichen angegeben (siehe Beispiel 4).

Beispiel 4. Datei im CSV Format mit Komma-Trennung

```
"time","T_lab [C]","T_sample [C]","RH_lab [%]"
"0","20","20.2","46"
"0.5","20.1","20.3","43"
"1.0","22","25","40"
"3.0","19","15","65"
```

Der Inhalt dieser Datei entspricht der aus Beispiel 3.

Für beide Formatvarianten gilt: für Variablen ohne gegebene Einheiten, d.h. ohne [...] in der Spaltenüberschrift, wird eine unbekannte/undefinierte Einheit - angenommen.

4.2.2. Zeitpunkte und Zeiteinheiten

Die Zeitpunkte können in beliebigen Intervallen angegeben sein. *MasterSim* geht davon aus, dass Sekunde als Basis-Zeiteinheit verwendet wird. Das bedeutet, dass Variablen intern zu einer Simulationszeit in Sekunden ausgetauscht werden. Wenn eine Eingangsdatei eine andere Einheit für die Zeit festlegt, konvertiert *MasterSim* diese Zeitangaben beim Einlesen der Datei in Sekunden.

Die folgenden Zeiteinheiten werden von MasterSim erkannt:

ms	Millisekunden
s	Sekunden
min	Minuten
h	Stunden
d	Tage
a	Jahre (reguläre Jahre, 365 reguläre Tage, kein Schaltjahr/-tag)

Die standardmäßige Zeiteinheit ist Sekunde

Im Falle einer fehlenden Zeiteinheit in der Kopfzeile der ersten Spalte (wie in Beispiel 4) nimmt *MasterSim* die Zeiteinheit **Sekunden** (s) an.

4.2.3. Interpretation der von den Datei-Lese-Slaves bereitgestellten Daten

Die von einem Datei-Lese-Slave exportierten Variablen sind zunächst keinem Datentyp zugewiesen, wie das sonst bei FMU Ergebnisvariablen ist. Daher prüft *MasterSim* während der Initialisierung nach den Verbindungen, die mit Datei-Lese-Slaves gemacht worden sind. Falls Verbindungen definiert sind übernimmt *MasterSim* den Datentyp der verknüpften Eingangsvariable auch für die Ausgangsvariable des Datei-Lese-Slaves.

Falls man versucht, die gleiche Ausgabevariable eines Datei-Lese-Slaves auf verschiedene Eingangsvariablen mit *unterschiedlichen Variablentypen* zuzuweisen, bricht *MasterSim* mit einer Fehlermeldung ab.

Während der Simulation, wenn der Datei-Lese-Slave ausgewertet wird, gelten die folgenden Regeln: für die Berechnung der Ergebnisgrößen.

Boolean-, Integer- und Enumeration-Datentypen

Für die Datentypen **Boolean**-, **Integer**- und **Enumeration** wird keine Interpolation vorgenommen. Die Werte in der Tabelle werden konstant zurückgeliefert, bis in einer Zeile ein neuer Wert definiert wird.

Beispiel 5 verdeutlicht die Auswertung der Werte.

Beispiel 5. Konstante Auswertung bei nicht-interpolierten Datentypen

Datentabelle für die einheitenlose Variable v:

```
Zeit [s]   v [-]
 1         4
 3         4 ①
 3         7 ②
 6         4
```

① Der Wert am Ende des Intervalls endet zum Zeitpunkt 3

② Der Wert zu Beginn des Intervalls startet mit Zeitpunkt 3; dieser Wert sollte von t>=3 genutzt werden.

Auswertung dieser Datentabelle für ausgewählte Zeitpunkte:

```
v(1) = 4
v(2) = 4
v(2.99999) = 4
v(3) = 7
v(4) = 7
v(5.99999) = 7
```

```
v(6) = 4
```

Bei konstanter Auswertung ohne Interpolation könnte die Zeile `3 4` aus der Datei weggelassen werden.

Gleitkomma-Werte

Werte vom Datentyp **Real** (Gleitkommazahlen) werden linear interpoliert. Analog zum obigen Beispiel zeigt Beispiel 6 die Auswertung bei Verwendung der linearen Interpolation.

