MasterSim Benutzerhandbuch

Andreas Nicolai <andreas.nicolai@tu-dresden.de>

Version 0.8.2, Nov 2019

Inhaltsverzeichnis

1. Einführung und Grundbegriffe

MasterSim ist ein Co-Simulations-Masterprogramm, dass eine 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, unter der fmi-standard.org-Web-Seite.

Grundsätzlich verbindet *MasterSim* verschiedene Simulationsmodelle und tauscht Daten zwischen Simulation-Slaves während der Laufzeit aus. Die folgende Grafik illustriert die Programmfunktion und ihren elementaren Ablauf.

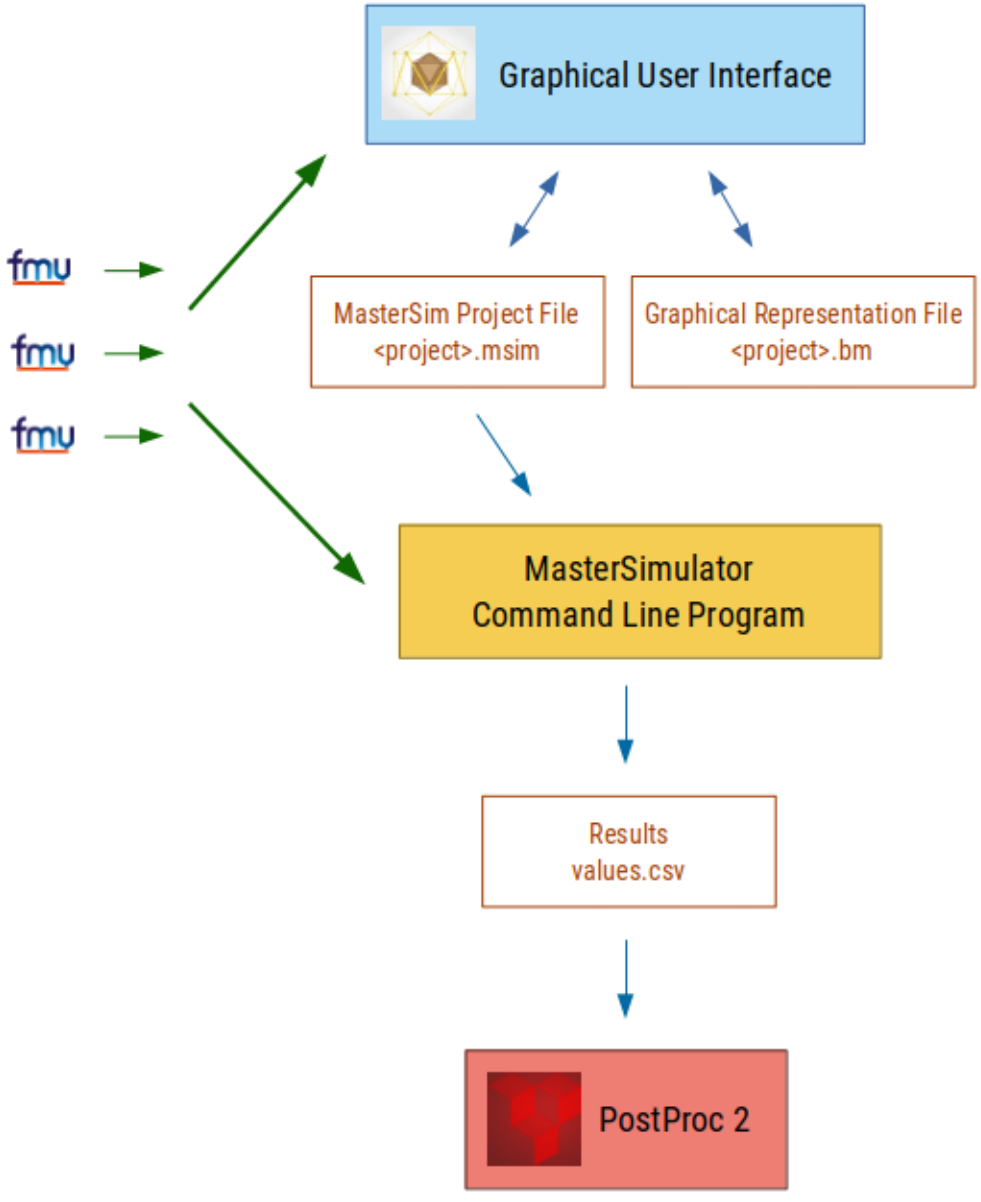

Figure 1. Diagramm über den Datenfluss und die Dateienanwendung

1.1. Die Teile des Programms

MasterSim besteht aus zwei Teilen:

1. einer grafischen Benutzeroberfläche (GUI) und
2. dem Simulationsprogramm *MasterSimulator* für die Befehlszeilen

Das GUI 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. Letzteres ist optional und nicht notwendig für die Simulation. Die Simulation wird durch das Befehlszeilen-Programm *MasterSimulator* ausgeführt, welches Projekt-Dateien liest, ausgewiesene FMUs importiert und die Simulation umsetzt. Die erzeugten Ergebnisse, sowohl von *MasterSimulator* selbst als auch diejenigen möglicher anderer Slaves werden dann von allen weiterverarbeitenden Werkzeugen genutzt (z. B. PostProc 2), um die Ergebnisse zu visualisieren und zu analysieren.

Die Trennung zwischen der Benutzeroberfläche und dem aktuellen Simulator macht es sehr einfach, *MasterSim* in einer geskripteten Umgebung oder für eine systematische Variantenuntersuchung zu nutzen, wie sie weiter unten im Abschnitt Workflows 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 abbauende/aufbauende Status

Es werden keine asynchronen FMU-Varianten unterstützt.

Typischerweise ist *MasterSim* für Linux und MacOS nur als 64-Bit-Anwendung verfügbar. Für Windows ist *MasterSim* auch als 32-Bit und 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. Konstruktionskriterien/ Schwerpunkte

- Betriebssystemübergreifend: Windows, MacOS, Linux
- keine Abhängigkeit von extern installierten Datenbanken, alle Quellen sind in der Version enthalten, insbesondere keine Abhängigkeit von FMI-unterstützenden Datenbanken (Ausnahme: Standard C++, und die Qt-5-Datenbanken); das Ausgliedern und Erstellen von Quell-Codes sollte leicht sein (ebenso das Packen für andere Plattformen)

- vollständige, in die *MasterSim*-Datenbank eingebettete Master-Funktionalität, nutzbar über das Kommandozeilen-Werkzeug und das Programm GUI
- eingebettete Nachrichtenverarbeitung, um GUI-Einbindung und Log-Datei-Support zu unterstützen, keine direkten `printf()`- oder `std::cout´ -Ausführungen
- unterstützt FMU-Fehlerbehebung: kann das Entpacken deaktivieren (ständig: dll/so/dylib-Dateien), der Quellzugriff erlaubt die Fehlersuche während des Ladens von gemeinsam genutzten Datenbanken und angefügten Fehlersuchprogrammen
- High-Level- C++-Code (lesbar und wartungsfreundlich)

TODO: readable Korrektur im Original

- enthält Instrumente, um Zähler und Timer für den Leistungsvergleich von Master-Algorithmen und FMU wiederherzustellen
- der Code ist angepasst für die Fehleranalyse des Master-Algorhitmus - alle Variablen der typspezifischen Datenfelder im *easy analysis in debugger*

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*

Es gibt eine besondere Funktion in *MasterSim*, welche hilfreich ist, um FMUs zu benutzen, die ihre eigenen Ausgabedaten verfassen. Um ein beschreibbares, *slave-spezifisches* Ausgabeverzeichnis für jeden Slave zur Verfügung zu stellen, setzt *MasterSim* den Parameter `ResultsRootDir` (normalerweise mit dem Referenzwert 42) für jeden Slave in diesem Verzeichnis. So lange ein Slave einen solchen Parameter definiert, besitzt der FMU-Code ein verlässliches, gültiges Verzeichnis zum Hineinschreiben seiner Daten. Siehe auch unter Verzeichnis *Slaves*.

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 gemeinsamen Datenbanken

Slave/Simulator	beschreibt ein Simulationsmodell, dass durch ein FMU realisiert/erstellt wird; dabei können mehrere Slaves durch ein einziges FMU realisiert werden, falls die Fähigkeit **canBeInstantiatedOnlyOncePerProcess** richtig eingestellt ist
Master	beschreibt den gesamten Rahmen der Simulationskontrolle, der die generelle Verwaltungsarbeit übernimmt
Simulationsszenario	definiert eine Reihe von Slaves und deren Verbindungen (Datenaustausch) ebenso gut wie andere Eigenschaften, wie z.B. Start- und Endzeit, algorithmische Optionen, Output-Einstellungen; alternativ **system** genannt
Grafiknetzwerk	eine andere Beschreibung für die räumliche Struktur miteinander verbundener Slaves
Masteralgorithmus	beschreibt die Implementierung eines mathematischen Algorithmus, der die gekoppelte Simulation zeitgleich fördert; kann ein Wiederholungs-Programm enthalten
Fehlerkontrolle	meint eine lokale Fehlerüberprüfung (Schritt-für-Schritt), genutzt als schrittweises Korrekturschema
Masterzeit	der Zeitpunkt der Mastersimulation startet mit 0; die Maßeinheit ist nicht strikt definiert (es muss eine gemeinsame Festlegung zwischen FMUs geben, normalerweise in Sekunden; Ausnahme: wenn Datei-Lese-Slaves verwendet werden; siehe Abschnitt CSV-FileReader-Slaves).
Gegenwärtige (Master-)Zeit	die Zeit, in der der Master-Status eingeschaltet ist; ändert sich **einzig** am Ende eines erfolgreichen `doStep()`- oder `restoreState()`- Aufrufs.

1.5. Arbeitsablauf

Wie alle anderen Simulationsmodelle, 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 kleinen oder keinen Modifikationen in der Projektdatei. Manchmal ist ist es sehr komfortabel, die selbe Projektdatei zu nutzen und zu verändern, aber ein anderes Arbeitsverzeichnis (in dem Ausgänge gespeichert werden) zu bestimmen, damit das Resultat verschiedener Varianten verglichen werden kann.
(Siehe das `--working-dir`-Befehlszeilenargument, beschrieben in Abschnitt Arbeits-und Ausgangsverzeichnis).

1.5.1. Ersteinrichtung eines Simulationsszenarios

Das ist die einfache Vorgehensweise:

TODO: Sie oder du?

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. Bestimmen Sie die Simulationsparameter.
6. Führen Sie eine Simulation durch.
7. Prüfen Sie die Ergebnisse.

1.5.2. Nur publizierte FMU-Parameter sind modifiziert

Ein sehr einfacher Fall und, wenn von FMUs unterstützt, durchaus eine bewährte Methode. In *MasterSim* müssen nur die den publizierten Parametern zugewiesenen Werte geändert werden (dies kann auch direkt in der Projekt-Datei getan werden, z. B. mittels Skript) und die Simulation kann wiederholt werden.

1.5.3. FMUs ändern das interne Verhalten, aber nicht die Oberfläche

Dies ist am häufigsten der Fall. Hier bleiben die Namen der Eingangs- und Ausgangsgrößen unverändert. Auch die publizierten Parameter bleiben gleich. Jedoch ändert sich das interne Verhalten der Betriebsart aufgrund der Anpassung des internen Modellverhaltens, wonach das FMU nochmals exportiert wurde. Da *MasterSim* nur noch auf FMUs Bezug nimmt, können FMU-Dateien in solchen Fällen einfach ersetzt und der Simulator ohne weitere Anpassungen gestartet werden.

1.5.4. 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. FMUs ändern die Oberfläche

Wenn eine importierte FMU einen Teil ihrer Oberfläche ändert (z. B. sind Ein- oder Ausgangsgrößen modifiziert), dann wird dies in der Benutzeroberfläche durch Hervorhebung der falschen Verbindungen angezeigt. Wenn nur der Teil einer Größe verändert wurde, editieren Sie am besten die Projekt-Datei und benennen dort die Größenbezeichnung um. Ansonsten einfach die Verbindung entfernen und eine neue schaffen.

Wenn sich der Variablentyp in eine 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.

1.6. Ein Überblick über den Simulations-Algorithmus

MasterSim hat folgende zentrale Bausteine:

- Initialisierung (Lesen der Projekt-Datei, Extraktion von FMUs, Überprüfung …)
- Ausgangsbedingungen
- Korrekturschleife während der Laufzeit
- Master-Algorithmus (d.h. er versucht Maßnahmen zu ergreifen)
- Fehleranalyse
- Ausgangsschreiben nach Festlegung

Diese Bausteine werden nachfolgend näher erläutert.

1.7. Initialisierung

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

Danach wird die Projekt-Datei gelesen und alle diesbezüglichen FMUs werden ausgewählt. Wenn Verweise auf CSV-Dateien auftauchen (siehe Abschnitt CSV-FileReader-Slaves), sind diese Dateien gegliedert und für Kalkulationen eingerichtet.

Auszüge aus dem FMU-Archiv können mit der Befehlszeilen-Option `--skip -unzip` (siehe Abschnitt Modifikation/Fixierung des FMU-Inhalts) übersprungen werden.

TODO: library = Datenbank?

Als erster Schritt der aktuellen Co-Sim-Initialisierung werden alle FMU-Slaves realisiert (dynamische Datenbanken werden geladen und Symbole importiert, danach wird `fmiInstantiateSlave()` oder `fmi2Instantiate()` aufgerufen für jeweils FMI 1.0- und FMI 2.0-Slaves). Es folgt eine Sammlung aller Austauschvariablen und das Erstellen einer variablen Kartierung.

Jeder während der Initialisierung aufgedeckte Fehler führt zu einem Abbruch des Simulators.

1.7.1. Ausgangsbedingungen

Die erste Aufgabe des Simulators ist es, für alle Slaves konsistente Anfangswerte zu schaffen. Das ist eine bereits nicht unbedeutende Aufgabe und nicht in allen Fällen ist der Erfolg garantiert. Der einzige Vorgang, für den FMI-1- und FMI-2-Slaves zum Einsatz kommen können, ist der, schrittweise die Eingangs- und Ausgangsgrößen in allen Slaves zu erhalten und zu setzen, in wiederholender Weise, bis keine Änderungen mehr beobachtet werden.

TODO: sinnvolle Übersetzung "loop over" ?

Der Algorithmus in *MasterSim* ist:

```
Lassen Sie alle Slaves folgende Schritte durchlaufen:
  - rufen Sie setupExperiment() für den aktuellen Slave auf
  - setzen Sie alle Variablen der Kausalitäten EINGANG oder PARAMETER auf ihre
normalen Werte, wie sie in bei modelDescription.xml gegeben sind
  - setzen Sie alle Parameter auf den in der Projektdatei spezialisierten Wert (falls
Werte zugewiesen worden)

für FMI 2: befehlen Sie allen Slaves: enterInitializationMode()

ein Zyklus  mit drei Wiederholungen:
  lassen Sie alle Slaves folgende Schritte durchlaufen:
    nehmen Sie alle Ausgänge der aktuellen Slaves und speichern Sie sie in der
umfassenden Variablen-Abbildung
  lassen Sie alle SLaves folgende Schritte durchlaufen:
    setzen Sie für alle Eingangsvariablen Werte aus der umfassenden Variablen-
Abbildung ein

für FMI 2: befehlen Sie allen Slaves: exitInitializationMode()
```

Beachten Sie: Der anfängliche Berechnungsalgorithmus ist derzeit ein Gauss-Jacobi-Algorithmus und als solcher nicht übermäßig stabil oder effizient.

