AUTOMATISCHE ÜBERSETZUNG KOMPLEXER FLUGSTEUERUNGSSOFTWARE VON SIMULINK/STATEFLOW NACH SCADE – EIN ERFAHRUNGSBERICHT[1]

G. Walde*, R. Luckner*

***Technische Universität Berlin (TUB)**
Institut für Luft- und Raumfahrt
Fachgebiet Flugmechanik, Flugregelung und Aeroelastizität

Zusammenfassung

Der Entwicklungsprozess sicherheitskritischer Software für komplexe Flugzeugsysteme, wie für das Flugsteuerungssystem, unterliegt einem strengen und aufwändigen Prozess. Dieser ist im Rahmen der Zulassung des Flugzeuges einzuhalten. Ein wichtiges Ziel ist es, diesen Prozess effizient zu gestalten. Ein Ansatz dazu ist die automatische Generierung des Quellcodes aus Modellen, die als Blockschaltbilder oder Zustandsdiagramme bereits im Entwurfsprozess der Funktionen - meistens mit den Werkzeugen Matlab, Simulink und Stateflow - entstehen. In der Luftfahrt wird für die Codegenerierung häufig der Codegenerator KCG der SCADE Suite eingesetzt, da dieser qualifizierbar ist. Dafür muss das Modell in Simulink/Stateflow aus dem Auslegungsprozess zunächst nach SCADE übersetzt werden. In diesem Beitrag wird die Machbarkeit der automatischen Übersetzung von Simulink/Stateflow nach SCADE mit dem SCADE Suite Gateway for Simulink für ein existierendes Modell untersucht. Als Anwendungsbeispiel dienen die Regelgesetze eines automatischen Flugsteuerungssystems, deren Modell aufgrund seiner Größe und der verwendeten Simulink/Stateflow-Funktionalitäten herausfordernd und deshalb als Beispiel gut geeignet ist. Damit kann der Übersetzungsprozess im Bezug auf Einsetzbarkeit und seine Grenzen evaluiert werden und es können notwendige Änderungen im Modellierungsprozess abgeleitet werden, die die Übersetzung vereinfachen.

Nomenklatur

EC	Embedded Coder® (Produkt von The Mathworks™)	RTWec	Real-Time Workshop Embedded Coder® (Produkt von The Mathworks™)
FMRA	Fachgebiet für Flugmechanik, Flugregelung und Aeroelastizität der Technischen Universität Berlin	SCADE	Safety Critical Application Development Environment (Produkt von Esterel Technologies™)
GUI	grafische Oberfläche, engl. *Graphical User Interface*	SF	Stateflow® (Produkt von The Mathworks™)
LAPAZ	Luftarbeitsplattform für die allgemeine Zivilluftfahrt	SL	Simulink® (Produkt von The Mathworks™)
MCF	engl. *Master Configuration File*	SW	Software
MTOM	max. Abflugmasse, engl. *Maximum Take-Off Mass*	UCF	engl. *User Configuration File*

1 EINLEITUNG

Der Verifikationsprozess von sicherheitskritischer Avionik-Software nach RTCA DO-178C [9] hat einen erheblichen Anteil an den Entwicklungskosten von Avionikgeräten. Er liegt zwischen 35% (siehe [1]) und 75% (siehe [2]). Durch eine Automatisierung von Prozessschritten unter Einsatz nach RTCA DO-330 [11] qualifizierter Werkzeuge können diese Kosten gesenkt werden, da im Gegensatz zu nicht-qualifizierten Werkzeugen Schritte zur Verifikation der automatisch generierten Ergebnisse ganz oder teilweise entfallen können.

Die Werkzeuge MATLAB® und Simulink®/Stateflow® werden häufig in der modellbasierten Entwicklung von Regelungssystemen eingesetzt, da sie die Auslegung, (grafische) Modellierung und Simulation der Systeme unterstützen. Aus solchen Simulink/Stateflow-Modellen kann mit dem Codegenerator Embedded Coder® (Erweiterung von MATLAB) automatisch Code generiert werden. Der Codegenerator wird jedoch nicht als qualifizierbares Werkzeug (nach RTCA DO-330 [11]) angeboten. Stattdessen werden Verifikationswerkzeuge mit einem sogenannten *Qualification Kit* angeboten, siehe [3]. Mit diesen Werkzeugen kann z.B. der generierte Quellcode statisch analysiert werden, um das Auftreten bestimmter Fehlerklassen zu finden bzw. auszuschließen.

Für den Software-Entwicklungs- und Verifikationsprozess ist es vorteilhaft, nach dem Reglerentwurf ein qualifizierbares Werkzeug für die Codegenerierung, z.B. den Codegenerator KCG® der SCADE Suite®, zu verwenden. Wenn dafür bestehende Simulink/Stateflow-Modelle genutzt werden sollen, müssen diese zunächst in die SCADE-Sprache übersetzt werden.

Im Forschungs- und Technologievorhaben CERTT-FBW23 wird die Machbarkeit der automatischen Übersetzung von Simulink/Stateflow nach SCADE® mit dem SCADE Suite Gateway for Simulink® für ein existierendes Modell untersucht. Als Anwendungsbeispiel

[1] Simulink® und Stateflow® sind eingetragene Marken von The MathWorks, Inc.
SCADE Suite® und SCADE Suite Gateway™ for Simulink® sind eingetragene Marken von Esterel Technologies.

dienen die Regelgesetze für die Flugsteuerung des Luftarbeitsflugzeuges Stemme S15, die in den Forschungs- und Technologievorhaben LAPAZ und LAPAZ II entwickelt und im Flugversuch erprobt wurden. Das Modell ist aufgrund seiner Größe und der verwendeten Simulink/Stateflow-Funktionalitäten herausfordernd und deshalb als Beispiel gut geeignet.

1.1 Motivation

Ein häufiger Anwendungsfall in der Luftfahrtbranche für die Übersetzung von Simulink/Stateflow nach SCADE ist die Überführung eines Auslegungsmodells aus dem Systementwicklungsprozess in ein Entwurfsmodell (engl. *Design Model,* nach [10]) im modellbasierten Softwareentwicklungsprozess. Die RTCA DO-331 fordert bezüglich Modellierungssprachen:

> *"For each modeling language, reference the data that unambiguously defines the syntax and semantics of the language with respect to the type of software life cycle data the model represents. This may require limiting the use of some features of the modeling language."* RTCA DO-331, Kapitel MB.11.23.b (Software Model Standards).

Hierbei hat die Sprache SCADE einen Vorteil gegenüber Simulink und Stateflow, da sie formal beschrieben ist, siehe Kapitel 3.2.3 in [7]. Der Sprachumfang ist auf vergleichsweise wenige, dafür jedoch sichere Sprachelemente begrenzt. In Verbindung mit einem qualifizierbaren Codegenerator (SCADE KCG®) wird dadurch die Nachweisführung nach RTCA-DO178C [9] und RTCA DO-331 [10] vereinfacht, da Verifikationsschritte entfallen können, siehe [7].

Der oben genannte Anwendungsfall soll durch die Evaluation des SCADE Suite Gateway for Simulink in diesem Beitrag betrachtet werden.

Eine weitere Anwendung ist die Übersetzung von Simulink und Stateflow nach SCADE als Zwischenschritt zur weiteren Übersetzung z.B. nach Lustre zu Verifikationszwecken (z.B. *Model Checking*). In [14], [15] und [16] werden hierfür komplexe Modelle aus dem Avionikbereich von Simulink und Stateflow über SCADE oder weitere Sprachen nach Lustre übersetzt. Die Übersetzung, die teilweise mit dem SCADE Suite Gateway for Simulink erfolgt, wird nicht im Detail beschrieben. Stattdessen geht es im Wesentlichen um die Verifikation von Eigenschaften des übersetzten Modells mit Hilfe formaler Methoden.

[17] und [18] beschreiben die Übersetzung von Simulink nach Lustre mit einem prototypisch entwickelten Werkzeug. Es werden z.B. die Grundprinzipien (*Type Inference*, *Clock Inference*, *Hierarchical Translation*) der Übersetzung von Simulink nach Lustre beschrieben. Dem hier eingesetzten Werkzeug SCADE Suite Gateway for Simulink (im Folgenden Gateway genannt) liegen auch die oben genannten Prinzipien zugrunde (siehe [4]), obwohl es sich um ein anderes Werkzeug handelt. Das ist nicht verwunderlich, da Lustre und SCADE geschichtlich eng zusammen hängen. Die Sprache Lustre bildet den textuellen Unterbau der grafischen Sprache SCADE, siehe Kapitel 4.1 in [5]. Weitere Informationen und Quellen zum geschichtlichen Hintergrund von SCADE sind in [5] und [6] zu finden.

1.2 Fragen der Evaluation

Folgende Fragen sollen in der Evaluation des Gateways untersucht werden.

1) Ist das Gateway für die Übersetzung bereits existierender, komplexer Modelle geeignet? Mit dem Begriff komplex ist hier die Eigenschaft gemeint, dass ein Modell eine noch überschaubare Größe (eins bis zwei Ebenen, weniger als hundert Blöcke) deutlich überschreitet und eine Vielzahl verschiedener Sprachelemente nutzt.
2) Wie leicht können Probleme bei der Übersetzung identifiziert und verstanden werden?
3) Welche Probleme existieren bei der Übersetzung?
4) Sind diese ursächlich in den eingesetzten (oder nicht eingesetzten) Modellierungsrichtlinien zu finden?
5) Welche Richtlinien für die Modellierung in Simulink und Stateflow müssen eingehalten werden, um Probleme bei der Übersetzung zu vermeiden?
6) Wie hoch ist der (erzielbare) Automatisierungsgrad bei der Übersetzung?
7) Wie kann die Vollständigkeit und Korrektheit der Übersetzung verifiziert werden?

2 ANWENDUNGSBEISPIEL

Als Anwendungsbeispiel dienen die Regelgesetze für die Flugsteuerung des Luftarbeitsflugzeuges Stemme S15, die in den Forschungs- und Technologievorhaben LAPAZ und LAPAZ II entwickelt und im Flugversuch erprobt wurden. Der Flugversuchsträger, den BILD 1 zeigt, hat eine Spannweite von ca. 18 m, ein MTOM von ca. 1000 kg und kann eine Nutzlast von ca. 150 – 300 kg aufnehmen.