Beispiel 6. Lineare Interpolation bei Gleitkommazahlen

```
v(1) = 4
v(2) = 4
v(2.99999) = 6.99999 ①
v(3) = 7
v(4) = 6 ②
v(5.99999) = 4.00001
v(6) = 4
```

① Wenn doppelte Zeitpunkte gefunden werden, überschreibt der zweite den ersten Wert, sodass die Zeile `3 4` ignoriert wird. Daher wird die Bewertung der Werte im Intervall 2...3 ebenso mit linearer Interpolation durchgeführt.

② Die lineare Interpolation zwischen den Werten v(3)=7 und v(6)=4 bei t=4 ergibt 6.

Wenn Sie Stufenfunktionen mit **Real**-Werten nachbilden möchten, nutzen Sie einfach ein sehr kurzes Wechsel-Intervall, z. B. `v(1) = 4; v(2.9999) = 4; v(3) = 7`. *MasterSim* wird nach wie vor den linearen Anstieg zwischen t=2.9999 und 3 verwenden, was aber unerheblich für die Ergebnisse sein sollte.

Starke Änderungen des Anstiegs in benachbarten Intervallen (Unstetigkeiten in der ersten Ableitung) können bei Fehlerschätzer-basierter Zeitschrittanpassung zu sehr kleinen Zeitschritten führen und damit die Simulationszeit drastisch verlängern. Wann immer möglich (und ohne die Ergebnisse signifikant zu verfälschen) sollte man versuchen, größere Sprünge zwischen Anstiegen zu vermeiden oder geeignet zu glätten.

Falls Sie einen Master-Algorithmus mit Zeitschrittanpassung in *MasterSim* verwenden, sollten Sie den maximalen Zeitschritt (max. Länge des Kommunikationsintervalls) auf einen Wert festlegen (Parameter **hMax**, siehe Abschnitt Einstellungen für die Simulation), der kleiner als ihr kleinstes Zeitintervall in der Eingangsdatei ihres Datei-Lese-Slaves ist. Ansonsten könnte *MasterSim* unter Umständen ein Intervall überspringen und gar nicht merken, dass es zwischenzeitlich eine Änderung in den Eingangsdaten gab. Hierbei würden Daten fehlen und wahrscheinlich falsche

> Ergebnisse erzeugt.
>
> Beispiel: wenn Sie mit stündlichen Klimadaten arbeiten, wählen Sie 30 Minuten als maximale Länge für ein Kommunikationsintervall.

Zeichenketten-Variablen

Zeichenketten (**String**)-Variablen werden wie **ganzzahlige** Werten behandelt.

4.3. Verbindungsgraph

Der Verbindungsgraph beschreibt die Verknüpfung aller Slaves über Ein- und Ausgangsvariablen und legt damit den Datenaustausch zwischen den Slaves fest. Jede `graph` Zeile legt den Datentransfer zwischen einer Ausgangs- und einer Eingangsvariable fest.

Syntax der Definition:

```
graph <outputvar> <inputvar> [<offset> <scale factor>]
```

Ausgangs- und Eingangvariablen werden jeweils aus Slave-Namen und Variablennamen zusammengesetzt:

```
graph <slave-name>.<variable-name> <slave-name>.<variable-name>  [<offset> <scale factor>]
```

Der optionale Verschiebungs- und Skalierungsfaktor legt eine Umrechenvorschrift zwischen Ausgangsvariable und Eingangsvariable fest. Wenn eine Umrechnungsvorschrift angegeben wird, müssen immer beide Werte in der Zeile angegeben sein.

Die folgende Umrechnungsgleichung wird verwendet:

```
input = offset + scale * output
```

Falls zum Beispiel ein FMU-Slave *Sensor* eine Temperatur in Kelvin liefert und ein anderer FMU-Slave *Heater* die Temperatur in Grad Celsius erwartet, können Sie die Verbindung wie folgt festlegen:

```
graph  Sensor.temperature  Heater.temperature  -273.15  1
```

wodurch *MasterSim* folgende Gleichung ausführt:

```
input (in C) = -273.15 + 1 * output (in K)
```

Auf ähnliche Weise können Sie das Vorzeichen einer Variable in einer Verbindung umkehren, wenn Sie zum Beispiel Wärmeströme oder Masseströme durch Röhren verbinden. Angenommen der

Wärmestrom ist positiv in Richtung der Oberflächennormalen festgelegt und Sie verbinden *SurfaceA.HeatFlow* und *SurfaceB.HeatFlow*, dann wird die Verknüpfung unter Berücksichtigung der Vorzeichenumkehr so definiert:

```
graph  SurfaceA.HeatFlow  SurfaceB.HeatFlow  0  -1
```

4.3.1. FMU-Parameter

Sie können Parameter der FMUs (oder konkreter, die der individuellen FMU-Slaves) festlegen, indem Sie das `parameter`-Schlüsselwort benutzen.

