Solar-Controller optimieren mit Simulink-Polysun Cosimulation

Andreas Wolf[1], Roland Kurmann[1], Conrad Gähler[2], Andreas Witzig[3]

[1] Vela Solaris AG, Stadthausstrasse 125, Winterthur, Schweiz,
Tel.: +41 55 220 71 00, E-Mail: info@velasolaris.com, www.velasolaris.com

[2] Siemens Schweiz AG, Building Technologies Division, Zug, Schweiz

[3] Zürcher Hochschule für Angewandte Wissenschaften ZHAW, Institute of Computational Physics, ICP, Winterthur, Schweiz

1. Einleitung

Die Steuerung gebäudetechnischer Energiesysteme ist für die Maximierung des Solarertrags sowie für die Optimierung der Anlageneffizienz von grösster Bedeutung. In realen Projekten zeigt sich, dass die Programmierung der Steuerung in der Planung und Projektierung oft nicht mit der effektiv in der Realisierungsphase umgesetzten Steuerung übereinstimmt (siehe normierte Standardplanungsphasen in Figur 1). Das vorausgesagte gute Systemverhalten wird damit oft nicht erreicht.

Figur 1: Normierte Planungsphasen [3]. Im Kontakt mit Benutzern von Solar-Planungssoftware wird beobachtet, dass die Tools in der Regel nur in Phase 1-3 eingesetzt werden. Falls es gelingt, echte Regler in diesen Phasen realitätsnaher abzubilden, werden die Simulationen auch bis in Phase 5 und 6 eingesetzt und die in der Planung erwarteten Solargewinne werden in Realität auch erreicht.

Es ist deshalb von grosser Relevanz, dass die Steuerung aus der Simulation exakt mit der in Realität eingesetzten Steuerung übereinstimmt. Im Folgenden werden zwei Möglichkeiten dargestellt, wie für das Planungstool Polysun die Steuerung durch einen bereits existierenden Algorithmus vorgegeben werden kann.

2. Polysun Plugin-Controller

Als erste Stufe einer Programmierschnittstelle bietet Polysun die benutzerspezifische Erweiterung der Steuerungen. Steuerungs-Plugins können in Matlab, Python oder in der Programmiersprache Java erstellt werden, was es erlaubt, komplizierte Systemzusammenhänge abzubilden.

2.1 Matlab

Die Matlab-Steuerung ist eine Plugin-Steuerung, die die Steuerungsaufrufe aus Polysun an eine Matlab-Funktion weiterleitet. In der Steuerung wird der Name der Funktion in Matlab gesetzt. Diese Funktion muss im Pfad von Matlab oder im Startverzeichnis sein. Die Matlab-Funktion muss folgende Funktionsargumente haben:

```
function [controlSignals, logValues, timepoints] =
control(simulationTime, status, sensors, sensorsUsed, properties,
propertiesStr, preRun, controlSignalsUsed, numLogValues, stage,
fixedTimestep, verboseLevel, parameters)
```

Die in der Matlab-Steuerung angegebene Matlab-Funktion wird in jedem Zeitschritt von Polysun aufgerufen. Sie muss in jedem Zeitschritt aufgrund der Input-Parameter die Steuersignale für Polysun berechnen, die als Rückgabeparameter über das Interface in die Simulation zurückgeben werden.

2.2 Python

Die RPC-Steuerung ist eine Plugin-Steuerung, die die Steuerungsaufrufe aus Polysun über einen Remote Procedure Call (RPC) an einen RPC-Server weiterleitet. In der Steuerung werden der Name der Funktion und die URL des Servers gesetzt, z.B. `controlFlowrate` und `http://localhost:2102/control`. Die Funktion im RPC-Server muss folgende Funktionsargumente haben:

```
control(simulationTime, status, sensors, sensorsUsed, properties,
propertiesStr, preRun, controlSignalsUsed, numLogValues, stage,
fixedTimestep, verboseLevel, parameters) => controlSignals, logValues,
timepoints
```

Die in der Steuerung angegebene RPC-Funktion wird in jedem Zeitschritt von Polysun aufgerufen und muss in jedem Zeitschritt aufgrund der Input-Parameter die Steuersignale für Polysun zurückgeben.

