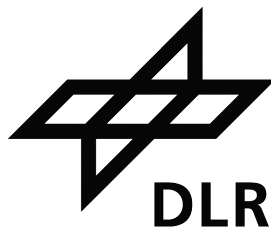
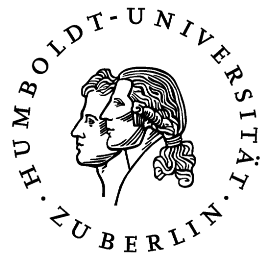
Studienarbeit

Verhaltensintegration in SMP2-Modelle

Alexander Röhnsch

30. November 2008





Axel Berres
Deutsches Zentrum für Luft- und Raumfahrt (DLR)
Simulations- und Softwaretechnik (SISTEC)
Rutherfordstraße 2
12489 Berlin

Dr. Klaus Ahrens
Lehrstuhl Systemanalyse
Institut für Informatik
Mathematisch-Naturwissenschaftliche Fakultät II
Humboldt-Universität zu Berlin

Inhaltsverzeichnis

1 Einführung **4**
1.1 Motivation . . . 4
1.2 SMP2 . . . 5

2 Beispielmodell **7**
2.1 Mathematische Beschreibung . . . 7
2.2 Konfiguration . . . 8

3 Beispielmodell in C++ **9**
3.1 SMDL-Dokumente . . . 9
3.2 SMP2-Code . . . 11
3.3 Simulation in SIMSAT . . . 12
3.4 Zusammenfassung . . . 13

4 Simulink **14**
4.1 Beispielmodell in Simulink . . . 15
4.2 Real-Time Workshop . . . 17
4.3 MOSAIC . . . 18
4.4 Simulation in SIMSAT . . . 23
4.5 Zusammenfassung . . . 24

5 Modelica **25**
5.1 Beispielmodell in Modelica . . . 25
5.2 Codeerzeugung . . . 27
5.3 SMP2-Kapselung . . . 28
5.4 Simulation in SIMSAT . . . 31
5.5 Zusammenfassung . . . 32

6 Vergleich und Ausblick **33**
6.1 Zusammenfassung . . . 34
6.2 Ausblick . . . 34

A Anhang **36**
A.1 Konvertierung von Simulink- und Modelica-Modellen . . . 36

Literatur **37**

Abbildungsverzeichnis

1 Darstellung der abgeschnittenen Sinuskurve. Die unterbrochene Linie zeigt eine normale Sinuskurve. Die durchgezogene Linie zeigt die abgeschnittene Sinuskurve. Die Fläche unter der ersten fünf Perioden der abgeschnittenen Sinuskurve ist blau ausgefüllt. 8
2 Modellstruktur in SMP2 9
3 Instanzierungsplan in SMP2 10
4 Ausführungsplan in SMP2 10
5 Simulation des Batteriemodells im Simulator SIMSAT 13
6 Batteriemodell in Simulink 14
7 Umgebung für Batteriemodell in Simulink 15
8 Simulationseinstellungen in Simulink 16
9 In Simulink berechneter Ladestand 17
10 Steuerung der Funktionen aus dem Modellcodes des Real-Time Workshops. Quelle: [RTW User Manual 3] 18
11 Screenshot der Konfiguration des Real-Time Workshops 19
12 Screenshot des Transformationsdialogs von MOSAIC 20
13 Simulationslauf des Simulink-Modells in SIMSAT 24
14 Batteriemodell in Dymola 25
15 Umgebungsmodell für Batterie in Dymola 26
16 In OpenModelica berechneter Ladestand 27
17 Simulationslauf des Modelica-Modells in SIMSAT 31

1 Einführung

Diese Studienarbeit untersucht verschiedene Möglichkeiten für die Transformation einer Simulationskomponente von ihrer Blockdiagramm-Repräsentation in ausführbaren Simulationscode, der konform zum ESA-Standard SMP2 ist.

Die einzelnen Transformationsschritte werden anhand eines Beispielmodells veranschaulicht.

1.1 Motivation

Das DLR entwickelt im Rahmen des Projekts *Virtueller Satellit* eine Werkzeugkette, die eine frühzeitige (dynamische) Simulation und Beurteilung von geplanten Raumfahrtsystemen ermöglichen soll.

Diese Werkzeugkette trennt die Arbeitsschritte Definition, Modellierung und Simulation einer Komponente. Während der Systementwickler die benötigten Komponenten einer Simulation grob definiert, können Ingenieure verschiedener Disziplinen diese Komponenten mit ihrem fachspezifischen Wissen anschließend genau modellieren. Sie sollen sich dazu einer grafischen Modellierumgebung bedienen können, deren Konzepte ihnen vertraut sind. Schließlich steht so eine Sammlung vollständig modellierter Komponenten zur Simulation bereit. Wie die einzelnen Komponenten zu einer Simulation verknüpft werden, und wie Komponenten im Nachhinein geändert werden können, sei dabei in dieser Arbeit außer Betracht gelassen.

Komponenten werden für jeden zu durchlaufenden Bearbeitungsschritt unterschiedlich repräsentiert, um Anforderungen an den jeweiligen Arbeitsschritt zu erfüllen. So werden beispielsweise zur Simulation detailliert Laufzeitinformationen verwertet. Die Definition sollte sich dagegen nur auf einen groben Aufbau einer Komponente konzentrieren.

Diese Studienarbeit betrachtet die Umwandlung einer zur Modellierung geeigneten Repräsentation - die Blockdiagramm-Darstellungen der Werkzeuge Simulink und Modelica - in eine mit dem ESA-Standard SMP2 simulierbare Repräsentation. Zur Veranschaulichung dieser Umwandlungen dient ein einfaches Modell, das den Ladestand einer Batterie unter Berücksichtigung einer veränderlichen Verbrauchsleistung wiedergibt.

Der grundlegende Ansatz dieser Transformation besteht darin, von vorhandenen Simulationsumgebungen Quellcode für die Berechnung des Modellverhaltens erzeugen zu lassen. Die Ausführung dieses Codes wird dann durch SMP2-Mechanismen ersetzt und so auf einen generischen Zielsimulator übertragen. Weiter müssen dem Simulator noch anzuzeigende Werte von Zustandsvariablen übermittelt werden.

In den folgenden Kapiteln wird deshalb zunächst der Standard SMP2 vorgestellt. Danach wird das Beispielmodell erklärt und mit SMP2 simuliert, um die Voraussetzungen für die eigentlichen beiden Transformationen zu zeigen. Anschließend wir das Beispiel jeweils als Simulink-Modell und als Modelica-Modell formuliert und SMP2-konform simuliert.

1.2 SMP2

Simulation Model Portability 2.0 (SMP2) ist ein unter der Leitung der ESA entwickelter Standard, der Entwicklungsaufwand bei Simulationsprojekten verringern soll [SMP2].

Er definiert zunächst Schnittstellen zwischen *Modellen*, *Simulatoren* und *Simulationsdiensten*. Die Trennung dieser Rollen führt zum einen zu **gegenseitiger Unabhängigkeit**. So kann eine Simulation auf verschiedenen, SMP2-konformen Simulatoren laufen. Auch ist die Entwicklung eines Simulators unabhängig von den Modellen, die später darauf simuliert werden sollen. Zum anderen können auf Grundlage einer einheitlichen Schnittstelle Modelle zur **Wiederverwendung** in mehreren Projekten entwickelt werden. Die Schnittstellen regeln ferner die Nutzung von Simulationsdiensten, wie z.B. dem *Logger* zum Protokollieren von Nachrichten, und unterstützen mehrere Mechanismen zur Kommunikation zwischen Modellen.

Ein Modell, ein Simulator oder ein Simulationsdienst werde als **SMP2-konform** bezeichnet, wenn die jeweiligen SMP2-Schnittstellen implementiert sind. Vereinfacht betrachtet sind dies die Schnittstellen `IModel`, `ISimulator` und `IService`. Die Schnittstelle für Modelle besteht z.B. aus Methoden, über die Modelle konfiguriert oder beim Simulator registrieren werden. Die Funktionen der Simulator-Schnittstelle gewähren vor allem Zugriff auf Simulationsmodelle. Tatsächlich gibt es aber eine Fülle weiterer Schnittstellen, um mit einem Modell verschiedenste Mechanismen zu unterstützen, z.B. dynamische Änderung von Modellfunktionen zur Laufzeit oder objektorientierte Konzepte wie Vererbung.

Alle Schnittstellen des SMP2-Standards sind durch die *Interface Definition Language* (CORBA IDL) definiert. Sie werden im so genannten *SMP2-Komponentenmodell* (SMP2 Component Model) beschrieben. Der Begriff Komponentenmodell leitet sich hierbei von genutzten Ideen der *Komponentenbasierten Entwicklung* ab. Konkrete Funktionen werden in eigenen Schnittstellen gekapselt. Diese Trennung ermöglicht eine leichtere Wiederverwendung von Modellen und auch eine einfache Änderung eines konkreten Mechanismus. Die konkrete Definition der Schnittstellen im SMP2-Komponentenmodell soll an dieser Stelle aber nicht weiter betrachtet werden.

Für vorliegende Arbeit ist vielmehr die **Interaktion zwischen Modellen und Simulator** interessant. Damit ein Modell seine Berechnungen ausführen kann, registriert es beim Simulator bestimmte *Aktivierungsfunktionen*, so genannte *EntryPoints*. Der Simulator ruft diese Funktionen des Modells nach einem bestimmten Schema auf, das zusätzlich definiert wird. Ein Modell registriert auch *Zustandsvariablen* (SMP2 Fields). Der Simulator erhält so Zugriff auf die Werte der Variablen zu jedem Zeitpunkt der Simulation und kann sie seinerseits dem Benutzer anzeigen oder anders weiter verarbeiten. Der Simulator ist also während der Simulation vor allem mit der Berechnung der Simulationszeit, dem Aufruf der Aktivierungsfunktionen zu gegebenen Zeitpunkten, dem Anzeigen der Zustandsvariablen registrierter Modelle und der Bereitsstellung des Zugriffs auf Simulationsdienste und Modelle beschäftigt. Konkrete Simulationsberechnungen finden ausschließlich in den Aktivierungsfunktionen und darin aufgerufenen Funktionen statt.

Zur einfachen Beschreibung des Aufbaus SMP2-konformer Modelle definiert SMP2 die *Simulation Model Definition Language* (SMDL), ein Metamodell für SMP2-konforme Modelle und Simulationsparameter. Mit dem Metamodell können mehrere Dokumente formuliert werden.

Der **Catalogue** beschreibt die Modellstruktur, also insbesondere seine Zustandsvariablen und Aktivierungsfunktionen. Es lassen sich auch Vererbungs-, Besitz- und weitere Beziehungen zwischen Modellen ausdrücken. Der Catalogue eignet sich nicht zur vollständigen Implementierung des Modellverhaltens. Dieses muss später im Modellcode implementiert werden.

Das **Assembly** beschreibt eine konkrete Instanzierung von Modellen. Es legt also Anzahl und Art anfangs vorhandener Simulationskomponenten fest und belegt ihre Zustandsvariablen mit Werten.

Der **Schedule** definiert ein Aufrufschema für die Aktivierungsfunktionen eines Simulationsmodells. Dazu werden einmalige oder regelmäßig wiederkehrende zeitliche Ereignisse definiert, die bei Aktivierung eine bestimmte Modellfunktion aufrufen. Beispielsweise könnte eine Initialisierungsfunktion bei 0 Sekunden der Simulationszeit aufgerufen werden, eine Berechnungsfunktion soll dagegen ab Sekunde 5 alle 0,005 Sekunden aufgerufen werden.

Während Angabe von Assembly und Schedule nicht notwendig für die Simulation sind, bieten sie durch die Trennung der Parameter und der Steuerung eines Modells von der eigentlichen Modellbeschreibung die Möglichkeit dynamischer Konfigurierung. Die Modelle sind so später leichter wieder verwendbar.

Vor dem Hintergrund der Transformation von Modellstrukturen ist leider nicht klar, ob SMDL kompatibel zur *Meta Object Facility* (MOF) ist. In der Dokumentation zu SMP2 werden dazu keine Angaben gemacht. Für eine Modell-Modell-Transformation von oder nach SMDL sollte also umfassend geprüft werden, ob SMDL MOF-kompatibel ist. Dann könnte man auf vorhandene Methoden der Transformation zurückgreifen, wie z.B. QVT.

