Konfigurationsprüfung für Standardsoftware mit Hilfe von Merkmalmodellen

Verification of standardsoftware configurations with feature models

Dipl.-Ing. (FH) **Wolfgang Frieß**, AUDI AG, Ingolstadt
Dipl.-Wi.-Inf. (FH) **Stefan Kubica**, Audi Electronics Venture GmbH, Ingolstadt
Dr. techn. **Andreas Krüger** (MBA), AUDI AG, Ingolstadt
Prof. Dr.-Ing. habil. **Wolfgang Schröder-Preikschat**, Friedrich-Alexander-Universität, Erlangen

Kurzfassung

Die Konfiguration der Standardsoftware in heutigen Kfz-Elektroniksystemen ist bedingt durch die Wechselwirkungen, die zwischen den einzelnen Modulen auftreten, ein komplexer Vorgang. Bisher gibt es noch kein zufriedenstellendes Konzept, mit dem sich eine korrekte Konfiguration toolunterstützt sicherstellen lässt. In diesem Beitrag soll ein entsprechendes Konzept vorgestellt werden.

Abstract

In automotive electronics systems, the configuration of standardsoftware is a complex process due to the interdependencies between the several modules. Up to now, there is no satisfying, tool-supported concept to ensure a correct configuration. This paper presents such a concept.

1. Einleitung

Bei der Entwicklung neuer Steuergeräte wird neben der eigentlichen Anwendungs-Software auch die zugrundeliegende Standardsoftware immer komplexer und umfangreicher. Diese realisiert die anwendungsunabhängigen Grundfunktionen eines Steuergeräts, wie zum Beispiel die Anbindung an die verschiedenen Bussysteme. Standardsoftware besteht aus verschiedenen Modulen, die sich für den jeweiligen Anwendungsfall individuell parametrieren und kombinieren lassen. Bei dieser Konfiguration gibt es zusätzlich Abhängigkeiten und Wechselwirkungen zwischen den einzelnen Modulen sowie projektspezifische Anforderungen, die berücksichtigt werden müssen. Sicherzustellen, dass die Standardsoftware korrekt in die Steuergeräte eingebunden wird, ist Ziel des

Automobilherstellers, da eine fehlerfrei funktionierende Standardsoftware die Grundlage für ein fehlerfrei funktionierendes Gesamtsystem Fahrzeug darstellt.

2. Herausforderungen der Standardsoftware

Viele Automobilhersteller haben derzeit ihre eigene, also OEM-spezifische Standardsoftware (OEM = Original Equipment Manufacturer). Die einzelnen Module der Standardsoftware werden in der Regel von verschiedenen Softwarefirmen implementiert. Das Ergebnis ist ein modularer Baukasten, den die OEM's in Kooperation mit den Steuergeräte-Zulieferern in ihren diversen Fahrzeugbaureihen verwenden. Als Beispiel für eine OEM-spezifische Standardsoftware zeigt Bild 1 den Standardsoftware Core des Volkswagen Konzerns.

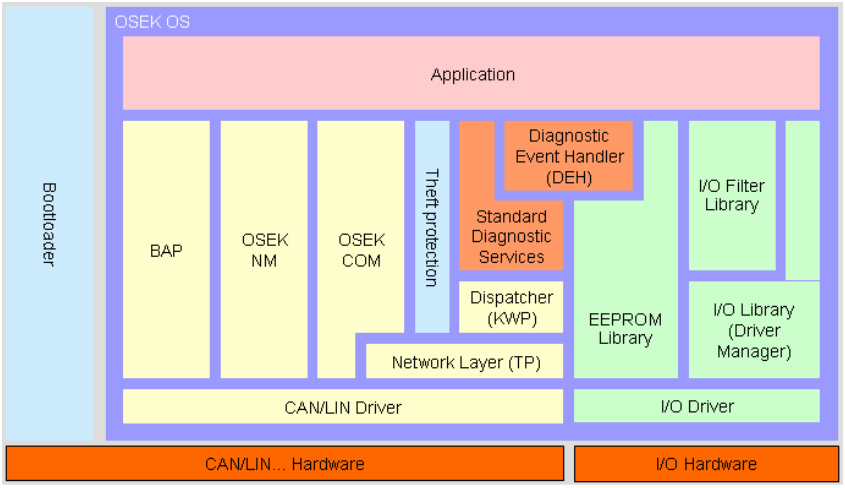

Bild 1: Standardsoftware Core des Volkswagen Konzerns / standardsoftware core of the volkswagen group