Definition der Syntax:

```
parameter <slave-name>.<variable-name> <value>
```

Für **boolesche** Parameter müssen Sie `true` als Wert verwenden (Kleinschreibung beachten) oder irgend einen anderen Wert (zum Beispiel `false`) für **falsch**.

Für ganzzahlige Werte (Typ **Integer**) müssen Sie einfach den Wert als Ziffer festlegen.

Werte für Parameter vom Typ **Real**" werden in der Einheit erwartet, die in der `modelDescription.xml` -Datei für den entsprechenden Parameter festgelegt worden sind. Eine Einheitenumrechnung wird hier **nicht** unterstützt.

Für **String**-Parameter wird alles nach dem Variablennamen als Zeichenkette angesehen (bis zum Ende der Zeile). Beispiel:

```
parameter building_model.projectFile C:\\My projects\\p2\\This tall building.project
```

Leerstellen können enthalten sein. Aber die Rückwärts-Schrägstrich-Zeichen (*backslash*) müssen als `\\` kodiert sein. Dies ist notwendig, damit Sonderzeichen wie Zeilenumbrüche durch `\n` kodiert werden können, wie im folgenden Beispiel gezeigt:

```
parameter building_model.configPara First line\n    Some more lines with indentation\nlast line.
```

Dies wird den folgenden String setzen:

```
First line
    Some more lines with indentation
last line
```

In dem seltenen Fall, wenn man Zeichenketten-Parameter mit führenden und abschließenden Leerzeichen festlegen möchte, kann man die Zeichenkette in ""

einschließen. Beispiel:

```
parameter building_model.configPara "   Leerzeichen am Anfang und Ende    "
```

4.4. BlockMod - Dateiformat der Netzwerkdarstellungsdatei

Die bm-Datei ist eine simple XML-Datei und beschreibt die graphische Anordnung und die Visualisierung des modellierten Simulations-Szenarios.

Ein einfaches Netzwerk, wie:

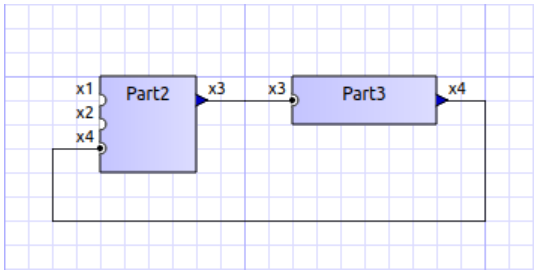

Abbildung 13. Beispiel für eine einfache grafische Präsentation eines Netzwerks

wird wie folgt in einer BlockMod Netzwerk-Beschreibungsdatei abgelegt:

```
<?xml version="1.0" encoding="UTF-8"?>
<BlockMod>
    <!--Blocks-->
    <Blocks>
        <Block name="Part2">
            <Position>224, -160</Position>
            <Size>64, 64</Size>
            <!--Sockets-->
            <Sockets>
                <Socket name="x1">
                    <Position>0, 16</Position>
                    <Orientation>Horizontal</Orientation>
                    <Inlet>true</Inlet>
                </Socket>
                <Socket name="x2">
                    <Position>0, 32</Position>
                    <Orientation>Horizontal</Orientation>
                    <Inlet>true</Inlet>
                </Socket>
                <Socket name="x4">
                    <Position>0, 48</Position>
                    <Orientation>Horizontal</Orientation>
                    <Inlet>true</Inlet>
                </Socket>
                <Socket name="x3">
                    <Position>64, 16</Position>
                    <Orientation>Horizontal</Orientation>
                    <Inlet>false</Inlet>
                </Socket>
```

```
            </Sockets>
        </Block>
        <Block name="Part3">
            <Position>352, -160</Position>
            <Size>96, 32</Size>
            <!--Sockets-->
            <Sockets>
                <Socket name="x3">
                    <Position>0, 16</Position>
                    <Orientation>Horizontal</Orientation>
                    <Inlet>true</Inlet>
                </Socket>
                <Socket name="x4">
                    <Position>96, 16</Position>
                    <Orientation>Horizontal</Orientation>
                    <Inlet>false</Inlet>
                </Socket>
            </Sockets>
        </Block>
    </Blocks>
    <!--Connectors-->
    <Connectors>
        <Connector name="new connector">
            <Source>Part2.x3</Source>
            <Target>Part3.x3</Target>
            <!--Connector segments (between start and end lines)-->
            <Segments>
                <Segment>
                    <Orientation>Horizontal</Orientation>
                    <Offset>0</Offset>
                </Segment>
            </Segments>
        </Connector>
        <Connector name="auto-named">
            <Source>Part3.x4</Source>
            <Target>Part2.x4</Target>
            <!--Connector segments (between start and end lines)-->
            <Segments>
                <Segment>
                    <Orientation>Vertical</Orientation>
                    <Offset>80</Offset>
                </Segment>
                <Segment>
                    <Orientation>Horizontal</Orientation>
                    <Offset>-288</Offset>
                </Segment>
                <Segment>
                    <Orientation>Vertical</Orientation>
                    <Offset>-48</Offset>
                </Segment>
            </Segments>
        </Connector>
    </Connectors>
</BlockMod>
```

