Software ohne Fehler

Code-Generierung für sicherheitsrelevante Anwendungen mit dem Werkzeug Ascet

In der Automobilindustrie ist die modellbasierte Software-Entwicklung und Code-Generierung seit mehr als zehn Jahren Stand der Technik. In vielen Anwendungsbereichen hat sie die herkömmliche Entwicklungsmethodik abgelöst. Auch wenn die modellbasierte Entwicklung alleine die Sicherheit von Software nicht garantieren kann, so trägt deren inhärente Systematik wesentlich zu einer erfolgreichen Entwicklung von sicherheitsrelevanter Software bei.

Von Dr. Darren Buttle

Der Ausgangspunkt für die Einführung der automatischen Code-Generierung war die Idee, die Anzahl iterativer Entwicklungsschritte zu verringern und dadurch die Entwicklungszeit zu verkürzen. Durch die Verwendung von Modellen als Eingangsgröße für die Code-Generierung anstelle von textbasierten Anforderungsdokumenten für die manuelle Programmierung wird ein höherer Grad an Eindeutigkeit erzielt. Zusätzlich ermöglicht die Simulation auf Basis von Modellen das frühzeitige Erkennen von Fehlern im Entwurf von Steuerungen und Regelungen. Das Ergebnis ist, dass Code innerhalb kürzerer Zeit mit einer kleineren Fehlerquote erzeugt werden kann.

Bei der Entwicklung von sicherheitsrelevanter Software sind weitere Aspekte der Code-Generierung von Bedeutung: Sicherheitsstandards, wie beispielsweise die ISO 26262, enthalten Richtlinien für den Einsatz geeigneter Methoden und Maßnahmen in der Software-Entwicklung. Ziel ist es, das Fehlerrisiko zu minimieren und die Wahrscheinlichkeit für den Nachweis der Fehler, welche von den bestehenden Sicherheitsvorkehrungen nicht abgefangen werden, zu maximieren. Teil 6 des Standards ISO 26262 enthält die Richtlinien für die Software-Entwicklung, welche die Spezifikation von Anforderungen, die Architektur und den Entwurf der Software sowie deren Implementierung und Test behandeln.

Der vorliegende Artikel diskutiert, wie übliche Fehler beim Entwurf und in der Implementierung mit Hilfe der automatischen Code-Generierung vermieden und eine frühzeitige Absicherung des Entwurfs erreicht werden kann. Die Beispiele beschreiben Anwendungen des modellbasierten Entwicklungswerkzeugs Ascet von ETAS, welches seit vielen Jahren in der Automobilindustrie etabliert ist.

Vermeiden von Fehlern im Software-Entwurf

Die folgenden Aspekte der modellbasierten Entwicklung helfen dabei, Fehler im Entwurf der Software von Steuerungen und Regelungen zu vermeiden:

- Modelliersprachen abstrahieren die Details der Implementierung auf einer neuen semantischen Ebene, die es erlaubt, das korrekte Zusammenwirken logisch nachzuvollziehen. Eine Modelliersprache ist zwar grundsätzlich nicht in der Lage, die Fehlerfreiheit einer Implementierung zu garantieren, jedoch lassen sich Implementierungsfehler durch eine sorgfältige Modellierung deutlich verringern.
- Frühzeitige Verifikation eines Modells in der Simulation.

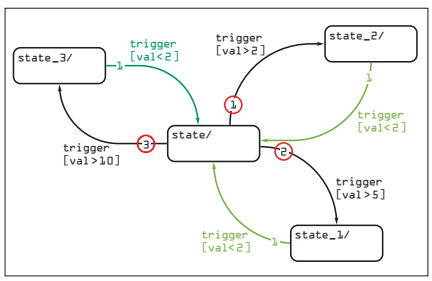


Bild 1. Ascet hilft dabei, auch in komplexen Systemen den Überblick zu behalten.

Modularität, Abstraktion und Kapselung

Komplexität ist eine der maßgeblichen Ursachen von Entwicklungsfehlern. Häufig lässt sich ein Entwurf in seiner Gesamtheit nicht von allen daran Beteiligten in allen Feinheiten verstehen. Komplexe Systeme lassen sich jedoch besser verstehen und beherrschen, wenn die Details auf der unteren Ebene abstrahiert und das System in Teile zerlegt wird, die leicht zu handhaben und zu testen sind.