BILD 1. Luftarbeitsflugzeug Stemme S15

Die LAPAZ – Projekte hatten drei Partner, Stemme AG als Verbundführer, Institut für Luftfahrtsysteme der Universität Stuttgart (zuständig für die fehlertolerante Plattform) und das Fachgebiet für Flugmechanik, Flugregelung und Aeroelastizität (FMRA) der Technischen Universität Berlin. Das FMRA entwickelte das flugmechanische Simulationsmodell der Stemme S15, die Flugsteuerungsgesetze, die Mensch-Maschine-Schnittstelle und einen Hardware-in-the-Loop – Flugsimulator.

Die modellbasierte Entwicklung erfolgte in LAPAZ mit den Werkzeugen Matlab, Simulink und Stateflow in der Version R2008b. Der Quellcode wurde automatisch mit

dem Codegenerator Real-Time Workshop Embedded Coder erstellt.

Im Nachfolgeprojekt CERTT-FBW23 wurde das Modell ohne Änderungen nach Matlab R2013b überführt. Für die Evaluierung wurden im Laufe der Übersetzung nach SCADE nicht-funktionale Änderungen am Model in Simulink/Stateflow durchgeführt, um die Anzahl übersetzbarer Bestandteile zu erhöhen.

2.1 Funktionen und Aufbau

Der Flugregler des Luftarbeitsflugzeuges Stemme S15 umfasst folgende Funktionen, die im BILD 6 im Anhang als Blockschaltbild dargestellt sind:

- 3D-Navigation (3DNav),
- automatischer Start (TO),
- automatische Landung (LAND),
  - Schleppgaslandung (Standardverfahren)
  - Segelfluglandung ohne Motor (Notverfahren),
- Landeabbruch,
- automatisches Rollen entlang der Mittellinie während Start und Landung,
- vollautomatische Mission (TO → 3DNAV → LAND),
- bodennahe Arbeitsflüge,
- automatische Höhenhaltung,
- automatische Geschwindigkeitshaltung,
- automatische Kurshaltung,
- Böenlastabminderung,
- Schutzfunktionen.

BILD 7 im Anhang zeigt die Software-Architektur der Flugsteuerungsgesetze auf oberster Ebene.

Die Flugregelungsfunktionen wurden iterativ entwickelt und erfolgreich im Flugversuch erprobt. Erkenntnisse aus abgeschlossenen Flugversuchen flossen in nachfolgende Reglerversionen ein. Mehr als 30 Versionen wurden ausgeliefert. Es erfolgten ca. 180 automatische Flüge, ca. 110 automatische Starts und Landungen auf Gras- und Betonbahnen und ca. 10 vollautomatische Missionen. Die Versuche wurden für verschiedene Beladungszustände und Wetterbedingungen durchgeführt. Weitere Informationen können [12] und [13] entnommen werden. Beim Anwendungsbeispiel handelt es sich um die letzte, im Flugversuch erprobte Reglerversion cntr_v04d03.

2.2 Eigenschaften des Anwendungsbeispiels

Die erfolgreiche Flugerprobung zeigt, dass funktional auf Flugzeug- und Systemebene bereits ein hoher Reifegrad erreicht wurde. Auf Softwareebene werden insbesondere nicht-funktionale Anforderungen, wie das Einhalten von Modellierungsrichtlinien, noch nicht zufriedenstellend erfüllt. Während der Entwicklung fehlten Modellierungsrichtlinien, insbesondere zur Beschränkung der Komplexität, zur Beschränkung der Nutzung von Stateflow und Embedded Matlab Code, sowie Vorgaben zur Trennung von Daten- und Kontrollfluss. Diese wurden erst nachträglich eingeführt. Sie konnten für das Anwendungsbeispiel in der Projektlaufzeit nicht mehr vollständig umgesetzt werden. Weiteres Verbesserungspotential besteht in der stärkeren Nutzung eigener Bibliotheken für häufig verwendete Funktionen und der strikten Einhaltung von Namenskonventionen. Darüber hinaus werden in [4] Richtlinien für Simulink-Modelle und Stateflow-Charts definiert, die eingehalten werden müssen, damit ein Modell nach SCADE übersetzbar ist. Diese lagen zum Zeitpunkt der Modellierung nicht vor, sodass zunächst unklar ist, inwieweit sie eingehalten werden.

In TAB 1 sind Metriken des Modells und des Quellcodes angegeben, die zeigen wie komplex das Modell ist.

Modell-Metriken (Anzahl der Elemente vom Typ)		
Typ	**ohne Bibliotheken**	**mit Bibliotheken**
Element (alle Arten inklusive Linien und Annotationen)	> 21000	> 22000
Block	> 4400	> 4600
Subsystem	195	>230
Stateflow-Chart	250	> 300
Embedded Matlab Funktion in Simulink	16	> 30
Ebenen (ohne Stateflow-Charts)	8	-
Codemetriken		
Lines of Code[2]		ca. 50000[3]

TAB 1. Metriken zur Größe des Anwendungsbeispiels

Ein kleiner Teil des Quellcodes ist handgeschrieben. Dieser wird im Modell als sog. *Custom Code* eingebunden und nur in Stateflow benutzt.

3 DIE WERKZEUGKETTE

3.1 SCADE Suite

Für die Evaluierung wird die SCADE Suite in der Version R16.1a mit den Erweiterungen KCG 6.4, SCADE Suite Gateway for Simulink und Simulink Wrapper eingesetzt. Seit der Version R16 der SCADE Suite ist darin die neue Sprachversion SCADE 6.5 enthalten. Allerdings unterstützt das Gateway die Sprache SCADE 6.5 noch nicht, sodass für die Übersetzung von SL/SF-Modellen nach SCADE die Sprache SCADE 6.4 eingesetzt werden muss.

Der wesentliche Unterschied der Sprache SCADE 6.5 zur Sprache SCADE 6.4 besteht in der Erweiterung um numerische Datentypen mit unterschiedlichen Implementierungsgrößen, siehe TAB 2. Bis zur Sprache SCADE 6.4 existieren stattdessen nur Datentypen die keine Implementierungsgröße und Vorzeichenbehaftung unterscheiden. Sie haben im mathematischen Sinne eine unendlich hohe Auflösung und einen unendlich hohen Wertebereich, siehe Seite 108 in [4]. Für die Codegenerierung wird diesen „mathematischen" Datentypen aus SCADE ein Datentyp mit einer bestimmten Implementierungsgröße zugeordnet, z.B. der Datentyp `double` für Signale vom SCADE Typ `real`.

Weitere Informationen zu Neuerungen sind den *Release Notes* der SCADE Suite R16 zu entnehmen, siehe [8].

[2] hauptsächlich automatisch generierter Quellcode (C) mit dem Codegenerator RTWec aus Matlab der Version R2008b
[3] ohne Kommentarzeilen und ohne Standardbibliotheken

Simulink R2013b	SCADE Sprache 6.4	SCADE Sprache 6.5
boolean	bool	bool
uint8 uint16 uint32 - int8 int16 int32 -	int	uint8 uint16 uint32 uint64 int8 int16 int32 int64
single double	real	float32 float64

TAB 2. Numerische Datentypen in Simulink und SCADE

3.2 Matlab/Simulink/Stateflow

Für die Analyse des zu übersetzenden Modells, die Funktionsentwicklung und die Korrektur des Modells wird Matlab R2013b eingesetzt. Das Gateway ist in der eingesetzten Version kompatibel zu Modellen, die in Matlab von Version R2008b bis Version R2013a erstellt wurden. Um ein Modell aus einer neueren Matlab-Version übersetzen zu können, muss es deshalb zunächst in eine kompatible Version exportiert werden. Danach muss das exportierte Modell in der Zielversion von Matlab einmal geöffnet und abgespeichert werden, damit die Kompatibilität gewährleistet ist.

3.3 Begriffe

In TAB 3 und TAB 4 werden Begriffe definiert, die in Simulink, Stateflow und SCADE benutzt werden.

Begriff in SL/SF	Definition
Block	Ein Block stellt eine elementare Funktion (z.B. Multiplikation, Integrator) auf graphische Weise mit Eingängen und Ausgängen dar.
Subsystem	Ein Subsystem ist eine Gruppe von Blöcken, die zu einer funktionalen Einheit zusammengefasst sind. Es dient dazu ein komplexes Modell durch Hierarchiebildung übersichtlich aufzubauen.
Bibliothek (engl. *Library*)	Eine Bibliothek ist eine Sammlung von Blöcken oder Subsystemen, mit häufig benötigen Funktionen. Diese können mehrfach in einem Modell eingebunden (instanziiert) werden.
Stateflow-Chart	Ein Stateflow-Chart ist ein endlicher Zustandsautomat, welcher aus Zustandsübergangsdiagrammen oder Flussdiagrammen (diese enthalten keine Zustände) bestehen kann. Die Abarbeitung erfolgt Ereignisgesteuert.

TAB 3. Begriffe in Simulink/Stateflow

Begriff in SCADE	Definition
Operator	Ein Operator ist ein funktionales Element, welches Instanzen anderer Operatoren enthalten kann. Es entspricht in seiner Funktion einem Block oder Subsystem in Simulink oder einem Stateflow-Chart.
Package	Packages entsprechen Namensräumen. Sie können Elemente wie Operatoren, Konstanten, Sensoren, Typen und Sub-Packages enthalten.
Bibliothek	Bibliotheken in SCADE entsprechen den Bibliotheken in SL/SF.
SSM	SSM steht für *Safe State Machine* (engl.), dem Ausdruck für Zustands-automaten in SCADE.
Projekt	Ein Projekt definiert die Abhängigkeiten eines SCADE Modells zu Bibliotheken, Packages und Operatoren.
MCF	Die MCF-Datei (engl. *Main Configuration File*) wird bei der Installation der SCADE Suite mit dem SCADE Suite Gateway for Simulink mitgeliefert. Sie enthält Regeln zur Übersetzung von SL-Blöcken nach SCADE.
UCF	Eine UCF-Datei, engl. *User Configuration File*, enthält nutzerspezifische Übersetz-ungsregeln, Typendefinitionen und Ergänzungen zur MCF-Datei für den Import von SL/SF nach SCADE. Weitere Informationen sind in [4], Appendix A.