Aufgrund der Komplexität und Vielzahl von Standardsoftware Modulen ergeben sich sehr viele Konfigurationsmöglichkeiten. Dies erfordert detailliertes Wissen des Systemintegrators über die Funktionsweise der Standardsoftware Module, da neben den Konfigurationsanforderungen an die einzelnen Module auch die Wechselwirkungen zwischen den Modulen berücksichtigt werden müssen. Unterschiedliche, oft herstellerspezifische Konfigurationsformate für die diversen Module erschweren die Modulintegration zusätzlich. Dabei können auch Module mit der gleichen Funktionalität, abhängig vom Hersteller, unterschiedliche Konfigurationsformate haben.

Gerade die Wechselwirkungen zwischen Modulen werden von aktuellen Konfigurationskonzepten kaum berücksichtigt, sind aber für ein funktionierendes

Softwaresystem von großer Bedeutung. Ein Beispiel für derartige Wechselwirkungen sind die Module CAN-Treiber (Controller Area Network) und BAP (Bedien- und Anzeigeprotokoll) [1]. Das Modul BAP ist ein Kommunikationsprotokoll, welches auf einem Bustreiber aufsetzt. Dies kann beispielsweise ein CAN-Treiber sein. Damit BAP korrekt arbeiten kann, müssen bestimmte Parameter in der Konfiguration des CAN-Treibers eingestellt werden. Da die Module von verschiedenen Herstellern implementiert werden, ist es schwierig die Einhaltung dieser Randbedingungen toolunterstützt zu überprüfen.

Um die Vorteile einer standardisierten Software Plattform noch besser nutzen zu können, hat sich das AUTOSAR Konsortium zum Ziel gesetzt, eine industrieweite Standardisierung zu erreichen. Durch eine solche Standardisierung soll es zukünftig einfacher sein, gleiche Standardsoftware Module von verschiedenen Software Herstellern zu beziehen und diese in ein Steuergerät zu integrieren. Dafür wird auch ein entsprechendes, standardisiertes Konfigurationskonzept entwickelt werden.
Ein grundsätzliches Problem, welches eine toolunterstützte Berücksichtigung von Abhängigkeiten zwischen verschiedenen Modulen erschwert, ist das Fehlen einer einheitlichen, modulübergreifenden Modellierungsmöglichkeit für alle Anforderungen an die Konfiguration. Durch eine allgemeine Konfigurationsbeschreibung, die unabhängig von einer konkreten Standardsoftware Architektur ist, können Abhängigkeiten zwischen den einzelnen Modulen beschrieben und geprüft werden.
Im Folgenden soll deshalb eine Modellierungsform vorgestellt werden, welche unabhängig von den verwendeten Konfigurationswerkzeugen arbeitet und für beliebige Software Architekturen eingesetzt werden kann. Grundlage dabei sind Merkmalmodelle, wie sie bei Software Produktlinien eingesetzt werden. Das Ziel von Software Produktlinien ist das möglichst effektive Erzeugen von vielen, ähnlichen Software Produkten. Ebenso wie bei Standardsoftware Konfigurationen kommt dabei dem Variantenmanagement eine große Bedeutung zu.

3. Merkmalmodelle – Grundlagen und Anwendung

Merkmalmodelle sind ein zentraler Bestandteil bei der Erstellung von Software Produktlinien [2]. Sie werden dabei zur Modellierung der Merkmale einer Domäne, sowie ihren Wechselwirkungen untereinander, eingesetzt. In einem Merkmalmodell werden die Merkmale, also die strukturellen und funktionalen Eigenschaften der untersuchten Systeme kategorisiert. Es wird dabei zwischen gemeinsamen und variablen Merkmalen unterschieden. Ein gemeinsames Merkmal ist eine Eigenschaft, die alle Produkte einer

Produktlinie aufweisen. Ein variables Merkmal ist eine Eigenschaft, worin sich Produkte einer Produktlinie voneinander unterscheiden können. Merkmale sind hierarchisch angeordnet, wobei die Wurzel des Baumes den Konzeptknoten enthält, der stellvertretend für die gesamte Systemfamilie steht.

