Spezifikationstechniken für Software-Schnittstellen in Fahrzeug-Infotainmentsystemen

Simon Gerlach
Volkswagen AG
Brieffach 1148
D-38436 Wolfsburg, Germany
simon.gerlach@volkswagen.de

Armin Widegreen
Fachhochschule Stralsund
Zur Schwedenschanze 15
D-18435 Stralsund, Germany
armin.widegreen@fh-stralsund.de

Abstract: Die Benutzeroberfläche und die ihm zugrundeliegenden Softwarefunktionen werden bei Fahrzeug-Infotaintmentsystemen oftmals von unterschiedlichen Teams realisiert. Die Schnittstelle zwischen diesen beiden Softwareteilen muss daher exakt spezifiziert und zwischen allen Projektbeteiligten abgestimmt werden können. Den vorhandenen Beschreibungstechniken fehlen jedoch Ausdrucksmittel, um damit eine vollständige Spezifikation einer solchen Schnittstelle zu formulieren. Da sie zudem oftmals für eine ausschließlich maschinelle Verarbeitung konzipiert sind, sind sie für den manuellen Spezifikations- und Abstimmprozess ungeeignet.

In diesem Aufsatz wird daher eine formale Beschreibungssprache vorgestellt, mit der sowohl der Aufbau einer Schnittstelle als auch deren Nutzung beschrieben werden kann. Darüber hinaus ermöglicht sie es, weitere Informationen zum Abstimmungs- und Freigabestatus der Schnittstellenelemente zu hinterlegen. Die Notation der Sprache wurde mit dem Ziel entworfen, dass sie auch ohne spezielle Werkzeuge und ohne Kenntnisse der Zielplattform und des darin genutzten Kommunikationsframeworks genutzt werden kann. Aus den damit erstellten Schnittstellenspezifikationen kann per Codegenerator automatisiert zielsystemspezifischer Code erzeugt werden. Nach der Vorstellung der Sprache wird erläutert, wie sie in den Entwicklungsprozess eines Serien-Infotainmentsystems eingeführt wurde und welche Auswirkungen sich dadurch auf den Spezifikations- und Abstimmungsvorgang ergeben.

1 Hintergrund und Motivation

Moderne Infotainmentsysteme im Fahrzeug sind für die Kunden ein zunehmend bedeutendes Kaufkriterium und tragen wesentlich zur Marge der Hersteller bei. Sie bieten immer mehr Funktionen und aufwändigere Benutzeroberflächen (HMI). Ihre zunehmenden Softwareumfänge zusammen mit den kurzen Modellzyklen verlangen den Einsatz effizienter Entwicklungsmethodiken.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4

1.1 Neuartige Arbeitsteilung zwischen Fahrzeugherstellern und Lieferanten

Die klassische Aufgabenteilung bei der Entwicklung dieser Fahrzeug-Infotainmentsysteme bestand darin, dass die Spezifikation der Geräte durch die Fahrzeughersteller (OEM) erfolgte. Mit der Umsetzung dieser Spezifikation wurden spezialisierte Zulieferer beauftragt [SBK, HB08]. Diese erstellen Teile der Software selbst, kaufen fertige Teile zu oder beauftragen ihrerseits weitere Zulieferer mit der Umsetzung.

Aufgrund der besonderen Bedeutung für die Markenidentität und die Gebrauchstauglichkeit des Gesamtsystems ist derzeit jedoch zu beobachten, dass die OEMs zunehmend größeren Einfluss auf die Realisierung der HMIs nehmen wollen [Ham07]. Daher wird in einigen Fällen das HMI sogar vom OEM selbst „In-House“ entwickelt [HB08, OK07, Ver04]. In der Verantwortung des Zulieferers liegen dabei jedoch weiterhin die Laufzeitumgebung für die HMI-Software und ebenso alle weiteren Softwareteile, die vom Kunden nicht unmittelbar wahrgenommen werden können.