TODO: Fehler im Original: iterations

Wenn Sie mehr als 3 Slaves in einer Sequenz mit direkter Zufuhr von variablen Ein- zu Ausgängen verbunden haben, z. B. wenn die Ausgänge den Eingängen via Algebraischer Verbindungen zugeordnet sind, werden die 3 Wiederholungen des Gauss-Jacobi-Algorithmus eventuell nicht genügen, um alle Slaves korrekt zu initialisieren.

Dennoch, der Anteil an einer uneindeutigen Angabe im FMI-Standard, wird von Co-Simulations-Slaves nicht eingefordert, um deren Ausgangsstatus zu aktualisieren, wann immer sich die Zufuhr ändert. 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: **without call** zu `doStep()` und **with a call** zu `doStep()`.

MasterSim zieht es vor, die Funktionalität von FMI 1.0 zu übernehmen (d. h. keine schrittweise Wiederholung), nur um Ein- und Ausgänge zu synchronisieren, unter der Voraussetzung, dass die Ausgänge sich nicht ändern (für die meisten FMUs sowieso), wenn die Eingänge auf andere Werte eingestellt sind. Unter dieser Bedingung sind 3 Wiederholungen immer ausreichend.

TODO: Übersetzen von input/output sinnvoll?; communication=Datenübertragung?

1.7.2. Start- und Endzeit der Simulation

MasterSim betrachtet die Simulationszeit 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 ist auf 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 Simultionszeit *3600* laufen, welche dann als *Datenübertragungsintervall der Endzeit* 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 Mitteilungsintervall 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 zum FMU auch per `setupExperiment()`-Aufruf (das Argument `stopTimeDefined` ist durch *MasterSim* immer auf `fmiTrue` gesetzt) überführt.

TODO: Übersetzung Solver sinnvoll?

1.8. Die Umstellung der Zeitschritte

Irgendwann ist die Simulation abgeschlossen, der Solver gibt den Zyklus der umgestellten Zeitschritte an. Wenn die Umstellung der Zeitschritte über die Markierung **adjustStepSize** (siehe Simulator settings) gesperrt ist, wird die Wiederholung des Inhalts nur einmal ausgeführt. Für FMI-1.0-Slaves oder FMI-2.0-Slaves ohne die Fähigkeit zur Speicherung/Wiederherstellung des Slave-Status, ist die Wiederholung ebenfalls nicht möglich (tatsächlich löst das Abfragen eines Wiederholungs-Algorithmus für diese Slaves einen Fehler während der Initialisierung aus).

Innerhalb des Zyklus versucht der ausgewählte *Master-Algorithmus* einen einzelnen Schritt mit der gegenwärtig vorgeschlagenen Zeitschrittgröße (für eine konstante Schrittmethode, wird der **hStart**-Parameter genutzt)zu machen. Der *Master-Algorithmus* involviert eventuell eine wiederholende Auswertung der Slaves (siehe unten).

Für einen sich wiederholenden Master-Algorithmus ist es vielleicht möglich, dass die Methode nicht innerhalb des gegebenen Limits konvergiert (siehe Parameter **maxIterations**).

TODO: time step übersetzen?

1.8.1. Zeitschritt-Verringerung, wenn der Algorithmus nicht konvergiert

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

```
h_new = h/5
```

Der Faktor 5 ist so ausgewählt, dass die Zeitschrittgröße schnell reduziert werden kann. Zum Beispiel, wenn eine Unterbrechung auftritt (z. B. ausgelöst durch eine schrittweise Änderung diskreter Signale) muss der Simulator die Zeitschritte schnell auf einen niedrigen Wert reduzieren, um die Schrittänderung zu überspringen.

Die Schrittgröße ist dann vergleichbar mit den Schritten des niedrigeren Datenübertragungs-Limits (Parameter **hMin**). Dies ist notwendig, um zu verhindern, dass die Simulation in extrem langsamen Zeitschritten stecken bleibt. Wenn die Schrittgröße unter den Wert von **hMin** reduziert würde, würde bei der Simulation die Fehlermeldung **wird abgebrochen** auftreten.

In manchen Fällen kann die Interaktion zwischen zwei Slaves das Konvergieren jedweder Master-Algorithmen verhindern (sogar den Newton-Algorithmus). Dennoch kann in diesen Fällen der verbleibende Fehler unerheblich sein und die Simulation kann in kleinen Schritten langsam über die problematische Zeit hinweggehen und danach die Schritte vergrößern. In diesen Fällen können Sie den Parameter **hFallBackLimit** festlegen, welcher größer sein muss als **hMin**. Wenn *h* auf einen Wert unter diese *zulässige* Mitteilungs-Schrittgröße reduziert ist, wird der Master-Algorithmus erfolgreich zurückkehren, nachdem alle Wiederholungen ausgeführt worden sind. Demnach wird der Schritt als *sich angenähert* behandelt und die Simulation geht zum nächsten Intervall weiter.

TODO: Fehler Original s.o. tiptoe; acceptable

Die oben angeführte Publikation illustriert das Verhalten der Simulation beim Benutzen der Parameter.

1.8.2. Fehlerkontrolle und Zeitschritt-Regulierung

Wenn eine Fehlertestmethode (**ErrorControlMode**) festgelegt ist, folgt einem konvergierendem Schritt eine lokale Fehlersuche. Derzeit basiert diese Fehlerprüfung auf der Schritt-Verdopplungs-Technik und kann als solche nur eingesetzt werden, wenn die Slaves FMI-2.0-Setzung/-Erhaltung der Statusfunktion unterstützen.

Grundsätzlich läuft der Test folgendermaßen ab:

```
- Setzten Sie den Slave-Status zurück, um den Lauf des Kommunikationsintervalls zu
starten.
- Nehmen Sie zwei Schritte (mit dem vollen Master-Algorithmus pro Schritt)
- Berechnen Sie Fehlerkriterien 1 und 2
- Setzen Sie den Status zurück zum Status nach dem ersten Master-Algorithmus
```

Also, der Fehlertest benötigt zwei weitere Durchgänge des *Master-Algorithmus* per Datenübertragung. Für wiederholende Master-Algorithmen oder den Newton-Algorithmus kann der Aufwand für den Fehlertest erheblich sein.

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

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

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

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

Wenn Sie einen Fehlersuche-Algorithmus in *MasterSim* benutzen, sollten Sie ein Zeitschrittlimit für den Rückzug setzen. Andernfalls könnte *MasterSim* versuchen, die Dynamiken der Schrittänderung zu beseitigen, indem es die Zeitschritte auf extrem niedrige Werte justiert.

1.9. Master-Algorithmen

Ein *Master-Algorithmus* ist grundsätzlich die mathematische Prozedur, um die gekoppelte Simulation einen Schritt voran zu bringen. Solch ein Co-Simulations-Master-Algorithmus verfügt über ein charakteristisches Set an Regeln, um Werte von einem FMU abzurufen, wann und wie diese Werte an andere FMUs überführt werden und die Kriterien des Konvergierens von Wiederholungen.

MasterSim führt mehrere Standard-Algorithmen durch. Eine detaillierte Diskussion über die 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 (in german)

1.9.1. Gauss-Jacobi

Basis-Algorithmus:

```
alle Slaves sollen folgende Schritte durchlaufen:
wiederholen aller Ausgangswerte


alle Slaves sollen folgende Schritte durchlaufen:
  setzen aller Eingangswerte
  dem Slave sagen. einen Schritt zu tun
```

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

Anstatt einen Schritt zur Datenübertragung für 10 Sekunden zu nutzen und Gauss-Jacobi für 2 Wiederholungen zu nutzen, ist es effizienter Wiederholungen 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, jedoch läuft die Simulation akkurater ab (und stabiler) mit dem 5-sekündigem Datenübertragungsintervall.

1.9.2. Gauss-Seidel

Basis-Algorithmus:

```
Wiederholungsschleife:
  Durchlaufstationen aller Slaves:
    setzen der Eingangswerte für Slaves aus der globalen Werteliste
    den Slave veranlassen, einen Schritt zu tun
    wiederherstellen des Ausgangs aus dem gegenwärtigen Slave
    Erneuern der globalen Variablenliste
  eine konvergierende Prüfung durchführen
```

Zyklen

MasterSim enthält eine Funktion, die die Rechenleistung reduziert, wenn viele FMUs involviert sind und nicht alle direkt miteinander verbunden sind. Die folgende Figur zeigt ein Simulationsszenario, in dem die Berechnung in Stufen ausgeführt werden kann.

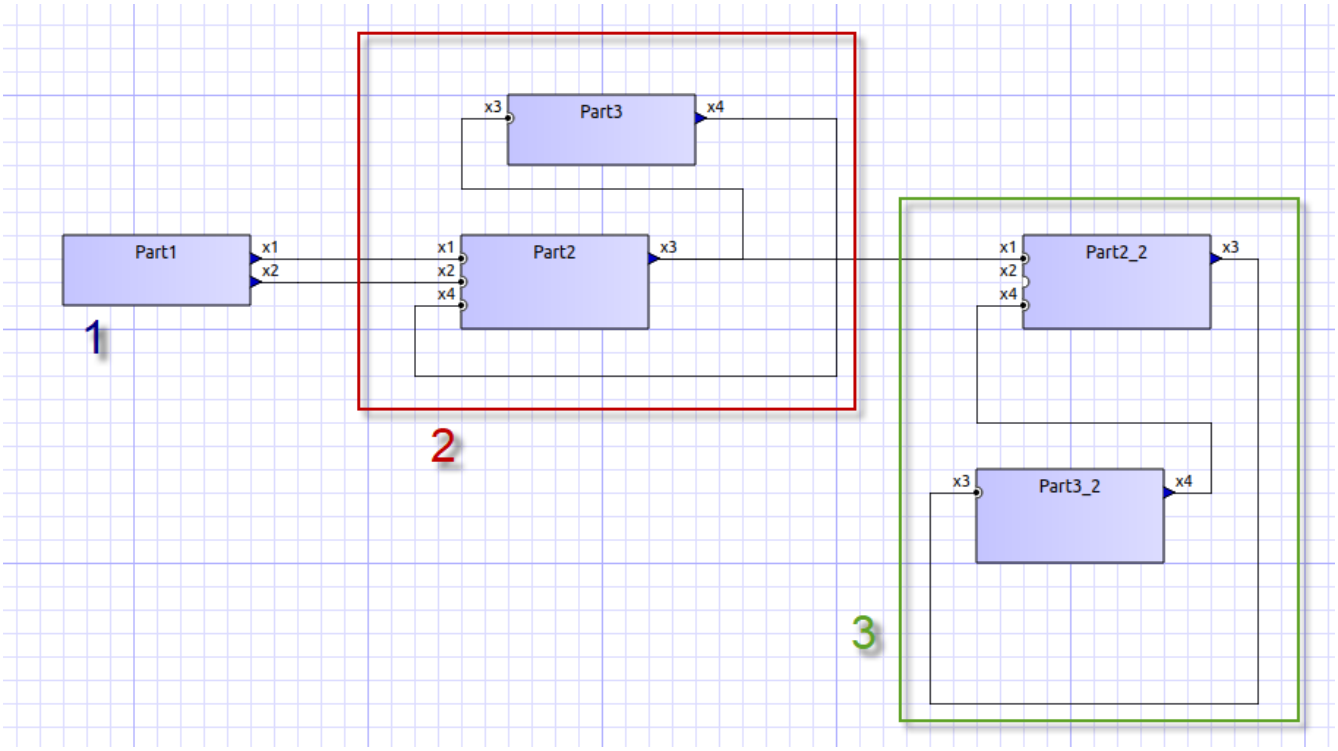

Figure 2. Zyklen in sich wiederholenden Algorithmen

(1) Dieses FMU erzeugt nur Ausgänge und kann nur ein einziges Mal im Gauss-Seidel-Algorithmus untersucht werden.

(2) Diese zwei FMUs tauschen Werte aus, sie sind in einem *Zyklus*. Wenn der Gauss-Seidel-Algorithmus mit aktivierter Wiederholung ausgeführt wird, brauchen nur diese beiden FMUs aktualisiert werden und sie müssen Werte austauschen, denn sie erfordern keinen Eingang von anderen FMUs (außer für das Erste, dessen Ausgangsvariablen sind bereits bekannt).

(3) Die letzten beiden FMUs sind auch in einem Zyklus gekoppelt, aber nur miteinander. Sie werden in der letzten Phase/Zyklus wiederholt. Bis die Ergebnisse der anderen drei FMUs berechnet worden und bekannt sind, müssen wieder nur drei FMUs im Zyklus sein.

TODO: Korrektur Original: relation

Die Anzahl an FMUs in einem Zyklus zu begrenzen, reduziert nicht nur den gesamten Aufwand, sondern berücksichtigt auch die Starre der Kopplung. In einem Zyklus können 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 nicht linearen Verbindung gekoppelt sein oder sensibler auf Änderungen der Eingangswerte reagieren (= starre Kopplung) und zehn oder mehr Wiederholungen können benötigt werden. Dieses, das Vereinzeln der Zyklen, kann die Rechenleistung bei der Gauss-Seidel signifikant reduzieren.

Jedes FMU kann einem Zyklus zugewiesen sein, welcher nummeriert ist (Beginn bei 0) und in der Reihenfolge der Zyklusnummer ausgeführt wird (siehe Simulatordefinition im Abschnitt Simulator-/Slave- Definitionen).

1.9.3. Newton

Basis-Algorithmus:

```
Wiederholungsschleife:
  Berechnen Sie in der ersten Wiederholung die Newtonmatrix via Angleichung des
Differenzquotienten

  Lassen sie alle Slaves Folgendes durchlaufen:
    Legen Sie alle Eingangswerte fest
    Befehlen Sie dem Slave, einen Schritt zu machen

  Lassen alle Slaves Folgendes durchlaufen:
    Rufen Sie alle Ausgangswerte ab

  Lösen Sie das Gleichungssystem
  Berechnen Sie die Abweichung der Variablen

  Führen Sie einen Konvergenz-Test durch
```

Zyklen werden genauso behandelt wie mit dem Gauss-Seidel.

Für den Fall, dass nur ein einziger FMU innerhalb des Zyklus ist, wird der Newton-Master-Algorithmus dieses FMU nur einmal auswerten und die Ergebnisse als bereits konvergiert behandeln. Natürlich wird in diesem Fall keine Newton-Matrix benötigt und verfasst. Allerdings wird es in dem (seltenen) Fall, dass ein solches FMU seine Eingangswerte mit *seinen eigenen Ausgängen* verbindet, vielleicht zu Problemen führen, bis die potentiell ungültigen FMU-Bedingungen akzeptiert werden.