TAB 4. Begriffe in SCADE

Das Wort Modul wird im Folgenden als Überbegriff für ein SL-Subsystem, ein SL/SF-Bibliothekselement oder ein SF-Chart verwendet.

3.4 SCADE Suite Gateway for Simulink

Die folgenden Erläuterungen sind an die Dokumentation des Gateways in [4] angelehnt.

Das Gateway besteht aus zwei Bestandteilen. Erstens aus dem Simulink Translator®, dieser übersetzt Simulink-Blöcke und Subsysteme aus mdl- und slx-Dateien, sowie Parameter aus m-Dateien. Zum zweiten besteht es aus dem Stateflow Importer®, dieser importiert Stateflow-Charts, die in einem Simulink-Modell enthalten sind. Die Übersetzung von Stateflow-Charts muss immer gesondert von der Übersetzung von Simulink-Blöcken erfolgen. Zudem hat der Stateflow Importer nicht den Anspruch, Stateflow-Charts vollständig und korrekt zu übersetzen, da der Sprachumfang der Zustandsmaschinen in SCADE deutlich geringer ist, als der Sprachumfang von Stateflow, siehe [4].

Der Übersetzungsprozess eines Modells erfolgt modular. Er beginnt auf der untersten Ebene des zu übersetzenden Modells und wird dann aufsteigend bis zur obersten Ebene durchgeführt (*bottom-up*). Jedes SL/SF-Modul wird in SCADE einzeln als Operator in ein vom Nutzer festgelegtes Package importiert. Bereits importierte

Module können für die Übersetzung einer darüber liegenden Ebene wiederverwendet werden, siehe Kapitel 2 in [4]. Mit Hilfe von nutzerspezifischen Übersetzungsregeln und sog. *Import Dependencies* kann der Aufbau eines SCADE-Modells aus existierenden Packages und Operatoren gesteuert werden. Die nutzerspezifischen Übersetzungsregeln werden in UCF-Dateien angegeben. Diese können für einzelne Blöcke oder ganze Module über den Pfad (Typ 1) oder über andere Attribute, wie z.B. den Blocktyp (Typ 2) angegeben werden. Mit dem zweiten Typ können Regeln aufgestellt werden, die global gelten sollen. Die in 3.3 genannte MCF-Datei verwendet nur Regeln dieses Typs. Regeln, die über den Blockpfad festgelegt sind, haben die höchste Priorität und überschreiben alle anderen Regeln die über Attribute festgelegt sind und ebenfalls anwendbar wären. Auszüge von UCF-Dateien für die Definition nutzerspezifischer Datentypen und Übersetzungsregeln des Anwendungsbeispiels sind in BILD 11 und BILD 12 im Anhang zu finden.

Zusammengefasst erfolgt die Übersetzung eines parametrisierten Modells bestehend aus Simulink und Stateflow in neun Schritten:

1) Analyse der Hierarchie des Modells zur Festlegung der Importreihenfolge
2) Parameter aus m-Dateien importieren
3) UCF für eigene Datentypen (z.B. Busse) und modellweite Übersetzungsregeln erstellen
4) Module aus eigenen Bibliotheken importieren
5) Stateflow-Charts importieren
6) Nicht-übersetzbare Elemente, die in SL/SF nicht geändert werden können oder sollen, manuell in SCADE nachbilden
7) Subsysteme des Modells modular importieren und bereits importierte Module (z.B. Parameter, eigene Bibliothekselemente, SF-Charts) wiederverwenden. Hierfür kann es notwendig sein, UCF-Dateien zu erstellen und anzugeben.
8) Gesamtmodell importieren
9) Anpassung des SCADE-Modells, z.B.:
   – Lesbarkeit
   – Abgleich der Codegenerierungseinstellungen (z.B. *Inlining*) mit dem Simulink-Modell

Die Verifikation muss ebenfalls von der unteren zur oberen Ebenen erfolgen. Diese sollte jeweils für alle Module separat, für alle Hauptmodule separat und zum Schluss für das Gesamtmodell geschehen.

Details zur Übersetzung und weitere Informationen, z.B. zum Re-Import, zum Simulink Wrapper®, zu Modellierungsrichtlinien für Simulink-Modelle und Stateflow-Charts, zur *Traceability* zwischen dem Simulink-Modell und dem SCADE-Modell und zur unterstützten Syntax von Matlab m-Dateien sind [4] zu entnehmen.

4 ANSATZ UND IMPLEMENTIERUNG

Die Größe des Modells des Anwendungsbeispiels in Verbindung mit den angewendeten Modellierungsrichtlinien (vgl. Kapitel 2.2) führt dazu, dass die Übersetzung mit dem Gateway nicht für alle Module im Ursprungszustand erfolgreich ist. Erschwerend kommt hinzu, dass das Gateway bei schweren Fehlern (z.B. Aufgrund nicht unterstützter Blöcke) die Übersetzung abbricht und in diesem Fall keine Zwischenergebnisse (teilweise übersetztes SCADE-Modell) produziert. Lediglich Warnungen und Fehler lassen sich dann auswerten. Weiterhin muss der erste Import über die GUI (engl. *Graphical User Interface*) erfolgen. Bei der modularen Übersetzung eines so großen Modells ist der Aufwand der Übersetzung und Analyse der Ergebnisse sehr hoch. Den Autoren ist kein Werkzeug am Markt bekannt, welches die Übersetzbarkeit von Simulink/Stateflow nach SCADE entsprechend der Richtlinien in [4] vollständig prüfen kann. Ohne einen Überblick zur Übersetzbarkeit des Anwendungsbeispiels und zum Vorhandensein nicht unterstützter Sprachelemente kann keine Aufwandsabschätzung erstellt werden. Für eine realistische Planung der Arbeiten ist dieser Überblick jedoch notwendig. Aus diesem Grund wird in den folgenden Abschnitten ein Ansatz vorgestellt, mit dem ein solcher Überblick erhalten werden kann.

4.1 Prozess- und Werkzeuganforderungen

Folgende Anforderungen an den Prozess sowie die zu entwickelnden Werkzeuge wurden aufgestellt.

1) **Korrektheit und Vollständigkeit:** Die Übersetzung von SL/SF nach SCADE soll korrekt und vollständig erfolgen.
2) **Automatisierung:** Die Übersetzung soll für alle Simulink-Subsysteme und alle Stateflow-Charts eines Modells von Matlab aus vollautomatisch gestartet werden können. (Hinweis: Die Aufteilung in Simulink-Subsysteme und Stateflow-Charts ist wegen ihrer getrennten Behandlung im Gateway notwendig.)
3) **Effiziente Auswertung:** Die Ergebnisse der Übersetzung sollen automatisiert gesammelt und aufbereitet werden, um einen schnellen Überblick über den Status der Übersetzung zu erhalten.
4) **Modularer Import 1:** Der automatische Übersetzungsvorgang soll modular erfolgen.
5) **Modularer Import 2:** Das SCADE-Gesamtmodell soll vollautomatisch aus bereits importierten Modulen aufgebaut werden können, wobei auch manuell erstellte Module berücksichtigt werden sollen.
6) **Manuelle Korrektur:** Es soll möglich sein, manuell übersetzte oder korrigierte Module von der automatischen Übersetzung auszuschließen, um ein Überschreiben des manuell erzeugten Moduls zu verhindern.

4.2 Ansatz

Beim modularen Import werden nicht vorhandene, über *Import Dependencies* angegebene, Kind-Elemente, nicht als Fehler behandelt, die zu einem Abbruch führen. Mit Hilfe des modularen Imports kann demzufolge erreicht werden, dass Fehler bei der Übersetzung direkt den übrigen Bestandteilen des zu übersetzenden Moduls zugeordnet werden können. Werden nach dem *Trial and Error*-Verfahren alle Subsysteme und Stateflow-Charts einzeln importiert, kann ein erster Überblick über die Übersetzbarkeit gewonnen werden.

Wie bereits erwähnt, muss der erste Import eines Moduls über die GUI des Gateways erfolgen. Aufgrund der Größe des Anwendungsbeispiels mit einer Vielzahl an Modulen ist der Aufwand dafür zu hoch. Es existiert jedoch die Möglichkeit eines *Workarounds*, mit der die Übersetzung von der Kommandozeile aus erfolgen kann. Hierfür hat das Unternehmen Esterel Technologies ein Beispiel für

Simulink basierend auf Shellskripten für Unix entwickelt und zur Verfügung gestellt. Es basiert auf einem *Dummy*-SCADE-Projekt mit einem *Dummy*-SCADE-Package für den Simulink-Import. Die Dateien des Beispiels enthalten Platzhalter für Parameter der Import-GUI für Simulink, die mit Hilfe der Shellskripte durch Parameter für den Import eines konkreten Simulink-Subsystems ersetzt werden. Dieser *Workaround* wurde erfolgreich nach Matlab unter Windows portiert und weiterentwickelt. Die Erzeugung der Import-Parameter wurde automatisiert, sodass der *Workaround* für die vollautomatische Übersetzung von Simulink/Stateflow-Modulen eingesetzt werden kann. Neben dem Import können z.B. auch die Modellprüfung in SCADE, sowie die Codegenerierung über die Kommandozeile gestartet werden. Die Modellprüfung wurde bereits implementiert. Sie erfolgt unmittelbar nach dem Import. Die prinzipielle Funktionsweise ist in BILD 8 im Anhang dargestellt.

Ist eine vollautomatische Übersetzung in der Entwicklungsumgebung des Modells (Matlab) möglich, so kann in Kombination mit einer automatischen Richtlinienprüfung und ggf. -korrektur, ein effizienter, iterativer Prozess aufgebaut werden, um die Übersetzbarkeit des Modells herzustellen. Der angedachte Übersetzungsprozess ist in vereinfachter Form in BILD 9 dargestellt.