Das *SMP2 C++ Mapping*, oder auch einfach *Language Mapping*, transformiert einen SMDL-Catalogue mit der Modellstruktur in C++-Code. Es werden Klassen, Variablen und Funktionen deklariert, wie sie im Modell angegeben sind und von den Schnittstellen des Komponentenmodells gefordert werden. Erforderliche Abläufe, wie z.B die Anmeldung beim Simulator, oder die Registrierung der Aktivierungsfunktionen und Zustandsvariablen werden über das Language Mapping automatisch bereitgestellt. So kann aus einer Modellbeschreibung vollständiger Code erzeugt werden. Darin fehlt nur die Implementierung der Aktivierungsfunktionen, also die eigentliche Berechnung des Modellverhaltens.

Ein weiteres SMDL-Dokument namens **Package** referenziert eine Instanzierungsbeschreibung in einem Assembly. Daraus lässt sich mit dem Language-Mapping Code erstellen, der das Modell generisch mit der Factory-Methode instanziert. Zusätzlich werden dabei alle zur vollständigen Kompilierung des Modells erforderlichen Makefiles erzeugt.

2 Beispielmodell

Zur Veranschaulichung der Transformation dient ein einfaches Modell, das den **Ladestand einer Batterie** berechnet. Es erhält als Eingang die zeitabhängige Funktion eines imaginären Verbrauchs, also die elektrische Leistung, die an der Batterie abgegriffen wird. Die Einheit ist Watt `[W]`. Das Modell berechnet die ebenfalls zeitabhängige elektrische Energie der Batterie (im Folgenden mit Ladestand bezeichnet). Die Einheit ist Wattsekunde `[Ws]`.

2.1 Mathematische Beschreibung

Elektrische Leistung ist die Änderungsrate einer elektrischen Energie. Sei $E(t)$ die elektrische Energie der Batterie, so beschreibt die Ableitung $E'(t)$ die Änderung dieser Energie, bzw. die Leistung, die an der Batterie erbracht wird. Nun ist die Batterie aber kein Verbraucher, sondern eine Spannungsquelle. Es soll die Leistung $P(t)$ betrachtet werden, die die Batterie für andere Verbraucher aufbringt. Diese ist $-E'(t)$. Umgestellt erhält man einen ersten Zusammenhang mit:

$$E(t) = \int -P(t)dt$$

Zusätzlich sind noch weitere Eigenschaften zu berücksichtigen: Die Batterie soll nicht geladen werden können und ein angelegter Ladestrom soll auch keine weitere Wirkung zeigen. Eine negative Verbrauchsleistung wird also ignoriert.

$$E(t) = \int -max(P(t), 0)dt$$

Dabei sei die Funktion max definiert als:

$$max(x) = \begin{cases} x, \text{falls } x \geq 0 \\ 0 \text{ sonst} \end{cases}$$

Zudem habe die Batterie einen anfänglichen Ladestand von E_0:

$$E(t) = E_0 + \int -max(P(t), 0)dt$$

Schließlich soll die Batterie auch nicht unter einen Ladestand von 0 fallen können:

$$E(t) = max(E_0 + \int -max(P(t), 0)dt, 0)$$

Es wird also lediglich die verbleibende Energie der Batterie berechnet. Insbesondere gibt es keinen Fehler wenn die Batterie leer ist, aber weiterhin Leistung abgegriffen wird.

Die gewählte Modellierung ist zulässig solange einzig der Ladestand der Batterie betrachtet wird. Soll das Modell in einen Schaltkreis integriert und das Verhalten des Schaltkreises untersucht werden, so ist das hier verwendete datenflussbetonte Blockdiagramm weniger geeignet. Es unterstellt einen kausalen Zusammenhang zwischen Ein- und Ausgängen der Blöcke: Ausgänge berechnen sich aus Eingängen. Ein Stromfluss ließe sich mit diesem Ansatz aber nicht einfach modellieren. Die Eigenschaften eines Stromflusses lassen sich nicht mit "angeforderten Strom" und "gelieferten Strom" beschreiben. Im Schaltkreis ist das Verhalten des Stromflusses und damit auch das Entladeverhalten der Batterie sowie das Verbrauchsverhalten der Teilnehmer durch Elemente wie die Widerstände der

Schaltelemente, die angelegte Spannung und elektrische Eigenschaften der Teilstromkreise bestimmt. Es sei auf die Möglichkeit der Sprache *Modelica* verwiesen, Schaltkreise mit speziellen, s.g. akausalen Blockdiagrammen zu modellieren. Dies ermöglicht eine einfache, korrekte Simulation von Schaltkreisen.

2.2 Konfiguration

Die Batterie erhalte eine Anfangsladung von 10 Ws. Das Eingangssignal $P(t)$ sei ein einfacher Sinus. So können alle modellierten Eigenschaften gut betrachtet werden. Alle Kurvenabschnitte kleiner 0 werden demnach auf 0 gesetzt und es ist eine Stagnation des Ladestandes zu erwarten. Bei positiven Kurvenabschnitten sollte der Ladestand fallen.

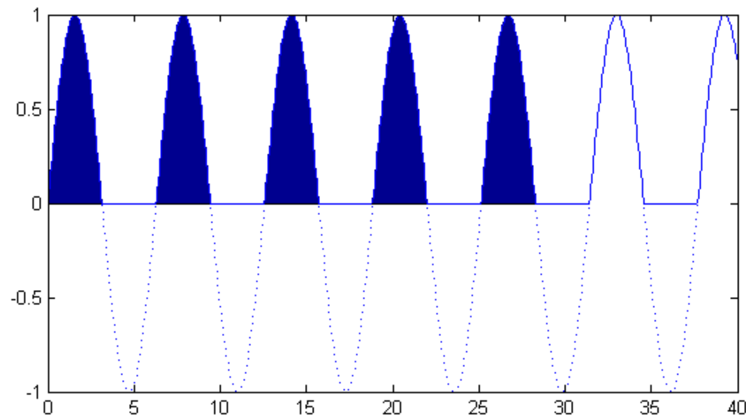


Abbildung 1: Darstellung der abgeschnittenen Sinuskurve. Die unterbrochene Linie zeigt eine normale Sinuskurve. Die durchgezogene Linie zeigt die abgeschnittene Sinuskurve. Die Fläche unter der ersten fünf Perioden der abgeschnittenen Sinuskurve ist blau ausgefüllt.

Eine Sinus-Periode besteht aus einem positiven Kurvenabschnitt, gefolgt von einem negativen. Die Fläche einer Periode des abgeschnittenen Signals ist also die Fläche des positiven Kurvenabschnitts:

$F_P = \int_0^{2\pi} max(sin(t), 0)dt = \int_0^{\pi} sin(t)dt = 2$

Die Anfangsladung von 10 Ws ist also spätestens nach $\frac{10}{F_P} = 5$ Perioden erschöpft. Da die ganze Leistung der fünften Periode aber schon nach der halben Periodenlänge abgefallen ist, ist die Anfangsladung schon nach 4,5 Perioden erschöpft. Die Leerung ist deshalb bei $4{,}5 * 2\pi = 28{,}27s$ zu erwarten.

Nach der Leerung dürfte sich der Ladestand nicht mehr ändern, da die Batterie nicht geladen und der Ladestand nicht unter 0 fallen kann.

3 Beispielmodell in C++

3.1 SMDL-Dokumente

Das eben beschriebene Modell soll über die SMP2-Schnittstelle simuliert werden. Dazu wird mit dem SMP2-Metamodell die Modellstruktur in einem Catalogue, die Instanzierung in einem Assembly und die Ausführung in einem Schedule beschrieben.

Catalogue

Der Simulator soll das Modell über zwei Aktivierungsfunktionen (EntryPoints) steuern. Die eine dient zum Aktualisieren des Eingangssignals, die andere zur Berechnung des Ausgangssignals. Ihre Namen heißen dementsprechend `Update_Source_Signal` und `Calculate_Output`. Weiterhin wird eine Zustandsvariable für den Zustand des Integrators benötigt. Da aber auch das Eingangssignal und das Ausgangssignal im Simulator angezeigt werden sollen, werden die drei Variablen (Fields) `Integrator_State`, `Source_Signal` und `Output_Signal` definiert. Folgende Abbildung zeigt (unvollständig) die gewählte Struktur, wie sie vom Simulator SIMSAT [SIMSAT] dargestellt wird.

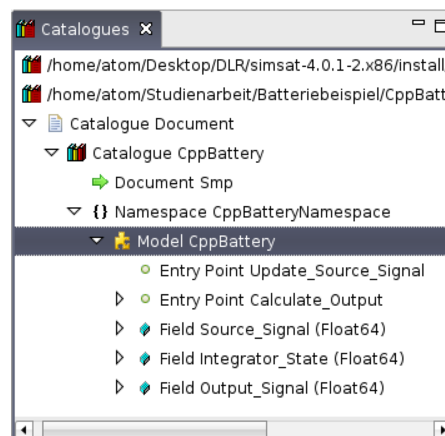


Abbildung 2: Modellstruktur in SMP2

Assembly

Für das Beispielmodell wird eine Instanz angelegt, sowie die drei Zustandsvariablen `Source_Signal`, `Integrator_State` und `Output_Signal`. Für jede der Zustandsvariable wird ihr Anfangswert als Parameter festgelegt. Im Beispiel wird ein Wert von 10 für den Integratorzustand festgelegt, was den 10 Ws Anfangsladung des Batteriemodells entspricht.

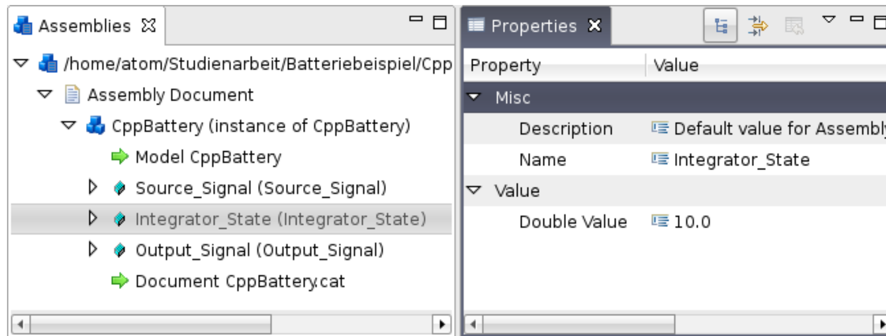


Abbildung 3: Instanzierungsplan in SMP2

Schedule

Die zwei deklarierten Aktivierungsfunktionen sollen vom Simulator regelmäßig aufgerufen werden. Dazu wird im Schedule jeweils ein *Task* definiert, der die beiden Funktionen aktiviert. Dazu wird ein Simulationszeitereignis definiert, dem die *Tasks* zugeordnet sind. Folgende Abbildung zeigt die SIMSAT-Darstellung dieses Schedules.

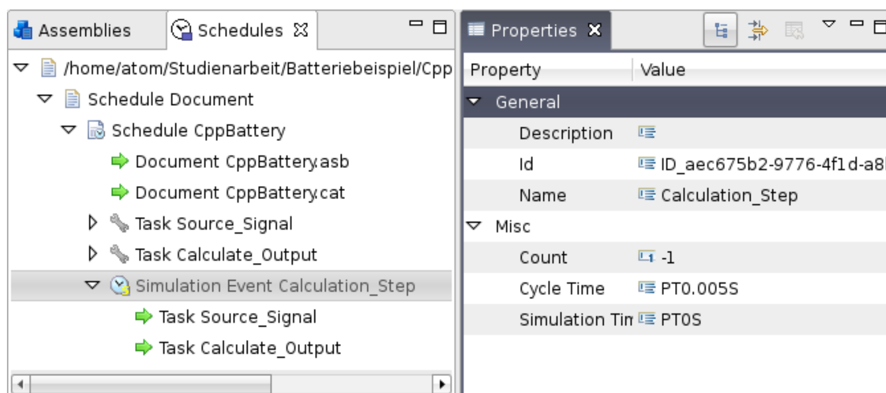


Abbildung 4: Ausführungsplan in SMP2

Die eigentlichen Eigenschaften des Ereignisses werden wie folgt festgelegt. `-1` als Wert für den Parameter `Count` schreibt eine unbegrenzte Anzahl an Aktivierungen vor. Mit `Cycle Time` wird als Intervall zwischen zwei Ereignissen 0,005 s festgelegt. [1] `Simulation Time` gibt schließlich den Startzeitpunkt für die Ausführungsabfolge vor.