1.10. Ausgänge schreiben

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

Wenn Sie Ausgänge wirklich nach jedem internen Schritt haben wollen, setzten Sie **hOutputMin** auf 0.

2. Grafische Benutzeroberfläche

MasterSim verfügt über eine einigermaßen komfortable grafische Benutzeroberfläche, um *Simulations-Szenarios* festzulegen und anzupassen. Mit *Simulations-Szenario* meine ich die Definition derjenigen FMUs, die einen Slave (oder Slaves) definieren und realisieren wie man Eingangs-und Ausgangsvariablen verbindet und alle Eigenschaften im Zusammenhang mit Rechenalgorithmen. Grundsätzlich alles, das gebraucht wird, um eine Co-Simulation durchzuführen.

2.1. Startseite

Die Software beginnt mit einer Startseite, welche grundsätzlich 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).

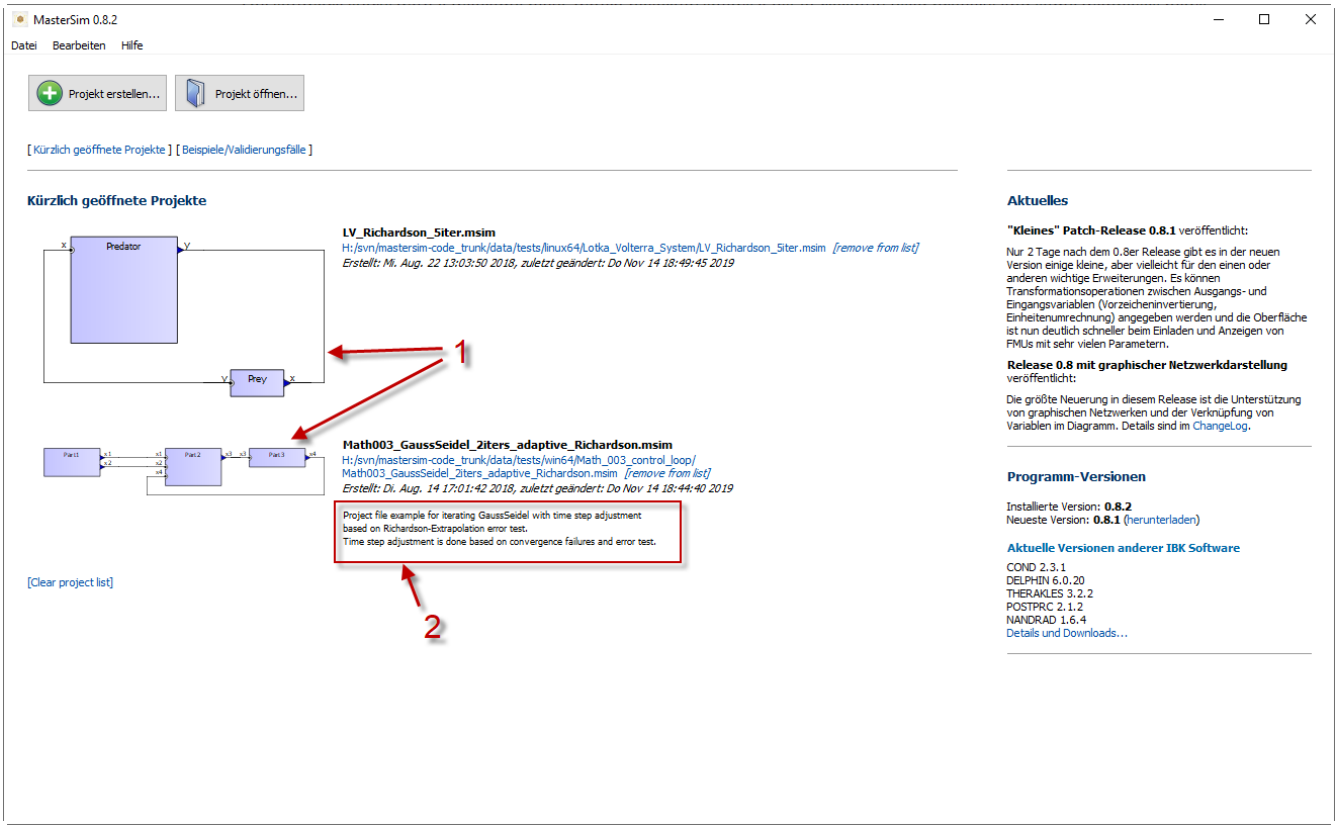

Figure 3. Startseite mit kürzlich verwendeten Projekten und webbasierten Neuigkeiten

(1) Miniaturansicht mit der Vorschau eines Simulations-Szenarios

(2) Kurze Beschreibung des Projekts. Die Beschreibung entstammt der Kommentarzeile der Projektzeilenüberschriften (siehe Project file format).

Die auf der Startseite angezeigten Miniaturansichten werden erzeugt/aktualisiert, wenn das Projekt gesichert wurde. Die Dateien sind innerhalb des Benutzerverzeichnisses platziert:

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

und die Bilddatei ist benannt wie die Projektdatei, erweitert um den Anhang `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 werden im Installationsverzeichnis angegeben, welche gewöhnlich schreibgeschützt sind).

2.2. Symbolleiste und nützliche Tastenkombinationen

So schnell ein Projekt hergestellt/geöffnet ist, wird eine der Projektinhaltsansichten gezeigt und eine Symbolleiste an der linken Seite des Programms angezeigt. Die Zeichen der Symbolleiste haben die folgenden Funktionen (sie werden auch beim Antippen angezeigt, wenn man mit dem Mauszeiger über den Button fährt):

Programm Information	Zeigt die Programminformation
Create new	Erzeugt ein neues Projekt (Kürzel **Ctrl** + **N**)
Open project	Öffnet eine `*.msim`-Projektdatei (Kürzel **Ctrl** + **O**)
Save project	Sichert das aktuelle Projekt (Kürzel **Ctrl** + **S**) (sichert außerdem die Netzwerkpräsentation)
Open PostProc	Öffnet das Nachbearbeitungswerkzeug, angegeben im Einstellungsdialog. Obwohl ich empfelen würde PostProc 2 zu nutzen. Sie können hier jede andere Nachbearbeitungs-Software starten oder sogar ein automatisches Analyseskript. Setzen Sie einfach die entsprechende Kommentarzeile in den bevorzugten Dialog.

FMU Analysis

MasterSim kann 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 ein FMU im Dateiensystem aktualisiert haben und diese Änderungen in der *MasterSim*-Benutzeroberfläche reflektieren wollen (alternativ laden sie das Projekt einfach neu).

TODO: Korr.Orig. s.o. *thos*

Slave definition view

Wechselt zum Slaves definition view. Hier definieren Sie, welche FMUs wichtig sind und weisen Slaves Parameterwerte zu. Außerdem können Sie eine grafische Darstellung des Netzwerkes kreieren.

Connection view

Wechselt zum Connection view. Hier können Sie die Verbindungen zwischen den Slaves verwalten und spezielle Attribute (Transformationen) zwischen den Verbindungen zuweisen.

Simulation settings view

Wechselt zum Simulation settings view. Alle Simulationsparameter und numerischen Algorithmusoptionen sind hier spezialisiert. Ebenso startet die tatsächliche Simulation von dieser Ansicht.

Undo/Redo

Die nächsten zwei Buttons steuern die Funktionen rückgängig oder noch einmal machen der Benutzeroberfläche. Alle im Projekt gemachten Änderungen können zurückgenommen und noch einmal gemacht (Kürzel sind **Ctrl** + **Z** für rückgängig machen und **Ctrl** + **Shift** + **Z** für erneut tun).

Language switch

Die nächsten Buttons öffnen ein Kontextmenü mit einer Sprachauswahl. Sie müssen die Anwendung neu starten, um die neue Sprachwahl zu aktivieren.

Quit

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 Kürzel

Hier ist eine Liste der nützlichen programmweiten Tastaturkürzel:

Table 1. Programmweite Tastenkombinationen

Windows/Linux	MacOS	Command
Ctrl + **N**	⌘ + **N**	erstellt ein neues Projekt
Ctrl + **O**	⌘ + **O**	lädt ein Projekt
Ctrl + **S**	⌘ + **S**	speichert ein Projekt
Ctrl + **Shift** + **S**	⌘ + **Shift** + **S**	speichert das Projekt mit einem neuen Dateinamen
Ctrl + **Z**	⌘ + **Z**	rückgängig machen
Ctrl + **Shift** + **Z**	⌘ + **Shift** + **Z**	erneut versuchen
F2	**F2**	öffnet die Projektdatei im Texteditor
F9	**F9**	startet die Simulation (kann von jeder Ansicht genutzt werden, keine Notwendigkeit zunächst zur Ansicht der Simulationsumgebung zu wechseln!)
	⌘ + **.**	Öffnet den Präferenzen-Dialog

2.3. Die definierte Ansicht von Slaves

Der Eingang der Simulations-Szenarien ist in drei Ansichten aufgeteilt. Das Erstellen einer Simulation startet mit dem Importieren von Slaves. Somit ist die erste Ansicht (und die wichtigste), die für Slaves festgelegte.

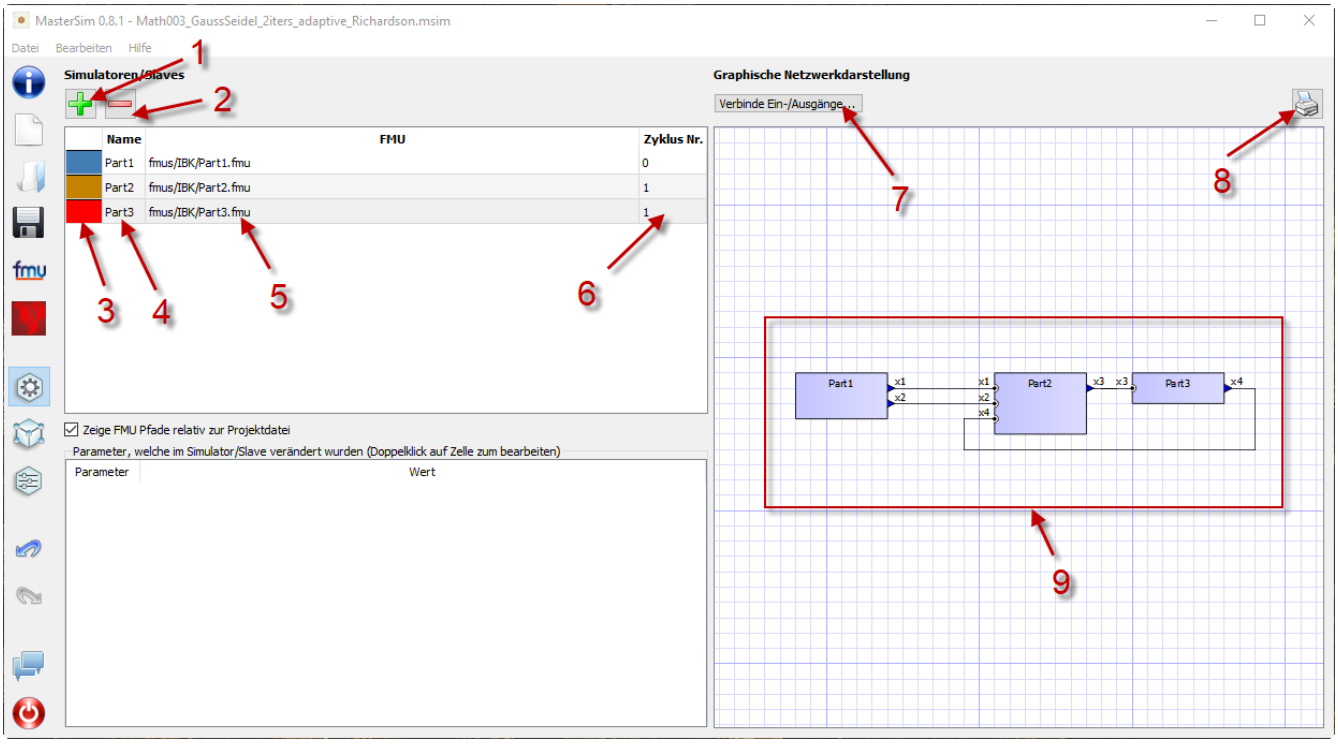

Figure 4. Die festgelegte Slave-Ansicht zeigt eine Liste importierter FMUs, zugewiesene Slave-ID-Namen und eine optionale grafische Darstellung.

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 die gegenwärtig ausgewählten Slaves (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 Basisinamen des FMU-Datei-Pfades 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 *"Show FMU paths relative to project file"*. Außerdem muss der Projektname gespeichert worden sein, bevor relative Pfade angezeigt werden können.

(6) Definiert, in welchem Zyklus das FMU berechnet werden soll (im Standard sind alle Slaves im Zyklus 0 und damit alle als gekoppelt angenommen calculated. Siehe Beschreibung in Master- Algorithmus.

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

(8) Kopiert Netzwerkschemata für den Drucker oder als PDF-Datei.

(9) Dies ist ein grafisches Netzwerkschema - rein optional, aber es hilft, zu verstehen, was Sie tun.

Falls Sie mehrere Blöcke gleichzeitig umgestalten wollen, können sie viele Blöcke mittels **Ctrl** + **Click** am Stück auswählen. Wenn Sie einen der Blöcke verschieben, werden sich die anderen ausgewählten Blöcke ebenso bewegen.

2.3.1. Slaves hinzufügen

Neue Slaves werden zugefügt durch das Auswählen von `fmu`- oder `csv`- oder `tsv`-Dateien. *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 das selbe FMU mehrere Male importieren, es sei denn das FMU hat die markierte Fähigkeit **canBeInstantiatedOnlyOncePerProcess** auf *false* gesetzt. In diesem Fall werden die Slaves unterschiedliche ID-Namen haben, trotzdem sie zur selben FMU-Datei referenziert werden. Parameter und das visuelle Auftreten können für einen Slave der selben FMU unterschiedlich gesetzt sein.