Unabhängig von dieser veränderten Aufgabenteilung ist es üblich, dass von den OEMs zur Kostenoptimierung mehrere Zulieferer parallel mit der Entwicklung des Infotainmentsystems beauftragt werden [HB08]. Die so entstehenden Geräte müssen jedoch für den Kunden identisch wirken. Um dies zu erreichen und dabei gleichzeitig Entwicklungskosten durch die Vermeidung von Mehrfachentwicklungen zu reduzieren wird versucht, die gleiche HMI-Software auf den Plattformen verschiedener Zulieferer einzusetzen.

1.2 Definition von Geräteschnittstellen erforderlich

Sowohl eine In-House Entwicklung des HMIs als auch der Einsatz desselben HMIs für verschiedene Zulieferer-Plattformen führen dazu, dass das HMI und die übrige Software von unterschiedlichen Stellen realisiert werden. Daher ist es wichtig, dass zwischen beiden Softwareteilen eine eindeutige Schnittstelle vereinbart wird [PBKS07, Bri04], über die Daten ausgetauscht und dem HMI Zugriff auf die Gerätefunktionen ermöglicht wird.

Damit das HMI auf den unterschiedlichen Plattformen der Zulieferer lauffähig ist, müssen diese eine spezifikationsgemäße Implementierung der Schnittstelle bereitstellen.

1.3 Standardisierungsinitiativen

Im Rahmen der AUTOSAR-Partnerschaft wird auf eine ähnliche Situation im Bereich der Steuergeräte im Fahrzeug reagiert. Auch hier existierten zuvor eine Vielzahl inkompatibler Hard- und Software-Plattformen. Gleiche Softwarefunktionen mussten daher für den Einsatz auf unterschiedlichen Plattformen mehrfach entwickelt werden. Die OEMs waren zudem bisher nicht in der Lage, Softwarefunktionen eigenständig zu entwickeln, ohne dass diese an die Hardware einzelner Zulieferer gebunden wäre. AUTOSAR standardisiert daher eine Softwarearchitektur, die auf unterschiedlicher Hardware realisierbar ist und da-

ISBN 978-3-88579-286-4

durch die darauf basierende Software portierbar macht [Fue10].

Der Infotainmentbereich wurde von AUTOSAR explizit ausgeschlossen, da hierbei andere Anforderungen als bei den übrigen Steuergeräten im Fahrzeug bestehen. Seit langem wird jedoch auch hierfür eine Standardisierung der Schnittstellen für die gängigsten Gerätefunktionen prognostiziert [Bri04]. Derzeitig wird von der GENIVI-Allianz versucht, eine standardisierte Infotainmentplattform zu definieren, auf deren Basis produktindividuelle HMIs entwickelt werden können. Dafür werden einheitliche Software-Schnittstellen festgelegt, über die das HMI die Gerätefunktionen ansprechen kann [Fue10].

Solange man sich auf die darin angebotenen Standardfunktionen beschränkt, entfällt somit eine individuelle Vereinbarung der Schnittstelle. Grundsätzlich besitzen jedoch alle Standardisierungsversuche den Nachteil, dass sie naturgemäß keine spezifischen Sonderfunktionen enthalten. Diese werden jedoch benötigt, um über neuartige Funktionalität Alleinstellungsmerkmale zu realisieren. So erscheint die Standardisierung eines Radio-Moduls zwar möglich, in innovativen Bereichen wie den Fahrerassistenzsystemen oder der Integration mobiler Geräte und Web-Dienste bestehen jedoch zu große Unterschiede zwischen den Anbietern. Zudem führt die hohe Dynamik in diesem Gebiet dazu, dass nicht schnell genug Standardisierungen durchgeführt werden können. Schon allein aus Wettbewerbsgründen würden entsprechende Gremien von den OEMs und Zulieferern nicht rechtzeitig eingebunden werden.

Man kann somit zwar die Inhalte der Schnittstellen nicht standardisieren, wohl aber die darin verwendeten Kommunikationsmechanismen.

2 Stand der Praxis

Bei Volkswagen hat konzernintern eine solche Standardisierung bereits stattgefunden. Ein entsprechendes Kommunikationsframework wird produkt- und markenübergreifend seit mehreren Gerätegenerationen eingesetzt.