Eine sehr weit verbreitete Merkmalmodell-Notation, die als Grundlage für viele Erweiterungen im Bereich der Software Produktlinien dient, ist die FODA (Feature-Oriented Domain Analysis, [3])-Notation. Die hierarchischen Beziehungen zwischen Merkmalen, lassen sich, wie in Bild 3 am Beispiel einer Zentralverriegelung dargestellt, folgendermaßen klassifizieren:

Notwendige Merkmale: Ein notwendiges Merkmal muss immer ausgewählt werden, sobald das übergeordnete Merkmal ausgewählt oder wenn das notwendige Merkmal dem Konzeptknoten untergeordnet ist.

Optionale Merkmale: Ein optionales Merkmal kann ausgewählt werden, sobald das übergeordnete Merkmal ausgewählt oder wenn das optionale Merkmal dem Konzeptknoten untergeordnet ist.

Alternative Merkmale: Aus einer Gruppe alternativer Merkmale muss genau ein Merkmal ausgewählt werden, sobald das übergeordnete Merkmal auswählt oder wenn die Gruppe dem Konzeptknoten untergeordnet ist.

Oder Merkmale: Aus einer Gruppe von Oder-Merkmalen muss mindestens ein Merkmal ausgewählt werden, sobald das übergeordnete Merkmal ausgewählt oder wenn die Gruppe dem Konzeptknoten untergeordnet ist. Zusätzlich können aus der Gruppe beliebig viele weitere Merkmale ausgewählt werden.

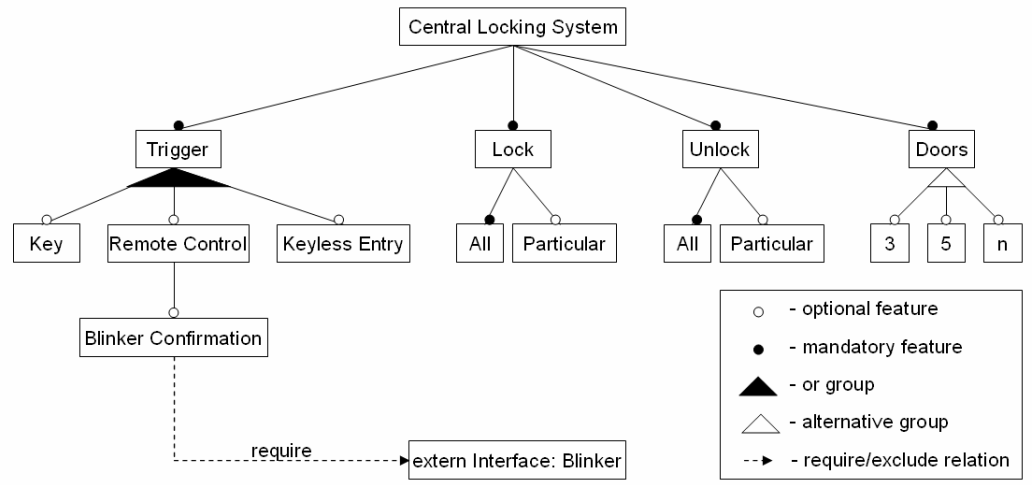

Bild 3: Beispiel Merkmalmodell / feature model example

Die Beziehungen zwischen Merkmalen beschränken sich nicht nur auf Hierarchie-, sondern auch auf Abhängigkeitsbeziehungen. Wenn ein Merkmal ein anderes Merkmal benötigt, kann dies beispielsweise über eine *require*-Beziehung modelliert werden. Falls sich im Gegensatz dazu zwei Merkmale ausschliessen, kann dies über eine *exclude*-Beziehung dargestellt werden. Der FODA-Ansatz wurde von [4] um die Möglichkeit, einem Merkmal Attribute zu übergeben, erweitert. Diese Attribute sind eine Möglichkeit um Merkmalen Werte hinzuzufügen. Für die hier vorgestellte Merkmal-Notation wird diese Methodik für die Übergabe von Parametern (z.B. Timing-Parameter) an ein Merkmal realisiert. Des Weiteren wurden die *require*- und *exclude*-Beziehungen um Bedingungen erweitert. Somit ist es möglich, die Beziehungen zwischen den Merkmalen genauer zu definieren. Die Erweiterung um Parameter und komplexe Require-Beziehungen ist für den Anwendungsfall Standardsoftware von besonderer Bedeutung und soll im Folgenden näher erläutert werden.