2.3.2. Eigenschaften/Parameterwerte der Slaves

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

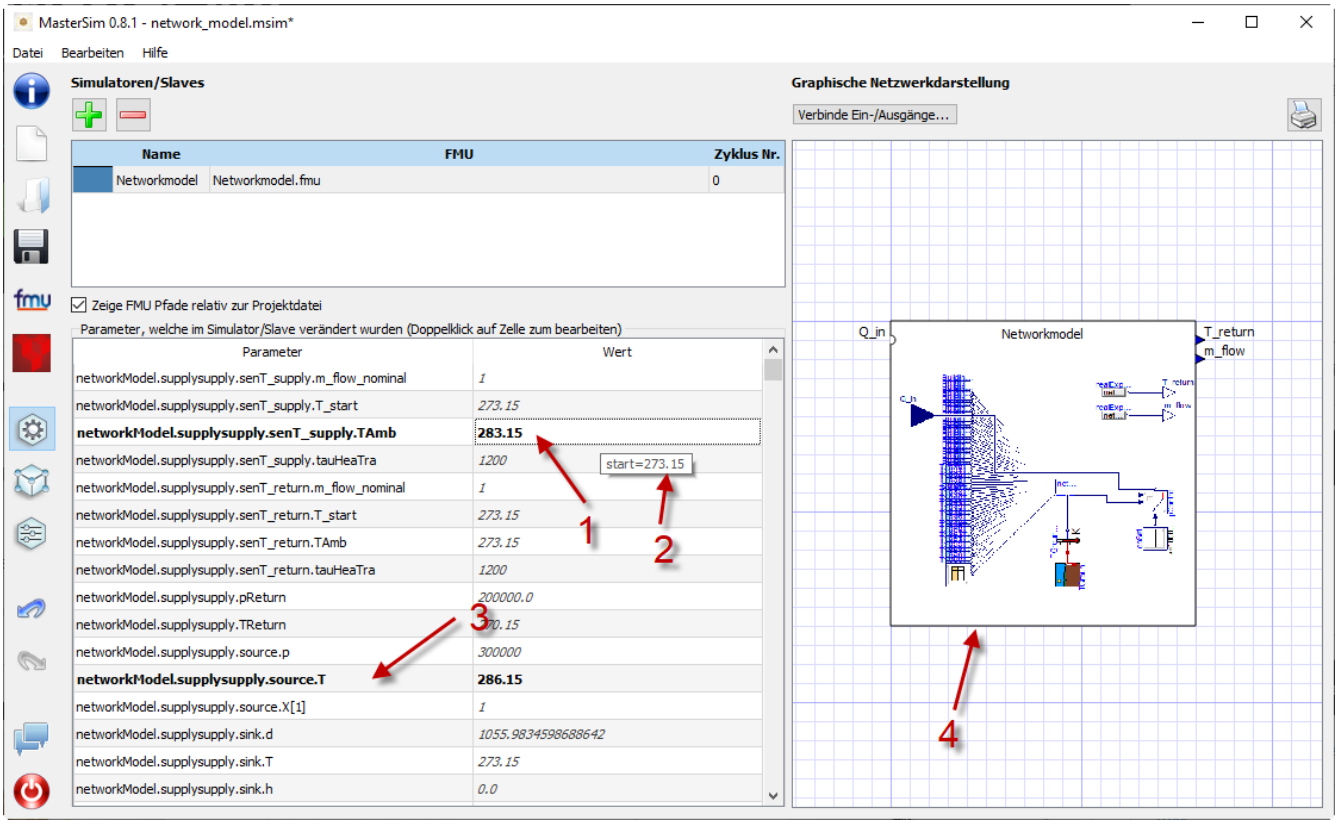

Figure 5. Tabelle mit Slave-spezifischen Parameterwerten

(1) Schwarze und fette Schriftzeichen geben 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 Werkzeughinweis mit den Standardparametern. Dies kann genutzt werden, um den Standardwert zu sehen, für den Fall, dass ein Parameter geändert worden ist.

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

Parameter können durch **Doppeklicken** der Wertezelle editiert werden und einen Wert eingeben. Das Bereinigen des Inhalts der Zelle setzt die Parameter auf ihre standardmäßigen Werte zurück.

2.3.3. 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.

Sie können in die Netzwerkansicht heraus- und hineinzoomen, indem Sie mit den Scroll-Knopf der Maus nutzen. Es wird zu der Stelle gezoomt, an der sich der Mauszeiger befindet.

Das Netzwerk zeigt ***Blöcke*** (angepasst an die Simulatoren/Slaves) und in jedem Block einen weiteren ***Sockel***. Sockel zeigen die Eingangs-/Ausgangsvariablen eines jeden Simuations-Slaves an. Die Blöcke werden in unterschiedlichen Farben angezeigt, die individuellen block states anzeigend.

Verbindungen in der Netzwerkansicht herstellen

Sie können eine neue Verbindung zwischen Ein- und Ausgängen von Slaves herstellen, indem Sie zunächst das Netzwerk in den *connection mode* versetzen durch das Drücken des Buttons **(7)**. Wenn der Verbindungsmodus hergestellt ist, wandelt sich der Zeiger innerhalb des Netzwerkansicht-Fensters zu einem Kreuz. Sie können die Maus dann über einen Ausgangssockel bewegen (Triangel). *Drücken und halten* Sie den Maus-Button und zeigen Sie die Verbindung zu einem *freien* Eingangssockel (leerer Halbkreis). Wenn die Verbindung einmal hergestellt worden ist, wird der Verbindungsmodus wieder ausgeschaltet und die Blöcke und Verbindungen können bewegt werden.

Sie können den *Verbindungsmodus* verlassen, indem Sie den Rechtsklick in der Netzwerkansicht drücken.

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

Blockstatus

Da *MasterSim* nur auf FMUs Bezug nimmt, wird deren eigentlicher Inhalt their actual content (z. B. Anschlusseigenschaften von `modelDescription.xml`)nur ersichtlich, wenn sie importiert worden. Der FMU-Import und der Analyseschritt werden automatisch vorgenommen, wenn ein Projekt geöffnet ist und ein neuer FMU-Slave hinzugefügt wird.

Beim Importieren des FMU versucht die Benutzeroberfläche das FMU-Archiv zu entpacken und dessen Inhalt zu analysieren. Wenn die `modelDescription.xml`-Datei korrekt gelesen werden kann, wird *MasterSim* anbieten den Block-Editor zu öffnen. Innerhalb des Editors können Sie grundlegende Geometrie des Blocks (SLave-Darstellung) und die Gestaltung der Sockel (die Position der Eingangs-/Ausgangsvariablen). Sie können die Aufforderung ignorieren und die visuelle FMU-Darstellung verlassen ohne darstellerische Festlegungen. Grundsätzlich kann ein FMU drei Status haben, die in der UI unterschiedlich visualisiert sind:

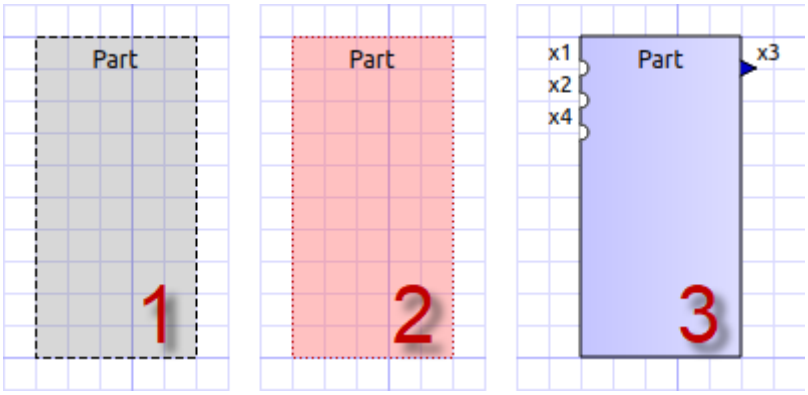

Figure 6. Unterschiedlicher Status von Blöcken und deren visuelles Erscheinen

(1) Die entsprechende `fmu`-Datei existiert nicht oder kann nicht gelesen werden (kein Pack-Archiv, kann nicht extrahiert werden, beinhaltet keine `modelDescription.xml` -Datei oder keine gültige Datei,...viele Dinge können hier schief gehen)

(2) Die Modellbeschreibung wurde für diesen Slave erfolgreich analysiert, aber die Blockdefinition stimmt nicht mit dem Anschluss überein (bis jetzt). Typischerweise hat die Definition eines zugehörigen Blocks noch keine Sockel definiert oder gestaltet, wenn ein FMU zum ersten Mal importiert wird. Es wird dann eine einfache rote Box angezeigt. Sie können diese Box **doppelt anklicken**, um den Block-Editor zu öffnen.

(3) Der Block ist festgelegt worden und die Sockel passen zu diesem und werden durch die Modellbeschreibung angezeigt (Name und Eingangs-/Ausgangstypen).

2.3.4. Block-Editor

Der Block-Editor erlaubt Ihnen, die grundlegende, rechteckige Gestalt Ihres FMU festzulegen und die Sockel zu gestalten. Der Block-Editor wird entweder direkt nach dem Import eines FMU geöffnet oder indem Sie auf einen Block in der Netzwerkansicht **doppelklicken**.

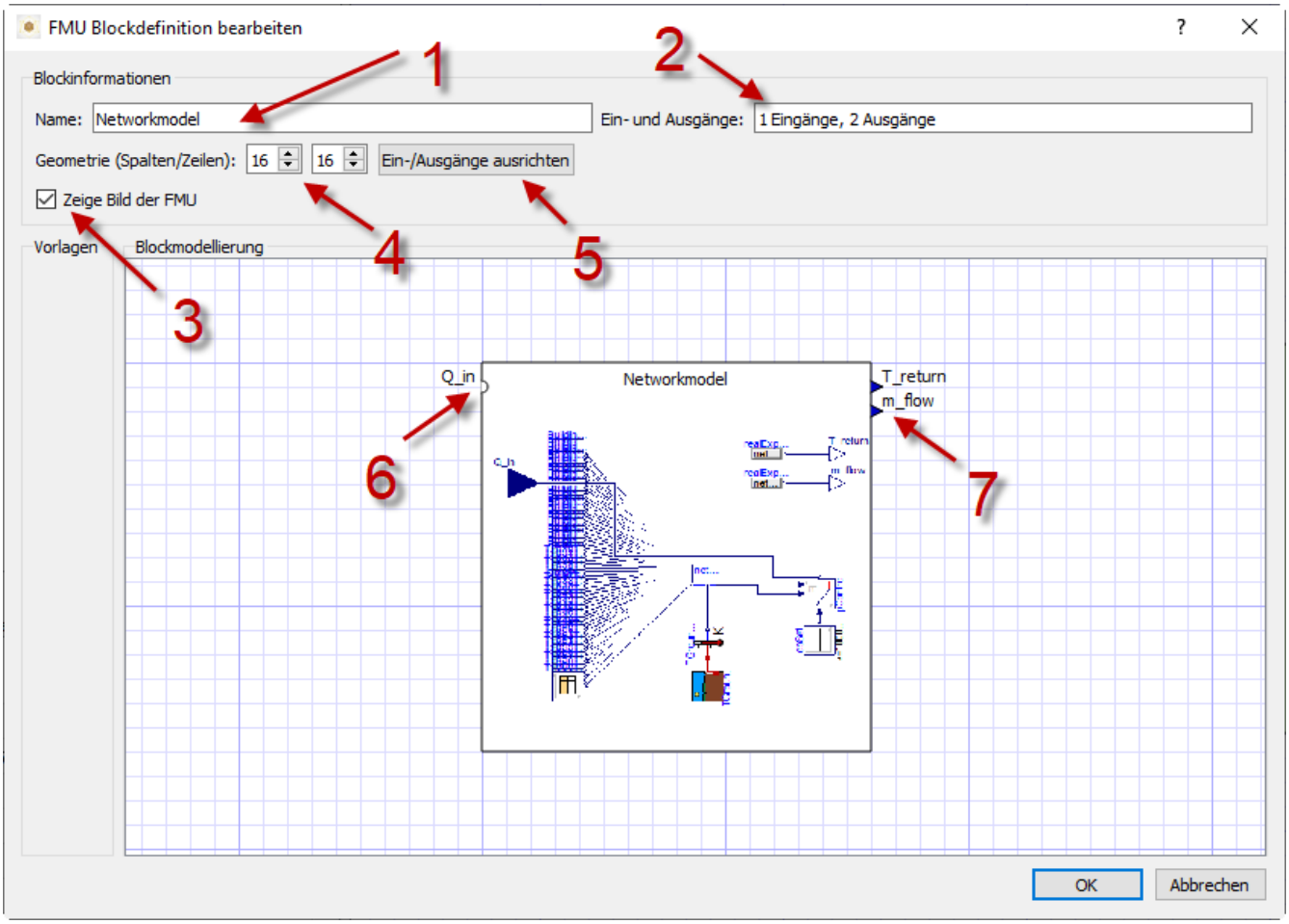

Figure 7. Editor für die Blockgeometrie und die Gestaltung des Sockels

(1) Slave-ID-Name

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

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

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

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

(6) Verweist auf einen Eingangssockel (Eingangsvariable)

(7) Verweist auf einen Ausgangssockel (Ausgangsvariable)

In einer der nächsten Programmversionen wird es möglich sein, das Erscheinungsbild eines Blocks als Vorlage für die zukünftige Nutzung ähnlicher oder gleicher FMUs zu speichern. Gegenwärtig müssen Sie den Block jedes Mal, wenn Sie einen FMU importieren, konfigurieren. Ebenso ist die verbesserte Anwendung und der benutzerdefinierte Sockel-Speicherort noch nicht umgesetzt.

2.4. Ansichten verknüpfen

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

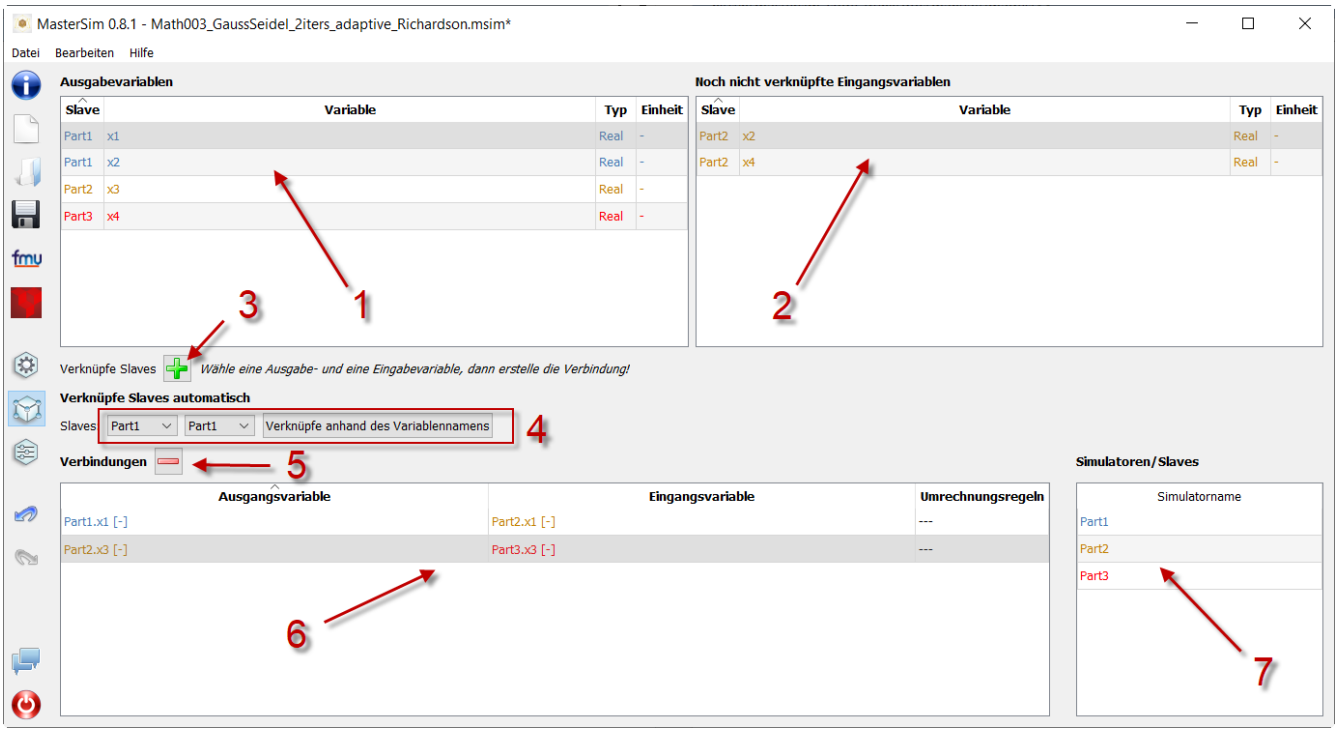

Figure 8. Verknüpfte Ansicht mit aufgezeigten Eingangs- und Ausgangsvariablen für alle Slaves und festgelegten Verbindungen

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

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

(3) Wählen Sie zunächst eine Ausgangsvariable und die Eingangsvariable aus, die mit dem Ausgang verbunden werden soll und drücken Sie dann diesen Button, um eine Verbindung herzustellen.