Durch die Wahl einer festen Schrittweite werden im Allgemeinen die numerischen Eigenschaften der Lösung nicht korrekt bestimmt. Egal wie fein die Schrittweite gewählt sei;

[1] `PT0,005S` ist eine Zeitangabe nach ISO 8601. `P` deutet auf eine Zeitspanne und `T` leitet eine als Zeit zu interpretierende Zeichenkette ein. `S` ist die Angabe der Zeiteinheit Sekunde

ist sie nicht variabel, gibt es immer Charakteristika, die zwischen zwei Schritten nicht erkannt werden können. Für das vorgestellte einfache Beispiel reicht eine feste Schrittweite aber aus. Der zu erwartende Kurvenverlauf wurde bereits vorgestellt.

3.2 SMP2-Code

Das SMP2 Language Mapping erzeugt aus den Informationen eines Catalogues C++-Quelltext, der die SMP2-Schnittstelle standardkonform bedient. Darin wird unter anderem die Kommunikation mit dem Simulator bereits abgewickelt. Im Besonderen werden die deklarierten Aktivierungsfunktionen registriert und alle Variablen instanziert und registriert. Beispielhaft ist im folgenden Auszug die vom Simulator aufzurufende Funktion `void Publish(IPublication*)` zu sehen, in der die Registrierung der drei Variablen und der zwei Aktivierungsfunktionen stattfindet.

```
void CppBattery::Publish( ::Smp::IPublication* receiver )
    throw ( ::Smp::IModel::InvalidModelState )
{
    ::Smp::Mdk::Management::ManagedModel::Publish( receiver );

    receiver->PublishField(
        "Source_Signal", "", &Source_Signal );
    receiver->PublishField(
        "Integrator_State", "", &Integrator_State );
    receiver->PublishField(
        "Output_Signal", "", &Output_Signal );

    receiver->PublishOperation( "Update_Source_Signal", "",
        ::Smp::Publication::Uuid_Void );
    receiver->PublishOperation( "Calculate_Output", "",
        ::Smp::Publication::Uuid_Void );
}
```

Wurden alle zur Berechnung des Modellverhaltens erforderlichen Variablen bereits im Catalogue definiert, beschränkt sich die weitere Implementierung nun auf die Aktivierungsfunktionen. Diese werden im Folgenden kurz besprochen.

Für die Implementierung der Funktion `Update_Source_Signal` wurde als Eingangssignal einfach ein Sinus mit einer Periodenlänge von 2πs gewählt, abhängig von der Simulationszeit. Die Ermittlung der Simulationszeit ist hier vereinfachend als einzelner Funktionsaufruf dargestellt. Dahinter steckt ein Aufruf des SMP2-Simulationsdienstes `TimeKeeper`, von dem die aktuelle Zeit in verschiedenen Skalen abgefragt werden kann.

```
void CppBattery::_Update_Source_Signal()
{
    // calculate sine from simulation time
    Source_Signal = sin( GetSimulationTime() );
}
```

Bei der Berechnung des Ausgangssignals lässt sich die im mathematischen Modell hergeleitete Formel wiedererkennen. Der Anfangsladestand E_0 soll als Modellparameter unabhängig vom Quelltext sein und kann über die Variable `Integrator_State` direkt im Assembly festgelegt werden. Für die Integration wird die zusätzlich definierte Funktion `Float64 Integrate(Float64)` aufgerufen, die den Integrationsalgorithmus kapselt.

```
void CppBattery::_Calculate_Output()
{
    Output_Signal = max( 0.0, Integrate( - max( 0.0, Source_Signal) ) );
}
```

Der Integrator ist hier vereinfachend als diskrete Summation realisiert. Der Zeitschritt von 0,005 s ist hier explizit im Code angegeben. Er sollte wie der Werte der Anfangsladung im Assembly als Parameter verfügbar gemacht werden. Hier wurde darauf verzichtet, um die Behandlung mit dem Wert der Anfangsladung vergleichen zu können.

```
::Smp::Float64 CppBattery::Integrate(::Smp::Float64 input)
{
    // discrete implementation of Integrator
    // it is assumed that we are called once each 0.005 second
    Integrator_State += input * 0.005;
    return Integrator_State;
}
```

3.3 Simulation in SIMSAT

Durch Kompilieren des Codes mit den implementierten Aktivierungsfunktionen und Linken mit dem SMP2 MDK erhält man eine Bibliothek des Modells. Diese kann zusammen mit dem Assembly und dem Schedule (und einer Projektdatei, die diese drei Dateien referenziert) von einem SMP2-konformen Simulator ausgeführt werden. Von den zwei gegenwärtig verfügbaren SMP2-Simulatoren *SIMSAT* und *EuroSim* benutzen wir *SIMSAT* [SIMSAT]. Abbildung 5 zeigt einen Screenshot des mit dem Beispielmodell geladenen *SIMSAT*. Links ist eine Liste sichtbar, in der sich die registrierten Variablen wiederfinden. Diese können im rechten Bereich, im s.g. *Data Viewer*, als Wert und/oder als Kurve verfolgt werden.

Mit dem Sinus-Signal als Eingang wird deutlich, wie nur die positiven Kurvenabschnitte zu einem Absinken des Ladestandes führen (Abbildung 5 unten rechts). Bei negativen Abschnitten stagniert der Ladestand. Weiterhin bleibt der Ladestand nach Erreichen des 0-Wertes konstant 0. Die Batterieladung ist erwartungsgemäß bei 28 s erschöpft.

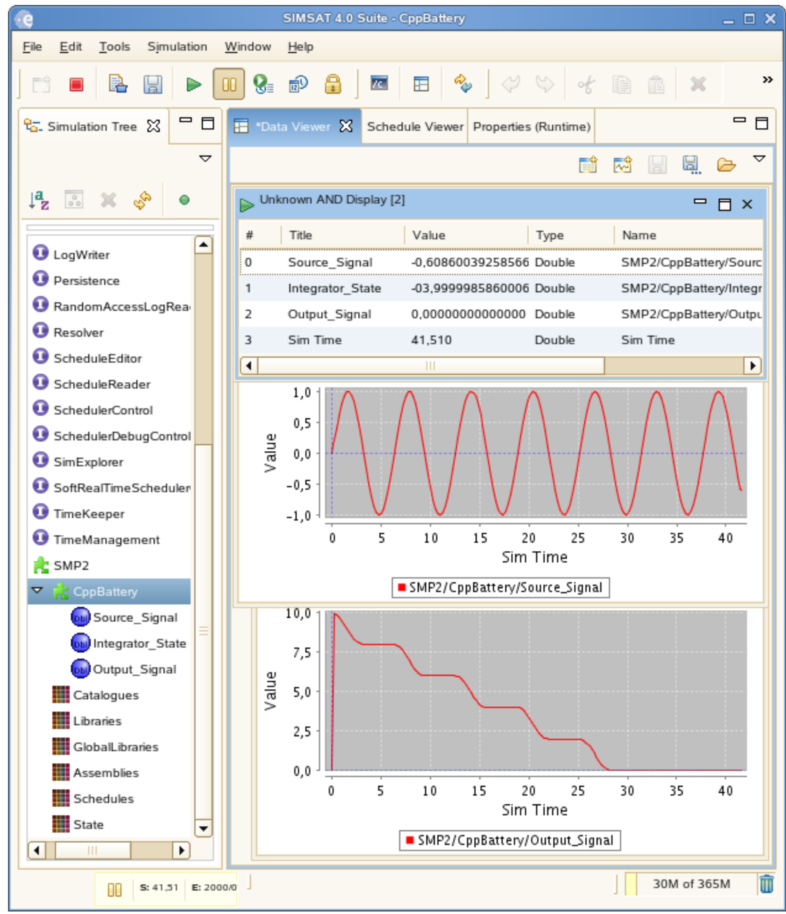


Abbildung 5: Simulation des Batteriemodells im Simulator SIMSAT

3.4 Zusammenfassung

Mit dem SMP2-Metamodell wurden zur Berechnung des Ladestandes eines vereinfachten Batteriemodells Catalogue, Assembly und Schedule erstellt. Das Verhalten wurde in den Aktivierungsfunktionen in C++ implementiert. Das kompilierte Modell wurde dann in *SIMSAT* simuliert.

Im Folgenden wird gezeigt, wie die Verhaltensimplementierung in C++ durch die Modellierung des Verhaltens in der grafischen Modellierumgebung für Blockdiagramme *Simulink* ersetzt werden kann.

4 Simulink

In diesem Abschnitt wird gezeigt, wie ein mit dem Programm *Simulink* modelliertes Verhalten als SMP2-Komponente transformiert und simuliert werden kann.

Dazu wird zunächst das Beispielmodell in Simulink modelliert und simuliert. Aus dem Modell wird dann mit dem Programm *Real-Time Workshop* Code generiert, der das Modellverhalten äquivalent zur Simulation in Simulink berechnet. Dieser Code wird mit dem Programm *MOSAIC* in einem SMP2-Modell gekapselt. Dieses Modell wird schließlich im Simulator *SIMSAT* simuliert.

Modellierung und Simulation mit Simulink

Simulink erweitert MATLAB um die Möglichkeit, dynamische Systeme grafisch zu modellieren. Die Modellierung bedient sich so genannter Funktionsblöcke. Ein Block kann als grafische Darstellung einer Funktion f verstanden werden. Er erhält Eingaben X und berechnet aufgrund dieser Eingaben, eigener Zustände Z und Parameter P zeitabhängig bestimmte Ausgaben Y. Eingaben und Ausgaben werden hierbei auch Signale genannt.

$Y = f(X, Z, P, t)$

Die Darstellung eines Systems als Blockdiagramm betont vor allem den Datenfluss. Es ist fraglich, welche Vor- oder Nachteile darin zu sehen sind. Cellier misst dieser Darstellung eine bessere Übersicht gegenüber einer expliziten Formulierung von differential-algebraischen Gleichungen bei [Cellier 2006, Seite 7].

Zur Simulation wird das Blockdiagramm in ein System von Differentialgleichungen (ODE - ordinary differential equations) umgeformt. Die Simulation besteht aus der wiederholten Lösung des Gleichungssystems. Die Lösung wird bei nicht simplen Systemen (z.B. solche die einen Integrator beinhalten) mit Näherungsverfahren berechnet. Das Intervall zwischen je zwei Berechnungsschritten ist die Schrittweite. Simulink bietet die Möglichkeit, die Schrittweite fest anzugeben, oder während der Simulation bedarfsabhängig zu ändern. Simulink stellt mehrere unterschiedliche Näherungsverfahren für Simulationen mit fester oder variabler Schrittweite zur Verfügung.

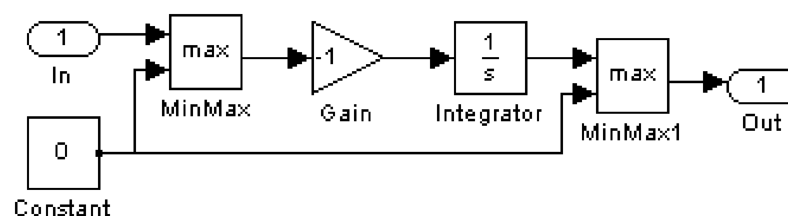


Abbildung 6: Batteriemodell in Simulink

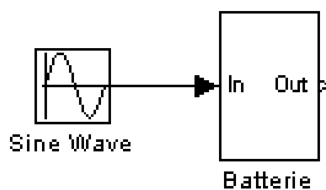


Abbildung 7: Umgebung für Batteriemodell in Simulink

4.1 Beispielmodell in Simulink

Zunächst wird das Beispiel des Batterieladestandes als Simulink-Modell implementiert (Abbildung 6).

Im Blockdiagramm der Abbildung lässt sich die Berechnungsformel für den Ladestand wiedererkennen:

$E(t) = max(E_0 + \int -max(P(t), 0)dt, 0)$

Das Eingangssignal wird an den ersten Maximum-Block geleitet, der wiederum den jeweiligen Wert weiterreicht, wenn er größer als 0 ist. In einem weiteren Block wird der Wert negiert, anschließend integriert, an den zweiten Maximum-Block und schließlich als Ausgangssignal aus dem Block hinaus gereicht. Dieses Batteriemodell stellt nach außen wieder einen Block dar, dessen Einbettung Abbildung 7 darstellt. Dort wird der Batterie als Eingangssignal einfach ein Sinussignal vorgeschaltet.