Das Format ist ziemlich selbsterklärend. Das erste und das letzte Segment einer Verbindung (`Connector` tag) wird automatisch abhängig von der Sockelposition auf dem Block festgelegt, und wird dadurch

nicht in der Netzwerk-Beschreibungsdatei gespeichert.

BlockMod ist eine Open-Source-Bibliothek zum Modellieren solcher Netzwerke. Die Wiki-Seite des Projekts enthält mehr ausführliche Informationen über das Datenformat und die Funktionalität.

5. Testreihen und Validierung

Das *MasterSim* Quelltext-Repository enthält ein Unterverzeichnis `data/tests` mit einem Regressionstest. Diese Testreihe wird genutzt, um zu prüfen, ob alle Algorithmen wie erwartet arbeiten und ob Änderungen am Quelltext versehentlich etwas kaputt machen.

MasterSim durchläuft außerdem die FMU-Quer-Validierung (fmi-cross-check). Die Dateien und Skripte, die mit diesem Test in Verbindung stehen, befinden sich im Unterverzeichnis `cross-check`.

5.1. Regressionstests

5.1.1. Verzeichnisstruktur

```
/data/tests                     - Basisverzeichnis für Tests
/data/tests/<platform>/<test>   - Basisverzeichnis für eine Testreihe
```

`platform` ist (gegenwärtig) eine von: `linux64`, `win32`,`win64` und `darwin64`

Jede *Testreihe* hat eine Reihe von Unterverzeichnissen:

```
fmus                - enthält für den Test benötigte FMU-Archive
description         - (mathematische) Beschreibung des Test-Problems
```

Innerhalb der *Testreihe* können verschiedene ähnliche *Testfälle* mit modifizierten Parametern oder angepassten Problemlösungs-Szenarien gespeichert sein. *Testfälle* werden in der Regel als Testreihen gruppiert, wenn sie die gleichen FMUs oder andere Eingangsdateien benutzen.

Für jeden *Testfall* existiert eine *MasterSim*-Projektdatei mit der Erweiterung `msim`. Das Skript, welches die Testfälle durchführt, verarbeitet alle `msim`-Dateien, die in der Unterverzeichnisstruktur unter der aktuellen Plattform gefunden werden.

Um die Überprüfung in einem Testfall zu bestehen, muss eine Reihe von Referenzergebnissen vohanden sein, die in einem Unterverzeichnis mit folgendem Namen gespeichert sind:

```
<project_file_without_extension>.<compiler>_<platform>
```

Zum Beispiel:

```
FeedthroughTest.gcc_linux
FeedthroughTest.vc14_win64
```

Diese Verzeichnisse sind grundsätzlich nach einer durchlaufenen Simulation umbenannte Arbeitsverzeichnisse, in denen alles außer `summary.txt` und `values.csv` beseitigt worden ist.

5.1.2. Durchlauf der Tests

Die Tests laufen automatisch nach Erstellen der Software ab. Ansonsten kann man die Testreihe auch über ein Skript selbst starten (je nach Plattform):

- `build/cmake/run_tests_win32.bat`
- `build/cmake/run_tests_win64.bat`
- `build/cmake/run_tests.sh`

5.1.3. Aktualisierung der Referenzergebnisse

Aus einem Testverzeichnis heraus kann man das Script `update_reference_results.py` aufrufen und dabei die plattformspezifische Verzeichnisendung als Argument angeben.