(4) Hier können Sie viele Verknüpfungen zwischen zwei Slaves verschiedener Namen erstellen (siehe Erklärung unten)

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

(6) Zeigt alle bisher erzeugten Verknüpfungen. Durch einen **Doppelklick** auf die letzte Spalte wird die Umwandlungstätigkeit in Auftrag gegeben.

(7) Eine Tabelle mit allen Slaves und deren Farben (um die Identifikation der Variablen nach Farbe zu unterstützen)

2.4.1. Die Besonderheiten automatischer Verbindungen

Diese Funktion ist sehr hilfreich, wenn FMUs miteinander verbunden sind und Ausgangs- und Eingangsvariablen zweier Slaves den selben Namen haben. Dies ist insbesondere hilfreich, wenn Sie viele Eingangs- und Ausgangsvariablen zwischen zwei Slaves verbinden müssen. Falls Sie ein FMU auf diese Weise erzeugen, können Sie den folgenden Ablauf nutzen:

1. wählen sie in den Kombinationsfeldern die zu verbindenden Slaves aus
2. drücken Sie den Verbindungs-Button

Eine Verbindung wird erstellt, wenn:

- der Variablenname passt
- der Datentyp der Variable passt
- eine Variable einen Kausalitäts-*Eingang* hat und die andere einen Kausalitäts-*Ausgang*

Example 1. Automatische Verknüpfung zweier Slaves

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

Die automatische Verbindung erstellt:

- Slave1.Raum1.Temperatur → Slave2.Raum1.Temperatur
- Slave1.Raum1.Heizkraft → Slave2.Raum1.Heizkraft

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

2.4.2. Eine Verbindung mit einer Umwandlungstätigkeit beauftragen

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 Editieren der Umwandlungsfaktoren und Versätze zu öffnen. Siehe Abschnitt Verbindungsgrafik für eine detaillierte Beschreibung.

2.5. Die Ansicht der Simulationseinstellungen

Alle Einstellungen, die die gegenwärtigen Co-Simulations-Algorithmen kontrollieren, werden hier festgelegt. Eine detaillierte Beschreibung der Einstellungen und ihrer Anwendung findet man im Abschnitt Master-Algorithmus.

Abschnitt Project file reference - Simulator settings beschreibt die zugehörigen Eintragungen in der *MasterSim*-Projekt-Datei.

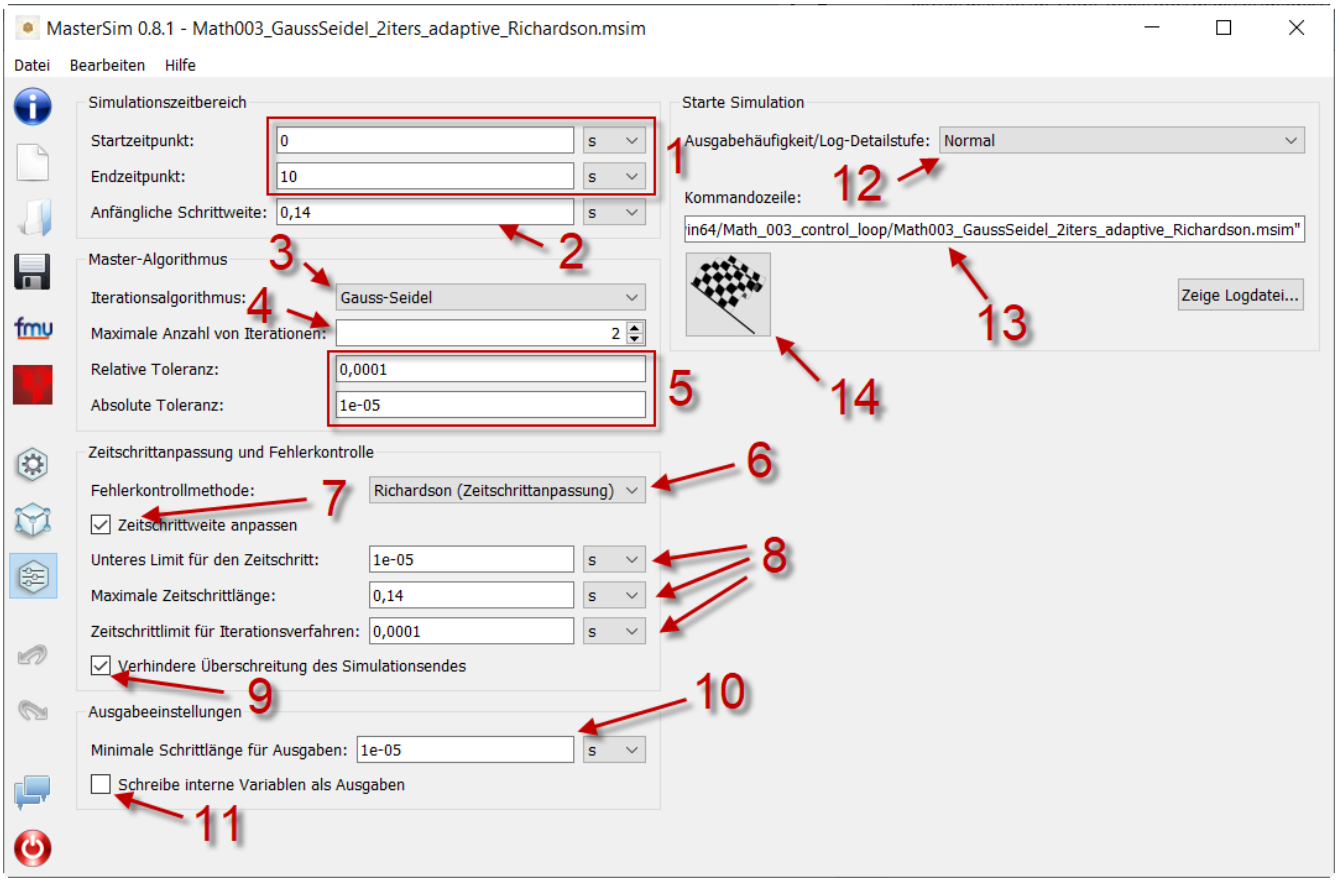

Figure 9. Simulationseinstellungen und die Startansicht der Simulation

TODO: Korr.Orig. s.u. adjustment; convergence

(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 der Datenübertragung genutzt, bis das Ende der Simulationszeit erreicht wurde.

(3) Auswahl des Master-Algorithmus

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

(5) Die relativen und absoluten Toleranzen werden für einen Verbindungstest wiederholender Algorithmen und, wenn freigegeben, für eine lokale Fehlerprüfung und die Zeitschritt-Anpassung genutzt.

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

(7) Wenn getestet, wird *MasterSim* den Zeitschritt anpassen, dies verlangt FMUs, um die Fähigkeit **canHandleVariableCommunicationStepSize** zu unterstützen.

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

(9) Wenn getestet, wird *MasterSim* die Schrittgröße an das letzte Intervall so anpassen, dass es den Endzeitpunkt der Simulation als Ende des letzten Intervalls der Datenübertragung *exakt* wiedergibt, ohne Rücksicht auf das Kennzeichen **(7)** (siehe Diskussion in Abschnitt Time step adjustment).

(10) Legt das minimale Intervall fest, dass durchlaufen werden muss, bevor ein neuer Ausgang geschrieben wird. Es hilft, die Anzahl an Ausgängen im Falle variabler Zeitschritte zu reduzieren, wenn diese Zeitschritte viel kleiner als ein aussagekräftiges Ausgangsraster werden können.

(11) Wenn getestet, schreibt *MasterSim* auch die Werte innerbetrieblicher Variablen zu den Ausgangs-Dateien, ansonsten nur die Variablen von Kausalitäts-*Ausgängen*. Hauptsächlich nützlich für das Entpacken/die FMU-Analyse, oder um interne Werte zu erhalten, die nicht von der FMU selbst für Ausgangs-Dateien geschrieben worden.

(12) Lässt Sie das Level an Vielfältigkeit der Lösungsausgaben des Bedienfeldes kontrollieren (siehe Befehlszeilen-Argumente).

(13) Befehlszeile, die genutzt wird, den Simulator zu betreiben. Kann für die automatische Verarbeitung in ein Shell-Skript oder eine Batch-Datei kopiert werden.

(14) Der große, dicke Start-Button. **Ready, Steady, Go!**

Wenn Sie die Simulation starten, wird ein Konsolenfenster mit einer Fortschritts-/Warnungs-/Fehlermeldung für die laufende Simulation auftauchen. Da einige Simulationen sehr schnell sein können, zeigt sich nach ungefähr 2 Sekunden das Protokollfenster mit dem gegenwärtigen Inhalt des Bildschirmprotokolls.

Beachten Sie, dass die Simulation vielleicht noch im Hintergrund laufen könnte, selbst wenn das Protokollfenster bereits gezeigt wurde. Wenn Sie die Simulation mehrere Male starten, bringen Sie mehrere Simulationsprozesse parallel hervor. Das wäre nur dann eine Verschwendung, wenn die Simulation ins selbe Verzeichnis schreiben und jede andere Datei überschreiben würde.

2.6. Einstellungs-Dialog

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

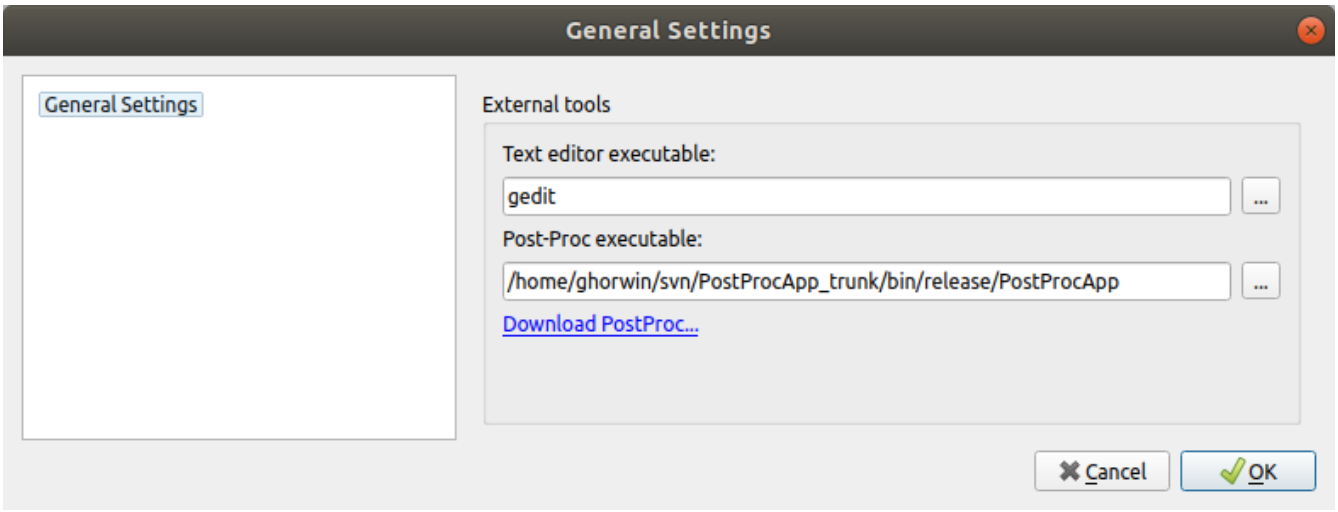

Figure 10. Einstellungsdialog mit Eingangsoptionen für den Texteditor und das ausführbare Nachbearbeitungsprogramm

Wenn Sie eine Textdatei im externen Texteditor bearbeiten und die Datei speichern, bringen Sie das nächste Mal die *MasterSim*-Benutzeroberfläche in den Fokus. Sie wird anregen, das modifizierte Projekt neu zu laden.

3. *MasterSimulator* - Das Befehlszeilen-Programm

Die gegenwärtige Simulation wird mit dem Befehlszeilen-Programm von `MasterSimulator` ausgeführt. Es wird grundsätzlich in folgenden Schritten ausgeführt:

1. Es liest die msim-Projektdatei (die Netzwerkdarstellung wird ignoriert, da sie nur visuelle Informationen enthält).
2. Danach wird ein Arbeitsverzeichnis für den Simulations-Testfall erzeugt (der Pfad kann angepasst werden, siehe: `--working-dir`-Befehlzeilen-Option unten).
3. Die FUMs werden extrahiert (dies kann übersprungen werden, wenn es fertig extrahiert ist, siehe Befehlszeilen-Option `--skip-unzip` unten)
4. Die Simulation läuft, wie in der Projekt-Datei festgelegt, ab

Benutzen von *MasterSim* in einer verschriftlichten Umgebung

Da die *MasterSim*-Projekt-Datei ein einfacher Text ist und der Auflöser durch die Befehlszeile gestartet werden kann, ist es möglich, *MasterSim* in einer verschriftlichten Umgebung und in automatisierten Prozessen zu nutzen (z. B. für optimierte Berechnungen).

3.1. Befehlszeilenargumente

Allgemeine Syntax für das MasterSimulator-Befehlszeilen-Werkzeug:

TODO: Korrektur Original: main page?

```
Syntax: MasterSimulator [flags] [options] <project file>

Markierungen:
  --help            Druckt die Hilfsseite.
  --man-page        Druckt die Hauptseite (Quellversion) des stündlichen Ausgangs.
  --cmd-line        Druckt die Befehlszeile wie sie vom Befehlszeilen-
Syntaxanalysierer verstanden wurde.
  --options-left    Druckt alle Optionen, die dem Befehszeilen-Syntaxanalysierer
unbekannt sind.
  -v, --version     Zeigt die Versionsinformation.
  -x, --close-on-exit  Schließt das Konsolenfenster nach dem Beenden der Simulation.
  -t, --test-init   Startet die Initialisierung und stoppt sie direkt danach.
  --skip-unzip      Entpackt nicht die FMUs und erwartet sie nicht entpackt im
Extraktionsverzeichnis.

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

3.1.1. Arbeits- und Ausgangsverzeichnis

Wenn kein Arbeitsverzeichnis vorgegeben wird, wird der Verzeichnis-Pfad vom Projektdatei-Pfad erzeugt mit beseitigten Erweiterungen. Zum Beispiel:

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

# Ausgänge werden hier vermerkt
/simulations/myScenario/...
```

Im Bereich Struktur und Inhalt des Inhaltsverzeichnis wird der Inhalt des Arbeitsverzeichnisses erklärt.