Die Kette dieser Blöcke wird für jeden Simulationsschritt einmal (und Teilketten davon bei Integration häufiger) durchgerechnet. Der Anfangsladestand E_0 ist hier nicht dargestellt. Er ist über den Anfangswert des Integratorblocks implizit gegeben.

Modelltrennung und Wiederverwendung ist in Simulink möglich, indem die Modelle als so genannte Simulink-Library zur Verfügung gestellt werden. Dieser Prozess ist einigermaßen umständlich, erfordert er doch mehrere Aktionen des Benutzers. Eine einfache Wiederverwendung von bestehenden Modellen ist in Simulink sonst nur durch Duplizieren der Modellimplementierung möglich, womit dann mehrere Instanzen desselben Modells gepflegt werden müssen.

Eine dritte Möglichkeit sind Verweise auf bestehende Modelle. Dann werden referenzierte Modelle allerdings vorkompiliert und als externe Funktion eingebunden. Optimierungen zwischen Modellen könnten dann nicht berücksichtigt werden. Abgesehen davon gelang es auch nicht, das Beispielmodell ohne Fehler mit diesem Ansatz zu simulieren.

Die Simulation mehrerer getrennter Komponenten mit Werteaustausch zieht unweigerlich weitere Probleme nach sich, die in dieser Studienarbeit nicht besprochen werden können. Siehe dazu auch Kapitel 6

Konfiguration und Simulation

Für die Simulation des Beispielmodells in Simulink sind verschiedene Einstellungen zu wählen.

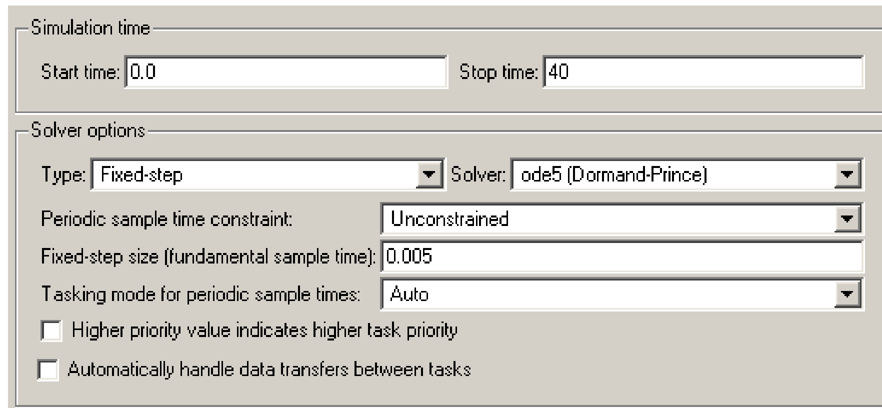


Abbildung 8: Simulationseinstellungen in Simulink

Das Beispiel soll 40 Sekunden lang simuliert werden, um die Entladung der Batterie (bei ca. 28 s erwartet) gut beobachten zu können.

Ferner wird zur Lösung der Differentialgleichungssysteme ein Algorithmus mit fester Schrittweite gewählt. Dies erfordert das später beschriebene Programm MOSAIC. Es unterstützt Simulation in Echtzeit, wofür eine feste Schrittweite benötigt wird.

Als Lösungsalgorithmus schlägt MOSAIC den genauesten der in Simulink verfügbaren Algorithmen vor [MOSAIC, S. 19]. Die Schrittweite von 0,005 Sekunden ist ein von MOSAIC vorgeschlagener Kompromiss zwischen Genauigkeit und Geschwindigkeit der Berechnung.

Als Startwert für den Integratorblock wurde ein Wert von 10 gewählt. Sonst sind alle standardmäßigen Parameter der Simulink-Blöcke und der Simulationseinstellungen so belassen worden.

Ergebnis

Die Kurve der Abbildung zeigt keine nennenswerten Unterschiede zu dem in C++ berechneten Verhalten. Alle Modelleigenschaften sind erfüllt worden.

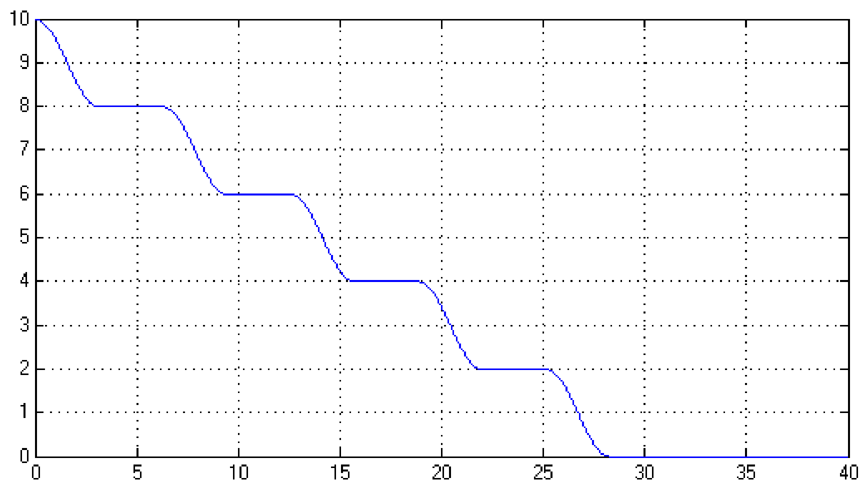


Abbildung 9: In Simulink berechneter Ladestand

4.2 Real-Time Workshop

Das als Ergänzung zu Simulink vertriebene Programm *Real-Time Workshop* (RTW) erzeugt C-Code aus einem Simulink-Modell. Das berechnete Verhalten ist mit dem bei einer Simulink-Simulation berechneten identisch. Der zeitliche Verlauf der Modellvariablen wird aber nicht wie in Simulink angezeigt, sondern in einer Datei gespeichert.

Der maßgebliche Vorteil besteht in der **vollständigen Verfügbarkeit des Modell-Quelltextes** mit allen notwendigen Berechnungen (im Folgenden RTW-Code genannt). Er kann so in anderen Code eingebettet und weiterverwendet werden.

Modellausführung im RTW-Code

Die Berechnung des Modellverhaltens findet in spezifischen Funktionen statt, die von der RTW-eigenen **Laufzeit-Bibliothek** heraus aufgerufen werden. Laufzeit-Bibliothek und Modellcode ergeben zusammen kompiliert eine ausführbare Datei, die die Berechnung durchführt [RTW 3, S. 6-4 ff].

Die spezifischen Modellfunktionen stellen unterschiedliche Aspekte der Berechnung zur Verfügung. So werden z.B. die Funktionen `MdlStart` und `MdlTerminate` einmal zu Anfang bzw. Ende der Berechnung aufgerufen, und sorgen für Initialiserung bzw. Freigabe von Daten. Die Funktionen `MdlOutputs`, `MdlUpdate` und `MdlDerivatives` werden hingegen je Zeitschritt mindestens einmal aufgerufen. Sie berechnen jeweils die Modellausgaben, die diskreten, bzw. die kontinuierlichen Zustände des Modells [RTW 3, S. 6-17 ff].

Alle an der Berechnung beteiligten Variablen werden vom Real-Time Workshop in einer großen globalen Datenstruktur names *rtModel* gehalten.

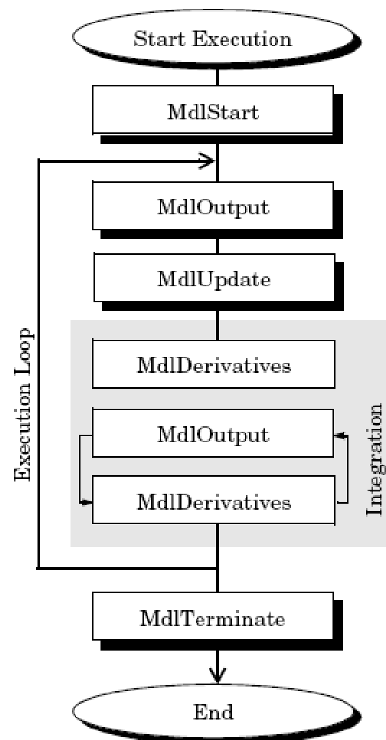


Abbildung 10: Steuerung der Funktionen aus dem Modellcodes des Real-Time Workshops. Quelle: [RTW User Manual 3]

Einstellungen

Weitere Einstellungen sind zur Erzeugung des Codes notwendig. So verlangt MOSAIC z.B. C statt C++-Code und die spezifische Zielplattform *Generic Real-Time Target with dynamic memory allocation*. Allgemein lässt sich der zu erzeugende Code auf eine bestimmte Plattform (AUTOSAR, Tornado, UNIX), einen bestimmten Zweck (Z.B.: Rapid Simulation, Rapid Prototyping) anpassen. Die Einstellungen lassen sich auch über die MATLAB-Shell treffen [RTW 7, S. 2-5, 2-9], [MOSAIC, S. 16 bis 28].

4.3 MOSAIC

Das vom niederländischen NLR (Nationaal Lucht- en Ruimtevaartlaboratorium) entwickelte und vertriebene Programm *MOSAIC* (Model-Oriented Software Automatic Interface Converter) bettet den im Real-Time Workshop erzeugten Code in ein SMP2-Modell ein. Dabei kann man sich einer grafischen Oberfläche oder der Kommandozeile zur Konvertierung der Modelle bedienen [MOSAIC].

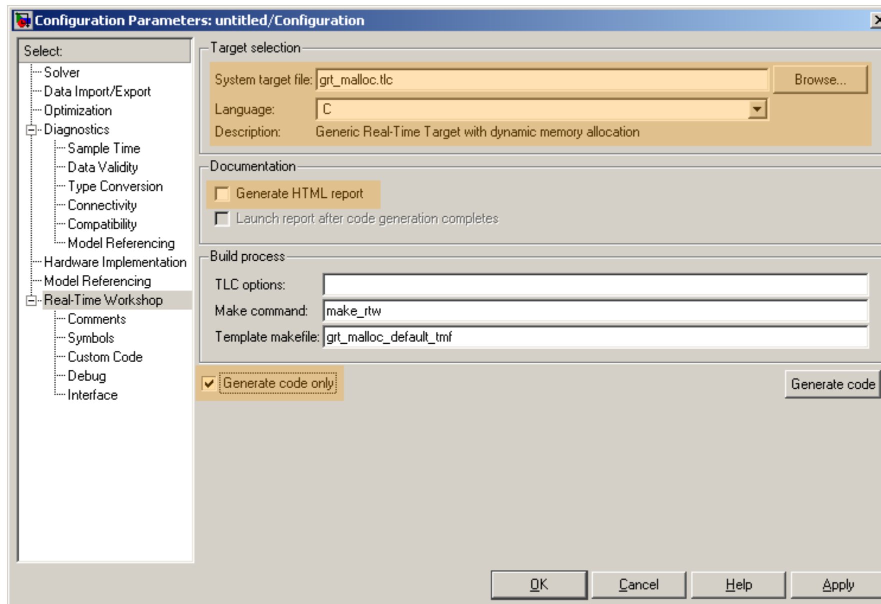


Abbildung 11: Screenshot der Konfiguration des Real-Time Workshops

SMP2-Ausführung von RTW-Code

MOSAIC ersetzt die Laufzeit-Bibliothek und damit die Ausführungssteuerung des Modellcodes durch eine Steuerung mit SMP2-Aktivierungsfunktionen (EntryPoints).

Dazu wurden zunächst **Schnittstellenfunktionen** deklariert, die wesentliche Steuerungsmechanismen als Codefragmente direkt aus dem Quelltext der RTW-Laufzeit-Bibliothek übernehmen (wie z.B. aus `grt_malloc_main.c`). Diese sind in der erzeugten Datei <modelname>_interf.c zu finden. Weitere Funktionalität wird von bestimmten Dateien der Laufzeit-Bibliothek beim Kompilieren eingebunden, weshalb MOSAIC das Kopieren einiger solcher Dateien benötigt [MOSAIC, S. 13 bis 15].