2.3 Java

Plugin-Steuerungen können selbst in der Programmiersprache Java erstellt werden. Voraussetzung zum Erstellen von Steuerungs-Plugins sind Java-Programmier-kenntnisse und eine Java-Entwicklungsumgebung.

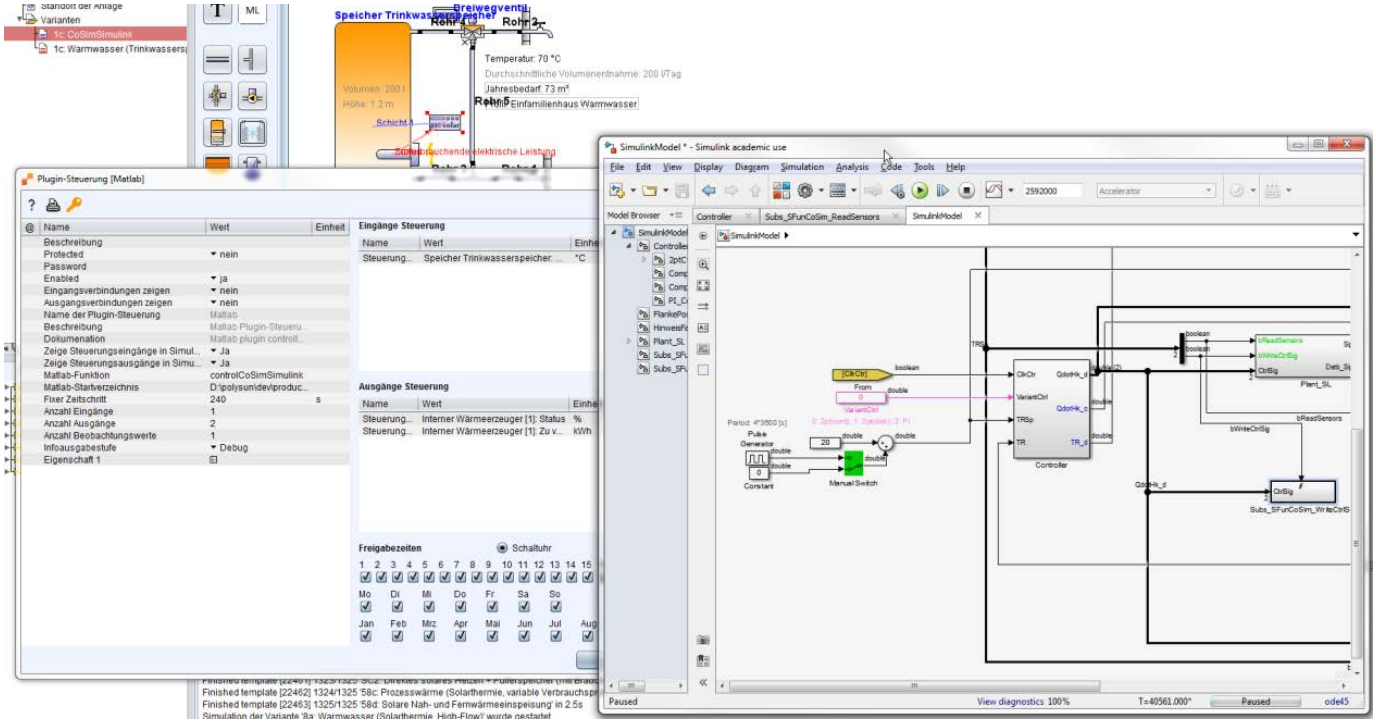
Plugin-Steuerungen müssen das Java Interface `IPluginController` implementieren. Die wichtigste Methode darin ist ist `control().` Diese Methode wird in jedem Zeitschritt aufgerufen und berechnet die Steuerungssignale aus den Sensorsignalen. Die Signatur ist

```
control(int simulationTime, boolean status, float[] sensors, float[]
controlSignals, float[] logValues, boolean preRun, Map<String, Object>
parameters).
```

Die Werte für die gesetzten Steuerungseingänge werden im `sensors`-Array in jedem Zeitschritt zur Verfügung gestellt. Daraus berechnet die Plugin-Steuerung die Kontrollsignale (`controlSignals`-Array) für die gesetzten Steuerungsausgänge.

