# Sichere Integration von Fremdsoftware: Lösungskonzepte für eine Problemstellung aus der Praxis

Jörn Schneider, Vincent Schulte-Coerne

Robert Bosch GmbH, Forschung und Vorausentwicklung, Stuttgart
joern.schneider@de.bosch.com, vincent.schulte-coerne@de.bosch.com

## 1 Einleitung

Integration von Fremdsoftware wird zunehmend zu einer Schlüsselfähigkeit der Automobilzulieferer, die es effizient zu beherrschen gilt. Hierbei existiert ein breites Spektrum von Anwendungsfällen mit unterschiedlichsten Anforderungen an Sicherheit (im Sinne von Safety und Security) und Zuverlässigkeit. Beispiele reichen von extern entwickelten Komfortfunktionen bis zu Softwarekomponenten des OEM zur Realisierung markenspezifischen Verhaltens bei sicherheitsrelevanten Systemen.

Eine wesentliche Aufgabe bei der Fremdsoftwareintegration ist es, die Wechselwirkungen zwischen einzubettender Software und umgebendem System auf beabsichtigte Wirkketten zu beschränken. Konkret wird hier der Fall betrachtet, dass Fremdanwendungen und eigenentwickelte Software auf dem gleichen Steuergerät ablaufen. Durch die Nutzung gemeinsamer Ressourcen (z.B. Speicher, Rechenzeit, oder Bibliotheksfunktionen) ergeben sich unerwünschte Einflussmöglichkeiten.

Eine von uns intern durchgeführte Studie [Sc03] konkreter Projekte aus der Produktentwicklung liefert eine Reihe von Anforderungen aus der Praxis, z.B.: Eignung der Lösungskonzepte für Fremdsoftwarezulieferung in Form von Object-Code, möglichst geringer zusätzlicher Ressourcenverbrauch auf dem Steuergerät und Verträglichkeit mit aktuellen und zukünftigen Szenarien der Zusammenarbeit mit Automobilherstellern, -zulieferern und Unterauftragsnehmern. Teilweise können diese Anforderungen durch den Einsatz formaler Methoden erfüllt werden. Insbesondere Ansätze zur statischen Programmanalyse, z.B. abstrakte Interpretation, sind hierfür prädestiniert.

Im folgenden Abschnitt wird das Problem realisierungsbedingter Fremdsoftwareeinflüsse anhand von drei Teilproblemen genauer betrachtet. Das von uns vorgeschlagene Lösungskonzept setzt auf eine Kombination formaler Methoden mit plattformbasierten Schutzmechanismen und wird in Abschnitt 3 beschrieben. Das Konzept erlaubt eine umfassende Absicherung gegen unerwünschte Einflüsse zugelieferter Software. Der letzte Abschnitt dient der Zusammenfassung.

# 2 Problemstellung

## 2.1 Strukturelle Integrität des Datenflusses

Unzulässige Veränderungen von Anwendungsdaten durch eine Fremdsoftware können schwerwiegende Folgen bis hin zum Tod von Verkehrsteilnehmern haben. Gleiches gilt für Konfigurationsänderungen der HW oder Variationen innerer Zustände von Plattformsoftware, etwa über Datenschnittstellen zu Treibern oder Betriebssystem.

Bei der Integration von Fremdsoftware gibt es aber auch Gründe Lesezugriffe einzuschränken. Beispielsweise werden bei regelungstechnischen Anwendungen nicht messbare physikalische Werte mit Hilfe interner Modelle näherungsweise berechnet. Die Modellgüte ist auf die jeweilige Anwendung optimiert. Wenn Fremdanwendungen mit höheren Güteanforderungen solche internen Werte verwenden, treten zwangsläufig Fehler auf. Bei der Integration von Fremdsoftware wirft dies auch Haftungs- und u.U. Lizenzrechtliche Fragen auf.