TODO: sinnvolle Übersetzung für verbosity

3.1.2. Wortarten der Konsole und des Protokolldatei-Ausgangs

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 Detailgenauigkeit wird durch den `--verbosity-level`-Parameter kontrolliert, welcher die Menge der durch den Master erzeugten Fehlinformationen einschränkt.

Das Niveau der verbosity auf 0 ermöglicht so ziemlich jede Ausgabe, das verbosity-Niveau auf 1 ist der Standard mit einem normalen Fortschrittsnachrichtenausgang. Nutzen Sie ein höheres verbosity-Level, wenn das Ziel die Fehlersuche ist, oder um Fehler einzugrenzen.

Die verbosity der Protokolldatei ist während der Initialisierung immer auf 3 gesetzt und während der Simulation zurückgesetzt auf den Standardwert 1 oder den Befehlszeilenwert (um eine Verlangsamung der Simulation aufgrund des exzessiven Ausgabeschreibens zu vermeiden).

3.1.3. Spezielle Optionen bei Windows

Normalerweise bleibt das Konsolenfenster (bei Windows) am Ende der Simulation offen, sofern nicht die `-x`-Befehlszeilen-Markierung angegeben wurde.

3.2. Struktur und Inhalt des Arbeitsverzeichnis

Im Standard wird die folgende Verzeichnisstruktur genutzt. Angenommen es gibt eine Projekt-Datei `simProject.msim`, die für die Beschreibung des Simulationsszenario 2 bereitgestellte FMUs `part1.fmu` und `part2.fmu` nutzt, welche innerhalb des Projekts jeweils als Simulations-Slaves **P1** und **P2** Bezug nehmen. Nehmen wir an, diese Dateien sind in einem Unterverzeichnis verortet:

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

Während des Betriebs des Master-Simulators wird ein Arbeitsverzeichnis erstellt. Standardmäßig entspricht der Dateipfad zu diesem Arbeitsverzeichnis dem Projekt-Dateinamen ohne weitere Anhänge.

```
/sim_projects/pro1/simProject/             - working directory
/sim_projects/pro1/simProject/log/         - log and statistic files
/sim_projects/pro1/simProject/fmus/        - unzipped fmu subdirectories
/sim_projects/pro1/simProject/fmus/part1   - unzipped part1.fmu
/sim_projects/pro1/simProject/fmus/part2   - unzipped part2.fmu
/sim_projects/pro1/simProject/slaves/      - output/working directory for fmu slaves
/sim_projects/pro1/simProject/slaves/P1    - output directory for slave P1
/sim_projects/pro1/simProject/slaves/P2    - output directory for slave P2
/sim_projects/pro1/simProject/results/     - base directory for simulation results
```

Der Basisname des Verzeichnis, hier `simProject` kann geändert werden, indem das Befehlszeilenargument `--working-directory` genutzt wird.

3.2.1. Verzeichnis `log`

TODO: verbosity level?

Dieses Verzeichnis beinhaltet 3 Dateien:

`progress.txt`
: beinhaltet den gesamten Simulations-Fortschritt

`screenlog.txt`
: beinhaltet die Ausgabe von *MasterSim*, geschrieben im Konsolenfenster mit dem erforderlichen Umfang an Worten (während der Initialisierung wird immer eine detaillierte Ausgabe an die Log-Datei geschrieben, selbst wenn auf dem Bildschirm nichts zu schreiben notwendig ist bei einer geringen Wortanzahl)

`summary.txt`
: nachdem die Simulation *erfolgreich* vollendet ist, beinhaltet diese Datei eine Zusammenfassung der relevanten Lösungs- und Algorithmusstatistiken.

Das Format des `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
            ...              ...             ...
```

TODO: Korrektur Original (s.u.): Tab groß?

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 erklärt.

TODO: Original s.o.: welcher Abschnitt?

3.2.2. Verzeichnis `fmus`

Innerhalb dieses Verzeichnisses werden die importierten FMUs extrahiert, jedes in ein Unterverzeichnis mit dem Basisnamen des 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

# _MasterSim_ erzeugt Verzeichnisse
.../fmus/s1
.../fmus/s1_2                                ③
```

① zweite Realisierung des gleichen FMU

② anderes FMU mit gleichem Basisnamen

③ Anhang 2 und 3 ist durch *MasterSim* zugefügt

Grundsätzlich wird jede FMU-Datei nur einmal ausgewählt.

Überspringen des FMU-Extraktions-Schritts

MasterSim unterstützt die Befehlszeilen-Option `--skip-unzip`, welche sehr nützlich ist, um fehlerhafte FMUs im `modelDescription.xml` oder fehlende Ressourcen zu reparieren. Wenn solch ein FMU auftaucht, können Sie MasterSimulater einmal durchlaufen lassen, um die FMUs ins Verzeichnis zu extrahieren, dann die schlechten Dateien im jeweiligen extrahierten 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 durch die Editions- und Test-Lauf-Prozedur schnell wiederholen, bis alles funktioniert.

Siehe außerdem Abschnitt Modifikation/Fixierung des FMU-Inhalts.

3.2.3. Verzeichnis `Slaves`

Oft schreiben nicht-triviale Simulations-Slaves ihre eigenen Ausgabe-Dateien, anstatt die gesamten Ausgabedaten per FMI-Ausgabevariablen zum Master zu verschieben. In Fällen in denen PDEs gelöst werden und tausende Variablen erzeugt werden, könnte dies tatsächlich nicht möglich sein.

Da ein Slave-FMU mehrere Male realisiert werden kann, ist die feste Programmierung eines Ausgabepfads innerhalb des FMU im Allgemeinen keine gute Idee (obgleich gegenwärtig noch immer Praxis). Ausgaben ins gegenwärtige Arbeitsverzeichnis zu schreiben ist ebenso ungeschickt, da das Arbeitsverzeichnis zwischen den Aufrufen der FMUs eventuell durch den Master geändert werden muss - und dies wird am besten vermieden.

Leider unterstützt der FMU-Standard keine Option, ein solches offizielles Ergebnis-Verzeichnis festzulegen. *MasterSim* löst dies, indem es Slave-spezifische Verzeichnispfade in einem Reihenparameter, genannt `ResultsRootDir`, einführt, falls das FMU einen solchen Parameter angibt. Wenn keine Wertemenge in der Projekt-Datei für diesen Parameter festgelegt ist, wird *MasterSim* den für den Slave erzeugten Pfad im Arbeitsverzeichnis fixieren. Das FMU kann auf den von *MasterSim* kreierten Pfad gestützt und beschreibbar sein. Natürlich, wie bei jedem Parameter, können Sie manuell einen Wert für diesen Parameter setzen.

3.3. Rückkehr-Codes des *MasterSimulator* -Programms

MasterSimulator setzt zurück:

0 auf Erfolg

1 auf Fehler (alles von schlechten oder fehlenden FMUs, oder Fehlern während der Berechnung,...), `screenlog.txt` wird Details beinhalten.

3.4. Simulationsausgabe

3.4.1. Slave-Ausgabewerte

MasterSim kreiert zwei Ergebnisdateien innerhalb des `results`-Unterverzeichnis.

`values.csv`
: Anzahl-Ausgabe aller Ausgabevariablen von allen Slaves (egal, ob sie verbunden sind oder nicht).

`strings.csv`
: Werte aller Ausgabevariablen der Typenreihen aller Slaves.

TODO: Korrektur im Original s.u.: Wo beginnt der folgende Satz?

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

TODO: "type" lieber generell mit Modell übersetzen? Korrektur Original s.u. 2 mal "generates" klingt nicht gut

Reihen-Ausgabe-Dateien werden nur erzeugt, wenn die Ausgabe dieser Typen erstellt wird. CSV-Dateien nutzen Tabulatorzeichen als Trennzeichen. In der ersten Spalte steht immer der Zeitpunkt, der Spaltenkopf gibt die Zeiteinheit an.

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 denen der csv-Dateien, die als Datei-lese-Slaves genutzt werden, siehe Abschnitt CSV-FileReader-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.

Synonyme Variablen

Einige FMUs (d.h. solche, die von Modelica Modellen erstellt wurden) können verschiedene (interne) Variblen aufweisen, welche den selben Referenzwert teilen. Das passiert, wenn die symbolische Analyse des Modelica Modells diese Variablen als die selben erkennen konnte. In diesem Fall, schreibt MasterSim keine duplizierte Ausgabevariable (dies wäre eine Verschwendung von Festplattenkapazitäten und Simulationszeit, siehe Ticket #47), sondern erstellt eine Datei `synonymous_variables.txt` mit einer Tabelle synonymer Variablen.

TODO: Korr.Orig. s.u.: Warum hier "fmu" klein, sonst immer in großen Lettern

Die Tabelle wird als einfache Textdatei geschrieben mit durch Tabs 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 es 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
```

3.4.2. Finale Statistik/Zusammenfassung

MasterSim beinhaltet interne Profilierungsfunktionen, welche die Evaluierungszeiten der verschiedenen Teile der Software überwachen. Ebenso werden Ausführungsgrafen für unterschiedliche entscheidende Funktionen gezeigt. Die Statistik wird ins Konsolenfenster kopiert (für das Wortartlevel > 0) und in der Log-Datei `screenlog.txt` im folgenden Format:

```
Solver-Statistiken
-------------------------------------------------------------------------------
Wanduhrzeit                              =   78.044 ms
-------------------------------------------------------------------------------
Ausgabenschreibung                          =   76.767 ms
Master-Algorithmus                          =    0.666 ms         324
Annäherungsfehler                        =                     41
Annäherung an überschrittene Wiederholungsobergrenze =                     41
Fehlermesszeit und gezählte Fehler          =    0.214 ms          85
-------------------------------------------------------------------------------
Teil1                                 doStep =    0.101 ms        1229
                                    getState =    0.070 ms        1116
                                    setState =    0.020 ms         509
Teil2                                 doStep =    0.079 ms        1496
                                    getState =    0.039 ms        1116
                                    setState =    0.024 ms         776
Teil3                                 doStep =    0.071 ms        1496
                                    getState =    0.038 ms        1116
                                    setState =    0.040 ms         776
-------------------------------------------------------------------------------
```

Ebenso wird dieselbe statistische Information in die `summary.txt`-Logsdatei kopiert, in ein eher *maschinenfreundliches* Format (mit Zeitangaben immer in **Sekunden**):

TODO: Übersetzten oder lieber nicht, da es im Programm selbst auch so angezeigt werden wird?