Beispielsweise würde der Aufruf aus dem Verzeichnis

```
data/tests/linux64/Math_003_control_loop
```

so aussehen:

```
> ../../../../scripts/TestSuite/update_reference_results.py gcc_linux
```

Das Script aktualisiert nun alle Referenzergebnisse in diesem Verzeichnis. Falls die Referenzergebnisse mehrerer Testreihen bearbeitet/aktualisiert werden sollen, so führt man das Skript `update_reference_results_in_subdirs.py` aus einem der oberen Verzeichnisse durch, zum Beispiel von: `data/tests/linux64` (anderen Scriptnamen beachten!):

```
> ../../../scripts/TestSuite/update_reference_results_in_subdirs.py gcc_linux
```

5.2. Regeln für Quervergleiche und die FMI Standard.org Übersicht

Siehe Dokumentation im Unterverzeichnis `cross-check`.

5.3. Verschieden Möglichkeiten, Test-FMUs zu erstellen

5.3.1. Native C++ FMUs

Ein geeigneter Weg, um eine einfache, sehr spezielle Test-FMU zu erhalten, ist die Verwendung des FMI Code Generator-Werkzeugs. Es ist ein Python-Skript mit einer grafischen Benutzeroberfläche, in der FMU-Variablen und -Eigenschaften festgelegt werden können. Mit dieser Information erstellt der Code-Generator ein Quelltextverzeichnis mit einer Programm-Quelltextvorlage und den benötigten Skripten für das Erstellen und Verpacken der FMUs. Damit ist es sehr einfach, eigene FMUs zu erstellen (siehe auch Dokumentation/Tutorial auf der Webseite https://github.com/ghorwin/FMICodeGenerator).

5.3.2. Modelica-basierte FMUs

FMUs aus kommerziellen Modelica-Entwicklungsumgebungen

Benötigt eine entsprechende passende Lizenz. Das Exportieren von FMUs aus SimulationX oder Dymola ist recht einfach, aber für die Windows-Plattform beschränkt.

OpenModelica kann ebenso FMUs exportieren. Nachfolgend sind die Schritte erläutert, um so eine FMU zu erstellen.

5.3.3. Erstellen einer FMU mit OpenModelica

Zunächst wird ein Modelica-Modell erstellt.

Versehen Sie Variablen, die als Ausgangsgrößen von der FMU bereitgestellt werden sollen, mit dem Stichwort `output`.

Zum Beispiel:

```
model BouncingBall "The 'classic' bouncing ball model"
  type Height=Real(unit="m");
  type Velocity=Real(unit="m/s");
  parameter Real e=0.8 "Coefficient of restitution";
  parameter Height h0=1.0 "Initial height";
  output Height h "Height";
  output Velocity v(start=0.0, fixed=true) "Velocity";
  output Integer bounceCounter(start=0);
  output Boolean falling;
initial equation
  h = h0;
equation
  v = der(h);
  der(v) = -9.81;
  if v < 0 then
    falling = true;
  else
    falling = false;
  end if;
  when h<0 then
    reinit(v, -e*pre(v));
    bounceCounter = pre(bounceCounter) + 1;
  end when;
```

```
annotation(
    experiment(StartTime = 0, StopTime = 5, Tolerance = 1e-6, Interval = 0.01));
end BouncingBall;
```

Erstellungsvariante 1: Die FMU manuell erstellen

Öffnen Sie zunächst OMShell, tippen Sie dann die folgenden Befehle, um das Modell zu laden und erzeugen Sie eine Co-Simulations-FMU:

```
>> loadFile("/path/to/modelica/models/BouncingBall/BouncingBall.mo")
>> translateModelFMU(BouncingBall, fmuType="cs")
"/tmp/OpenModelica/BouncingBall.fmu"
```

Die Ausgabe `/tmp/OpenModelica/BouncingBall.fmu` zeigt an, dass die FMU erfolgreich erstellt wurde.