Um die strukturelle Integrität des Datenflusses zu wahren, muss der Datenfluss zwischen Fremdsoftware und Restsystem auf die essentiellen, d.h. auf die aufgrund von Anforderungen identifizierten, Kommunikationskanäle und -richtungen beschränkt werden. Mögliche Kommunikationskanäle umfassen steuergerätinterne (z.B. gemeinsame Speicherbereiche) und steuergerätexterne Kanäle (z.B. über Aktorik, Sensorik, oder andere Steuergeräte). Ein besonderes Problem stellen mittelbare Zugriffe (z.B. DMA oder Zugriff über eine Hardware Abstraction Layer) dar.

## 2.2 Strukturelle Integrität des Kontrollflusses

Die Verwendung von nicht dafür vorgesehenem Code durch Fremdsoftware birgt auch dann Probleme, wenn es sich nicht um einen Programmierfehler handelt. **Kompatibilitätsprobleme:** Durch Unterlaufen des *Information hiding* Prinzips kann es bei der Anpassung anwendungsinterner Strukturen zu Fehlfunktionen kommen. **Inkonsistente Zustände:** Interne Routinen sind oft auf Aufrufssituationen zugeschnitten bzw. hinterlassen das System in einem Zustand, der nur in speziellen Situationen zulässig ist. Die Verwendung interner Routinen außerhalb des zulässigen Kontextes kann zu schweren Fehlern führen. **Rechteausweitung:** Über unzulässige Aufrufe privilegierter Funktionen kann eine Fremdsoftware Zugriff auf verbotene Systemteile erlangen. **Rechtliche Probleme:** Ähnlich wie im vorigen Abschnitt diskutiert, kann auch die Verwendung von nicht vorgesehenem Programmcode Haftungs- und Lizenzrechtliche Probleme aufwerfen.

Die Absicherung des Kontrollflusses ist auch bei vorhandener Datenflussabsicherung notwendig. Die Manipulation von freigegebenen Daten mit nicht für die Verwendung durch die Fremdsoftware vorgesehenem Code ist keineswegs unbedenklich.

### 2.3 Sicherstellung des Echtzeitverhaltens

Ein essentieller Aspekt von Echtzeitanwendungen ist die Einhaltung maximaler Antwortzeiten. Diese lassen sich nur dann garantieren, wenn die Verzögerungen durch andere Anwendungen und durch Plattformsoftware im dafür vorgesehenen Rahmen bleiben. Konkret bedeutet dies, dass die maximal zugestandene Rechenzeit und Aktivierungshäufigkeit einer Fremdsoftwarekomponente sicher eingehalten werden muss.

## 3 Lösungsansätze

### 3.1 Offline Analyse

Mit Hilfe von Methoden zur statischen Programmanalyse [NNH99] lassen sich konservative, d.h. sichere aber möglicherweise pessimistische, Aussagen über die betrachteten Eigenschaften (Datenfluss, Kontrollfluss und Echtzeitverhalten) machen. Der Einsatz solcher Methoden wird erleichtert durch die vergleichbar einfachen Softwarestrukturen von Systemen mit harten Echtzeitanforderungen im Automobilbereich. Nicht eingesetzt werden i.d.R.: Dynamische Speicherallokation, Zeiger, dynamische Prozesserzeugung und Prozessauslagerung.

**Datenfluss** Beim Datenfluss müssen das interne Speicherzugriffsverhalten und die Zugriffe auf externe Kommunikationskanäle analysiert werden. Insbesondere mittelbare Zugriffe (etwa über DMA bzw. über eine HAL) und Datenflüsse bei der HW-Konfiguration (z.B. von Peripheriebausteinen) sind dabei zu berücksichtigen. Entsprechenden Analysetools muss die jeweilige Systemkonfiguration (HW-, Plattformsoftware-, und Anwendungssoftwarestrukturen) zur Verfügung gestellt werden. An Aufgaben stellt sich hier insbesondere die Auswahl bzw. Definition geeigneter Systemkonfigurationssprachen und die Entwicklung entsprechend mächtiger Analysetools.