4. Merkmalmodelle für Standardsoftware

Die Konfigurationsmöglichkeiten von Standardsoftware Modulen können auch als Merkmale einer Standardsoftware Produktlinie aufgefasst werden. Darum liegt es nahe, diese auch mit denselben Modellen zu beschreiben. Merkmalmodelle realer Software Systeme sind normalerweise sehr komplex. Die in Bild 3 verwendete grafische Darstellungsweise der Modelle wird bei großen Systemen schnell unübersichtlich. Aktuelle Tools bevorzugen deshalb eine Darstellungsweise ähnlich einer Verzeichnisstruktur bei Dateisystemen. Das Grundkonzept von Merkmalmodellen lässt sich mit der grafischen Darstellung in Anlehnung an FODA aber besser darstellen, weshalb im Folgenden diese Darstellungsweise weiter verwendet wird.

Um die Anwendung von Merkmalmodellen im Bereich Standardsoftware zu demonstrieren, wird das Beispiel aus Kapitel 2 noch einmal aufgegriffen. Die Software, die das Bedien- und Anzeigeprotokoll (BAP) realisiert, benutzt einen Treiber um die BAP-Botschaften über ein Bussystem zu verschicken. Das Bedien- und Anzeigeprotokoll ist auf verschiedenen Bussystemen wie CAN oder MOST [5], lauffähig. Die CAN-Software ist ein konfigurierbares Modul, welches die Möglichkeit bietet, sogenannte „Transmit Queues“ zu nutzen. Falls BAP-Botschaften über den CAN-Bus verschickt werden sollen, dürfen jedoch keine Transmit Queues verwendet werden, da dadurch das zeitliche Verhalten beeinflusst werden könnte.

Es bestehen also folgende Anforderungen an eine korrekte Standardsoftware Konfiguration:

1. Wenn BAP verwendet wird, muss auch ein Bus Treiber vorhanden sein.
2. Wenn CAN verwendet wird, dürfen keine Transmit Queues verwendet werden.

Um die Einhaltung dieser Anforderungen toolunterstützt prüfen zu können, wird eine Beschreibung benötigt, wie eine korrekte Konfiguration auszusehen hat. Diese Beschreibung erfolgt mittels eines Merkmalmodells. Wie in Kapitel 3 beschrieben, wird zwischen gemeinsamen und variablen Merkmalen unterschieden. Die Module MOST-, CAN- und BAP-Software können als variable Merkmale einer Systemfamilie Standardsoftware betrachtet werden. Die Module haben ebenfalls konfigurierbare Merkmale, die von der jeweiligen Implementierung abhängig sind. Im Folgenden sollen nur die Merkmale berücksichtigt werden, die für dieses Beispiel relevant sind. Damit ergibt sich für den CAN-Treiber ein optionales Merkmal „Transmit Queues". Die oben genannten Anforderungen sind damit alleine allerdings noch nicht beschrieben. Abhängigkeiten zwischen einzelnen Merkmalen werden, wie in Kapitel 3 beschrieben, in Merkmalmodellen mit require- und exclude- Beziehungen modelliert. Mit diesen ist es möglich, Abhängigkeiten zwischen beliebigen Merkmalen, unabhängig von Ihrer Anordnung im Modell darzustellen. Bild 4 zeigt das entsprechende Merkmalmodell.