```
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
```

Wall clock time

gesamte Simulationszeit, die nach der Initialisierung aufgebracht wurde. Die Dauer für Entpacken und Laden der geteilten Bibliothek wird nicht einbezogen (`WallClockTime`)

Output writing

Zeit, die für das Schreiben von Ausgabedateien und das Berechnen damit zusammnenhängender Werte gebraucht wurde. (`FrameworkTimeWriteOutputs`)

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

Anzahl der Zeiten, die ein wiederholender Master-Algorithmus scheitert, sich innerhalb der erlaubten Anzahl an Wiederholungen anzunähern oder abweicht. Dies gilt einzig für sich wiederholende Master-Algorithmen. (`ConvergenceFails`)

Convergence iteration limit exceeded

Zeiten, die ein sich wiederholender Master-Algorithmus scheitert, sich innerhalb der erlaubten Anzahl an Wiederholungen anzunähern (sie sollte weniger oder gleich der Anzahl der Annäherungsfehler sein). Dies gilt einzig für sich wiederholende Master-Algorithmen. (`ConvergenceIterLimitExceeded`)

Error test time and failure count

Anzahl der Zeit, in der der Fehlertext scheitert (`ErrorTestFails`) und die insgesamt genutzte Zeit, um die Fehlertests durchzuführen, inklusive der Zeit, um den FMU-Status zurückzusetzen und für Schritte der Neubewertung. (`ErrorTestTime`). Dies gilt nur für Master-Algorithmen mit aktivierter Fehlerkontrolle (Richardson-Varianten).

Die übrigen Linien zeigen Zeiten und Zählungen individuell für jeden Slave. Diese Linien zeigen die genutzte Zeit in den Funktionsaufrufen bis `doStep()`, `getState()` und `setState()` für diesen Slave und die jeweilige Aufrufzählung. Die den Status betreffenden Funktionen werden nur für sich wiederholende Master-Algorithmen genutzt, wenn die FMUs FMI-2.0-Merkmale unterstützen. Bedenken Sie, dass diese Funktionen sowohl vom Master-Algorithmus als auch vom Fehlertest aufgerufen werden (wenn möglich).

Ausgabe-Scheiben und **Master-Algorithmus** sind die beiden Hauptkomponenten des MasterSimulator-Pogramms, sodass ihre addierten Zeiten nahe der Wanduhrzeit liegen sollten.

Die dritte Spalte in der Bildschirm-Datei-Statistik beinhaltet Zähler. Der Zähler für den Master-Algorithmus ist die Anzahl der Zeit, in welcher der Master-Algorithmus einen Schritt macht. Damit ist dies die gesamte Zählung der Schritte. Neuversuche und Wiederholungen *innerhalb* des Master-Algorithmus werden hier nicht beachtet.

Der letzte Abschnitt der Statistik listet Zeiten und Zähler für individuelle FMU-Slaves und die meisten relevanten Funktionen.

Sie sollten diese Profilierungswerte nutzen, um die Simulation abzustimmen und, im Fall einer sehr langsamen Simulation, herausfiltern, welche FMUs die meiste Zeit benötigen. Ebenso helfen sie bei der Identifikation, falls eine der schnellen Funktionen (den Status zu setzen und zu erhalten) zu viel Zeit verbraucht.

4. Das 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 dem selben Namen und der Erweiterung `bm` kann am selben Ort wie die Projekt-Datei angezeigt sein. Diese beinhaltet die grafische Darstellung des Simulations-Szenarios. Da die grafische Netzwerkanzeige *rein optional* ist, kann die `bm`-Datei beliebig weggelassen/ignoriert/gelöscht werden.

Zukünftige Entwicklungsarbeit

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.

Das Format ist ein Klartext-Format (UTF8-kodiert) mit folgendem Inhalt:

Beispiel 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 Tokens jeder Zeile sind durch Leerräume (Tabulatoren oder Leerzeichen) voneinander getrennt. Die Zeilen, die mit einem Rautenzeichen # beginnen, werden kommentiert und ignoriert.

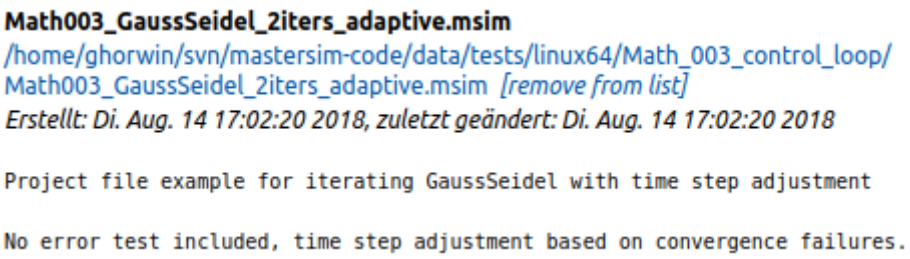

Figure 11. Auf der Startseite angezeigte Projekteigenschaften

Alle Befehlszeilen, bevor die ersten tatsächlichen Eingabezeilen als Kopfzeile erkannt werden. In diesen Zeilen wird vorausgesetzt, dass auf die Stichworte `Created:` und `LastModified:` ein mehr oder weniger bedeutungsvoller (aber nicht standardisierter) Daten- oder Zeitstrang auf der Benutzeroberfläche gezeigt wird. Andere Kopfzeilen werden als Projektbeschreibung betrachtet, welche in der Projektzusammenfassung auf der Startseite der grafischen Benutzeroberfläche gezeigt werden (siehe Figur Auf der Startseite angezeigte Projekteigenschaften).

Nach der Kopfzeile mit der Beschreibung, ist die Ordnung der Einträge/Zeilen willkürlich.

4.1. Einstellungen des Simulators

Unten 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-Algorithmus.

Parameter werden als mit einer Einheit versehene Zahl angegeben, außer die Zähler (Maximum an Wiederholungen) oder Fehlergrenzen (welche sowieso ohne Einheit angegeben werden).

Optionen, die Zeitschritte betreffend:

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

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

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

hMin (*default=1e-5 s*) niedrigere Begrenzung des Zeitschritts im anpassungsfähigen Zeitschritt-Modus, wenn der Zeitschritt unter diese Grenze reduziert wird, stoppt der Master

hFallBackLimit (*default=1e-3 s*) für einen Gauss-Seidel mit Zeitschritt-Festlegung: wenn die Zeitschritte unter diese Grenze fallen, wird der nicht wiederholende Gauss-Seidel genutzt (um diskontinuierliche Ausgänge zu kompensieren), sollte > **hMin**

hStart (*default=10 min*) initialer Zeitschritt, wird als konstante Schrittgröße verwendet, wenn nicht der anpassungsfähige Zeitschritt-Modus in Gebrauch ist.

Falls **hStart** in der Projekt-Datei auf 0 gesetzt ist, wird berechnet, dass es 1/1000 der Simulationsdauer ist, festgelegt durch (**tEnd - tStart**).

TODO: Korr.Orig.s.u. minimum, disabling

hOutputMin

(*default=10 min*) minimale Dauer, die verstreichen muss, bevor der nächste Ausgabewert geschrieben wurde. Wenn Schrittgrößen der Kommunikation größer als **hOutputMin** sind, werden die Ausgänge eventuell übersprungen, aber das reguläre Ausgabeintervall bleibt erhalten.

outputTimeUnit

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

adjustStepSize

(*default=false*) aktiviert/verhindert den anpassungsfähigen Zeitschritt-Modus, wenn der Fehlerkontroll-Modus **ADAPT_STEP** ist, eine fehlerhafte **adjustStepSize** ist ein Fehler.

preventOversteppingOfEndTime

Diese Markierung wird für bestimmte FMUs gebraucht, welche einen Test gegen das Übersteigen 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 Rundungsfehler zu einer sehr kleinen Überschreitung des Endzeitpunktes dazu addiert werden können. Dennoch sollten gute FMUs das Überschreiten angemessen händeln. Zugleich, um einen FMU-Fehler und einen Simulationsabbruch zu vermeiden, kann *MasterSim* die letzte Kommunikationsintervall-Größe so anpassen, dass exakt die Endzeit der Simulation an das FMU übermittelt wird. Wenn diese Markierung deaktiviert ist, muss wahrscheinlich die letzte Intervallschritt-Größe eingestellt werden, selbst wenn die Zeitschritt-Anpassung generell durch die Markierung **adjustStepSize** gesperrt ist.

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

GAUSS_JACOBI	Gauss-Jacobi-Algorithmus (nicht wiederholend)
GAUSS_SEIDEL	Gauss-Seidel-Algorithmus (wiederholend oder nicht wiederholend, abhängig von **it_max_steps**)
NEWTON	Newton-Algorithmus mit einer Annäherung des Differenz-Quotienten an die Jacobi-Matrix

Wiederholungs- und Konvergenzparameter:

maxIterations	(*default=1=disabled*) max. Anzahl an Wiederholungen, wenn == 1 wird keine Widerholung 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).  Noch nicht implementiert. Nicht benutzen!
ADAPT_STEP	ermöglicht implizit den anpassungsfähigen Zeitschritt-Modus und passt Zeitschritte an, wenn das Fehlerlimit überschritten ist.

4.1.1. Fortgeschrittene Konfigurationen

Die folgenden Konfigurationen werden zumeist für den Gebrauch mit einer Gegenproben-Prozedur verwendet.

preventOversteppingOfEndTime	(*default=true*) selbst für FMUs mit konstanten Schritten, wird der letzte Schritt gekürzt, um exakt den Endzeitpunkt zu treffen (das ist für solche FMUs erforderlich, die eine strikte Endzeitüberprüfung haben).
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 sicherlich einige Fähigkeiten durch FMUs unterstützt werden, siehe Abschnitt Master-Algorithmus.

4.2. Simulator-/Slave-Definitionen

Jeder Slave wird festgelegt durch:

```
simulator  <priority> <cycle>  <slave-name> <html-color-code> <path/to/fmu-file>
```

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-Algorithmus).

Die **Priorität** wird genutzt, um die Anordnung der Durchführung in einem Zyklus auszuwählen (für Gauss-Seidel). Trotzdem ist diese Funktion gegenwärtig nicht implementiert und Slaves innerhalb des selben Zyklus werden in der Anordnung bewertet, in der sie festgelegt sind.

Der Slave-/Simulatorname ist eine eindeutige Identifikation des FMU-Falls.

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 Ausgänge). Daher müssen sie keine Zeichen beinhalten, die in Dateisystemnamen nicht erlaubt sind.

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 (Opazität - ff =

vollkommen opak, 0 = vollkommen transparent). 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-Verweis zur eigentlichen FMU-Datei. Der Pfad zur FMU-Datei in Anführungszeichen angefügt werden, wenn der Pfad oder der Dateiname Leerzeichen enthält. Der Pfad kann absolut oder relativ zur `msim`-Projektdatei sein. Einige Slaves können durch die selbe FMU-Datei realisiert sein (wenn das FMU diese Funktion unterstützt). In diesem Fall nehmen einige Simulatorzeilen Bezug auf den gleichen FMU-Dateipfad.

4.2.1. CSV-FileReader-Slaves

Anstatt eines FMUs können Sie ebenso auf eine Datendatei verweisen (Erweiterung mit `tsv` oder `csv`). In diesem Fall wird *MasterSim* FileReader-Slaves realisieren und die Daten in der Datei wird wie ein FMU behandelt, dass nur Ausgänge zur Verfügung stellt, aber keinen Eingang und keine Parameter hat.

Effektiv unterstützt *MasterSim* zwei Varianten von csv-Dateien. In beiden Varianten werden Zahlen immer in der **englischen Nummernschreibweise** verfasst. Der Datei-Parser-Pfad versucht zunächst, den durch Tabulatoren getrennten Variantenwert zu nutzen, indem die ersten beiden Zeilen mit Tabulatorzeichen aufgeteilt werden. Wenn dies mehr als zwei Spalten ergibt und die gleiche Anzahl an Spalten in beiden Linien (die Kopf- und erste Datenzeile), wird eine Tabulator-getrennte csv/tsv-Variante übernommen. Andernfalls wird der Exeltyp der angeführten csv-Variante übernommen.

Tabulator-getrennte Werte

Das Format einer solchen Input-Datei folgt den selben Konventionen wie das Dateienformat, dass von PostProc2 unterstützt wird.

Die Datei startet mit einer einzelnen Zeile (der Kopfzeile), dem Identifizieren von Variablen-Namen und der Einheit im Format, wie:

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

wo `<tab>` das Tabulatorzeichen ist.

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

Beispieldatei:

```
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 der Zeichenfolge der Kopfzeile, ausgenommen der Einheiten (falls angegeben). Im Beispiel oben wird die Datei Ausgangsvariablen mit den Namen **T_lab**, **T_sample** und **TH_lab** anbieten.

Eine Datei mit diesem Format gilt automatisch, wenn eine Tabelle mit solchen Daten aus LibreOffice/Calc/Excel in einen einfachen Text-Editor kopiert und eingefügt wird.

Kommatrennung mit Anführungszeichen

In solchen Dateien ist das Trennungszeichen das , (Komma) und Werte werden durch Anführungszeichen angegeben. Zum Beispiel:

```
"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"
```

4.2.2. Zeitpunkte und Zeiteinheiten

Die Zeitpunkte können in zufällige Intervalle aufgeteilt sein. *MasterSim* erwartet zur Zeit, dass Simulationen in Sekunden ablaufen als Basis-Zeiteinheit. Das bedeutet im Internen, dass Variablen ausgetauscht werden, um einer Simulationszeit in Sekunden zu entsprechen. Wenn eine Eingangsdatei eine andere Einheit für die Zeit festlegt, konvertiert *MasterSim* diese Zeit 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 setzt *MasterSim* gegenwärtig die Zeiteinheit **Sekunden** (s) voraus.

4.2.3. Interpretation der von den FileReader-Slaves bereitgestellten Daten

TODO: Fehlt im Folgenden vielleicht etwas?

Variablen ohne gegebene Einheiten, z. B. das [...] fehlt in der Spaltenüberschrift, werden als unbekannte/undefinierte Einheit ausgewiesen. -.

Die von einem solchen FileReader-Slave exportierten Variablen wurden noch keinem Datentyp zugewiesen. Während der Initialisierung schaut *MasterSim* nach den Verbindungen, die mit FileReader-Slaves gemacht worden und teilt die Datentypen den auf *connected input variable* basierenden Variablen zu.

Während der Simulation, wenn der FileReader-Slave gefragt wird, einen Wert für eine Variable bereitzustellen, gelten die folgenden Regeln:

Boolean-, Integer- und Enumeration-Werte

Für **Boolean**-, **Integer**- und **Enumeration**-Werte wird keine Interpolation vorgenommen. Werte werden konstant zurückgeführt, bis der Wert zur Änderung festgelegt ist. Beispiel:

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

① Der Wert am Ende des Intervalls endet in der Zeit 3

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

Evaluation dieser Werteergebnisse:

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

Somit könnte die Zeile `3 4` aus der Datei weggelassen werden.

Reale Werte

Reale Werte sind linear interpoliert. Für das Datenbeispiel oben würden reale Werte folgendermaßen bewertet:

```
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 Schrittfunktionen mit **Realen** 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 die lineare Neigung zwischen t=2.9999 und 3 erkennen, was aber unerheblich für die Ergebnisse sein könnte.

Natürlicherweise ist die lineare Interpolation für **String**-Parameter nicht möglich, daher werden sie simultan zu **ganzzahligen** Werten behandelt.

Falls Sie einen anpassungsfähigen Schritt-Algorithmus in *MasterSim* verwenden, sollten Sie den maximalen Zeitschritt/die Länge des Datenübertragungsintervalls auf einen Wert festlegen, der kleiner als ihr kleinstes Zeitintervall in der Eingangsdatei ihres FileReader-Slaves ist. Ansonsten könnte *MasterSim* die Zeitschritte als höheren Wert festlegen und Werte/Intervalle überspringen. Hierbei würden Daten fehlen und wahrscheinlich falsche Ergebnisse erzeugt. Zum Beispiel: wenn Sie mit stündlichen Klimadaten arbeiten, wählen Sie 30 Minuten als maximale Länge für ein Datenübertragungsintervall.

4.3. Verbindungsgrafik

Die Verbindungsgrafik legt den Datenaustausch zwischen den Slaves fest. Jede grafisch dargestellte Definitionslinie legt den Datentransfer zwischen einer Eingangs- und einer Ausgangsvariable fest.

Syntax der Definition:

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

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

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

Der Offset- und Skalenfaktor legt den Umwandlungsvorgang zwischen Ausgangsvariablen und dem Wert, der für die Eingangsvariable bestimmt wurde, fest. Wenn eine solche Umwandlung einer Verbindung zugewiesen wird, müssen immer beide Werte dargestellt sein.

Die folgende Umwandlungsgleichung 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 angibt, können Sie die Verbindung wie folgt festlegen:

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

Was resultieren wird als:

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

Auf ähnliche Weise können Sie das Zeichen einer Verbindung umkehren, wenn Sie zum Beispiel Hitze und Massefluss durch Röhren verbinden. Angenommen der Hitzefluss ist positiv auf einer Oberfläche festgelegt und Sie verbinden *SurfaceA.HeatFlow* und *SurfaceB.HeatFlow*, dann wird die Verbindung mit der Zeichenumkehr festgelegt als:

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

4.3.1. FMU-Parameter

Sie können die Parameter der FMUs (oder spezieller, die der individuellen FMU-Slaves/-Exemplare) festlegen, indem Sie das `parameter`-Stichwort benutzen.

Definition der Syntax:

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

Für **boolesche** Parameter müssen Sie `true` festlegen (case-sensitive!) für **true** oder irgend einen anderen Wert (zum Beispiel `false`) für **falsch**.

Für **ganzzahlige** Werte müssen Sie einfach den Wert als Ziffer festlegen.

Werte für **wahre** Parameter werden in der Einheit erwartet, die in der `modelDescription.xml` -Datei für entsprechende Parameter festgelegt worden sind. Die Umwandlung der Einheit wird hier **nicht** unterstützt.

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

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

Leerstellen können eingefügt sein, aber die Rücktasten müssen als `\\` kodiert sein. Zeilenumbrüche werden durch `\n` kodiert wie im folgenden Beispiel:

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

Es wird den String setzen:

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

Wegen der eher leichten String-Kodierung können Sie keinen String festlegen, der mit Leerraum-Zeichen beginnt.

4.4. Blockmodell - das Dateiformat der Netzwerkpräsentation

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

Ein einfaches Netzwerk, wie:

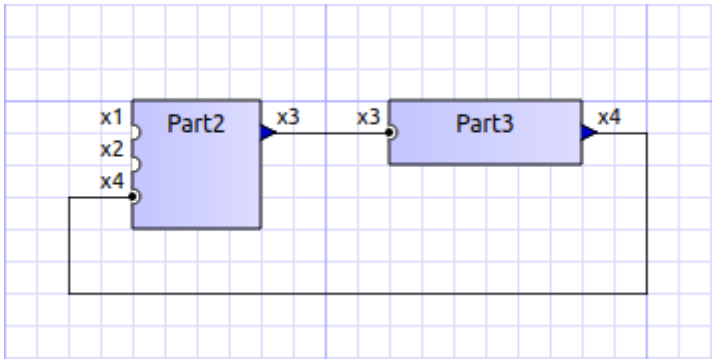

Figure 12. Beispiel für eine einfache grafische Präsentation eines Netzwerks

Es wird in der folgenden BlockMod Netzwerk-Präsentationsdatei festgelegt:

Blockmod Netzwerkpräsentationsdatei

```
<?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 wird automatisch festgelegt, abhängig von der Sockelposition auf dem Block und wird dadurch nicht in der Netzwerk-Präsentationsdatei 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. Konzept zur Testfolge

Die *MasterSim*-Quelle enthält ein Unterverzeichnis `Daten/Tests` mit einer Regressionstestfolge. Diese Tests werden genutzt, um zu prüfen, ob alle Algorithmen wie erwartet arbeiten und ob die Änderungen der Code-Basis versehentlich etwas kaputtmachen.