Folgender Codeabschnitt zeigt die Funktion `Batterie_Simulink_init`, deren Implementation ursprünglich aus `grt_malloc_main.c` stammt.

```
rtModel_Batterie_Simulink* Batterie_Simulink_init( ... )
{
    rtModel_Batterie_Simulink *S;
    [...]
    rtmiInitializeSizes(rtmGetRTWRTModelMethodsInfo(S));
    rtmiInitializeSampleTimes(rtmGetRTWRTModelMethodsInfo(S));
    RetVal = rt_SimInitTimingEngine(...);
    rt_ODECreateIntegrationData(rtmGetRTWSolverInfo(S));
    [...]
}
```

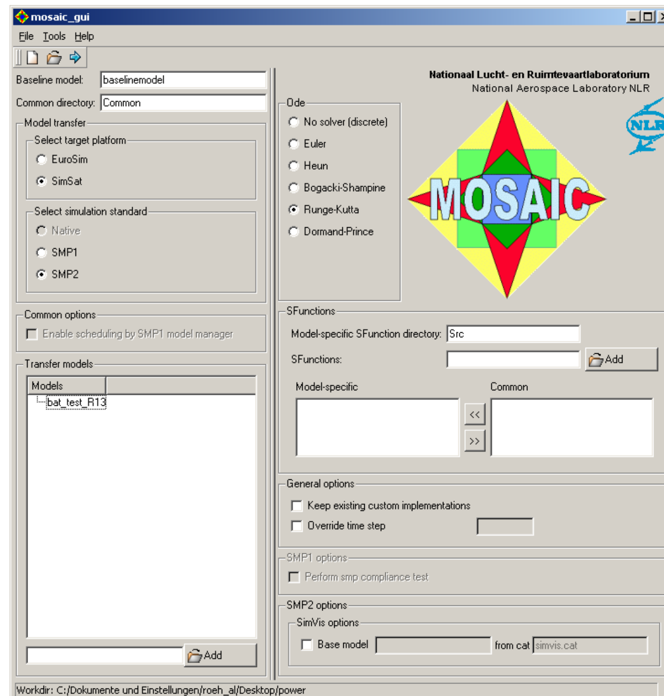


Abbildung 12: Screenshot des Transformationsdialogs von MOSAIC

Nach außen sehen diese Funktionen zunächst den RTW-Modellfunktionen ähnlich. Sie heißen `model_init`, `model_input`, `model_output`, `model_terminate` und `model_update`. Für das Beispielmodell wurde `model_input` nicht erzeugt, da es keine externen Eingaben erhält).

Für jede dieser Funktionen registriert MOSAIC nun eine **Aktivierungsfunktion** und steuert sie über denselben Schedule. Dabei werden `model_input`, `model_update` und `model_output` in dieser Reihenfolge für jeden Zeitschritt genau einmal ausgeführt. Diese führen wiederum die RTW-Modellfunktionen aus, mitunter mehrmals in einem Zeitschritt.

Die Funktionen `model_init` und `model_terminate` werden eigentlich gar nicht als Aktivierungsfunktionen benötigt, sondern immer direkt bei der Modellerstellung bzw. -zerstörung aufgerufen.

Veröffentlichung von Zustandsvariablen

MOSAIC veröffentlicht alle in den Arbeitsfeldern des RTW-Codes verfügbaren Eingabe- und Ausgabevariablen, die Variablen für kontinuierliche und diskrete Zustände, sowie Parameter und die vom Real-Time Workshop intern gehaltene Simulationszeit.

SMDL-Dokumente

MOSAIC legt einen Catalogue an, in dem die Aktivierungsfunktionen und die zu veröffentlichenden Variablen deklariert werden. Da die Modellsteuerung unabhängig von Modelleigenschaften ist, weist jeder erzeugte Catalogue die gleichen Aktivierungsfunktionen auf. Die zur Berechnung genutzten Variablen unterscheiden sich aber in Anzahl und Namen von Modell zu Modell.

Für das konkrete Beispielmodell wurden als Zustandsvariablen nur der Ausgang des zweiten MinMax-Blockes und der Ausgang des Gain-Blocks veröffentlicht (siehe Abbildung Simulink-Modell). Andere interne Blockvariablen stellte Real-Time Workshop mit den Standardeinstellungen nicht zur Verfügung. Deshalb wurde statt des Eingangssignals die Ausgabe des Gain-Blockes dargestellt. Gibt man Signalen des Simulink-Modells spezifische Namen, stehen sie im Code des Real-Time Workshop als konkrete Variablen zur Verfügung. Allerdings sind diese Variablen global und außerhalb der üblichen rtModel-Struktur deklariert. Deshalb wurde im Hinblick auf mögliche Seiteneffekte auf weitere Variablen verzichtet.

MOSAIC legt dazu auch noch ein Assembly mit einem Instanzierungsplan und einen Schedule mit einem Ausführungsplan an. Das **Assembly** wird nur zur Deklarierung der Modellinstanz genutzt. Die Variablen werden stattdessen explizit im Code instanziert.

Der **Schedule** führt die Aktivierungsfunktionen `input`, `update` und `output` in dieser Reihenfolge für jeden Zeitschritt aus. Um die gleiche Simulationsausführung wie bei der Simulink-Simulation zu erhalten, wurde ebenfalls ein Zeitschritt von 0,005 Sekunden verwendet. Die Laufzeit wurde allerdings nicht beschränkt. Das Programm SIMSAT verfügt intern über ein SMDL-Metamodell für SMP2 Version 1.2 und kann deshalb SMDL-Dokumente validieren. Der von MOSAIC erzeugte Schedule konnte von SIMSAT aber nicht validiert werden. Es ist zu vermuten, dass MOSAIC nicht mit der SMP2-Version 1.2 komforme Dokumente erzeugt.

Einschränkungen

Das Programm MOSAIC wird momentan nicht weiter entwickelt und stellt Anwender so vor einige Probleme.

Die verfügbare Version 7.1 erfordert das (veraltete) Release 14 mit Service Pack 3 von MATLAB, Simulink und RTW (jeweils in Version 7.1, 6.3 bzw. 6.3). Code der mit *Real-Time Workshop* eines anderen Releases erzeugt wurde, kann nicht verarbeitet werden.

Darüber hinaus erzeugt MOSAIC für die zwei unterstützten Simulatoren *EuroSim* und *SIMSAT* unterschiedliche Dateien, womit die von SMP2 angestrebte Simulatorunabhängigkeit nicht gewährleistet ist. Die Programm-Dokumentation begründet dies mit einer unterschiedlichen Interpretation von SMP2 in EuroSim und SIMSAT [MOSAIC, Fußnote S. 42].

Der von MOSAIC verwendete SMP2-Standard scheint in mehreren Aspekten veraltet zu sein. Zum einen ist die erzeugte Schedule-Datei nicht konform mit dem aktuellen

Standard, zum anderen scheinen im Code SMP2-Konzepte anders verstanden worden zu sein.

Weiterhin ist *MOSAIC* sowohl in Ausführung als auch vom erzeugten Code her an die Plattform Windows gebunden. Dies stellt sich als starke Einschränkung heraus. Denn der in dieser Studienarbeit verwendete Simulator *SIMSAT* wird ab Version 2 nur noch für Linux entwickelt. Um die von MOSAIC erzeugten SMP2-Modelle unter der aktuellen SIMSAT-Version 4 laufen zu lassen, sind daher zusätzlich zur Anpassung der SMP2-Verwendung auch Anpassungen in Hinsicht auf die Plattform erforderlich. Es ist zu erwarten, dass auch in *MOSAIC* erzeugte *Eurosim*-Modelle mit neueren Versionen nicht kompatibel sind.

Weitere Anforderungen stellt MOSAIC an die transformierbaren Simulink-Modelle. Damit *Real-Time Workshop 6.3* Code aus einem Simulink-Modell generieren kann, müssen bestimmte Beschränkungen eingehalten werden [MOSAIC, S. 16], [RTW 3, S. 1-4]. So können z.B. keine Blöcke genutzt werden, die durch MATLAB-Funktionen implementiert wurden, oder die Funktionen in MATLAB-Dateien aufrufen. Im MOSAIC User Manual werden weiterhin Probleme bestimmten Signalschleifen beschrieben [MOSAIC, S. 17]. Eine Schleife verbindet Eingang und Ausgang eines Blockes mit einer Signalkette über andere Blöcke. Wenn keine dieser Blöcke eine Zustandsverzögerung (etwa durch die Speicherung des Zustandes) aufweist, spricht man von einem "direct feed-through loop". Simulink kann diese Schleifen simulieren, anscheinend aber nicht der mit RTW Version 6.3 erzeugte Code, obwohl nur in [MOSAIC], nicht aber in [RTW 3] explizit beschrieben. In neueren Versionen des Real-Time Workshops könnte dieses Problem behoben worden sein. Neuere Versionen können aber nicht mit dem aktuellen MOSAIC verwendet werden.

Zusammenstellung der erfoderlichen Codedateien

Der von MOSAIC für ein Modell erzeugte Code besteht aus **RTW-Modellcode**, aus **SMP2-Code-Dateien** die mit dem SMP2-Language-Mapping erzeugt wurden und aus **Vermittlungscode**, der die RTW-Modellfunktionen auf SMP2-EntryPoints abbildet. Diese Dateien werden aufgrund von Regeln in von MOSAIC bereitgestellten Makefiles zu einer Bibliotheksdatei kompiliert.

Wegen genannter Inkompatibilitäten von MOSAIC mit dem aktuellen SIMSAT ließ sich diese aber nicht in SIMSAT der aktuellen Version 4.0.1–2 simulieren. Stattdessen mussten mehrere Anpassungen vorgenommen werden, um eine simulierfähige Bibliothek zu erzeugen.

Da die SMP2-Dateien als nicht standardkonform angenommen werden müssen, wurden sie mit dem aktuellen SMP2-Language-Mapping aus dem von MOSAIC erzeugten Catalogue neu erzeugt und ersetzt. Dabei wurden auch neue Makefiles erzeugt, die statt der von MOSAIC bereitgestellt genutzt wurden. Sie mussten an einigen Stellen um geänderte Dateinamen korrigiert werden.

MOSAIC bindet beim Kompilieren der Modellbibliotheksdatei eine Windows-Bibliothek mit ausgewählten Dateien der RTW-Laufzeit-Bibliothek ein. Diese werden vom

RTW-Modellcode benötigt. Die entsprechenden Dateien mussten daher unter Linux neu kompiliert und in den Linkprozess der Modellbibliothek eingebunden werden, damit alle Abhängigkeiten sowohl beim Kompilieren als auch beim Linken erfüllt werden können. Auch hierfür mussten die Makefiles angepasst werden.

Anpassung des Quelltextes

Die so kompilierte Bibliothek führte beim Laden in SIMSAT immer noch zu Fehlern. Sie deuten auf eine geänderte Verwendung der SMP2-Mechanismen.

So scheint es ein Problem bei der Registrierung der Funktion `terminate` zu geben. Die Ursache des Fehlers ist aber nicht klar. Es lassen sich keine Unterschiede zur Definition der anderen Aktivierungsfunktionen ausmachen, der Fehler tritt aber nur bei der Registrierung dieser Funktion auf. Nach Entfernung der expliziten Registrierung im Code läuft das Modell in SIMSAT. Anscheinend führt SIMSAT aufgrund der Informationen im Schedule oder aufgrund der Informationen der Modell-Basisklasse `EntryPointPublisher` die Registrierung implizit durch, oder gar aufgrund gleicher Namen der Funktionen im Quelltext und im Catalogue. Tatsächlich simulierte SIMSAT das Modell immer noch problemlos, nachdem auch die Registrierung der anderen Modellfunktionen entfernt wurde.

MOSAIC registriert teilweise überflüssige Aktivierungsfunktionen. Während der Aufruf der Funktion `input` modellabhängig notwendig sein kann, werden die Funktionen `init` und `terminate` ausschließlich direkt aufgerufen und erfordern so eigentlich keine Registrierung. Dies kann aber nicht Grund des eben beschriebenen Fehler gewesen sein, da EntryPoints auch registriert werden können, wenn sie nicht vom Schedule aufgerufen werden.

SMDL-Anpassungen

Die von MOSAIC erzeugte Schedule-Datei kann nicht als mit SMP2 Version 1.2 konform validiert werden. Sie könnte entsprechend angepasst werden, so dass sie konform zu SMP2 1.2 ist. Stattdessen wurde jedoch mit dem Modelleditor in SIMSAT ein neuer Schedule erstellt, der die gleiche Zeitsteuerung beschreibt.