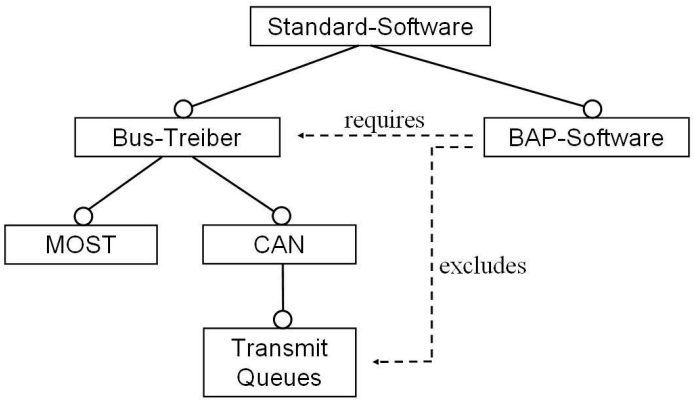

Bild 4: Standardsoftware Merkmalmodell mit Abhängigkeiten / standardsoftware feature model with dependencies

Um auch kompliziertere Abhängigkeiten zwischen Merkmalen oder zwischen Parametern von Merkmalen darstellen zu können, gibt es Ansätze, diese mittels einer Erweiterung der Sprache Prolog zu beschreiben. Damit wird es möglich, mathematische Bedingungen zwischen Parametern verschiedener Merkmale oder auch Abhängigkeiten zwischen drei

oder mehr Merkmalen zu beschreiben. Das Tool Pure::Variants [6] verwendet beispielsweise diesen Ansatz.

Neben den Abhängigkeiten zwischen Konfigurationsparametern gibt es auch Anforderungen an die Parameter selbst. Als Beispiel hierfür soll ein Standardsoftware Modul für das Netzwerkmanagement dienen. Es wird angenommen, dass das Modul die beiden Parameter „CycleTime“ und „TimeOut“ besitzt. Im Modell wird dies durch ein Merkmal „Network Management“ mit zwei Parametern beschrieben. Ausserdem werden folgende Anforderungen an eine gültige Konfiguration gestellt:

1. CycleTime darf die Werte 1,2,5,10 oder 50 annehmen.
2. Der Wert von TimeOut darf nur im Bereich zwischen 10 und 65535 liegen.
3. Zusätzlich muss der Wert von TimeOut ein Vielfaches von CycleTime sein.

Bild 5 zeigt, wie diese Anforderungen beim Tool Pure::Variants mittels eines erweiterten Prolog Dialekts beschrieben werden.

```
Id:       iKq8niHyDA9JtM0aM
Unique Name:    NET_MGMT_HSCAN
Visible Name:     Network Management HiSpeed CAN
Class:   ps:feature

Restrictions:
getAttribute('NET_MGMT_HSCAN','CycleTime',CT) and
getAttribute('NET_MGMT_HSCAN','TimeOut',TO) and
REMAIN is TO mod CT and
(
        memberchk(CT,[1,2,5,10,50])
        or
        warningMsg('CycleTime must be 1,2,5,10 or 50')
) and
(
        REMAIN =:= 0
        or
        warningMsg('TimeOut must be multiple of CycleTime')
) and
(
        (TO > 10 , TO < 65536)
        or
    warningMsg('TimeOut must be within range [10..65535]')
)
```

Bild 5: Beschreibung mit Prolog / description with prolog

Da Prolog im Bereich Embedded Software nur wenig verbreitet ist, wäre eine einfachere Beschreibungssprache wünschenswert. Aus diesem Grund gibt es auch Ansätze, die

Abhängigkeiten zwischen Merkmalen mittels OCL (Object Constraint Language) zu beschreiben [7]. OCL ist ein Teil der UML-Modellierungssprache und speziell für die Modellierung von Wechselwirkungen ausgelegt.

Da sich Merkmalmodelle rein auf Variabilitäten sowie den Anforderungen und Wechselwirkungen zwischen diesen beziehen, sind sie unabhängig von konkreten Konfigurationsformaten. Sie eignen sich deshalb besonders gut zur Beschreibung von modulübergreifenden Konfigurationsanforderungen und bei verschiedenen Formaten der Konfigurationsdateien. Für eine Konfigurationsprüfung muss allerdings noch die Anbindung an diese Konfigurationsformate geschaffen werden, da diese in realen Systemen die Konfigurationsinformationen enthalten.