Zur Erstellung von FMUs mit FMI 2.0 Unterstützung:

```
>> translateModelFMU(BouncingBall, fmuType="cs", version="2.0")
```

Erstellungsvariante 2: Skript-basierte automatische FMU-Erzeugung

Erstellen Sie eine Skript-Datei (`createFMU.mos`) mit dem folgenden Inhalt:

```
loadModel(Modelica, {"3.2.1"}); getErrorString();
loadModel(Modelica_DeviceDrivers); getErrorString();

setLanguageStandard("3.3"); getErrorString();

cd("./fmus");
loadFile("../reference_Modelica/BouncingBall.mo"); getErrorString();

setDebugFlags("backenddaeinfo");getErrorString();
translateModelFMU(BouncingBall, fmuType="cs"); getErrorString();
```

Lassen Sie das Skript laufen, mittels:

```
> omc createFMU.mos
```

6. Assistenzfunktionen für FMU-Entwicklung und Fehlerbeseitigung

Dieser Abschnitt beschreibt einige Funktionen von *MasterSim*, die sehr nützlich für Entwickler eigener FMUs oder zum Identifizieren von FMU-Berechnungsproblemen sind.

Dadurch, dass der *MasterSim*-Quelltext frei verfügbar ist, kann die ganze Kommunikation zwischen Co-

Simulations-Master und den Simulations-Slaves im Debugger verfolgt und analysiert werden. Doch auch ohne Entwicklungsumgebung und Quelltextzugriff, kann man mit *MasterSim* effizient FMU-Probleme beseitigen.

6.1. Einfache Veränderung/Reparatur von fehlerhaften FMUs

Häufig resultieren Fehler in FMUs aus fehlerhaften/unvollständigen `modelDescription.xml`-Dateien. Oder es fehlen Resourcen. Oder die im FMU-Archiv eingebetteten Resource sind veraltet, und müssen ersetzt werden (Datenbankdateien etc.).

In diesem Fall ist die klassische Arbeitsweise:

1. FMU entpacken
2. Inhalt bearbeiten
3. FMU-Verzeichnis wieder komprimieren
4. Komprimierte Datei umbenennen/in das ursprüngliche Verzeichnis kopieren

Dies kann recht mühsehlig und zeitaufwändig sein. *MasterSim* kann dieses Prozedere abkürzen.

Dazu startet man den *MasterSimulator* einmal regulär (und bricht dann ggfs. die Simulation ab). Die verwendeten FMUs wurden bei der Initialisierung in die jeweiligen Verzeichnisse entpackt (siehe auch Abschnitt Struktur und Inhalt des Arbeitsverzeichnis).

Nun kann man in diesem Verzeichnissen die benötigten Änderungen vornehmen, z.B. die `modelDescription.xml`-Datei im Texteditor bearbeiten oder Dateien austauschen. Auch kann man die Laufzeitbibliotheken der FMU selbst austauschen (oder neu compilieren).

Nun kann man *MasterSimulator* erneut durchlaufen lassen, jedoch diesmal mit der Befehlszeilen-Option `--skip-unzip`. *MasterSim* wird den Schritt des Entpackens überspringen und direkt auf die Dateien im extrahierten Verzeichnis zugreifen.

Beim Bearbeiten der `modelDescription.xml`-Datei kann man diese dabei auch im Texteditor geöffnet halten, was wiederum etwas Zeit spart.

7. Informationen für Entwickler

Die aktuellsten Informationen sind in der englischsprachigen Dokumentation enthalten.

Die in diesem Kapitel zur Verfügung gestellten Informationen werden eventuell in den *MasterSim Developers Guide* verschoben.

7.1. Erstellen von Bibliotheken und ausführbaren Programmen

7.1.1. Erstellen mit die Befehlszeile

Linux/MacOS

```
cd build/cmake
./build.sh
```

Im Falle fehlender Abhängigkeiten (verlangt zlib) müssen ggfs. die benötigten Entwicklungspakete installiert werden.

Windows

Es sind komfortable Skripte für die Erstellung mit Visual Studio 2015 und Qt5 (für die Benutzeroberfläche von *MasterSim*) enthalten. Andere Compiler, wie MinGW, arbeiten genauso gut, die Dateipfade/Suchpfade müssen allerdings manuell konfiguriert werden.