Die Schnittstelle zwischen dem HMI und den zugrundeliegenden Softwareteilen besteht darin aus einem Satz an Funktionen. Diese können von der jeweiligen Gegenstelle asynchron aufgerufen werden, um so Daten auszutauschen oder Aktionen auszulösen. Das Framework verbindet beide Teile per Laufzeitbindung. Um sie auf unterschiedlichen Prozessen oder gar Systemen ausführen zu können verwendet es Marshalling. Außerdem realisiert es einen Sprachübergang zwischen dem Java-basierten HMI und der nativen Implementierung der Gerätefunktionen.

Die Struktur der Schnittstelle wird festgelegt, indem auf Basis des Kommunikationsframeworks Paare von Java-Interfaces geschrieben werden, in denen jeweils die von der Gegenstelle angebotenen Funktionen eingeführt werden. Darüber hinaus muss die Verwendung dieser Funktionen spezifiziert werden. Dazu wird festgelegt, welche Abfolgen von Funktionsaufrufe zulässig sind und welche Wertebereiche ihre Parameter annehmen dürfen. Diese Informationen werden im Quellcode mittels spezieller Annotationen hinterlegt. Aus allen so angegebenen Informationen wird schließlich eine Dokumentation der Schnittstelle erzeugt. Sie dient als Grundlage für die Ausschreibung beider Softwareteile und als

Hilfsmittel für deren Entwickler.

Da die Schnittstelle für alle produkt- und markenspezifischen HMIs und alle beauftragten Zulieferer vereinheitlicht werden soll, muss sie in einem Gremium zwischen den Projektbeteiligten abgestimmt werden. Dazu wurden bisher von einem Team alle Änderungswünsche als angepasster Quellcode entgegengenommen und darüber beraten. Anschließend wurden die angenommenen Änderungen zusammengeführt und eine neue Version der Schnittstelle freigegeben.

3 Abstrakte Schnittstellenspezifikation

Durch eine Analyse des bestehenden Vorgehens konnten eine Reihe von Nachteilen identifiziert werden. So führt es zu einer starken Bindung an die eingesetzten Programmiersprachen und an das derzeit verwendete Kommunikationsframework. Die Einhaltung der Syntax der Programmiersprache führt dazu, dass die Schnittstellenspezifikation viel Overhead durch immer wieder ähnlichen Boilerplate-Code enthält. Zudem müssen dadurch einige Informationen redundant angegeben werden. Dies reduziert die Übersichtlichkeit und führt zu hohem Aufwand bei der Erstellung- und Zusammenführung. Die am Spezifikationsprozess beteiligten Entwickler müssen zudem vertiefte Kenntnisse des Frameworks besitzen, obwohl sie inhaltliche Abstimmungsarbeit leisten sollen. Da für Nicht-Experten die Struktur und Nutzung der Schnittstelle nicht direkt aus ihrem Quellcode erkennbar ist, ist außerdem die Überführung in ein Dokumentationsformat erforderlich.

Die beschriebenen Nachteile können vermieden werden, wenn die Schnittstelle stattdessen abstrakter und technologieunabhängig beschrieben wird.

Damit die dafür eingesetzte Technik im Spezifikations- und Abstimmprozess verwendet werden kann muss sie sämtliche Mechanismen und Konstrukte des derzeit verwendeten Kommunikationsframeworks bereitstellen. Gleichzeitig sollte sie jedoch weitestgehend technologieunabhängig sein, so dass die damit erstellten Spezifikationen als Grundlage für Nachfolgeprojekte wiederverwendet werden können, in denen möglicherweise andere Frameworks zum Einsatz kommen.

Sie muss möglichst einfach erlernbar sein, damit sie von allen Beteiligten mit geringem Einarbeitungsaufwand genutzt werden kann. Damit die Generierung einer zusätzlichen Dokumentation entfallen kann, muss sie zudem gut lesbar sein, gegenüber der bestehenden Vorgehensweise die Übersicht und das inhaltliche Verständnis der Spezifikation verbessern und Sachverhalte auf einem ausreichend hohen Abstraktionsniveau darstellen.

Sollten für ihre Verwendung spezielle Werkzeuge erforderlich sein, so muss der Aufwand für deren Bereitstellung möglichst gering sein.