4.3 Implementierung der Werkzeugkette

Anforderung 2 (Automatisierung) wird erfüllt, indem ein Werkzeug für Simulink und Stateflow erstellt wird, welches automatisch alle Simulink-Subsysteme bzw. Stateflow-Charts eines Modells findet, für diese Import-Parameter erzeugt und sie nacheinander, modular durch Aufruf des Gateways importiert. Einer der Import-Parameter ist die Liste der anwendbaren, nutzerspezifischen Übersetzungsregeln in Form von Konfigurationsdateien (UCF).

Anforderung 4 (Modularer Import 1) und 5 (Modularer Import 2) werden erfüllt, indem UCF-Dateien mit *Import Dependencies* und Übersetzungsregeln automatisch generiert werden. Ein Ausschnitt einer generierten UCF-Datei für den Import des Gesamtmodells zeigt BILD 12 im Anhang. Des Weiteren wird eine Liste mit erfolgreich übersetzten Modulen automatisch erstellt und aktualisiert. Diese Liste kann auch manuell bearbeitet werden, um z.B. manuell übersetzte Module einzutragen. Damit wird auch Anforderung 6 (Manuelle Korrektur) erfüllbar.

Anforderung 3 (Effiziente Auswertung) wird erfüllt, indem bei der Übersetzung automatisch Tabellen erzeugt werden, die mindestens die Informationen aus TAB 5 enthalten. Diese Informationen werden sowohl durch statische Analyse des Modells in Matlab (z.B. Anzahl enthaltener Elemente), als auch durch die Auswertung der Ausgaben des Gateways (z.B. Anzahl übersetzter oder nicht unterstützter Elemente) generiert. Des Weiteren wird das SCADE-Modell mit einem Werkzeug der SCADE Suite zur Prüfung der Syntax und Semantik (im Folgenden Checker genannt) geprüft und die Anzahl der Fehler im Modell bestimmt. Hierfür wurden Funktionen in Matlab implementiert und in das Übersetzungswerkzeug integriert, die alle Informationen zur Übersetzung sammeln und als xls-Datei ausgeben. Die anschließende Formatierung und Auswertung der Datei erfolgt manuell.

Allgemeine Informationen
Name des Moduls
Eindeutiger Bezeichner
Dauer der Übersetzung
Status der Übersetzung
Anzahl der Warnungen bei der Übersetzung
Vollständigkeit der Übersetzung
Status der Modellprüfung in SCADE
Anzahl an Fehlern bei der Prüfung
Link zum Fehlerbericht
Pfad des Moduls im Modell
Simulink-spezifische Informationen
Anzahl enthaltener und übersetzter Subsysteme
Anzahl enthaltener und übersetzten Blöcke
Anzahl an Stateflow-Charts
Anzahl an Embedded Matlab Funktionen
Anzahl der *Import Dependencies*
Stateflow-spezifische Informationen
Anzahl enthaltener und übersetzter Charts
Anzahl enthaltener und übersetzter Zustände
Anzahl enthaltener und übersetzter Transitionen
Anzahl enthaltener und übersetzter Junctions
Anzahl enthaltener und übersetzter Funktionen
Anzahl nicht unterstützter/übersetzter Funktionen
Anzahl nicht unterstützter/übersetzter Transitionen
Anzahl nicht unterstützter/übersetzter Junctions
Anzahl nicht unterstützter/ übersetzter Bedingungen
Anzahl nicht unterstützter/ übersetzter Aktionen

TAB 5. Informationen für die effiziente Auswertung der Übersetzung von SL/SF nach SCADE

4.4 Verifikation der Übersetzung

Für die Verifikation der Übersetzung sind die in TAB 5 genannten Informationen notwendig aber nicht ausreichend. Nur durch weitere Schritte kann ein vollständiger Nachweis von Anforderung 1 (Korrektheit und Vollständigkeit) erreicht werden.

Es muss geprüft werden, dass bei der Übersetzung nach SCADE und bei der Modellprüfung mit dem Checker keine Fehler aufgetreten sind. Der letzte Punkt ist eine notwendige Bedingung für die Codegenerierung. Weiterhin muss verifiziert werden, dass die Architektur des übersetzten Modells, der Daten- und Kontrollfluss, sowie die Datentypen und Variablennamen mit denen des Quellmodells übereinstimmen. Darüber hinaus soll die Verständlichkeit der Übersetzung der Stateflow-Charts verifiziert werden. Wenn nötig müssen Anpassungen erfolgen. Co-Simulationen, d.h. eine parallele Ausführung des Simulink-Modells und einer S-Funktion des SCADE-Modells, die mit denselben Eingängen stimuliert werden, sollen durchgeführt werden. Durch Vergleich der Ausgänge kann die funktionale Äquivalenz zwischen dem SCADE-Modell (bzw. dem Quellcode) und dem Simulink-Modell bzw. dem daraus generierten Quellcode verifiziert werden. Die S-Funktion kann mit Hilfe des Simulink Wrappers aus dem Quellcode generiert werden. Als letztes muss sichergestellt werden, dass keine Funktionalität bei der Übersetzung verloren gegangen oder hinzugefügt wurde. Dieses könnte z.B. durch anforderungsbasierte Test- bzw. Simulationsfälle und

durch Messen von Abdeckungsmetriken (*Requirements Coverage*, *Test Coverage*, *Model Coverage*) sichergestellt werden, siehe [9] und [10].

Der Aufwand der Verifikation ist sehr hoch. Für die Simulationen und die Ermittlung der Abdeckungsmaße ist jedoch eine Automatisierung und Werkzeugunterstützung denkbar.

5 ERGEBNISSE DER ERSTEN ÜBERSETZUNG

5.1 Ergebnisse des Simulink-Imports

BILD 10 im Anhang ist ein Auszug der Zusammenfassung des vollautomatischen, modularen Imports von Simulink-Subsystemen aus Matlab heraus. Es wurden insgesamt 195 Subsysteme in ca. 0,75 h mit dem Gateway importiert. 61 % der Subsysteme, die etwa 62 % aller Blöcke des Modells enthalten, konnten ohne Abbruch übersetzt werden, davon wurden alle vollständig übersetzt. Nur etwa 8 % der Subsysteme bestehen die interne Prüfung. Die niedrige Zahl liegt daran, dass bei den Übersetzungsregeln in UCF-Dateien Module angegeben sind, die noch nicht übersetzt werden konnten oder wurden. Bei einer ersten groben Sichtung der Ergebnisse sind verschiedene problematische Elemente aufgefallen. Daraufhin wurden der Ort und die Häufigkeit des Auftretens dieser Elemente im Modell mit selbst entwickelten Suchalgorithmen in Matlab bestimmt. Die Ergebnisse sind in TAB 6 zusammengefasst.

Inkompatibilität in Simulink	**Häufigkeit**
unbenannte Verbindungslinien zwischen Blöcken	ca. 70% der Linien sind nicht benannt (jede Verzweigung desselben Signals wurde mitgezählt)
Ein- oder Ausgänge haben denselben Namen, wie die damit verbundenen Signale	k.A. (Algorithmus noch nicht erstellt)
Diskreter Integrator mit inkompatiblen Einstellungen, z.B.: Initialisierungsmodus nicht auf „State and Output“ gestellt oder Verstärkungsfaktor lautet nicht exakt `1.0` sondern z.B. `single(1.0)` oder `single(0.8)`	ca. 38% der Subsysteme

TAB 6. Probleme beim Import von SL-Subsystemen

Die ersten beiden Probleme in TAB 6 führen meistens nicht dazu, dass der Import fehlschlägt. Sie führen jedoch dazu, dass die *Traceability* zwischen Simulink-Modell und SCADE-Modell aufwendiger wird, da Elemente in SCADE automatisch benannt bzw. umbenannt werden müssen. Zudem können sie zu Inkonsistenzen im SCADE-Modell führen, die manuell aufgelöst werden müssen.

Die Verwendung diskreter Integratoren mit inkompatiblen Einstellungen führt zum Abbruch der Übersetzung. Bemerkenswert ist, dass die meisten Subsysteme, bei denen die Übersetzung abgebrochen ist, mindestens einen dieser problematischen Integratoren enthalten. Es liegt deshalb nahe, dass fast alle Simulink-Subsysteme ohne Abbruch übersetzt werden können, wenn das Problem mit inkompatiblen Integratoren behoben wird. Es muss deshalb mit hoher Priorität gelöst werden.

Nur zwei der nicht übersetzbaren Subsysteme haben keinen Integrator mit den oben genannten Eigenschaften. Bei einem dieser Subsysteme schlug der *Type Inference* Algorithmus fehl, das genaue Problem konnte noch nicht identifiziert werden. Bei dem anderen Subsystem steht ein unnötiges Semikolon am Ende der Definition der Stützstellen einer Tabelle (engl. *Lookup-Table*), was nicht unterstützt wird.

5.2 Ergebnisse des Stateflow-Imports

Neben dem Import der Subsysteme in Simulink wurden mit dem Gateway auch alle Stateflow-Charts (SF-Charts) des Modells, die nicht in Bibliotheken definiert sind, vollautomatisch aus Matlab heraus importiert. Wie bereits beim Import der Simulink-Subsysteme, wurden auch hier nach einer ersten stichprobenartigen Sichtung einige Inkompatibilitäten in den SF-Charts aufgedeckt und der Ort sowie die Häufigkeit des Auftretens im Modell mit dafür erstellten Suchalgorithmen ermittelt. Die bisherigen Ergebnisse sind in TAB 7 zusammengefasst.