Da die SMP2-relevanten Quelltexte und SMDL-Dokumente unterschiedlicher Herkunft sind, muss darauf geachtet werden, dass sie gleiche Namen zur Identifikation der jeweiligen EntryPoints, Variablen und Modelle nutzen.

4.4 Simulation in SIMSAT

Mit der so kompilierten Modellbibliothek, dem von MOSAIC erzeugten Assembly und dem neu erzeugten Schedule konnte das Modell in SIMSAT simuliert werden. Die Abbildung zeigt den Simulationslauf der aus dem Simulink-Modell erzeugten SMP2-Komponente.

4.5 Zusammenfassung

Es ist gelungen, in Simulink modelliertes Verhalten über die Programme RTW und MOSAIC SMP2-konform zu simulieren. Dazu wurde mit RTW Code für das Simulink-Modell erzeugt und mit dem Programm MOSAIC SMP2-konform gekapselt. Im Code mussten dann noch Änderungen vollzogen werden, um unter Linux mit allen Abhängigkeiten kompiliert und unter SIMSAT 4.0.1–2 für Linux simuliert werden zu können.

Die Änderungen umfassten den Kompilierprozess und die SMP2-relevanten Dateien. Da diese teilweise modellabhängig sind, sind manche der Änderungen bei jeder Transformation eines Modells erneut durchzuführen.

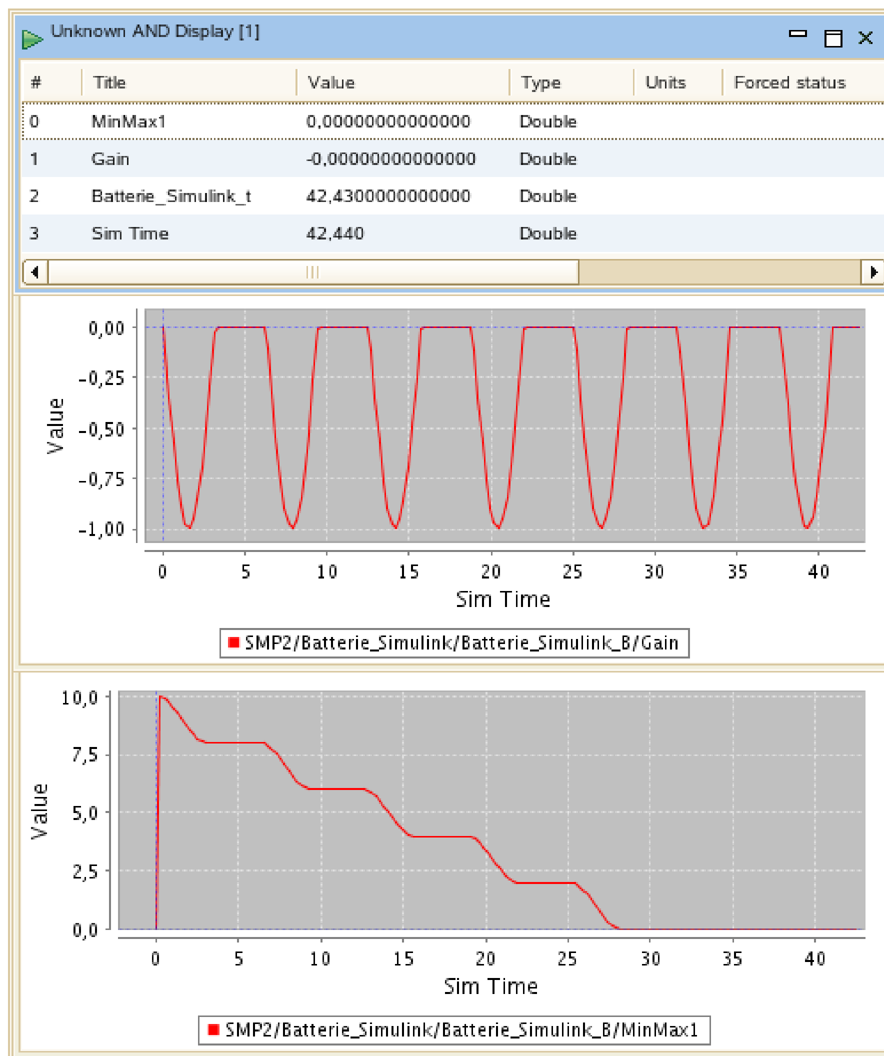


Abbildung 13: Simulationslauf des Simulink-Modells in SIMSAT

5 Modelica

Modelica ist eine objektorientierte Sprache für die Modellierung komplexer physikalischer Systeme [Fritzson 2004]. Sie wird seit 1996 unter maßgeblicher Beteiligung des DLR entwickelt. In der Industrie erlangt Modelica zunehmende Bedeutung als Alternative zu Simulink aufgrund mehrerer Vorteile. Ihre Objektorientierung ermöglicht einfache Änderung oder Spezialisierung bestehender Modelle. Darüber hinaus bringt Modelica Standardkomponenten verschiedener Bereiche, z.B. Elektrik, Mechanik oder Thermik mit. Mit ihnen lassen sich schnell Systementwürfe erstellen. Anschließend können verwendete Standardkomponenten entsprechend spezialisiert werden. Modelica und die Bibliothek mit Standardkomponenten sind frei verfügbar. Es gibt kommerzielle und frei verfügbare Programme, die die Modellierung unterstützen und die Simulation ermöglichen.

Eine Transformation von Modelica-Modellen in SMP2-konformen C++-Code ist bisher noch nicht dokumentiert. Grundsätzlich gibt es aufgrund der Offenheit der Sprache mehrere Modelica-Compiler, die aus einem Modelica-Modell C- oder C++-Code erzeugen. Es sollte daher möglich sein, in einem ähnlichen Ansatz wie *MOSAIC* die Steuerungslogik solchen Codes gegen eine SMP2-Steuerung auszutauschen.

5.1 Beispielmodell in Modelica

Mit *Modelica* ist prinzipiell eine Modellierung von Blockdiagrammen wie mit Simulink und mehr möglich.

Folgende Abbildung zeigt die Implementierung des Batterieladestandes als Modelica-Modell, im Programm *Dymola* grafisch modelliert und visualisiert.

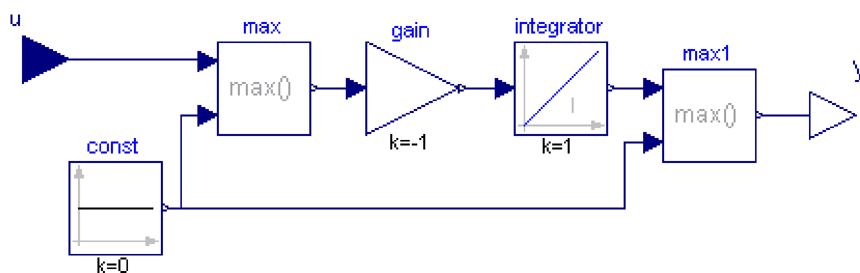


Abbildung 14: Batteriemodell in Dymola

Während das Programm Simulink zur Beschreibung seiner Simulationsmodelle sein eigenes Format benutzt, ist mit dem Begriff Modelica gerade nur die Sprache zur Beschreibung, also das Format selbst gemeint. Deshalb finden Modellierung und Simulation in Programmen statt, die Modelica-Programme verarbeiten können. Grafische und textuelle Repräsentation sind dabei äquivalent. Simulink-Modelle werden hingegen ausschließlich in der grafischen Umgebung Simulinks oder über Schnittstellen anderer Programme

bearbeitet, obwohl auch dort eine hierarchische Textrepräsentation zugrunde liegt. Für grafische Modellierung sind gegenwärtig die Programme *Dymola*, *MathModelica*, *SimulationX* und einige andere verbreitet. Mit einem Texteditor kann das Programm auch direkt geschrieben werden. Für die Simulation sind teilweise andere Programme erforderlich. Benutzt werden können *Dymola*, *OpenModelica*, *SimulationX*, *Mosilab* und einige wenige mehr.

Das obiger Abbildung entsprechende Modelica-Programm ist im Folgenden vollständig dargestellt. Im ersten Abschnitt werden alle genutzten Block-Instanzen angegeben (teilweise mit Angabe vom Default abweichender Parameter), im zweiten Abschnitt werden die mit `u` und `y` bezeichneten Ein- und Ausgänge der Blöcke dann verbunden.

```
model Batterie
  Modelica.Blocks.Interfaces.RealInput u;
  Modelica.Blocks.Interfaces.RealOutput y;
  Modelica.Blocks.Sources.Constant const(k=0);
  Modelica.Blocks.Math.Max max;
  Modelica.Blocks.Math.Gain gain(k=-1);
  Modelica.Blocks.Continuous.Integrator integrator;
  Modelica.Blocks.Math.Max max1;
equation
  connect(u, max.u1);
  connect(const.y, max.u2);
  connect(max.y, gain.u);
  connect(gain.y, integrator.u);
  connect(integrator.y, max1.u1);
  connect(const.y, max1.u2);
  connect(max1.y, y);
end Batterie;
```

Mit Modelica sind Modellgrenzen sehr einfach modellierbar. Als Beispiel wurde die Batterie als eigenständiges Modell entwickelt, das in einem Umgebungsmodell als Komponente genutzt und mit dem Wert eines Sinus-Blockes als Eingabe versorgt wird.

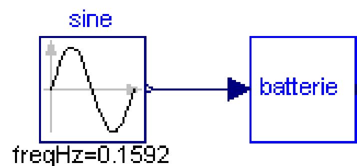


Abbildung 15: Umgebungsmodell für Batterie in Dymola

Die Frequenz des Sinus ist mit 0,1592 angegeben. Der Sinus-Block ist in Modelica standardmäßig mit einer Periode von 1 s bzw. einer Grundfrequenz von 1 Hz definiert.

Sonst wird jedoch üblicherweise 2π s als Periode bzw. $(2\pi)^{-1} \approx 0{,}1592$ als Grundfrequenz verwendet, so auch von Simulink. Um die Simulationsergebnisse mit den anderen Beispielmodellen vergleichen zu können, wird daher die Frequenz des Sinus-Blockes auf 0,1592 gesetzt.

Die Abbildung zeigt die Simulation obigen Modelica-Modells mit dem Sinus-Signal als Eingang in dem Programm OpenModelica. Das berechnete Verhalten ist mit dem in Simulink und C++ berechneten nahezu identisch.

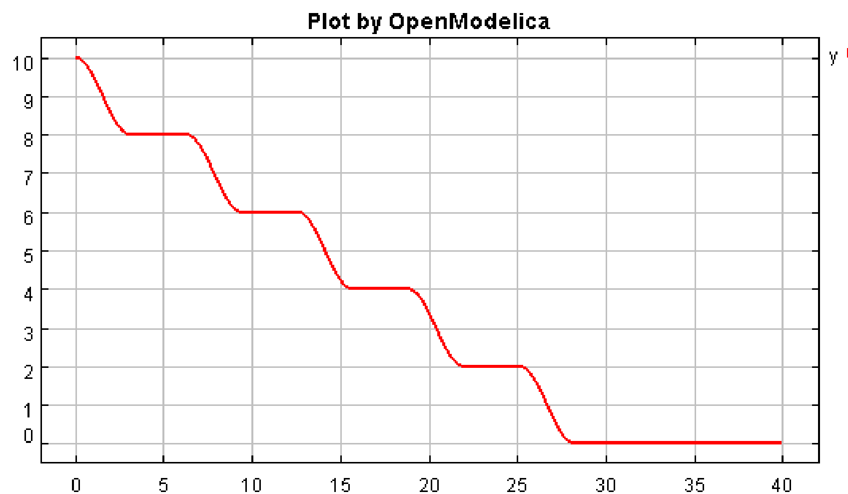


Abbildung 16: In OpenModelica berechneter Ladestand

5.2 Codeerzeugung

Viele der zur Simulation von Modelica-Modellen verwendbaren Programme erzeugen mit einem Modelica-Compiler Modellcode für die Simulationsausführung. Dieses Zwischenprodukt kann als Verhaltensimplementation in einem SMP2-Modell gekapselt werden. Dazu muss die Laufzeitbibliothek eines solchen Programms modifiziert werden.