Eine weitere wichtige Anforderung ist es, die Schnittstellenbechreibung mit möglichst geringem Aufwand maschinell weiterverarbeiten zu können. Dies ermöglicht es, vollautomatische Analysen durchzuführen um so beispielsweise Kenngrößen für die Projektleitung wie den Umsetzungsgrad oder den Stand der Abstimmung zu ermitteln. Außerdem muss der Code zur Realisierung der Schnittstelle mit dem gewählten Kommunikationsframe-

work automatisiert erzeugt werden können.

3.1 Untersuchung bestehender Beschreibungssprachen

Um die optimale Beschreibungstechnik zu bestimmen, wurden zunächst bestehende Vorgehensweisen zur Schnittstellenfestlegung aus anderen Bereichen hinsichtlich ihrer Tauglichkeit für diesen Einsatzzweck untersucht. Dadurch sollte von der dafür vorhandenen Werkzeugunterstützung und der Erfahrung von Entwicklern mit dieser Technik profitiert werden.

Zunächst wurde WSDL [W3C07] untersucht, eine XML-basierte Sprache zur Beschreibung von Web-Services. Für die Festlegung von Ablaufsequenzen kann diese mit WS-BPEL kombiniert eingesetzt werden. Da beide Sprachen jedoch ausschließlich für eine maschinelle Verarbeitung konzipiert wurden, sind sie für Menschen nur schwer lesbar. Dadurch würde der manuelle Spezifikations- und Abstimmprozess erschwert. Es wäre dafür eine spezielle Werkzeugunterstützung erforderlich, die bisher nicht verfügbar ist. Viele WSDL-Konstrukte dienen zur Angabe von Informationen, die ausschließlich für Web-Services relevant sind und für die direkte Kommunikation zwischen den beiden Softwareteilen im Infotainmentsystem nicht benötigt werden. Darüber hinaus fehlen Ausdrucksmittel, um die Schnittstelle so vollständig zu beschreiben, dass sie als Grundlage für eine Ausschreibung verwendet werden kann. Insbesondere kann keine Beschreibung der Semantik einer Funktion angegeben werden. WS-BPEL kann zwar einfache sequentielle Abläufe beschreiben, komplexe Kommunikationssequenzen und Aufrufmodalitäten der Funktionen können jedoch nicht spezifiziert werden.

OWL-S [W3C04, W3C09] dient zur Beschreibung von Web-Services auf Grundlage der OWL, einer Sprache zur Formulierung von Ontologien. Sie wird im BMW-FLUID Framework eingesetzt, um einen zum Entwicklungszeitpunkt unbekannten Dienst semantisch zu beschreiben, so dass dafür ein für das jeweilige Fahrzeug geeignetes HMI automatisch erzeugt werden kann [HSS07]. Zur detaillierten Beschreibung der Struktur einer Schnittstelle und deren Nutzung bietet die OWL-S jedoch keine geeigneten Ausdrucksmittel. Ähnlich wie WSDL ist die Sprache für eine ausschließlich maschinelle Verarbeitung konzipiert worden. Dementsprechend ist eine manuelle Erstellung eines OWL-S-Dokuments aufwändig und würde eine große Einarbeitungszeit für die Spezifikateure erfordern. Derzeit sind zudem erst wenige unterstützende Entwicklungsumgebungen und Transformationswerkzeuge vorhanden.

CORBA setzt für die Festlegung von Schnittstellen von Softwaremodulen die IDL ein. IDL-Dokumente werden manuell erstellt und dann automatisch in eine Programmiersprache transformiert. Im Gegensatz zu OWL-S und WSDL ist die IDL aufgrund ihrer besseren Lesbarkeit auch ohne zusätzliche Werkzeugunterstützung für einen manuellen Spezifikations- und Abstimmprozesse geeignet. Die Konstrukte der IDL lassen sich in verschiedene Zielsprachen abbilden. Sie sind Entwicklern daher aus der Programmierung bekannt und sind für sie somit schnell verständlich. Die IDL ermöglicht zwar die Festlegung der Signatur einer Funktion, sie bietet jedoch darüber hinaus kein Sprachmittel, um deren Aufgabe und

ISBN 978-3-88579-286-4

Verwendung vertragssicher zu beschreiben. Da bei der Kommunikation zwischen HMI und Applikationslogik möglicherweise Prozessgrenzen überwunden werden, muss hierbei von asynchronen Aufrufen ausgegangen werden. In CORBA finden Funktionsaufrufe hingegen stets synchron statt. Der IDL fehlen daher Möglichkeiten zur Festlegung von Kommunikationsabläufen und auch das Verhalten bis zur Antwort der Gegenstelle kann nicht beschrieben werden. Die verfügbaren Codegeneratoren, um IDL-Dokumente in verschiedene Zielsprachen zu wandeln, erzeugen stets CORBA-spezifischen Code.