Beim ersten Übersetzungsversuch konnten nur wenige SF-Charts vollständig übersetzt werden. Wie in TAB 7 zu sehen ist, sind unterschiedliche, meist sehr häufig vorkommende Probleme die Ursache. Elemente in SF-Charts vom Typ entsprechend Eintrag 1 bis 5 in TAB 7 werden bei der Übersetzung teilweise oder ganz ausgelassen. Das Gateway zählt jedoch auch die Transitionen und Zustände als übersetzt mit, bei denen Bedingungen oder Aktionen (siehe Kapitel 12 „Stateflow“ in [19]) aufgrund der oben genannten Inkompatibilitäten ausgelassen wurden. Aus diesem Grund werden die Warnungsausgaben beim Import daraufhin ausgewertet. Elemente, die dem letzten Eintrag in TAB 7 entsprechen, werden übersetzt, es entstehen jedoch Datentypkonflikte in SCADE (`bool` erwartet, `int` übersetzt), die bei der Prüfung mit dem Checker aufgedeckt werden.

Inkompatibilität in Stateflow	**Häufigkeit**
Benutzung von Wahrheitstabellen	1 SF-Chart
Benutzung von Embedded Matlab-Funktionen	ca. 6% der SF-Charts
Werte des Datentyps `single` in der Schreibweise `12.3F` oder `12.3f` (Buchstabe `f` bzw. `F` wird ignoriert o. führt zu Fehler)	ca. 35% der SF-Charts (bereits korrigiert, über 500 Vorkommnisse)
Busse als Eingang oder Ausgang (Verwendung strukturierter Datentypen)	ca. 50% der SF-Charts
Benutzung von Enumerationen aus *Custom Code* und Fehlen der Konvertierung auf den richtigen Datentypen	ca. 60% der SF-Charts (über 800 Vorkommnisse)
Literale `0` und `1` statt `false` und `true` in booleschen Ausdrücken	k.A. (Algorithmus noch nicht fertig)

TAB 7. Probleme beim Import von SF-Charts

Aufgrund der großen Anzahl an Vorkommnissen wurde entschieden, dass die Probleme werkzeugunterstützt behoben werden sollen. Für das Problem mit der Schreibweise von Werten des Datentyps `single` wurde ein automatisiertes Korrekturverfahren entwickelt und erfolgreich angewendet (vgl. Kapitel 6.4).

Nach dieser Korrektur wurde die Übersetzung wiederholt. Dabei wurden 250 SF-Charts in ca. 1 h importiert, wovon ca. 95 % ohne Abbruch übersetzt werden konnten. Wird die Anzahl übersetzter und vorhandener Objekte (*States*, *Transitions*, *Junctions*, *Functions*) verglichen, so wurden ca. 85 % der SF-Charts vollständig übersetzt. Bezogen auf die Gesamtanzahl der Objekte (mehr als 7600) konnte jedoch nur etwa 42 % des Stateflow-Umfangs übersetzt werden. Die deutliche kleinere Zahl kommt dadurch zustande, dass insbesondere bei den komplexen SF-Charts mit vielen Zuständen und Unterebenen die Übersetzung mit Fehlern abgebrochen ist. Dies war bei 9 SF-Charts der Fall, die 50 % des Umfangs, gemessen an der Objektanzahl (siehe oben), ausmachen. Werden bei der Betrachtung der Vollständigkeit nicht unterstützte bzw. nicht übersetzte Bedingungen (ca. 330) und Aktionen (ca. 630) mit berücksichtigt, so sinkt die Anzahl vollständig übersetzter SF-Charts auf ca. 29 %. Ca. 11 % aller SF-Charts haben die Modellprüfung in SCADE bestanden.

5.3 Probleme mit Datentypen

Wie in Kapitel 3.1 angegeben, unterstützt das Gateway in der aktuellen Version nur wenige Datentypen. Das Modell des Anwendungsbeispiels macht, wegen des begrenzten Speichers des Zielsystems, Gebrauch von allen in TAB 2 genannten Datentypen in Simulink und Stateflow, diese werden explizit angegeben. In der Regel wird für ein Signal der kleinstmögliche Datentyp bei ausreichender Genauigkeit ausgewählt. Die benutzten Datentypen müssen für die Übersetzung dem SCADE-Datentyp entsprechend TAB 2 zugeordnet werden, wobei Informationen zur Vorzeichenbehaftung und Auflösung verloren gehen. Konvertierungen zwischen verschiedenen ganzzahligen Typen oder verschiedenen Gleitkommatypen existieren nicht in der SCADE-Sprache 6.4, sodass entsprechende Blöcke in Simulink/Stateflow durch das Gateway in SCADE als Identität übersetzt werden. Dadurch kann sich zwischen Simulink/Stateflow und SCADE ein unterschiedliches Verhalten ergeben. Beispiele dafür sind in [4] auf Seite 106ff. angegeben.

6 ÄNDERUNGEN UND PROBLEMLÖSUNGEN

In diesem Kapitel werden Änderungen am Modell und dem Entwurfsprozess beschrieben, welche die Übersetzbarkeit von SL/SF-Modellen nach SCADE verbessern.

6.1 Modellierungsrichtlinien für SL/SF

Die folgenden Modellierungsrichtlinien wurden aus den aufgetretenen Problemen bei der Übersetzung des Anwendungsbeispiels abgeleitet.

- Nur unterstütze Matlab-Syntax in Parameterdateien verwenden (vgl. [4], Anhang C),
- Alle Signale eindeutig und unter Einhaltung von Namenskonventionen benennen,
- Präfix oder Postfix für Namen von Ein- und Ausgängen wegen eindeutiger Benennung nutzen,
- Verstärkungsfaktor eines Integrators vor dem Integrator angeben, wenn Wert ungleich `1.0` ist,
- Nutzung von SF-Charts minimieren,
- Komplexität von SF-Charts minimieren,
- Nutzung von *Custom Code* in SF-Charts minimieren
- Keine Busse in SF-Charts benutzen,
- Schreibweise `single(12.3)` statt `12.3f` in SF-Charts verwenden.

Für das Anwendungsbeispiel müssen diese und weitere Richtlinien (siehe [4]) nachträglich durch ein *Redesign* umgesetzt werden. Einige Änderungen am Modell wurden bereits durchgeführt. Beim Entwurf neuer Modelle sollten die Richtlinien frühzeitig angewendet werden und deren Einhaltung regelmäßig geprüft werden.

6.2 Datentypen

Eine Lösung des Problems fehlender Datentypen (vgl. Kapitel 5.3) könnte die Migration des SCADE-Modells von der Sprache SCADE 6.4 auf SCADE 6.5 sein. In der neuen Version stehen alle notwendigen Datentypen zur Verfügung. Der Aufwand der Migration und der Anpassungen am SCADE-Modell kann jedoch noch nicht abgeschätzt werden.

6.3 Problemlösung für Enumerationen

Im Anwendungsbeispiel sind Enumerationen in Header-Dateien (*.h) definiert, die die Zahlenwerte der verschiedenen Modi des Flugreglers festlegen. Diese werden von der Modelogik und von Schaltungen benötigt und sind der Grund, warum häufig SF-Charts im Modell verwendet werden. Diese Art der Implementierung kann jedoch nicht ohne weiteres übersetzt werden.

Eine Lösungsalternative für dieses Problem ist, die Enumerationen in Matlab statt in C zu definieren. Das Vorgehen wird an einem einfachen Modell verdeutlicht. Zunächst wird die Definition einer Enumeration in Matlab erstellt, siehe BILD 2. Diese Datei muss im Suchpfad liegen, damit z.B. ein SL-Modell darauf zugreifen kann.

```
classdef(Enumeration) def_en_modes < Simulink.IntEnumType
  enumeration
    en_GAMode(1)        %Go-Around Mode
    en_3DNavMode(2)     %3D-Navigation Mode
    en_LandMode(3)      %Autoland Mode
  end
end
```

BILD 2. Enumeration in Matlab (def_en_modes.m)

Das Beispielmodell hat einen Eingang („Inp_ui1Modes“ vom Typ „uint8“) und einen Ausgang („Out_b1Nav3DActive“ vom Typ „boolean“). Es soll logisch wahr ausgeben, wenn der Eingang den Wert = 2 (en_3DNavMode) annimmt, sonst soll es logisch falsch ausgeben. Es besteht aus einem einfachen SF-Chart ohne Zustände, welches in BILD 3 dargestellt ist. Für die Übersetzbarkeit nach SCADE ist die Konvertierung der Enumeration auf den richtigen Datentypen (`uint8`) notwendig. In Stateflow ist die Konvertierung nicht zwingend, im Sinne einer regelkonformen Modellierung für sicherheitskritische Systeme jedoch ratsam.

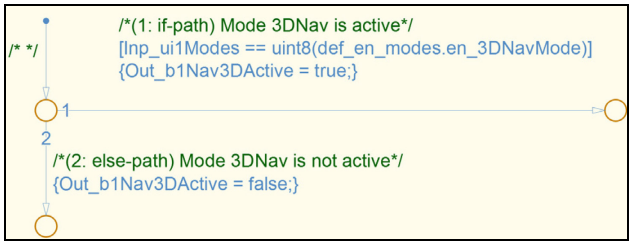

BILD 3. SF-Chart „SFChart_en“ mit Enumeration

Die Übersetzung nach SCADE erfolgt in 2 Schritten:

1) Import der Enumeration über „File/Import/Matlab Variables…/M-Files to enumerated types“ aus „def_en_modes.m“. Im Beispiel wird das Package „ML_def_en_modes“ als Ziel angegeben.
2) Import des SF-Charts „SF_SFChart_en “ über „File/Import/Stateflow Charts …“.

Damit die bereits importierte Definition der Enumeration für den Import des SF-Charts genutzt werden kann, muss das Package „ML_def_en_modes“ beim Stateflow Importer unter *Import Dependencies* (siehe BILD 4) angegeben werden.