3. Cosimulation Polysun-Matlab/Simulink

Als zweite Möglichkeit, den Regler detailliert abzubilden, wurde gemeinsam mit dem Steuerungshersteller Siemens eine Schnittstelle entwickelt, die die Steuerung extern zur Solarsimulation definiert und mit einer Cosimulation die Steuerungslogik und die Systemdynamik gekoppelt analysiert. Der Steuerungshersteller kann somit mit seinen eigenen Tools – im vorliegenden Beispiel Matlab/Simulink – die bereits vorhandenen Controller-Algorithmen in der Simulation zur Anwendung bringen.



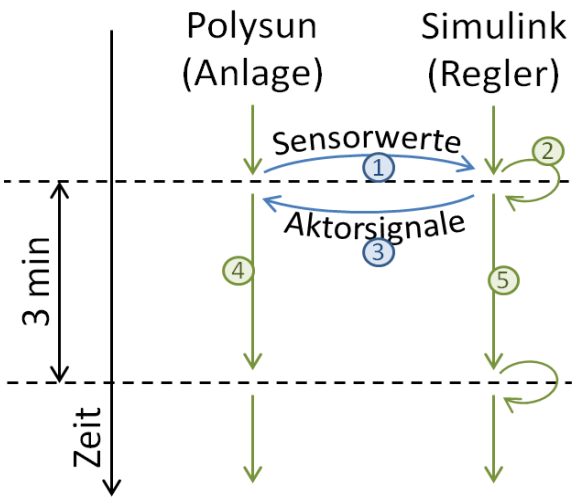
Figur 2: Polysun Simulation mit Matlab/Simulink-Cosimulation.

Aus Sicht der Polysun-Systemsimulation erfolgt die Kopplung folgendermassen: Sensoren (z.B. Temperaturfühler) und Aktoren (z.B. Pumpen) werden wie bisher im Solarsystem identifiziert und für die Cosimulation zeitabhängig ausgewertet. Aus Sicht der Matlab/Simulink-Controllersimulation werden die aktuellen Sensordaten übernommen, die die Regelstrecke eindeutig definieren. Die Regler können ein

„Gedächtnis“ haben (Integrationsglied im PI-Regler) und entsprechend der in Simulink aufgebauten Reglerlogik die Aktorsignale auswerten. Letztere werden in einem gemeinsamen Zeitschrittraster an die dynamische Systemsimulation zurückgegeben.

Die Zeitschrittverfahren der Systemsimulation und der Reglersimulation laufen gleichzeitig und parallel zueinander. In jedem Simulationsstrang läuft eine unabhängige Berechnung gemäss dem jeweiligen dynamischen Simulationsmodell. In der neuen Schnittstelle werden die Outputs der beiden Programme alle 3 Minuten zwischen den beiden Programmen ausgetauscht. Diese Synchronisationsschrittweite hat sich für die untersuchten Beispiele als ausreichend kurz erwiesen.

Ein wesentlicher Vorzug der hier vorgestellten Cosimulations-Schnittstelle gegenüber andern gängigen Ansätzen (z.B. Building Control Virtual Testbed BCVTB): Der Regler kann die Stellbefehle, die er aufgrund der Sensorsignale berechnet, sofort an die Strecke zurückgeben, so wie dies z.B. bei Proportional- oder Zweipunktreglern in der Realität auch der Fall ist. Dies wird durch die Aufrufs-Reihenfolge der beiden Programme innerhalb eines Synchronisations-Zeitschritts erreicht, wie in Figur 3 dargestellt.