Keine der untersuchten Sprachen ist somit geeignet, um eine Schnittstelle zwischen dem HMI und den Gerätefunktionen in geeigneter Form zu beschreiben. Stattdessen müssten sie dafür massiv angepasst werden, wodurch sie wiederum vom verbreiteten Standard abwichen und so nicht von der dafür verfügbaren Werkzeugunterstützung profitiert werden kann.

Dies träfe ebenso zu, wenn eine andere Sprache erweitert würde, deren primäre Aufgabe nicht die Schnittstellenbeschreibung ist (z.B. HTML). Zudem würde ein solches Vorgehen zu einer unnötig hohen Komplexität führen, da hierbei viele Sprachkonstrukte zur Verfügung stünden, die für den Einsatzzweck nicht benötigt würden. Aus diesem Grund wurde auch diese Möglichkeit verworfen.

3.2 Definition einer Spezifikationssprache für Software-Schnittstellen

Anstatt auf eine vorhandene Sprache zurückgreifen zu können, musste daher eine eigene Beschreibungstechnik entwickelt werden, die exakt die benötigten Ausdrucksmittel zur Verfügung stellt, um die Schnittstelle effizient beschreiben zu können. Hierfür wurden verschiedene Möglichkeiten untersucht.

Mittels eines Profils kann die UML um weitere Modellierungskonstrukte erweitert werden, mit denen sich die erforderlichen Informationen ausdrücken ließen. Die bereits in der UML vorhandenen Konstrukte können jedoch auf diese Weise nicht ausgeblendet werden. Daher wäre es möglich, dass Spezifikateure inhaltlich gleiche Sachverhalte auf verschiedene Weise ausdrücken, was eine automatisierte Verarbeitung erschweren würde. Die grafische Notation der UML wäre zudem zur Darstellung der Struktur sehr umfangreicher Schnittstellen unübersichtlich.

Mit Hilfe eines XML-Schemas kann ein XML-Dialekt festgelegt werden. Es stehen viele Werkzeuge und Bibliotheken für die Arbeit mit XML-Dokumenten zur Verfügung. Durch Schemata ist jedoch die Notation (konkrete Syntax) von XML nicht veränderbar. Für eine händische Bearbeitung ohne spezielle Werkzeuge ist diese jedoch nicht optimal geeignet. So müssen beispielsweise viele gebräuchliche Zeichen durch Escape Sequenzen ersetzt oder Texte in CDATA-Abschnitten eingebettet werden. Dies kann zu unleserlichen, verschachtelten Dokumenten führen, die den Abstimmprozess erschweren würden.

Zur Beschreibung der Schnittstellen wurde daher eine eigene Sprache (DSISL) mit Hilfe von Xtext erstellt. DSISL-Dokumente sind Klartextdateien, so dass im Abstimmprozess für Funktionen wie Merging und Diff Standardwerkzeuge eingesetzt werden können. Die Notation der DSISL wurde an die IDL angelehnt, weil diese von den Spezifikateu-

ren als leserlich und übersichtlich empfunden wurde. Alle Sprachelemente repräsentieren logische Konstrukte, die den Beteiligten aus ihrer täglichen Arbeit bekannt sind. Auf die Angabe technologiespezifischer Internas des Kommunikationsframeworks kann dagegen verzichtet werden, damit die Spezifikationen kompakt und verständlich bleiben und sich die Entwickler auf die inhaltliche Arbeit konzentrieren können.