Das Programm *Dymola* beinhaltet den leistungsfähigsten der verfügbaren Modelica-Compiler. Er unterstützt alle Aspekte der Sprache und stellt eine breite Auswahl von Lösungsalgorithmen zur Verfügung. Aber die Laufzeitbibliothek ist nicht im Quelltext verfügbar, da Dymola ein kommerzielles Programm ist. Deshalb kann es in dieser Arbeit nicht verwendet werden.

OpenModelica

Der Modelica-Compiler *OpenModelica* [Fritzson 2006] erzeugt ebenfalls als Zwischenprodukt Modellcode, der auf eine Laufzeitbibliothek zurückgreift und von dieser gesteuert

wird. Diese lässt sich aber einsehen und modifizieren, da OpenModelica als OpenSource unter der OSMC-Lizenz (Open Source Modelica Consortium) [OSMC] verfügbar ist. OpenModelica ist noch in der Entwicklung. Es unterstützt daher nicht den gesamten Modelica-Sprachstandard.

Modellausführung in OpenModelica

Der in *OpenModelica* enthaltene *OpenModelica Compiler* übersetzt ein Modelica-Modell in mehreren Schritten in **Modellcode**. In diesen Schritten werden zunächst Vererbungen, Parametrisierungen und andere Modellkonzepte aufgelöst. Dann wird dieses eingeebnete Modell in ein hybrides System algebraischer und differenzieller Gleichungen umgewandelt. Die Lösung dieser Gleichungen wird im Modellcode berechnet.

Der Modellcode besteht aus Funktionen bestimmter Bedeutung. Die **Laufzeitbibliothek** von Modelica kennt diese Funktionen und ruft sie für zur Berechnung bestimmter Teilschritte nach eigenem Ermessen auf. In der Laufzeitbibliothek findet also die Steuerung der Simulationsberechnung statt.

Um eine lauffähige Simulation zu erzeugen, wird der Modellcode mit der Laufzeitbibliothek gelinkt. Das Resultat ist sofort ausführbar, da die Laufzeitbibliothek eine main-Funktion definiert.

Ein Simulationslauf in OpenModelica ist durch eine angegebene Endzeit begrenzt. Die Simulationszeit wird nicht mit der Rechnerzeit synchronisiert. Es ist also keine Echtzeitsimulation möglich.

5.3 SMP2-Kapselung

Die Steuerung der Berechnung ließ sich in den Quelltexten der Laufzeitbibliothek so modifizieren, dass eine Funktion für die Initialisierung einer Berechnung, eine Funktion für das Durchführen eines Berechnungsschrittes und eine Funktion für das Aufräumen einer Berechnung extrahiert werden konnte. Diese Funktionen wurden mit allen benötigten Abhängigkeiten zu einer neuen Laufzeitbibliothek kompiliert. Wird Modellcode mit dieser Laufzeitbibliothek verlinkt, entsteht eine SMP2-konforme Komponente.

SMP2-Ausführung von OpenModelica-Code

Bei einer SMP2-konformen Simulation soll die Berechnungssteuerung durch Registrierung von Aktivierungsfunktionen auf den Simulator übertragen werden.

Es bieten sich folgende drei Aktivierungsfunktionen zur Berechnungsteuerung an: *Initialisieren*, *Berechnungsschritt ausführen* und *Aufräumen*.

In der Funktion `dassl_main` aus der Datei `solver_dasrt.cpp` wird die Simulationsberechnung mit dem DASSL-Lösungsalgorithmus (DASSL - differential-algebraic system solver, [Brenan 1996]) durchgeführt. Der größte Teil dieser Funktion wurde also auf die

drei neuen Funktionen `dassl_main_init`, `dassl_main_step` und `dassl_main_Finalise` aufgeteilt.

Diese Funktionen werden aber nicht direkt als Aktivierungsfunktionen verwendet. Stattdessen entstanden mit der Datei `SMP2-interface` die drei entsprechenden Funktionen `InitialiseHandler`, `StepHandler` und `FinaliseHandler`.

Man könnte erwarten, dass sie nur ihre jeweils entsprechende `dassl_main`-Funktion aufrufen und sonst nichts machen. Beim `StepHandler` ist das der Fall:

```
void StepHandler()
{
    dassl_main_step(dw, stop, stepSize);
}
```

Die anderen beiden Funktionen mussten aber noch Code aus anderen Dateien übernehmen, der für die jeweiligen Aufgaben nötig ist. In einer zukünftigen Implementierung sollte der übernommene Code besser auf Ebene der `dassl_...`-Funktionen gekapselt werden.

Allgemein ist für alle zu benutzenden Modelica-Modelle nur die Aktivierungsfunktion zu registrieren. Sie führt den Berechnungsschritt durch. `InitialiseHandler` und `FinaliseHandler` können direkt im Modellcode aufgerufen werden. Dafür bietet sich die `Configure`-Methode und der Destruktor der SMP2-Modelle an.

Für diese Studienarbeit ist die Aktivierungsfunktion `Modelica_Batterie_udpate` definiert worden, die unter Anderem die Funktion `StepHandler` aufruft.

Veröffentlichung von Zustandsvariablen

In einem von OpenModelica generierten Modellcode werden zu allen Modellvariablen Informationen bereitgestellt. Als Modellvariablen zählen hier die Ein- und Ausgänge aller benutzten Teilblöcke und rekursiv deren Teilblöcke. Für die Menge all dieser Variablen werden im Modellcode Felder zum Halten verschiedener Informationen definiert: Name der Variable, ihr Wert und auch Kommentare. Dabei haben die Informationen der gleichen Variablen auch gleiche Positionen. In weiteren Feldern sind auch die Parameter der Blöcke in ähnlicher Weise zugreifbar.

Während die Bezeichnung und Bedeutung dieser Felder feststeht, variiert ihr Inhalt (und die Reihenfolge der Variablen) von Modell zu Modell. Auch eine kleine Änderung des Modells kann nach Neukompilierung zu einer anderen Reihenfolge der Variablen in den Feldern führen. Es gilt also, anhand bekannter Parameter wie z.B. dem Namen einer Variablen, die Feldpositionen aller zur Anzeige gewünschten Variablen herauszufinden.

Für den Nachweis der prinzipiellen Eignung von OpenModelica-Code als Implementierungssubstitut im Rahmen dieser Studienarbeit wurden nur drei Variablen über SMP2

registriert: die Werte für Eingang und Ausgang des Modells, sowie die OpenModelica-interne Simulationszeit. Ihre Positionen in den Feldern werden hier der Einfachheit halber als konstant angenommen. Ihre Werte werden in der (einzigen) Aktivierungsfunktion entsprechenden SMP2-Variablen zugewiesen, die im Catalogue vereinbart und vom SMP2-Language-Mapping bereits registriert wurden.

```
void Modelica_Batterie::_Modelica_Batterie_update()
{
    // Call OpenModelica runtime step code
    StepHandler();

    // Update field variable by explicitly copying its value
    Model_Output = (Smp::Float64) globalData->algebraics[3];
    Model_Input = (Smp::Float64) globalData->algebraics[4];
    Model_Time = (Smp::Float64) globalData->timeValue;
}
```

SMDL-Dokumente

Für das Beispiel dieser Studienarbeit wurden alle SMDL-Dokumente manuell erstellt. Der Catalogue besteht aus der einen Aktivierungsfunktion `Modelica_Batterie_update` und den drei SMP2-Variablen `Model_Output`, `Model_Input` und `Model_Time`. Im Assembly wurden alle drei Variablen mit 0 initialisiert. Im Schedule wird die Aktivierungsfunktion über ein Simulationszeitereignis gesteuert, das alle 0,001 Sekunden also 5 mal häufiger als der eigentliche Zeitschritt stattfindet. Dies erfordert das weiter unten beschriebene Phänomen, in dem OpenModelica nur alle fünf Funktionsaufrufe die Simulationszeit weiterzählt.

Sammlung und Kompilierung des Codes

Der Modellcode besteht aus folgenden Dateien:

- SMP2-Modellcode: Aus dem Catalogue per SMP2-Language-Mapping erzeugt. Die Aktivierungsfunktion, die `Configure`-Methode und die Cleanup-Methode wurden entsprechend geändert. Makefiles für den Kompilierprozess und Instanzierungscode für das Modell wurden ebenfalls per Language-Mapping aus einem *SMDL-Package* erzeugt.
- OpenModelica-Modellcode: vom OpenModelica-Compiler aus dem Modelica-Modell erzeugt.
- OpenModelica-SMP2-Interface: besteht aus den bereits beschriebenen Funktionen `InitialiseHandler`, `StepHandler` und `FinaliseHandler`.

Bis auf die Änderungen im SMP2-Modellcode wurden alle Dateien einfach zusammenkopiert. Beim Kompilieren muss das OpenModelica-SMP2-Interface den OpenModelica-Modellcode finden, und der OpenModelica-Modellcode muss den Quelltext der OpenModelica-Laufzeitbibliothek finden. Die Makefiles mussten dann so angepasst werden, dass die Objekte des SMP2-Modell- und Packagecodes, des OpenModelica-Modellcodes, des OpenModelica-SMP2-Interfaces und der OpenModelica Laufzeitbibliothek erzeugt und miteinander zu einer Bibliotheksdatei verlinkt wurden.

5.4 Simulation in SIMSAT

Zur Ausführung in SIMSAT sind das Assembly, der Schedule, die Bibliotheksdatei und eine SIMSAT-Projektdatei nötig, in der Assembly, Schedule und Bibliothek referenziert sind. Zusätzlich wird die vom OpenModelica-Compiler erzeugte Datei mit den Initialparametern für das Modell benötigt. Diese muss im Arbeitsverzeichnis der SIMSAT-Projektdatei liegen, damit sie bei der Ausführung gefunden wird. Wird sie nicht gefunden, bricht SIMSAT den Ladevorgang mit einer Fehlermeldung ab.

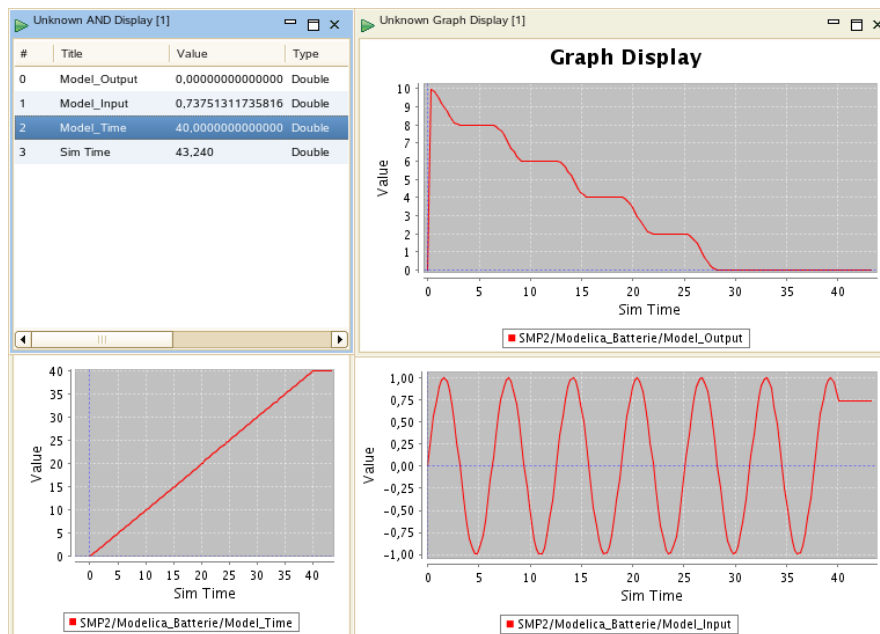


Abbildung 17: Simulationslauf des Modelica-Modells in SIMSAT

Bei der Ausführung des Beispielmodells fiel auf, dass OpenModelica immer fünf aufeinander folgende Berechnungsschritte als einen Zeitschritt behandelt. Um dies nachzuprüfen wurde die von OpenModelica intern gehaltene Simulationszeit über die SMP2-Variable `Model_Time` angezeigt. Das Modelica-Modell wurde auf eine Schrittweite von 0.005 s konfiguriert. Deshalb ist im Schedule ein Fünftel dieser Zeit als Schrittweite angegeben. Es resultiert die korrekte Berechnung des Batterieladestandes (Abbildung 17). Im

Gespräch mit Entwicklern von OpenModelica stellte sich heraus, dass über eine bestehende Variable der richtige Zeitschritt herausgefunden werden könne. Außerdem wurde der Vorschlag unterbreitet, die Modellausführung nicht über modifizierten Runtime-Library-Code zu steuern, sondern mit von Modelica unterstützten Modellmechanismen zu synchronisieren. Es ist in späterer Arbeit zu überprüfen, ob so die Steuerung zuverlässig auf SMP2-Mechanismen übertragen werden kann.