Ascet verwendet als Framework für das Erstellen und Zusammenführen von Entwürfen eine objektbasierte Modellierungssprache mit einer hierarchischen Klassenstruktur. Durch die genaue Definition der intermodularen Schnittstellen sind anderen Systemteilen nur die jeweils notwendigen Informationen zugänglich (**Bild 1**). Hierdurch wird sowohl der beabsichtigte als auch der unbeabsichtigte Zugriff auf interne Funktionen von Modulen vermieden.

Die Interaktion zwischen verschiedenen Modulen erfolgt durch Nachrichten, die gesendet oder empfangen

werden (**Bild 2**). Jede Nachricht verfügt über genau einen Sender und einen oder mehrere Empfänger. Dadurch werden mögliche Sequenzierungsprobleme zwischen Sendern und Empfängern vermieden und sichergestellt, dass die Empfänger immer die aktuellsten Daten erhalten.

Vermeiden von Mehrdeutigkeiten

Entscheidend ist, dass alle Beteiligten den Entwurf in der gleichen Art und Weise interpretieren. Hierbei sind zwei Aspekte zu beachten:

- Effiziente Kommunikation zwischen Experten über Domänengrenzen hinweg.
- Unmissverständliche Kommunikation zwischen Experten innerhalb einer Domäne.

Ascet adressiert den ersten Punkt, indem es unterschiedliche syntaktische Repräsentationen der zu Grunde liegenden Modellierungssprache zur Verfügung stellt (**Bild 3**). Beispielsweise können Regelungstechniker Kontroll- und Datenflussdiagramme entwerfen, während Software-Entwickler Modelle aus der Sicht der Programmierung aufbauen können.

Aus dem zweiten Punkt ergeben sich Konsequenzen für die Modellierungssprache: Eine grafische Notation kann zu Problemen führen, weil die Reihenfolge von Berechnungen nicht – wie in einer textbasierten Notation – intuitiv gegeben ist. Eine Festlegung per grafischer Anordnung, zum Beispiel von oben nach unten oder von links nach rechts, kann in die Irre führen. Hinzu kommt, dass Änderungen in der Anordnung, welche etwa durchgeführt werden, um die Bewertung zu unterstützen, unbeabsichtigte Änderungen im Entwurf und im Verhalten einer Steuerung oder Regelung nach sich ziehen können. Die grafische Notation von Ascet trifft keine impliziten Annahmen über die Berechnungsreihenfolgen, sondern verlangt deren explizite Angabe in Form von Sequenznummern.

Daraus ergeben sich die folgenden Vorteile:

- Ascet-Modelle sind per Konstruktion immer deterministisch.
- Das Verhalten von Ascet-Modellen ist unabhängig von der grafischen Anordnung. Die Semantik hängt ausschließlich von den Beziehungen zwischen den Modellelementen ab.
- Die Berechnungsreihenfolge wird im Modell dargestellt und in den Programm-Code bei dessen Generierung übertragen. Daraus ergibt sich automatisch ein Framework für Bewertungen. **Bild 4** zeigt die Nummerierung der Berechnungsreihenfolge am Beispiel von Übergängen eines Zustandsautomaten.

Sichere numerische Operationen

Ascet ermöglicht es, valide Wertebereiche, welche realen Umgebungsbedingungen entsprechen oder sich aus der Bedeutung von Berechnungsreihenfolgen ergeben, festzulegen und zu formalisieren. Beispielsweise dürfen Geschwindigkeitswerte nicht kleiner Null sein, ebenso wenig dürfen Temperaturwerte, die unterhalb von –273,15 °C liegen, als gültig interpretiert werden. Es können sogar komplette Gültigkeitsintervalle, z.B. für Sensoren, definiert werden.

Die Wertebereiche bilden auch die Grundlage für die automatische Generierung eines Laufzeitschutzes bei Grenzwertverletzungen und von Grenzwerten für die nachfolgenden Unit-Tests.

Zusätzlich dazu unterstützt Ascet Festkomma-Arithmetik. Gleitkomma-Arithmetik kann in sicherheitsrelevanten Anwendungen zu Problemen führen. So lässt sich die faktische Gleichheit von Werten schwer feststellen. In Folge dessen können beispielsweise beim Kürzen schwerwiegende Genauigkeitseinbußen auftreten.