Die DSISL führt Strukturierungsmöglichkeiten ein (Abbildung 1), um die Spezifikation in funktionale Module aufzuteilen (`Definition`, z.B. Radio, Media) und um darin zusammengehörige Funktionen zu gruppieren (`Service`, z.B. AM-FM-Tuner, DAB-Tuner, ...). Anhand dieser Struktur können variantenspezifische Schnittstellenumfänge festgelegt werden. Zudem kann diese Strukturierung auch zur Laufzeit verwendet werden, um darüber beispielsweise Startup-Sequenzen festzulegen.

```
definition SpeechIO;

service SpeechRec {
  description = [Speech recognition engine];
  authors = "Tom";
  history {
    31.12.2010 "DSISL-Generator" [File was automatically
      generated from existing specification];
  }
  ...
}

service SpeechSynthesizer {...
```

Abbildung 1: Beispiel für Strukturierungsmöglichkeiten in der DSISL

Die Kommunikation über die Schnittstelle erfolgt über Aufrufe der in den Services enthaltenen Funktionen. Diese werden in der DSISL zusammen mit ihren Parametern spezifiziert (Abbildung 2). Die Parameter dienen dem Datenaustausch zwischen den Softwareteilen und können sowohl einfache Datentypen als auch Bitfelder, Enumerationen oder komplexe Strukturen sein, die ebenfalls mittels der DSISL eingeführt worden sind.

Zur Verfolgung des Abstimmprozesses können die Produktvarianten angegeben werden, für die ein Schnittstellenteil bereits freigegeben wurde und durch welchen Verantwortlichen dies geschehen ist. Außerdem können Informationen zur Versionshistorie des Schnittstellenteils hinterlegt werden und es können sich Entwickler eintragen, die nach einer Änderung zu benachrichtigen sind.

Jeder Funktion wird einer der vordefinierten Kommunikationsmechanismen zugewiesen (im Beispiel `update_function notify_always`). Um die korrekte Verwendung der Funktionen zu beschreiben, können Angaben zur Aufrufmodalität und Eintrittsinvarianz gemacht werden. Darüber hinaus können in der DSISL Kommunikationssequenzen definiert werden, in denen die gültigen Aufeinanderfolgen mehrerer Funktionsaufrufe festgelegt sind. Alternativ kann dafür auch auf externe UML-Sequenzdiagramme verwiesen

```
update_function notify_always updateLanguage {
  description = [The language of the speech recognition
    system.];
  parameters {
    string language [The current language, e.g., "de_DE"];
    int skinId [id of the active skin (Enum SKIN_ID
      @link{speechrec.SpeechRec.enumerations.SKINID})];
  }
  errorcases {
    "" [none];
  }
  requirement = "@link doors://21341";
  pre-cond = [no preconditions];
  post-cond = [no postconditions];
  timing = [no timings specified];
  history {
    05.01.2009 "Peter" [Created initial version];
    30.11.2009 "Andre" [Added parameter skinId];
  }
  system {
    system_type = common;
    system_state = approved;
  }
  tandem {
    approved = BRAND1-VARIANT_X "Tom" 05.10.2009;
    approved = BRAND1-VARIANT_Y "Andre" 05.10.2009;
    approved = BRAND2-VARIANT_Z "Tom" 14.10.2009;
    approved = BRAND3-VARIANT_Z "Tom" 14.10.2009;
  }
}
```

Abbildung 2: Beschreibung einer Schnittstellenfunktion

ISBN 978-3-88579-286-4

werden.

An allen Strukturierungselementen, Funktionen und Datentypen der DSISL können über Kommentare weitergehende Informationen angegeben werden, wenn sich diese nicht mit den anderen Sprachmitteln ausdrücken lassen. Zur einfacheren Navigation für den Leser sind darin Hyperlinks zu anderen Schnittstellenteilen verwendbar. Es können auch Links auf Anforderungen in RM-Werkzeugen hinterlegt werden. Dies ermöglicht es, die zu einem Schnittstellenteil zugehörigen Anforderungen einfach aufzufinden, wenn sie für dessen Verständnis relevant sind oder sich aufeinander beziehen.

4 Einführung der DSISL in ein Serien-Projekt

Um die DSISL in den laufenden Entwicklungsprozess eines Radio-Navigationssystems der nächsten Generation einzuführen war es notwendig, die bereits bestehenden Schnittstellen automatisiert in die DSISL überführen zu können. Dafür wurde der vorhandene Quellcode geparst und daraus automatisiert DSISL-Dokumente erzeugt. Außerdem wurde ein Codegenerator entwickelt, um wiederum aus diesen DSISL-Dokumenten den Quellcode für das eingesetzte Kommunikationsframework zu erzeugen.