Kommerziell verfügbare Tools werden diesen Anforderungen derzeit nicht gerecht und können lediglich zur punktuellen Analyse des Datenflusses verwendet werden. Beispiele dazu sind: C Verifier (PolySpace Technologies), ein Werkzeug das potentielle Laufzeitfehler (u.a. Überschreitung von Feldbereichsgrenzen) erkennt und StackAnalyzer (AbsInt GmbH), ein Tool zur Ermittlung des maximal verwendeten Stackbereichs. Weiterhin fehlt es noch an Mitteln zur automatischen Verifikation der a priori festgelegten Datenflüsse und an passenden Visualisierungstools.

**Kontrollfluss** Kontrollflussanalyse ist sowohl eine notwendige Voraussetzung für Analysen des Datenflusses und des Echtzeitverhaltens als auch unabdingbar zur Sicherstellung korrekten Verhaltens einer Softwarekomponente bzgl. des Kontrollflusses. Damit können neben fehlerhafter Programmierung der Kontrollstrukturen auch die in 2.2 genannten Integritätsprobleme im Kontrollfluss frühzeitig erkannt werden.

**Echtzeitverhalten** Über ein geeignetes Schedulingverfahren (z.B. Online Scheduling mit statischen Prioritäten) und eine passende Methode zur Response Time Analyse [Sc00, Sc02] kann die Einhaltung maximaler Antwortzeiten von Tasks und Interrupt Service Routinen (ISRs) nachgewiesen werden. Voraussetzung dafür ist, dass die maximale zeitliche Beeinträchtigung durch andere Tasks und ISRs sowie das Echtzeitbetriebsystem ermittelt werden kann. Für alle potentiell störenden Softwareteile muss also eine obere Schranke der maximalen Laufzeit (*Worst Case Execution Time* (WCET)) und die maximale Aktivierungshäufigkeit bekannt sein. Die erstgenannte Größe kann i.d.R. nicht gemessen werden, da der Worst Case nicht bekannt ist und eine komplette Abdeckung von Eingangswertebereichen nicht praktizierbar ist. Stattdessen muss die WCET mit Hilfe eines Analyse Tools ermittelt werden [FKM+03].

### 3.2 Schutzmaßnahmen zur Laufzeit

Die Überwachung der zugewiesenen Einflussbereichsgrenzen und Verhinderung entsprechender Grenzübertretungen zur Laufzeit mit Hilfe des Echtzeitbetriebssystems ist der eher klassische Ansatz, wie er beispielsweise in Anwendungen aus dem Bereich Luft- und Raumfahrt eingesetzt wird. Die im Automobilbereich eingesetzten Überwachungsmaßnahmen sind traditionell eher anwendungsspezifisch. Bei sicherheitsrelevanten Systemen gibt es aufgrund neuer Anwendungen auch hier einen Trend zu plattformbasierten Maßnahmen.

Zur effizienten Realisierung von Schutzmaßnahmen zur Laufzeit ist eine entsprechende Unterstützung durch die HW (z.B. Memory Protection Unit und Timer) und das Echtzeitbetriebssystem notwendig. Echtzeitbetriebssysteme, die Anforderungen dieses Konzeptes in Teilen gerecht werden, sind beispielsweise die für den Luftfahrtbereich entwickelten Betriebssysteme Deos™ von Honeywell [CR02] und INTEGRITY® von Green Hills. Für den Einsatz im Automobil gibt es derzeit lediglich den Prototyp eines OSEK kompatiblen Betriebssystems von LiveDevices [ETA03].

Eine wesentliche Bedeutung kommt der Fehlerbehandlungsstrategie zu. Übliche Konzepte sehen die Terminierung von Tasks vor, die eine Verletzung festgelegter Regeln begehen. Damit wird zwar die Einhaltung der Einflussbereichsgrenzen sichergestellt, aber gleichzeitig die Funktionalität der Fremdsoftware außer Kraft gesetzt, wenn keine besonderen Maßnahmen getroffen werden.