Unterstützung von Echtzeit-Systemen

Ascet abstrahiert die Echtzeit-Details der Low-Level-Implementierung und kapselt gleichzeitig auszuführende Objekte in Prozesse.

Die Code-Generierung übernimmt den fehleranfälligen Vorgang des Abbildens paralleler Prozesse auf das darunter liegende Scheduling-Modell eines Echtzeit-Betriebssystems. Dabei wird sichergestellt, dass jedem Prozess eine zeitlich konsistente Datensicht zur Verfügung steht. Die automatische Generierung des Codes verhindert, dass der Entwurf mit unnötigen Informationen belastet wird. Gleichzeitig ist der Code robust gegenüber Entwurfsänderungen: Änderungen der Abtastraten eines Prozesses haben keinen Einfluss auf die Implementierung anderer Prozesse.

Absicherung des Entwurfs in der Simulation

Eine Modellierungssprache ist nicht in der Lage, alle Fehler im Entwurf der Steuerungs- und Regelungs-Software zu verhindern; zum Beispiel können falsch interpretierte Anforderungen oder unbegründete Annahmen nur im Verlauf der Entwicklung identifiziert werden. Um das geplante Systemverhalten sicherzustellen, müssen Verifizierungs- und Validierungsaktivitäten so umfassend und früh wie möglich einsetzen. Die Formalisierung des Entwurfs als ausführbares Modell ermöglicht frühzeitige Tests, mit deren Hilfe grundlegende Annahmen des Entwurfs abgesichert werden können. Bestimmte Merkmale, die über alle Test- und Integrationsebenen Bestand haben sollen, können schnell und einfach im Entwurf verankert werden. Gleichermaßen lässt sich das Verhalten von unterschiedlichen Entwurfsalternativen

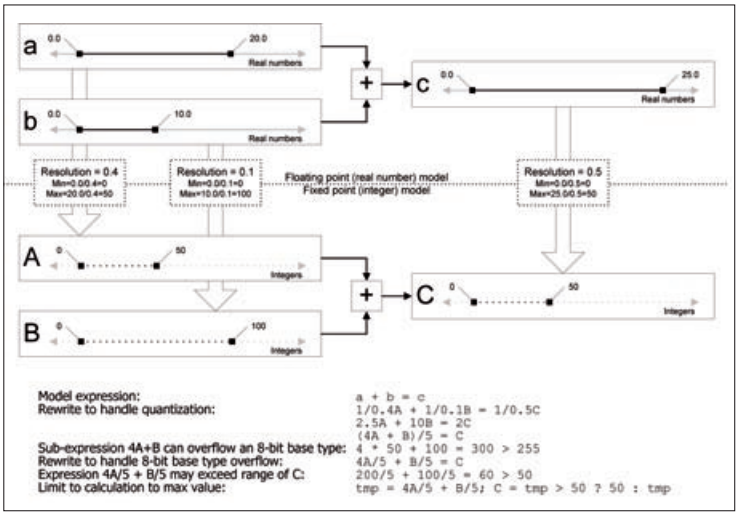
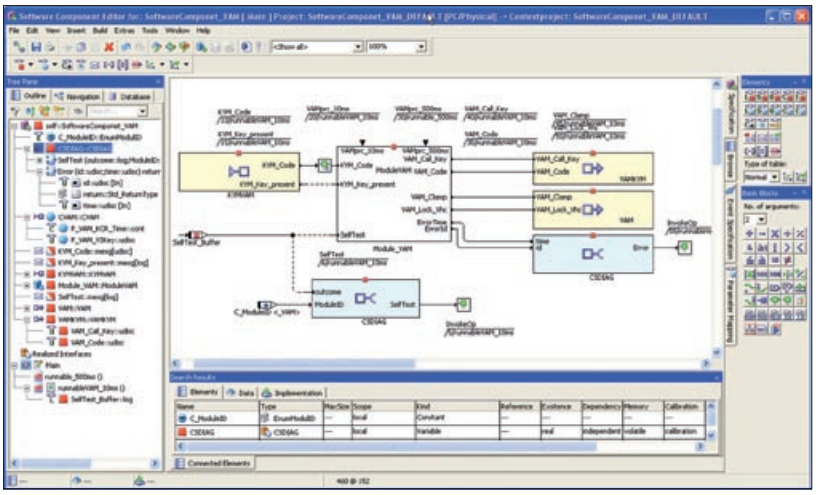


Bild 2. Ascet stellt die eindeutigen Beziehungen von Nachrichten sicher.