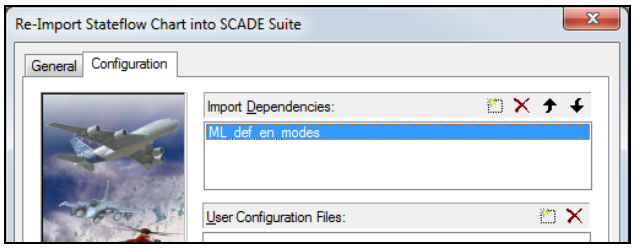

BILD 4. Import Dependencies beim Import

Das Ergebnis der Übersetzung ist in BILD 5 dargestellt. Es ist zu erkennen, dass sog. *Flowcharts* (SF-Charts ohne Zustände) durch das Gateway in nicht-grafische Text-Diagramme übersetzt werden.

```
#pragma ed <IfBlock oid="!ed/9a349/500A/1368/55eb08407401"/>
#end
activate Decision_1
   if (Inp_uilModes =
conv::Enum2Int(ML_def_en_modes::def_en_modes::en_3DNavMode))
#pragma ed <Action oid="!ed/9a34b/500A/1368/55eb08406d98"/>
#end
 then
      #pragma ed <Equation
oid="!ed/9a34d/500A/1368/55eb084071b4"/>
#end
      Out_b1Nav3DActive = true;
   #pragma ed <Action
oid="!ed/9a34f/500A/1368/55eb08401b53"/>
#end
   else
      #pragma ed <Equation
oid="!ed/9a351/500A/1368/55eb08405840"/>
#end
      Out_b1Nav3DActive = false;
returns .. ;
```

BILD 5. Übersetztes SF-Chart „SFChart_en“ in SCADE

Die Übersetzung wurde manuell mittels Sichtprüfung mit dem Ursprungsmodell verglichen und zusätzlich mit dem SCADE internen *Check* geprüft. Zum Abschluss wurde eine Simulation durchgeführt, wofür Code mit dem KCG Codegenerator generiert wurde. Für dieses einfache Modell ergaben sich keine Fehler bei der Übersetzung und dasselbe dynamische Verhalten wie in SL/SF.

Die vorgestellte Lösung hat mehrere Vorteile gegenüber der bisherigen Lösung mittels *Custom Code* in C:

– Die Übersetzbarkeit nach SCADE wird erreicht.
– Die Definition von Enumerationen kann in Matlab, Simulink und Stateflow genutzt werden, anstatt nur in Stateflow.
– Daraus ergibt sich als weiterer Vorteil, dass die Anzahl der SF-Charts im Modell reduziert werden kann (Für dieses einfache Beispiel ist ein einziger Simulink-Block „Compare to constant“ notwendig).

Als Nachteil ist aufzuführen, dass aus C-Programmen nicht ohne weiteres auf die Definition der Enumeration in Matlab zugegriffen werden kann.

6.4 Automatisierte Korrektur von SF-Charts

Der Stateflow-Importer erlaubt bei der Übersetzung nicht überall in SF-Charts die Schreibweise `12.3f` oder `12.3F` für Werte des Datentyps `single`. Stattdessen sollte die Schreibweise `single(12.3)` verwendet werden (der hier angegebene Wert ist als Beispiel zu betrachten). Das Modell des Anwendungsbeispiels wurde automatisch korrigiert. Dafür wurden Funktionen in Matlab implementiert, die das Auftreten dieser Schreibweise in Transitionen und Zuständen erkennen und korrigieren. Für die Umsetzung der Automatisierung wurden ca. drei Arbeitstage benötigt. Der Zeitaufwand der Korrektur und der manuellen Verifikation betrug 0,75 h.

Es wurden 88 SF-Charts an 506 Stellen automatisch korrigiert. Bei 25 % dieser SF-Charts hat die Korrektur dazu geführt, dass die Übersetzung ermöglicht wurde. Die übrigen 75 % enthalten weitere Probleme, die noch behoben werden müssen.

In naher Zukunft werden weitere Automatisierungen implementiert, um z.B. die in Kap. 5.2 identifizierten Probleme in SF-Charts zu korrigieren. Die Implementierung der Algorithmen zur Identifikation inkompatibler Literale in booleschen Ausdrücken (vgl. TAB 7) ist bereits weit fortgeschritten und wird ebenfalls durch eine automatisierte Korrektur erweitert. Anschließend ist die Korrektur der Enumerationen nach Kapitel 6.3 eingeplant.

7 ZUSAMMENFASSUNG UND AUSBLICK

Das wichtigste Ergebnis aus Sicht der Autoren ist der hohe Automatisierungsgrad, der für die Übersetzung von SL/SF-Modellen nach SCADE erzielt wurde. Für das Anwendungsbeispiel können alle SL-Subsysteme und alle SF-Charts vollautomatisch in ca. 2 h importiert werden. So kann das SL/SF-Modell nach Änderungen in kürzester Zeit nochmals übersetzt werden. Die erstellten Werkzeuge sind generisch und können problemlos auf andere Modelle angewendet werden.

Die Schwierigkeiten bei der Übersetzung des Beispiels lassen sich in drei Bereiche aufteilen. Ein Teil der Probleme folgt aus Einschränkungen des Gateways, als wichtigstes Beispiel sind hier die fehlenden numerischen Datentypen zu nennen. Ein anderer Teil, der insbesondere Stateflow betrifft, folgt aus dem deutlich geringeren Sprachumfang in SCADE gegenüber SL/SF. Eine weitere Ursache für Schwierigkeiten sind nicht eingehaltene Richtlinien des zu übersetzenden Modells, beispielsweise die Verwendung der Literale `0` und `1` für die booleschen Werte `false` und `true` in Stateflow.

Für das Anwendungsbeispiel kann noch keine abschließende Aussage über die Verteilung der drei Problemgruppen getroffen werden, da erst ein Teil der Probleme vollständig identifiziert wurden.

7.1 Lessons learned

Bisher können folgende Lehren aus der Evaluation gezogen werden:

1) Das Gateway ist ein Expertentool und erfordert einen Einarbeitungsaufwand von mehreren Wochen.

2) Zu übersetzende Modelle müssen aus Softwaretechnischer Sicht reif sein und strenge Richtlinien einhalten. Die Richtlinien sollten frühzeitig angewendet und regelmäßig geprüft werden.
3) Die Einschränkungen bei numerischen Datentypen müssen gelöst werden. Hierfür ist eine Aktualisierung des Gateways auf die Sprache SCADE 6.5 sehr wünschenswert.
4) Viele Stateflow-Funktionalitäten werden nicht unterstützt. Die Übersetzung mit dem Stateflow-Importer ist deshalb lückenhaft und häufig deutlich komplexer, als wenn sie manuell durchgeführt würde. Dadurch wird auch die Verifikation der Übersetzung erschwert.
5) Die Übersetzung des Anwendungsbeispiels ist noch nicht abgeschlossen und weiterer erheblicher Aufwand wird noch erwartet.
6) Die Übersetzung der sehr komplexen Stateflow-Charts der Modelogik wird vermutlich nicht automatisch erfolgen können und ein vollständiges *Redesign* in Simulink/Stateflow erfordern.
7) Für die Übersetzung komplexer Modelle sind Automatismen notwendig, um menschliche Fehler zu vermeiden und die Wiederholbarkeit zu verbessern.
8) Mit Werkzeugen, wie sie hier entwickelt und in diesem Beitrag kurz vorgestellt wurden, ist ein hoher Automatisierungsgrad der Übersetzung erreichbar.

7.2 Ausblick

Bisher liegt ein erster Überblick vor, welche Probleme bei der Übersetzung von SL/SF nach SCADE existieren. Die Analyse muss fortgesetzt und die Schwierigkeiten schrittweise gelöst werden, um anschließend die korrekte und vollständige Übersetzung zu verifizieren.

Ein Vergleich der Quellcodes, die mit den Codegeneratoren Embedded Coder und SCADE KCG generiert werden, hinsichtlich Nachverfolgbarkeit (*Traceability*), Lesbarkeit, Effizienz und Größe des generierten Quellcodes ist interessant. Hierfür muss aber erst das Modell vollständig übersetzt werden.

8 DANKSAGUNG

In diesem Beitrag wurden Ergebnisse des FMRA der Forschungsvorhaben LAPAZ und LAPAZ II genutzt. Beide Projekte sind abgeschlossen, sie wurden im Rahmen des nationalen Luftfahrtforschungsprogramms (LuFo IV) vom Bundesministerium für Wirtschaft und Technologie (BMWi) gefördert. Die Bearbeitung der Evaluation ist Teil des Forschungsvorhabens CERTT-FBW23. Es wird im Rahmen des nationalen Luftfahrtforschungsprogramms (LuFo V-1) vom Bundesministerium für Wirtschaft und Energie (BMWi) gefördert. Dem Unternehmen Esterel Technologies wird für die Aufnahme im akademischen Programm und die technische Unterstützung gedankt.