5. Anwendung zur Konfigurationsprüfung von Standardsoftware

Die Informationen, die zur Konfiguration der Standardsoftware Module benötigt werden, sind je nach Modul und Softwarefirma in verschiedenen Konfigurationsformaten abgespeichert. Gängige Formate sind beispielsweise OIL (OSEK Implementation Language) [8] oder DIL (Device Implementation Language) [9]. Aber auch firmenspezifische Formate oder einfache C-Header Dateien werden zur Konfiguration verwendet. Um eine Prüfung der Standardsoftware Konfiguration zu ermöglichen, muss diese Information eingelesen und gegen das Merkmalmodell geprüft werden. Bild 6 stellt dieses Konzept dar.

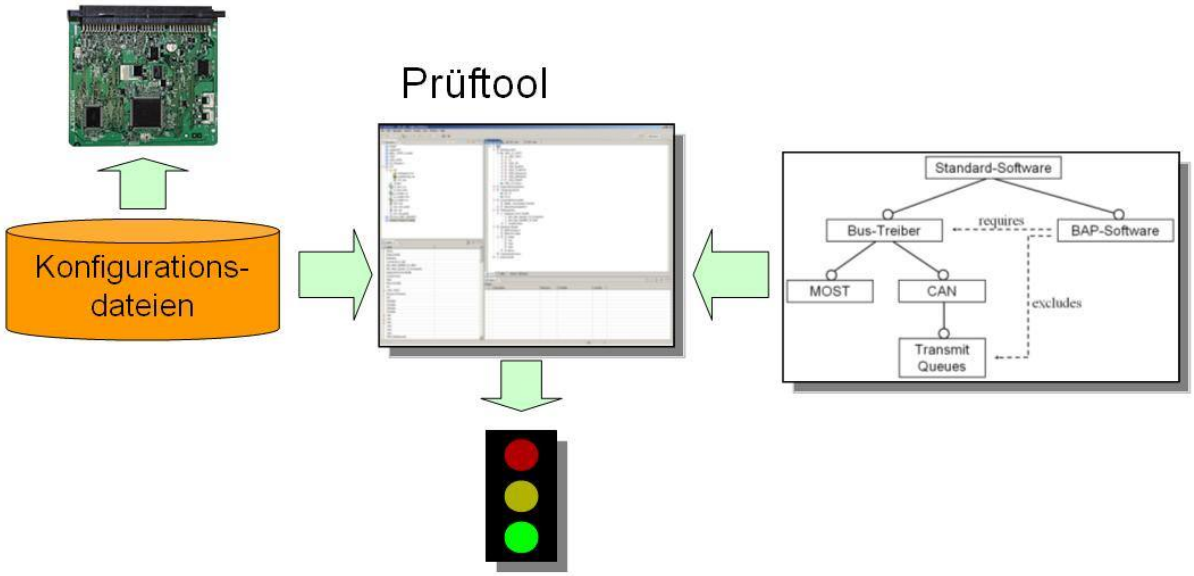

Bild 6: Prüfkonzept für Standardsoftware / verification concept for standardsoftware

Die Gesamtheit der Konfigurationsdateien der einzelnen Module stellen eine konkrete Standardsoftware Konfiguration dar. Das Merkmalmodell beschreibt im Gegensatz dazu die Gesamtheit aller gültigen Standardsoftware Konfigurationen. Bei der Prüfung wird festgestellt, ob die konkrete Konfiguration ein Teil aller gültigen Konfigurationen ist.

Um einen Vergleich der Konfigurationsdateien gegen das Merkmalmodell durchführen zu können, müssen die Merkmale der einzelnen Modulkonfigurationen von einem Werkzeug eingelesen werden. Wie in Bild 7 dargestellt, entsteht dadurch eine Liste von Merkmalen, die anschließend mit dem Modell verglichen wird.