### 3.3 Kombination von Offline Analysen und plattformbasierten Schutzmaßnahmen

Zur Integration von Fremdsoftware wird i.d.R. nur Objectcode geliefert, um geistiges Eigentum zu schützen. Dies erschwert den Einsatz von Methoden der statischen Programmanalyse teilweise erheblich. Ein simpler Ansatz dieses Problem zu umgehen, ist die Verwendung der Analysetools durch den Softwarezulieferer oder durch eine Zertifizierungsstelle. In Bezug auf WCET Analysen kommt hinzu, dass aus Kostengründen nicht jedes

System auf den Worst Case ausgelegt wird, dies wird i.d.R. nur bei sicherheitsrelevanten Anwendungen gemacht.

Vorteile beim Einsatz von Offline Analysen sind: Höhere Software Qualität, weniger Feldprobleme, kein Verbrauch von Steuergerätressourcen und keine aufwendige Fehlerbehandlung zur Laufzeit. Für die Überwachung zur Laufzeit spricht, dass sie auch ohne Quellcode möglich ist und auch gegen HW-Fehler schützt.

Wir propagieren den kombinierten Einsatz der oben beschriebenen Methoden. Die Vorteile der jeweiligen Methoden bleiben dabei im wesentlichen erhalten. Hinzu kommt, dass sich beide Ansätze hervorragend ergänzen, da sich für die Analyse gemachte Annahmen, etwa bzgl. möglicher Zustände der Umgebung, durch Laufzeitmaßnahmen absichern lassen.

## 4 Zusammenfassung

Der von uns vorgeschlagene Lösungansatz auf der Basis von statischer Programmanalyse und plattformbasierten Schutzmechanismen wird den in unserer Studie zur Fremdsoftwareintegration ermittelten Anforderungen der Praxis gerecht. Darüber hinaus ist der Ansatz auch für die Softwareintegration im Allgemeinen einsetzbar. Die konsequente Umsetzung dieses Lösungsansatzes würde eine deutliche Verbesserung des State-of-the-Art bei der Softwareentwicklung und -integration im Automotive Bereich bedeuten.

## Literatur


[CR02] Cofer, D. und Rangarajan, M.: Formal Modeling and Analysis of Advanced Scheduling Features in an Avionics RTOS. In: *Proceedings of the 2nd International Workshop on Embedded Software, EMSOFT'02*. Lecture Notes in Computer Science. Springer. 2002.

[ETA03] ETAS GmbH: *Overview of the ATG Protected OSEK OS (APOS)*. June 2003.

[FKM+03] Ferdinand, C., Kästner, D., Martin, F., Langenbach, M., Sicks, M., Wilhelm, S., Heckmann, R., Fritz, N., Thesing, S., Fontaine, F., Theiling, H., Schmidt, M., Evstiougov-Babaev, A., und Wilhelm, R.: Validierung des Zeitverhaltens von kritischer Echtzeit-Software. In: *Proceedings of the 1st GI Workshop Automotive SW Engineering and Concepts*. 2003.

[NNH99] Nielson, F., Nielson, H. R., und Hankin, C.: *Principles of Program Analysis*. Springer. 1999.

[Sc00] Schneider, J.: Cache and Pipeline Sensitive Fixed Priority Scheduling for Preemptive Real-Time Systems. In: *Proceedings of the 21st IEEE Real-Time Systems Symposium*. S. 195–204. November 2000.

[Sc02] Schneider, J.: *Combined Schedulability and WCET Analysis for Real-Time Operating Systems*. PhD thesis. Universität Saarbrücken. 2002. Shaker Verlag.

[Sc03] Schneider, J.: Recommendations and guidelines on techniques and processes for the safe integration of third-party software. Technical report. Robert Bosch GmbH, Research and Development. 2003. Internal Report.