LITERATUR

[1] V. Hilderman; L. Buckwalter: Avionics Certification: A Complete Guide to DO-178 (Software), DO-254 (Hardware). Avionics Communications Inc., USA, 1st edition, 2007.
[2] J.-C. Laprie: Dependability: The Challenge for the Future of Computing and Communication Technologies. Proceedings of the Dependable Computing Conference (EDCC-1), 1994, Springer-Verlag Berlin Heidelberg.
[3] http://de.mathworks.com/products/do-178/
[4] Esterel Technologies. Gateway Guidelines for Simulink®. 09/01/15, Revision: SC-SKG-EPG-R16 - DOC/rev/55682-01.
[5] G. Berry. SCADE: Synchronous design and validation of embedded control software. Proceedings of the GM R&D Workshop, Bangalore, India, January 2007, pp 19-33. Springer Netherlands.
[6] http://www.esterel-technologies.com/about-us/scientific-historic-background/
[7] Esterel Technologies. Methodology Handbook - Efficient Development of Safe Avionics Software with DO-178C Objectives Using Scade Suite®. Erste Ausgabe, Juni 2015.
[8] Esterel Technologies. Release Notes SCADE Suite® R16. Revision: SC-RN-R16 - 04/03/15.
[9] RTCA, Inc. RTCA DO-178C - Software Considerations in Airborne Systems and Equipment Certification. 2011.
[10] RTCA, Inc. RTCA DO-331 - Model-Based Development and Verification Supplement to DO-178C and DO-278A. 2011.
[11] RTCA, Inc. RTCA DO-330 - Software Tool Qualification Considerations. 2011.
[12] L. Dalldorff, R. Luckner, R. Reichel. A Full-Authority Automatic Flight Control System for the Civil Airborne Utility Aircraft S15 – LAPAZ. CEAS 2nd EuroGNC 2013, Conference in Guidance, Navigation & Control in Aerospace, 10.-12. April 2012, Delft.
[13] R. Luckner, L. Dalldorff, R. Reichel. A Utility Aircraft for Remote Sensing Missions with a High-Precision Automatic Flight Control System. ICARES 2014, 13-14 November 2014, Yogyakarta, Indonesia.
[14] S. P. Miller, M. W. Whalen, D. D. Cofer. Software model checking takes off. Communications of the ACM, Volume 53 Issue 2, February 2010.
[15] M. W. Whalen, J. D. Innis, S. P. Miller, and L. G. Wagner. ADGS-2100 Adaptive Display and Guidance System Window Manager Analysis. February 2006, NASA/CR-2006-213952.
[16] S. P. Miller, E. A. Anderson, L. G. Wagner, M. W. Whalen, M. P. E. Heimdahl. Formal Verification of Flight Critical Software. AIAA Guidance, Navigation and Control Conference and Exhibit, San Francisco, August 15-18, 2005.
[17] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis. Translating Discrete-Time Simulink to Lustre. Third International Conference EMSOFT, Philadelphia, PA, USA. Proceedings, LNCS 2855, pp. 84–99, Springer-Verlag, Berlin Heidelberg, 2003.
[18] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, P. Niebert. From Simulink to SCADE/Lustre to TTA: a layered approach for distributed embedded applications. Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'03). San Diego, California, USA, June 11-13, 2003.
[19] A. Angermann, M. Beuschel, M. Rau, U. Wohlfarth. Matlab - Simulink – Stateflow; Grundlagen, Toolboxen, Beispiele. 5. Auflage, 2007, Oldenbourg Verlag, München.

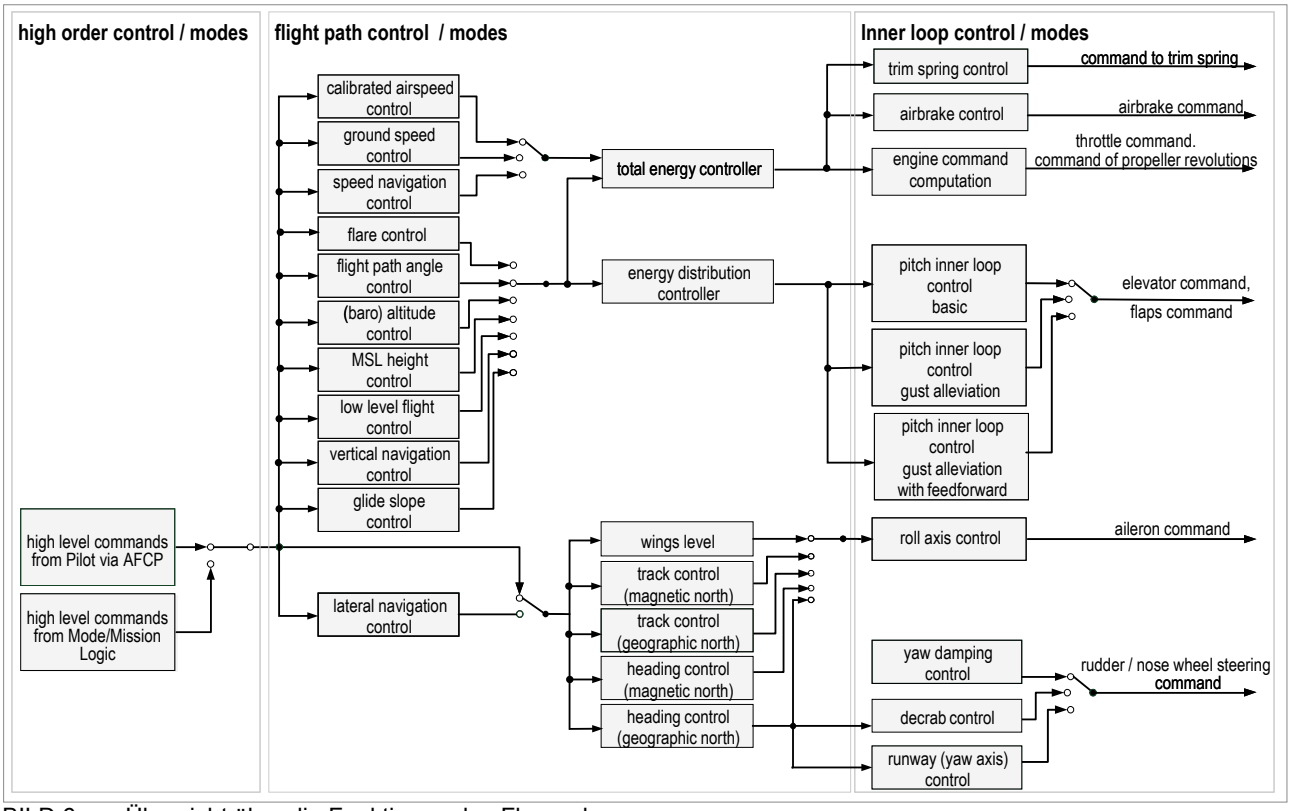

BILD 6. Übersicht über die Funktionen des Flugreglers

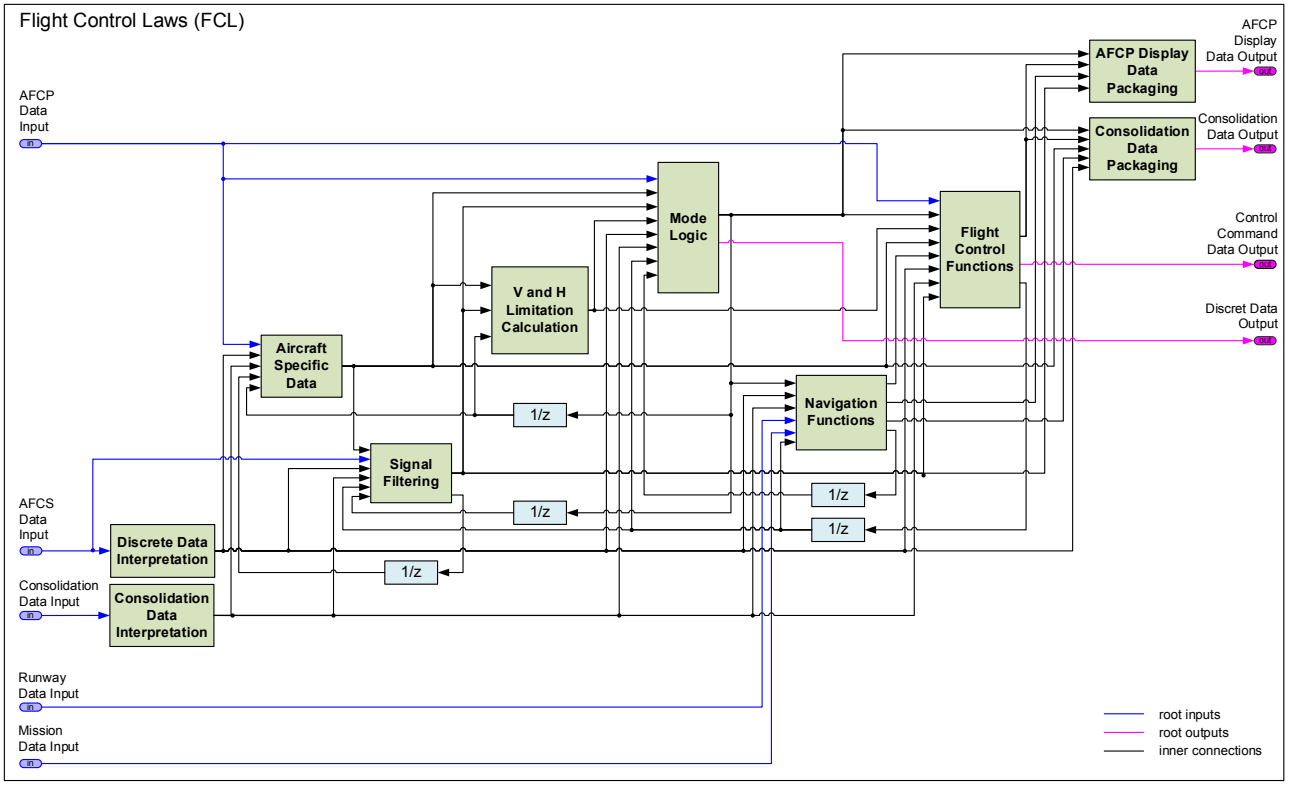

BILD 7. Vereinfachte Darstellung der Software-Architektur der Flugsteuerungsgesetze (zum Testen führt jedes Modul einen weiteren Bus mit Zwischenergebnissen heraus, diese sind hier nicht dargestellt)