Folgende Dateien sind im `build/cmake`-Verzeichnis:

```
build\cmake\build_VC.bat ①
build\cmake\build_VC_x64.bat ②
```

① for x86 builds

② for x64 builds

Damit die Skripte funktionieren, muss Qt am folgenden Ort installiert sein:

```
C:\Qt\5.15.2\msvc2019 ①
C:\Qt\5.15.2\msvc2019_64 ②
```

① für Aufbauten x86

② für Aufbauten x64

und **jom.exe** findet man unter:

```
c:\Qt\Tools\QtCreator\bin\jom.exe
```

`cmake` muss ebenso im Pfad vorhanden sein. Wenn diese Tools irgendwo anders installiert sind, können alternativ die Umgebungsvariablen `JOM_PATH` und `CMAKE_PREFIX_PATH` gesetzt werden.

Mit dieser Konfiguration können Sie nun entweder im 32-bit- oder 64-bit-Modus weiter bauen:

```
cd build\cmake
build_VC.bat
```

oder:

```
cd build\cmake
build_VC_x64.bat
```

Für unterschiedliche Visual-Studio-Versionen oder MinGW kopieren Sie die Batch-Datei und bearbeiten die Pfade in den Batch-Dateien.

7.1.2. Bibliotheken

Die *MasterSim*-Bibliothek und die Simulationsprojekte hängen von folgenden Bibliotheken ab, wobei abgesehen von Qt alle im Quelltext-Repository enthalten sind. Somit ist es nicht notwendig, irgendwelche Bibliotheken separat zu installieren:

MasterSimulator und **MasterSimulatorUI**

- IBK-Bibliothek (von IBK, TU Dresden, Germany)
- IBKMK-Bibliothek (von IBK, TU Dresden, Germany), Untergruppe der Mathe-Kernel-Bibliothek der IBK
- TiCPP-Bibliothek (eine angepasste Version vom IBK, TU Dresden, Germany)
- Minizip and zlib zum Entpacken von FMUs

nur **MasterSimulatorUI**

- Qt 5 Library
- BlockMod library (von IBK, TU Dresden, Germany, gehostet auf github)

Die frei verfügbare Version der Bibliotheken (mit Ausnahme von Qt) sind im Unterverzeichnis `third-party` abgelegt.

Die Bibliotheken im `third-party` Unterverzeichnis sind möglicherweise veraltet. Nutzen Sie bitte nur den Quelltext im `externals`-Unterverzeichnis des Repositories.

7.2. Entwicklungsumgebungen und Projekt-/Sitzungsdateien

7.2.1. Qt Creator

Die Entwicklung mit Qt Creator wird unterstützt (und explizit empfohlen) und Projektdateien stehen zur Verfügung. Individuelle Projektdateien befinden sich in Unterverzeichnissen:

```
<library/app>/projects/Qt/<library/app>.pro
```

Ausführbare Programme werden abgelegt in:

```
bin/debug       - Ausgabepfad für Entwicklungen mit dem Qt Creator
bin/release     - standardisierter Ausgabepfad für Erstellung mit cmake
```

7.2.2. Visual Studio

CMake-erstellte VC-Projektdateien

Die einfachste Variante, die immer funktionieren sollte, ist die CMake-erzeugte VC-Projektdatei und *Solution* zu verwenden.

Basisschritte: Öffnen Sie ein Konsolenfenster (VC Commandozeilen Fenster) und verwenden Sie dann CMake mit dem Visual Studio Erstellungsdatei-Generator.

Sie können die `build.bat`- oder `built_x64.bat`-Dateien für diesen Zweck wiederverwenden. Öffnen Sie ein Befehlszeilenfenster und wechseln in das Verzeichnis `build/cmake`.

1. Starten Sie entweder `build.bat` oder `build_x64.bat` und drücken Sie Ctrl+C, wenn die Erstellung startet (dann sind alle Umgebungsvariablen und Pfade bereits gesetzt).
2. Verlassen Sie das Unterverzeichnis und erstellen Sie ein neues Unterverzeichnis vc:

   ```
   > mkdir vc
   > cd vc
   ```

3. Öffnen Sie `cmake-gui`, geben Sie das Elternverzeichnis als Quellverzeichnis an und wählen Sie einen Visual Studio Generator.

```
> cmake-gui ..
```

Die so erstellte `*.sln` Datei kann man dann einfach in VC öffnen.

7.3. Weitere Entwickler-Informationen

Alle weiteren Entwicklerinformationen werden der Einfachheit halber nur in der englischen Dokumentation beschrieben.