MasterSim durchläuft ebenso die FMU-Mehrfachprüfung. Die Dateien und Skripte, die mit diesem Test in Verbindung stehen, befinden sich im Unterverzeichnis `cross-check`.

5.1. Regressionstest

5.1.1. Verzeichnisstruktur

```
/data/tests                     - root directory for tests
/data/tests/<platform>/<test>   - base directory for a test suite
```

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

Jede *Testfolge* hat eine Reihe von Unterverzeichnissen:

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

Innerhalb der *Testfolge* können verschiedene ähnliche *Testfälle* mit modifizierten Parametern oder angepassten Problemlösungs-Szenarien gespeichert sein. *Testfälle* werden in der Regel als Testfolgen gruppiert, wenn sie die selben 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 ab nach der Errichtung mit den Aufbauskripts. Ansonsten läuft ein geeignetes Skript:

- `build/cmake/run_tests_win32.bat`

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

5.1.3. Aktualisierung der Referenzergebnisse

Aus dem Inneren ruft das Skript ein Testverzeichnis auf: `update_reference_results.py` mit der Verzeichnisendung als Argument.

Zum Beispiel von innen:

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

ruft auf:

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

welches alle Referenzergebnisse in diesem Verzeichnis aktualisieren wird. Falls Sie die Referenzergebnisse einiger Testfolgen bearbeiten/aktualisieren wollen, führen Sie das folgende Skript aus einem der oberen Verzeichnisse durch, zum Beispiel von: `data/tests/linux64`:

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

5.2. Regeln für Gegenproben und die Liste der FMI Standard.org

Siehe Dokumentation im Unterverzeichnis `cross-check`.

5.3. Der Weg um Test-FMUs zu generieren

5.3.1. Der Gebrauch von C++ - FMUs

Ein geeigneter Weg, um ein einfaches, sehr spezielles Test-FMU zu erhalten, ist der Gebrauch des FMI Code Generator-Werkzeugs. Es ist ein Python-Werkzeug, mit einer grafischen Benutzeroberfläche, auf der FMU-Variablen und -Eigenschaften festgelegt werden können. Mit dieser Information erstellt der Code-Generator ein Quellcodeverzeichnis mit einem Vorlagen-Code und verbundenen Aufbauskripts - sodass es sehr einfach ist, eigene FMUs herzustellen. Siehe Dokumentation /Tutorial auf der Github-Seite.

5.3.2. FMUs, die von der SimulationsX exportiert werden.

Benötigt eine passende Lizenz. Das Exportieren von FMUs aus SimX ist sehr einfach, aber für die Windows-Plattform beschränkt.

5.3.3. Von OpenModelica exportierte FMUs

OpenModelica kann ebenso FMUs exportieren. Hier folgen die Schritte, um so ein FMU herzustellen.

Erstellen eines Modelica Modells

Erstellen eines Modelica Modells.

Annotieren Sie Variablen, die als Ausgangsgröße genutzt werden 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;
```

Variante 1: Das FMU manuell erstellen

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

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

Der Output lässt vermuten, dass die FMU-Datei `/tmp/OpenModelica/BouncingBall.fmu` erfolgreich erstellt wurde.

Für Version 2.0 des FMI-Standard-Gebrauchs:

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

Variante 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 teilweise nützlich für Entwickler eigener FMUs oder zum Identifizieren von Rechenproblemen eines dritten FMUs sind.

6.1. Modifikation/Fixierung des FMU-Inhalts

Wenn ein FMU eine ungültige `modelDescription.xml`-Datei besitzt, kann der Prozess zum Entpacken der FMU, Fixieren der Datei, erneutes Packen der FMU und das erneute Versuchen der Simulation mit einem Co-Simulations-Master zu viel Zeit kosten. Mit *MasterSim* läuft die Simulation nur einmal und extrahiert das FMU in das jeweilige `fmus`-Unterverzeichnis subdir. Dort kann man alle Dateien untersuchen und Fehler festmachen, oder fehlende Dateien zufügen.

Das nächste mal, wenn Sie den *MasterSimulator* durchlaufen lassen, führen Sie die Befehlszeilen-Option `--skip-unzip` aus und *MasterSim* wird den Schritt des Entpackens überspringen und direkt auf die Dateien im extrahierten Verzeichnis zugreifen.

---

7. Informationen für Entwickler

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 durch die Befehlszeile

Linux/MacOS

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

Im Falle fehlender Abhängigkeiten (verlangt zlib) wollen Sie vielleicht die zugehörigen Entwicklungspakete installieren.

Windows

Es sind komfortable Skripte für die Errichtung mit Visual Studio 2015 und Qt5 (für die Benutzeroberfläche von *MasterSim*) enthalten. Andere Kompilierer, wie MinGw, arbeiten genauso gut, die Dateienpfade müssen allerdings manuell konfiguriert werden.

Die Dateien sind im `build/cmake`-Verzeichnis verortet:

```
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\5.11.3\msvc2015 ①
c:\Qt\5.11.3\msvc2015_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 es irgendwo anders installiert ist,
können alternativ die Umgebungsvariablen `JOM_PATH` und `CMAKE_PREFIX_PATH` gesetzt
werden (siehe Aufbau-Batch-Dateien).
```

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

```
cd build\cmake
build_VC14.bat
```

oder:

```
cd build\cmake
build_VC14_x64.bat
```

Für unterschiedliche Visual-Studio-Versionen oder MinGW kopieren Sie die Batch-Datei und bearbeiten die Pfadgrößen. Eventuell wollen Sie auch diese Batch-Dateien überarbeiten, wenn sie verschiedene Installationspfade haben.

7.1.2. Externe Bibliotheken

Die *MasterSim*-Bibliothek und die Simulationsprojekte unterscheiden sich durch die folgenden

externen/dritten Bibliotheken (außer für Qt sind alle im Quell-Code-Speicher enthalten, somit ist es nicht notwendig, diese 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, angeboten von IBK, TU Dresden, Germany)
- Minizip and zlib zum Entpacken von FMUs

nur **MasterSimulatorUI**

- Qt 5 Library
- BlockMod library (von IBK, TU Dresden, Germany, untergebracht in github)

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

Die Bibliotheken in einem dritten Unterverzeichnis sind möglicherweise veraltet. Nutzen Sie bitte nur den Quell-Code im `externals`-Unterverzeichnis des Speicherortes.

7.2. Entwicklungsumgebungen und Projekt-/Sitzungsdateien

7.2.1. Qt Creator

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

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

Ausführbare Programme sind verortet in:

```
bin/debug        - Ausgabepfad für Entwicklungen mit dem Qt-Erschaffer
bin/release      - standardisierter Ausgabepfad für Cmake-Bildung
```

7.2.2. Visuelles Studio

CMake-erstellte VC-Projektdateien

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

Basisschritte: Öffnen Sie ein Konsolenfenster, die Installation des VC-Umgebungsaufbau und alle erforderlichen Pfade, nutzen 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 ändern dies in das Verzeichnis `build/cmake`.

1. starten Sie entweder `build.bat` oder `built_x64.bat` und drücken Sie Ctrl+C, wenn der Aufbau startet.
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 Erstellungsgenerator.

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

Vorbereitete VC-Projektdateien

Sie können ebenso die VC-Problemlösungsdatei in `build\VC14\MasterSim.sln` öffnen. **Note**: Sie müssen zunächst die CMake erstellen, um die Zlib `zconf.h`-Datei zu konfigurieren!

7.3. Hilfreiches Material bezogen auf die Entwicklung auf Linux

Hier folgen ein paar Anmerkungen, die ich gesammelt habe, als ich auf einige unerwartete Schwierigkeiten beim Erstellen dieses Master-Simulators gestoßen bin:

7.3.1. Überprüfen von Symbolen in gemeinsamen Bibliotheken

```
objdump -t <shared_library>
```

Um alle fmi2-Funktionen zu erhalten

```
objdump -t <shared_library> | grep fmi2
```

7.3.2. Verknüpfung gemeinsamer Bibliotheken mit statischen Teilen (die in ausführenden Programmen ebenso auftauchen)

Problem: Ursache des Problems: Sowohl FMU als auch Master nutzen die IBK-Bibliothek, welche wiederum statistische Teile/Singletons (z. B. Message-Handler) haben. Wenn FMU mit Exe während der Reinigung am Ende der Leitung verbunden wird, wird der Destructor des Singleton-Objekts zweimal aufgerufen, was einen Teilfehler verursacht.

Lösung: Dennoch keine, es scheint zu funktionieren, nachdem die "Verdopplung der So-Import-Prüfung" hinzugefügt wurde.

TODO: Hervorheben Eigennamen? z. B. cmake, Qt creator

7.3.3. FMU von Fehlern befreien

Vorausgesetzt Sie entwickeln die gemeinsame FMU-Bibliothek mit dem Qt Creator, können sie diesem Ablauf folgen:

1. Erstellen Sie ihr FMU entweder mit erhöhter Fehlerreinigung oder geben Sie Fehlerreinigungssymbole frei. Sie können auch eine externe Aufbauwerkzeug-Verkettung nutzen, z. B. Cmake.
2. Stellen Sie ihr FMU zusammen und entpacken Sie es im FMU-Archiv (Sie müssen dies nur einmal tun); Bedenken Sie: die gemeinsame Bibliothek innerhalb des FMU muss diejenige sein, die mit einem Qt Creator erstellt wurde
3. Erstellen Sie ihr MSIM-Test-Projekt.
4. Im Qt Creator öffnen und aktivieren Sie das MaterSimulator-Projekt, wählen Sie das MSIM-Projekt als Befehlszeilenargument aus und starten Sie die Fehlerbereinigung. Es wird die FMUs extrahieren und zu den Funktionen der gemeinsamen Bibliothek zufügen.

Sie können jetzt entweder Fehler entfernen und zu den Funktionen von FMU fmi2xxx übergehen oder die Quelldateien öffnen, die Sie für das Erstellen des FMU genutzt haben und Programmstopps setzten. Der Qt Creator wird diese automatisch aufnehmen und Sie können Fehler bereinigen/zum Master übergehen und ebenso zu den FMUs.

Beipiel: Während der Fehlerbereinigung eines gesonderten FMU-Projekts, welches statisch im Freigabemodus erstellt ist, aber während der Entwicklung dynamisch mit anderen Bibliotheken verbunden ist.

1. Das FMU ist erstellt (zunächst mit einer statisch verbundenen FMU `Test.so`) und das MasterSim-Projekt ist eingerichtet.
2. *MasterSim* läuft einmal und die Verzeichnisstruktur ist erstellt, das FMU ist extrahiert und wurde ohne angehängte Fehlerbeseitigung gestartet.
3. Nun erstellt das FMU-Entwicklungsprojekt im Qt Creator `libTest.so.2.0.1`, welches eine

Verbindung zu anderen dynamischen Bibliotheken im Entwicklungsverzeichnis schafft.

4. Die FMU-Datei wird in `Test.so` umbenannt und in das extrahierte FMU-Verzeichnis kopiert, wodurch es das statisch verknüpfte FMU überschreibt.
5. Der Suchpfad der Bibliothek zu der anderen dynamischen Bibliothek, zu welcher `libTest.so.2.0.1` führt, ist variabel zur Umgebung von MasterSim-Projekt `LD_LIBRARY_PATH` zugefügt.
6. `MasterSimulator` startet im Feherbereinigungs-Modus unter Nutzung der `--skip-unzip` -Befehlszeilen-Option.

7.4. Innerhalb von MasterSim

7.4.1. Datentypen

Die Simulation mit unterschiedlichen FMU-Datentypen (v1 and v2) sollte übereinstimmen. Ebenso sind die voreingestellten Typenüberschriften die selben für beide Versionen.

Die vorgesehene Plattform für diesen Master ist das Desktop-System (32bit/64bit), deshalb werden alle vom Master-Algorithmus erkannten Datentypen folgendermaßen zugeordnet:

- `fmi2Boolean` (`bool` in skalierten Schnittstellenfunktionen)
- `int`
- `double`
- `std::string`

7.4.2. Verbindungsgraf und variable Zuordnung

Variablen können einzig identifiziert werden von

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

und ein Graf kann definiert werden von:

```
A.x1        B.u1
A.d1        C.du1
B.x1        C.u1
B.x2        A.u2
C.x1        A.u1
```

Die erste Spalte sind Ausgabevariablen, die zweite Spalte verbundene Eingangsvariablen. x sind wahre Typen, d vom Typ *ganze Zahl*.

Jeder Slave besitzt für jeden Datentypen einen Vektor an Ausgabewerten (bool, int, real/double, string). Der Master besitzt außerdem für jeden Datentypen einen Vektor für Verbindungsvariablen.

Eine Abbildung der Variablen des lokalen Slave-Speichers zum globalen Vektor und vom globalen Vektor zu den Eingängen wird durch eine Abbildungstabelle individuell für jeden Datentypen realisiert:

Abbildung Ausgänge - wahre Typen

Slave	VariableName	global index	local index
A	x1	0	0
B	x1	1	0
B	x2	2	1
C	x1	3	0

Der Transfer von lokaler zu globaler Datenspeicherung ist dann ein simpler Algorithmus:

```
loop connectedVariableIndexes:
    copy(localArray[localIndex], globalArray[globalIndex])
```

Im Fall von Slave B wird das Feld connectedVariableIndexes [0, 1] sein.

Abbildung Eingänge - wahre Typen

Für Eingangsvariablen existiert eine gleichartige Abbildung. Kein Slave hat einen Eingangsvariablen-Zwischenspeicher, statt dessen sind die Variablen individuell gesetzt (siehe auch Newton-Algorithmus und die Jacobian-Generatio via DQ-Algorithmus).

Slave	VariableName	global index	local value reference
B	u1	0	55
C	u1	1	348432
A	u2	2	122
A	u1	3	321

Abbildungen von Eingang und Ausgang sind in der einzelnen Tabelle RealVariableMappings kombiniert.

Notiz: Der Ausgang eines Slaves ist eventuell direkt mit jeder eigenen Eingangsvariable verbunden, zum Beispiel:

```
fmu1.var2   fmu1.var15
```