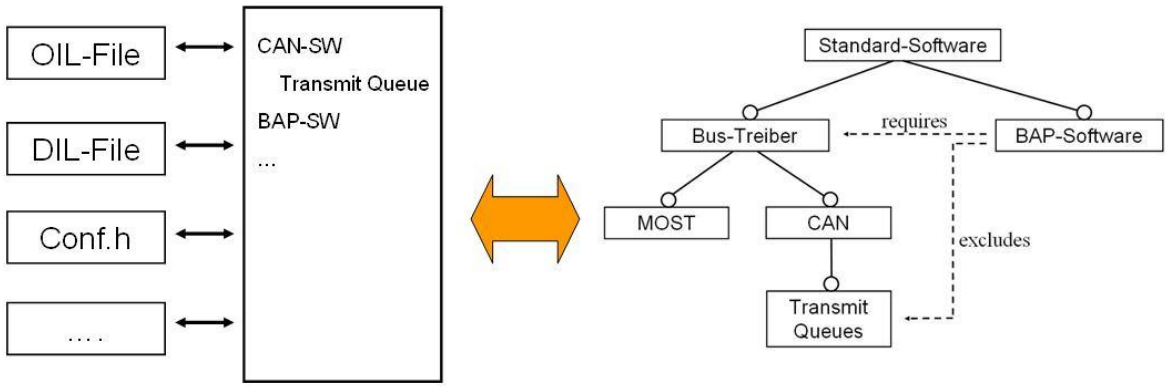

Bild 7: Einlesen von Konfigurationsdateien / import of configuration files

Durch die Trennung der konkreten Konfigurationsformate und dem allgemeinen Modell gültiger Standardsoftware Konfigurationen ist das Konzept universell für beliebige Software Architekturen und auch Mischformen, wie z.B. bei einem Migrationsszenario in AUTOSAR, geeignet. Je nach verwendetem Software Modul muss dieses lediglich als Merkmalmodell beschrieben sowie das Einlesen der Konfigurationsinformationen auf das verwendete Format angepasst werden.

Nach einer prototypischen Umsetzung des Prüfkonzepts zur Verifizierung und Optimierung des Konzeps soll eine Implementierung eines entsprechenden Prüftools folgen. Eingesetzt werden soll das Werkzeug zur Unterstützung von Standardsoftware Integrationsreviews in verschiedenen Steuergeräteprojekten. Ziel ist eine Reduzierung des Aufwands und eine Optimierung der Reviewqualität.

6. Zusammenfassung

Es wurde ein allgemeines Konzept zum Variantenmanagement und die Anwendung zur Prüfung von Standardsoftware Konfigurationen vorgestellt. Da Standardsoftware viele vernetzungsrelevante Funktionen umsetzt, kann eine zusätzliche Prüfung der Standardsoftware Konfiguration im Vorfeld einer Vernetzungsprüfung helfen, Fehler und deren Ursache möglichst frühzeitig zu finden und damit Kosten zu sparen. Das Anwendungsfeld Standardsoftware ist aber nur eines von vielen, da sich Merkmalmodelle überall dort einsetzen lassen, wo es darum geht, viele Varianten und deren Zusammenspiel zu beherrschen. Für den Automobilhersteller ist es letztendlich von großer Bedeutung, die Komplexität seiner Produktpalette auf jeder Abstraktionsebene bis hinunter zur Standardsoftware zu beherrschen, um Qualität auf höchstem Niveau sicherzustellen.

[1] Wenzel M., Wille F.M.: Bedien- und Anzeigeprotokoll trennt Funktion und Gestaltung. Elektronik Automotive, 6/2004, 30-32

[2] P. Clemens, L. Northop: Software Product Lines - Practices and Patterns. Addison-Wesley, Boston, 2002.

[3] Kyo C. Kang, Kyo C., Cohen, Sholom G., Hess, James A., Novak,William E., and Peterson, A. Spencer (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie Mellon University, Software Engineering Institute.

[4] Krzysztof Czarnecki, Thomas Bednasch, Peter Unger und Ulrich W. Eisenecker: Generative Programming for Embedded Software: An Industrial Experience Report. In: GPCE, Seiten 156–172, 2002

[5] MOST Cooperation: MOST Specification Rev. 2.3, August 2004, http://www.mostnet.de

[6] Pure::Systems GmbH, www.pure-systems.com

[7] Streitferdt, Detlef: Family-Oriented Requirements Engineering. Doktorarbeit, Technische Universität Illmenau, 2003.

[8] OIL: OSEK Implementation Language Specification Document, Version 2.5, Juli 2004, http://www.osek-vdx.org

[9] Hersteller-Initiative Software : API I/O Library, Version 2.0.3., März 2004, http://www.automotive-his.de