Figur 3:. Ablaufschema der Cosimulation mit Polysun und Matlab/Simulink. Bei Erreichen des Synchronisierungs-Zeitpunkts werden (1) die Sensorsignale von Polysun (Anlagenmodell) zu Simulink (Regler) übertragen. Dann werden (2) die Direct Feedthrough-Terme des Reglers gerechnet (P-Regler, 2-Punkt-Regler etc). Die daraus resultierenden Aktor-signale werden (3) sofort zu Polysun zurück übertragen, so dass sie für die Anlagen-Simulation für die weiter-führende Berechnung (4) unmittelbar berücksichtigt werden. Die Polysun-Simulation führt dabei mit variablem Zeitschritt so viele Simulationsschritte aus, bis zum Zeitpunkt t+3min die nächste Kopplung initiiert wird. Parallel dazu wird dann (5) auch der restliche Teil des Reglers (zeitkontinuierliche Integratoren, z.B. bei PI-Regler) für den nächsten Cosimulations-Zeitschritt simuliert.

Zum Datenaustausch wird die Simulink-Bibliothek `MatConsoleCtl` verwendet. Daraus werden so genannte Assert-Callbacks zur Unterbrechung der Simulationen verwendet [2]. Diese bieten die Funktion `set_param` an, um die Simulation zu starten, pausieren, weiterzuführen und zu stoppen. Die entsprechenden Commands `start`, `pause`, `continue` und `stop` sind interne Anfragen für entsprechende Aktionen. Sie werden im Simulationsablauf nicht sofort ausgeführt. Simulink erledigt zuerst jene Schritte, die nicht unterbrochen werden können, wie zum Beispiel Aufgaben aus dem numerischen Gleichungslöser. Dies muss beim Timing der Daten-Synchronisation berücksichtigt werden.

Die dynamischen Zeitschritte sowohl in Polysun als auch in Simulink werden entsprechend der automatischen Zeitschrittanpassung häufig auch kleiner als 3min, sie können jedoch in einer gekoppelten Simulation nie länger als 3min werden.

4. Resultate und Diskussion

Über die heutigen Praxisanwendungen hinaus ergeben sich in der Kopplung der verschiedenen Simulationstools interessante Möglichkeiten zur verbesserten Steuerung von Solarsystemen und zur Integration externer Algorithmen in die gekoppelte dynamische Systemsimulation. Die beiden vorgestellten neuen Kopplungsmöglichkeiten ergänzen bereits bestehende Interfaces [4] und setzen einen zusätzlichen Fokus auf die Steuerung von HLK-Systemen mit Solaranlagen.

Im Polysun Plug-In Controller wird angeboten, die Steuerungslogik extern vorzugeben. Die Zeitschrittsimulation wird von Polysun vorgegeben, wobei über das Plug-In der Zeitschritt beeinflusst werden kann. Im Vergleich zu ungekoppelten Simulationen (typische Simulationszeit = 30 Sekunden für eine Ganzjahres-Simulation) dauert die Simulation mit Plug-In Controller in der Regel etwas länger (typische Simulationszeit = 10 Minuten), was für die üblichen Anwendungen von Polysun als Planungstool immer noch als hinreichend schnell angesehen werden darf.

In der Kopplung zwischen Polysun und Matlab/Simulink werden zwei fortlaufende Zeitschrittsimulationen miteinander synchron abgearbeitet. Diese Vorgehensweise ist aus konzeptionellen Gründen sehr interessant und öffnet die Möglichkeit zur synchronen Kopplung zu diversen anderen Tools. Es muss jedoch im gegenwärtigen Stand mit Laufzeiten von 6-8 Stunden gerechnet werden, was den Anwendungsbereich vorläufig auf Spezialanwendungen beschränkt.

5. Referenzen

[1] Polysun-Dokumentation, http://www.velasolaris.com/files/tutorial_de.pdf

[2] Simulink-Dokumentation, https://ch.mathworks.com/help/simulink/ug/control-simulations-programmatically.html

[3] Schweizer Norm SIA 102: Ordnung für Leistungen und Honorare der Architektinnen und Architekten

[4] A. Witzig, et.al.: Solarsimulation in verschiedenen Anwendungsbereichen: Polysun als universelles Plugin, 19. OTTI Symposiums für Thermische Solarenergie, Mai 2009, Kloster Banz, Bad Staffelstein, Deutschland, http://velasolaris.com/files/publikation_2009-05-otti-solarthermie-polysun-plugin.pdf