I Bild 3. Grafische Modelle schaffen Übersicht.

in Situationen, die nur mit hohem Aufwand mit dem realen System getestet werden könnten, simulieren.

Die Ergebnisse von entwurfsbasierten Simulationen können als Bezug für Tests des Seriensystems dienen und Abweichungen zwischen Testresultaten, die mit dem Embedded-Target erzielt werden, und Ergebnissen der Simulation näher untersucht werden.

Sinn und Zweck der entwurfsbasierten Simulation ist es nicht, den Systemtest zu ersetzen, sondern die Erfolgswahrscheinlichkeit von Systemtests durch ein frühzeitiges Erkennen von kritischen Fehlern und Fehlfunktionen zu maximieren.

Automatische Code-Generierung sorgt für reproduzierbare Ergebnisse

Die Methode der automatischen Code-Generierung führt zu absolut reproduzierbaren Ergebnissen: Die gleiche Eingabe erzeugt ohne Ausnahme die gleiche Ausgabe. Darüber hinaus muss ein Code-Generator auf eine zuverlässige Art und Weise richtigen Code erzeugen: Die vorherrschende Sprache für die Programmierung von Software für Fahrzeugsteuergeräte ist C. Die manuelle Programmierung von sicherheitsrelevanter Software in C wird durch eine Vielzahl von Möglichkeiten für fehlerhafte Programmierung und Fällen, in denen das Verhalten oder die Kompilierung der Sprache undefiniert ist, erschwert. Durch eine automatische Code-Generierung lassen sich diese Probleme systematisch vermeiden. Empirisch wurde belegt, dass die Fehlerrate in generiertem C-Code eine Größenordnung unter der Fehlerrate von manuellem C-Code hoher Güte liegt [1, 2]. Dabei hängt die hohe Qualität nicht vom Können des Programmierers ab und wird auf konstant hohem Niveau an 365 Tagen im Jahr erreicht.

Natürlich können Fehler im Code-Generator zu Fehlern in der generierten Software führen. Aufgrund der Reproduzierbarkeit der Code-Generierung lassen sich diese Fehler aber vergleichsweise einfach identifizieren. Nach einer Korrektur des Code-Generators können die Instanzen der Fehler durch eine erneute Generierung systematisch und zuverlässig aus dem Code entfernt werden.

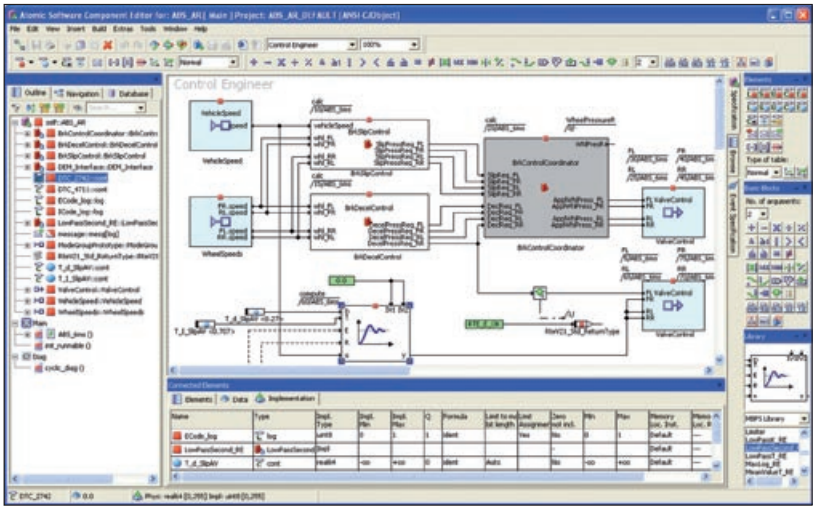
Automatisches Einhalten von Spracheinschränkungen