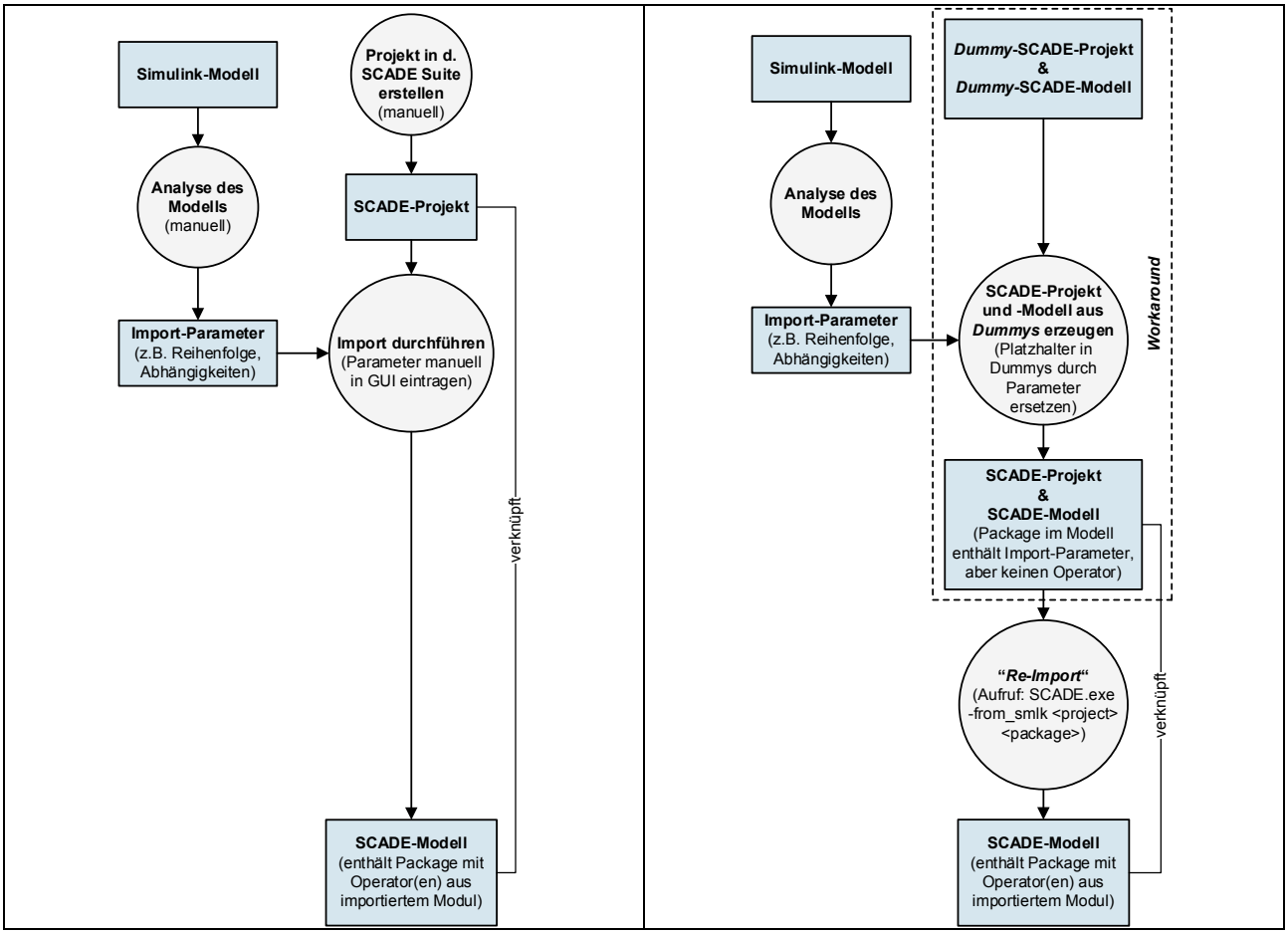

BILD 8. Vereinfachter Ablauf des Imports eines SL/SF-Moduls, links über die grafische Oberfläche des SCADE Suite Gateway for Simulink in der SCADE Suite, rechts automatisiert aus Matlab mit Hilfe eines Workarounds

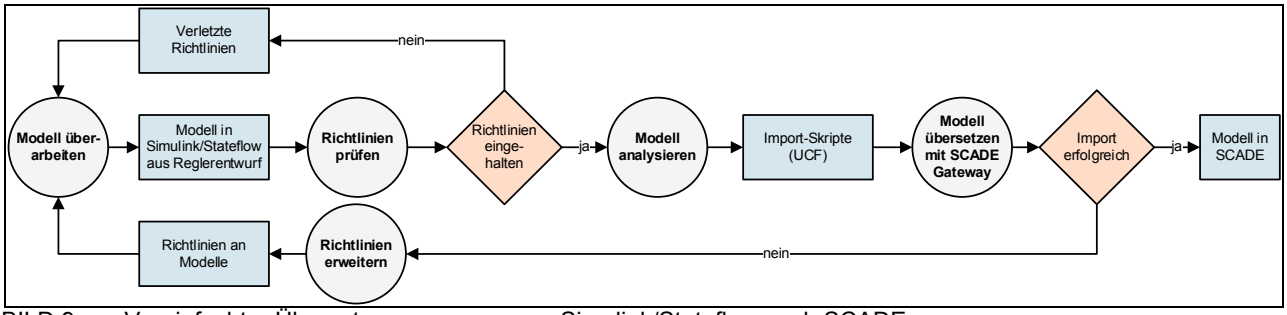

BILD 9. Vereinfachter Übersetzungsprozess von Simulink/Stateflow nach SCADE

Name	ID	Transl.-Time	Transl.-Status	Transl.-Compl.	Check Status	Warnings	Subs.	TLSubs	Blocks	TLBlocks	SFCharts	Path
flightcntr	SMLK_001	20,57	WAHR	WAHR	FALSCH	9	1	1	117	117	0	flightcntr_mdl/…
c2000_WpRwyValDet	SMLK_002	11,72	FALSCH	FALSCH	FALSCH	9	1	0	44	0	4	flightcntr_mdl/…
c3000_FCF	SMLK_003	14,23	WAHR	WAHR	FALSCH	16	1	1	65	65	2	flightcntr_mdl/…
c1000_inlp	SMLK_004	13,24	WAHR	WAHR	FALSCH	10	1	1	93	93	2	flightcntr_mdl/…
c1110_cmdet	SMLK_005	14,33	WAHR	WAHR	FALSCH	13	1	1	55	55	2	flightcntr_mdl/…
c1120_mpibas_et	SMLK_006	13,16	FALSCH	FALSCH	FALSCH	16	1	0	115	0	5	flightcntr_mdl/…
c1130_turncomp	SMLK_007	12,71	WAHR	WAHR	FALSCH	8	1	1	43	43	2	flightcntr_mdl/…
c1140_guaFun	SMLK_008	12,58	WAHR	WAHR	FALSCH	5	1	1	20	20	0	flightcntr_mdl/…
c1140_mpigua	SMLK_009	12,81	WAHR	WAHR	FALSCH	6	1	1	39	39	0	flightcntr_mdl/…
Filt_Th0	SMLK_010	12,64	FALSCH	FALSCH	FALSCH	0	1	0	8	0	0	flightcntr_mdl/…
c1141_Thetacom	SMLK_011	11,71	FALSCH	FALSCH	FALSCH	7	1	0	29	0	2	flightcntr_mdl/…
Filt_Th0	SMLK_012	12,65	FALSCH	FALSCH	FALSCH	0	1	0	8	0	0	flightcntr_mdl/…
c1142_HP-Filter	SMLK_013	11,72	FALSCH	FALSCH	FALSCH	4	1	0	23	0	1	flightcntr_mdl/…
c1150_sw_mpigua_mpibas	SMLK_014	12,59	WAHR	WAHR	FALSCH	5	1	1	21	21	1	flightcntr_mdl/…
c1160_gal	SMLK_015	13,66	WAHR	WAHR	FALSCH	12	1	1	35	35	0	flightcntr_mdl/…
c1161_alw	SMLK_016	13,14	WAHR	WAHR	FALSCH	10	1	1	30	30	1	flightcntr_mdl/…
Al_cg_correction	SMLK_017	13,14	WAHR	WAHR	WAHR	1	1	1	17	17	0	flightcntr_mdl/…
GAL-HP-FIlter2	SMLK_018	13,22	FALSCH	FALSCH	FALSCH	0	1	0	9	0	0	flightcntr_mdl/…
c1162_alw_verzoegerung	SMLK_019	12,80	WAHR	WAHR	FALSCH	1	1	1	17	17	1	flightcntr_mdl/…
Variable Time Delay1	SMLK_020	12,48	WAHR	WAHR	FALSCH	0	1	1	8	8	0	flightcntr_mdl/…

BILD 10. Ausschnitt aus der Zusammenfassung des vollautomatischen, modularen Imports von SL-Subsystemen

```
// Automatically generated UCF File
// Busdef-Nr.: 001 of 131
// Headerfile: -
Type(bu_aeroPar_type) {
  Field = r4CWets: r;
  Field = r4CWets2: r;
  Field = r4CAets: r;
  Field = r4CAets2: r;
  Field = r4CAets3: r;
  Field = r4CMets: r;
  Field = r4CAal: r;
  Field = r4CAal2: r;
  Field = r4CMet: r;
  Field = r4xARP: r;
  Field = r4zARP: r;
  Field = r4lmy: r;
};

...

// Busdef-Nr.: 004 of 131
// Headerfile: "def type IO AFCPcom FCL.h"
Type(bu_wpdt_type) {
  Field = b1wpdt_compl: b;
  Field = ui1wpElements: i;
  Field = ar_r8wpLat_deg: r^250;
  Field = ar_r8wpLon_deg: r^250;
  Field = ar_r4wpH: r^250;
  Field = ar_r4wpV_kmh: r^250;
  Field = ar_ui1wpfo: i^250;
};
...
```

BILD 11. Ausschnitt einer automatisch generierten UCF-Datei mit der Definition von eigenen, zusammengesetzten Datentypen

```
// Automatically generated UCF File for flightcntr_mdl/flightcntr

Block("flightcntr_mdl"/"flightcntr"){
  Interface (7 -> 5){
    Clock = `0.016`;
    ForceClock;
  };
  Controller;
  Translation(bu_AFCSinp_type,bu_consinRM_type, bu_COM2FCL_type,bu_ACspecdt_type,bu_wpdt_type,
bu_rwyTOdt_type,bu_rwyLANDdt_type -> u,u,u,u,u);
};

Block("flightcntr_mdl"/"flightcntr"/"c2000_WpRwyValDet"){
  Interface (6 -> 4){
    Clock = `0.016`;
    ForceClock;
  };
  Translation(bu_infdsp_type,bu_disdt_type,bu_consin_type,bu_wpdt_type,bu_rwyLANDdt_type,
bu_rwyTOdt_type -> bu_LANDprms_type,bu_calcrwyTOdt_type,bu_calcLANDdt_type,
bu_calcWPdt_type){SMLK_c2000_WpRwyValDet::c2000_WpRwyValDet();};
};
...
```

BILD 12. Ausschnitt einer automatisch generierten UCF-Datei mit der Definition von Übersetzungsregeln für den modularen Import