Weiterhin war zu beobachten, dass weitere Steuerungsmechanismen im OpenModelica-Code existieren, die eine Berechnung über das bei OpenModelica anzugebende Simulationsende hinaus verhindern. Im Beispiel bricht die Berechnung bei 40s ab, woraufhin alle Werte für die Folgezeit konstant bleiben. Daher sollte konsequenterweise die Aktivierungsfunktion im Schedule auf eine feste Aufrufanzahl festgelegt, oder die verbliebenen Steuerungsmechanismen aus dem OpenModelica-Code entfernt werden.

5.5 Zusammenfassung

Das von dem Programm OpenModelica erzeugte Verhalten eines Modelica-Modells wurde erfolgreich in ein SMP2-Modell umgewandelt und simuliert. Dazu wurde die Laufzeitbibliothek von OpenModelica entsprechend manipuliert. Diese Änderung ist für alle Modelle gleich. Für das Modell wurden zudem alle SMDL-Dokumente und daraus SMP2-Modell-Code erzeugt. Diese Schritte sind für jedes zu transformierende Modell erneut nötig.

6 Vergleich und Ausblick

Im Folgenden wird für alle vorgestellten drei Wege grob zusammengefasst, wie geeignet ihre Verwendung zur Modellierung im DLR-Projekt *Virtueller Satellit* ist. Außerdem wird zusammengestellt, welche Arbeit noch erforderlich ist, um jeden der Wege im Rahmen des Projektes automatisch einzusetzen.

Modellierung

Alle drei Wege benutzen das SMP2 C++ Mapping, um SMP2-konformen Quelltext zu erzeugen. Im Programm SIMSAT ist z.B. ein einsatzbereites C++ Mapping verfügbar. Die Modellierung des Modellverhaltens im erzeugten C++-Code erfordert deshalb keinen weiteren technischen Aufwand. Zusätzliche Werkzeuge oder Transformationen werden nicht benötigt. Die Automatisierung beschränkt sich auf das Auslösen des Language Mappings und des Kompilierens. Somit ist der C++-Ansatz am leichtesten zu benutzen. Allerdings hat sich herausgestellt, dass Fachingenieure, die im Rahmen dieser Projekte mit der Modellierung betraut werden, keine einschlägigen Erfahrungen mit Softwareentwicklung in C++ haben, und also C++ als Sprache zur Modellierung nicht geeignet ist.

Ingenieure sind hingegen im Umgang mit grafischen Modellierkonzepten solcher Programme wie Simulink oder Dymola vertraut. Diese Programme nutzen häufig Blockdiagramme für Darstellung und Modellierung dynamischer Systeme. Sie bieten auch direkt Konzepte der Modellierungsdomäne an. Zwar wird so die Modellierung erleichtert, aber zur Erzeugung SMP2-konformer Modelle sind Anpassungen am Simulationscode nötig, der von diesen Programmen erzeugt wurde.

Werkzeugentwicklung/Arbeitsautomatisierung

Soll Simulink zur Modellierung eingesetzt werden, muss zunächst ein grobes Simulink-Modell erzeugt werden, das die spezifizierten Ein- und Ausgänge einer Komponente beinhaltet. Hier bietet sich an, mit einem Simulink-Metamodell zu arbeiten. Eine entsprechende Transformation müsste noch entwickelt werden. Nach der Modellierung muss der Real-Time Workshop Code erzeugen. Sowohl dafür erforderliche Einstellungen als auch die Auslösung können über MATLAB-Scripte automatisiert werden. Anschließend muss die MOSAIC-Transformation ausgelöst werden. Dieser Schritt ist über die Kommandozeilenschnittstelle von MOSAIC automatisierbar. Soll mit einer MATLAB-Version oder Modellen gearbeitet werden, die von MOSAIC momentan nicht unterstützt werden, ist die Weiterentwicklung von MOSAIC erforderlich.

Wird Modelica zur Modellierung eingesetzt, ist eine ähnliche Metamodell- Transformation zur Bereitstellung der Modellstruktur als Modelica-Modell nötig. Zur Codeerzeugung kann der Aufruf des OpenModelica-Compilers über die Kommandozeile automatisiert werden. Die anschließend benötigte Transformation in eine SMP2-Komponente

ist prinzipiell möglich, wie in dieser Studienarbeit gezeigt werden konnte. Aber ein automatischer, verlässlicher Transformationsprozess, der ein möglichst breites Spektrum an Modelleigenschaften unterstützt, muss entwickelt werden. Hierfür ist eine genauere Aufwandsabschätzung nötig.

Der bei MOSAIC und im Modelica-Beispiel verfolgte Ansatz lässt sich generell auf andere geeignete Modellierumgebungen übertragen. Modellierumgebungen eignen sich, wenn sie Simulationscode erzeugen können, und wenn sowohl der Simulationscode als auch seine Steuerung modifiziert werden kann, so dass die Steuerung mit SMP2-Mechanismen umgesetzt werden kann und die Zustandsvariablen veröffentlicht werden können. Um zusätzlich Komponenten einzeln transformieren zu können, müssen solche Modellierumgebungen eine Möglichkeit bieten, Blockeingänge von außerhalb des Modells eingeben zu können.

6.1 Zusammenfassung

Das Verhalten von SMP2-Modellen ist mit C++ ohne viel technischen Aufwand implementierbar. Dieser Ansatz ist aber nicht für das Projekt *Virtueller Satellit* geeignet.

Simulink eröffnet einen geeigneten Modellieransatz. Die zur Umwandlung in ein SMP2-Modell benötigten Programme Real-Time Workshop und MOSAIC weisen unter Umständen Versionsprobleme auf. Die einzelnen, momentan von Hand ausgelösten Transformationsschritte müssen noch automatisiert werden.

Andere Werkzeuge bieten ebenfalls geeignete Modellieransätze und können für SMP2-Simulationen genutzt werden. Modelica bietet einige Vorteile gegenüber einer Modellierung mit Simulink. Eine SMP2-Transformation muss aber umfangreich entwickelt werden.

6.2 Ausblick

In dieser Studienarbeit wurden in Simulink oder Modelica erstellte Simulationen als SMP2-Modell verpackt und ausgeführt. In einem nächsten Schritt sollten auch Teilmodelle einzeln in SMP2-Modelle transformiert werden und dann gemeinsam simuliert werden. So könnten mit verschiedenen Werkzeugen modellierte Komponenten gemeinsam simuliert werden.

Dazu müssen Modellen externe Eingangssignale zugeschaltet werden können. Dies muss über SMP2-Mechanismen ermöglicht werden, um die Entkopplung der Modelle zu wahren. Weiterhin muss eine Möglichkeit geschaffen werden, Ausgangsvariablen und Eingangsvariablen verschiedener Modelle miteinander zu verknüpfen. Schließlich muss auch eine Logik implementiert werden, wann Teilschritte eines jeden Modells berechnet werden und wann Werte über die Modellverknüpfungen transportiert werden sollen.

Probleme ergeben sich, wenn Modelle unterschiedliche Schrittweiten oder gar variable Schrittweiten aufweisen. Schleifen in der Modellverknüpfung werden besondere Probleme

aufwerfen, wenn sich in der Schleife keine verzögernden Blöcke befinden, oder diese Eigenschaft nicht überprüft werden kann. Generell stellt sich daher die Frage, mit welcher Genauigkeit das modellierte Verhalten berechnet werden kann und wie die erhaltenen Ergebnisse zu interpretieren sind.

Die angestrebte SMP2-Simulation verschiedener Modelle entspricht im Prinzip einer Co-Simulation der Teilmodelle. Es ist deshalb auch zu erwarten, dass so erstellte Simulationen langsamer laufen als vollständig in z.B. Simulink entwickelte. Simulink kann alle Gleichungen der Simulation modellübergreifend optimieren. Dies ist nach Trennung in SMP2-Komponenten nicht möglich.

A Anhang

A.1 Konvertierung von Simulink- und Modelica-Modellen

Dymola-Code nach Simulink exportieren

Der aus Modelica-Gleichungen erzeugte, ausführbare C-Code eines in Dymola erzeugten Modelica-Modells kann als s.g. *DymolaBlock* in Simulink als *S-function MEX block* genutzt werden. Dafür wird eine grafische Schnittstelle in Simulink bereitgestellt, durch die ein *DymolaBlock* in Simulink-Diagrammen genutzt werden kann. Ein *DymolaBlock* funktioniert ab MATLAB Version 5.3 und Simulink 3.0.

Dymola-Simulator aus MATLAB heraus aufrufen

Dymola kann aus einem Modelica-Modell ausführbaren Code erzeugen. Dieser kann als Stand-Alone-Simulation kompiliert werden. Dabei wird eine ausführbare Datei namens `dymosim.exe` erzeugt. Diese lässt sich als Funktion direkt aus Matlab heraus aufrufen. Es müssen Experimentdaten und Simulationsparameter übergeben werden. Als Ausgabe wird eine Ergebnisdatei im MATLAB-Format erzeugt.

Simulink-Modelle in Modelica-Modelle konvertieren

Die Firma *Claytex* (http://www.claytex.com) entwickelt das Werkzeug *Simelica*, das Simulink-Modelle in eine äquivalente Modelica-Repräsentation konvertieren kann. Es enthält eine eigene Modellbibliothek, die die meisten der in Simulink zur Modellierung verfügbaren Komponenten bereitstellt. Das fertige Modelica-Modell soll ohne Einschränkungen in jeder beliebigen Modelica-Umgebung weiter benutzt werden können [Dempsey 2003].

Literatur

[Brenan 1996] K.E. Brenan, S.L. Campbell, L.R. Petzold: *Numerical Soluation of Initial-Value Problems in Differential-Algebraic Equations*, Society for Industrial and Applied Mathematics, Philadelphia 1996, republication of 1989 edition from North-Holland, New York

[Cellier 2006] F.E. Cellier, E. Kofman: *Continuous System Simulation*, Springer, New York 2006

[Dempsey 2003] M. Dempsey: *Automatic translation of Simulink models into Modelica using Simelica and the AdvancedBlocks library*, Proceedings of the 3rd International Modelica Conference, Linköping, November 3-4, 2003

[Fritzson 2004] P. Fritzson: *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley-IEEE Computer Society Press, 2004

[Fritzson 2006] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, K. Nystrom, L. Saldamli, D. Broman, A. Sandholm: *OpenModelica - A Free Open-Source Environment for System Modeling, Simulation, and Teaching*, Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design, Munich, 2006

[MOSAIC] W. Lammen, J. Moelands: *MOSAIC 7.1 User Manual - Automated Model Transfer from MATLAB7.1/Simulink to ESA's Simulation Model Portability SMP2 standard and the real-time simulation engine EuroSim Mk4*, National Aerospace Laboratory Netherlands, 2006, distributed electronically alongside MOSAIC 7.1

[OSMC] *Open Source Modelica Consortium License*: http://www.ida.liu.se/labs/pelab/modelica/OpenModelica/OSMC/OpenSourceModelicaConsortium-bylaws.pdf (abgerufen 25.10.2008)

[RTW 3] *Real-Time Workshop User's Manual Version 3.11*, Mathworks Inc., 1999, distributed electronically alongside MATLAB Release 11

[RTW 7] *Real-Time Workshop User's Manual Version 7.1*, Mathworks Inc., 2008, distributed electronically alongside MATLAB Release 2008a

[SIMSAT] *SIMSAT-Information*: http://www.egos.esa.int/portal/egos-web/products/Simulators/SIMSAT/ (abgerufen 25.10.2008)

[SMP2] *Simulation Model Portability 2.0 Handbook*, EGOS-SIM-GEN-TN-0099, ESA/ESOC, Darmstadt, 2005

[Tiller 2004] M. Tiller: *Introduction to Physical Modeling with Modelica*, Kluwer Academic Publishers, Norwell MA USA, 2004, 2nd print