Der Standard ISO 26262 definiert in Teil 6 Abschnitt 8 Implementierungsrichtlinien, die sich an die in MISRA-C:2004 enthaltenen Richtlinien für die Anwendung von C in kritischen Systemen anlehnen [3]. Die Richtlinien in ISO 26262 sind größtenteils unabhängig von der Art der Programmierung. Allerdings gibt es einige Hinweise auf Fälle, in denen automatisch generierter Code von den Richtlinien abweichen kann. Ungeachtet dessen ist es bei Projekten, die sowohl generierten als auch manuellen Code enthalten, von Vorteil, einen einheitlichen Satz von Programmierregeln anzuwenden. In diesem Fall müssen bei der Verwaltung von Code aus unterschiedlichen Quellen keine gesonderten Vereinbarungen getroffen und beachtet werden.

Ohne wesentliche Auswirkungen auf das Eingabemodell oder auf die Laufzeit unterstützt Ascet diesen Ansatz durch die Generierung von bis zu 100 Prozent MISRA-C:2004-konformem C-Code. Das vereinfacht die Überprüfung von Quell-Code, der sich aus manuellen und automatisch generierten Anteilen zusammensetzt.

Vermeidung von Laufzeitfehlern

Zur Vermeidung von Laufzeitfehlern verlangt die Regel 21.1 des Standards MISRA-C:2004 explizite Prüfungen. Durch das Einfügen derartiger Prüfungen an beliebigen Stellen verliert der Code an Effizienz. Andererseits ist das Identifizieren derjenigen Stellen, an denen solche Prüfungen erforderlich sind oder ohne nachteilige Auswirkungen weggelassen werden können, gleichermaßen schwierig und fehlerträchtig.



I Bild 4. Festlegung der Berechnungsreihenfolge am Beispiel von Übergängen eines Zustandsautomaten.

Code-Generatoren sind in der Lage, die Stellen im Code, an denen Laufzeitfehlerprüfungen eingebaut werden sollen, systematisch zu finden. Ascet generiert explizite Prüfungen, um folgende Laufzeitprobleme zu vermeiden:

- Division durch Null.
- Vorzeichenbehafteter Überlauf (der Versuch, die kleinste von der Hardware darstellbare ganze Zahl durch -1 zu dividieren).
- Unterlauf und Überlauf von Prozessor-Datentypgrößen.
- Überschreiten von Wertebereichsgrenzen (Verwendung von Wertebereichen, welche durch das Modell festgelegt sind, zur Ausnutzung von saturierender Arithmetik).
- Überschreiten von Indexgrenzen bei Arrays (Verhindern der Indizierung über den spezifizierten Index-Bereich hinaus).

Dr. Darren Buttle
arbeitet bei der ETAS GmbH in Stuttgart als Produkt-Manager für Embedded-Anwendungen im Bereich Software Engineering. Er war verantwortlich für die ISO-26262-Zertifizierung des Ascet-Code-Generators.

Markieren von potentiellen Problemen bei der Kompilierung

Viele Programmiersprachen – so auch C – legen keine Vorschriften für die Reihenfolge der Auswertung eines Ausdrucks fest. Dadurch entstehen Probleme bei der Verwendung von Funktionen mit Seiteneffekten, die sich durch eine Änderung des Funktionszustands beim Aufruf ergeben.

Ein Code-Generator ist nicht in der Lage, die Auswertungsreihenfolge bei der Kompilierung festzulegen. Er kann jedoch Code generieren, der durch die Auswertungsreihenfolge nicht verändert wird oder – wenn das nicht möglich ist – diejenigen Fälle kennzeichnen, in denen die Reihenfolge Einfluss auf das Ergebnis der Auswertung eines Ausdrucks hat. Dadurch lassen sich potentielle Risiken zielgerichtet erkennen. Der Code-Generator von Ascet unterstützt beide Methoden. *sj*

Literatur

[1] *German, A.:* Software Static Code Analysis Lessons Learned. CrossTalk Magazine, The Journal of Defense Software Engineering. November 2003.

[2] *Stürmer, I.; Weinberg, D.; Conrad, M.:* Overview of existing safeguarding techniques for automatically generated code. ACM Sigsoft Software Engineering Notes. 2005.

[3] MISRA:C-2004 Guidelines for the C language in critical systems. Edition 2 including Technical Corrigendum 1. MIRA Limited. July 2008.