Damit dabei die volle Abwärtskompatibilität sichergestellt werden konnte, mussten in der DSISL einige zusätzliche optionale Sprachkonstrukte eingeführt werden, die im ursprünglichen Sprachentwurf nicht vorgesehen waren. Mit ihnen können Werte explizit vorgegeben werden, die ansonsten während der Codegenerierung automatisch bestimmt werden.

4.1 Auswirkungen auf den Spezifikations- und Abstimmungsprozess

Zukünftig erfolgen die Spezifikations- und Abstimmprozesse nun auf Basis von DSISL-Dokumenten, wodurch sich jedoch keine Änderungen in ihrem Ablauf ergeben. Die zur Abstimmung eingereichten DSISL-Dokumente werden verglichen und über die Änderungswünsche wird beratschlagt. Anschließend werden sie zusammengeführt und darin die Abstimmungsergebnisse hinterlegt. Zuletzt wird die Spezifikation freigegeben und den Entwickler-Teams des HMIs und der zugrundeliegenden Gerätefunktionen übergeben.

4.2 Werkzeugunterstützung

Da DSISL ein Klartextformat ist, können alle erforderlichen Arbeitsschritte mit Standard-Werkzeugen durchgeführt werden. So kommen im Abstimmprozess übliche Diff- und Merge Tools zum Einsatz. Die Anwender können weiterhin ihre bevorzugten Texteditoren nutzen, um die Schnittstellenspezifikationen zu erstellen oder zu bearbeiten. Um jedoch den Komfort beim Arbeiten mit DSISL-Dokumenten zu erhöhen, wurden Eclipse-

basierte Editoren bereitgestellt, in denen Funktionen wie Syntax-Hervorhebung und Code-Vervollständigung verfügbar sind.

4.3 Automatisierte Weiterverarbeitung

Nach dem Erhalt einer neuen abgestimmten Version der Schnittstellenspezifikation erzeugen die Entwickler daraus den Quellcode für das verwendete Kommunikationsframework mit Hilfe des Codegenerators. Die technologiespezifischen Informationen werden von ihm automatisch ergänzt.

Die maschinelle Verarbeitung ermöglicht es aber auch, automatisierte Prüfungen durchzuführen. Auf diese Weise wird u.a. bereits während des Abstimmprozesses geprüft, ob alle Teile der Schnittstelle geeignete und eindeutige Namen besitzen oder ob die Querverweise in den Kommentaren gültig sind. Außerdem wurden auch automatische Auswertungen der Schnittstelle realisiert, um damit beispielsweise Trendanalysen für das Management oder Komplexitäts- und Aufwandsschätzungen zu erstellen.

5 Bewertung

Nach der Synthetisierung der DSISL-Dokumente aus dem bestehenden, manuell-geschriebenen Schnittstellencode wurde dieser Code anschließend mithilfe des Codegeerators automatisiert erneut erzeugt. Danach wurde ein Vergleich zwischen dem ursprünglichen und dem generierten Code durchgeführt, bei dem keine Unterschiede festgestellt wurden. Auf diese Weise konnte nachgewiesen werden, dass die Ausdrucksmächtigkeit der DSISL ausreicht, um alle Schnittstellen dieser Gerätegeneration zu beschreiben.

Bei der Vorstellung der DSISL gegenüber ihren zukünftigen Anwendern wurde diese gegenüber der Spezifikation in Quellcode als übersichtlicher empfunden, da sie auf die wesentlichen inhaltliche Aspekte reduziert ist und zudem Redundanzen und Overhead durch Boilerplate-Code vermeidet. Durch diese Gestaltung der Sprache und durch die zusätzlich zur Verfügung stehende Werkzeugunterstützung mit Eingabeunterstützung und automatisierten Überprüfungsmöglichkeiten kann davon ausgegangen werden, dass sie die Effizienz des Spezifikations- und Abstimmprozess verbessert. Da zu diesem Zeitpunkt im Entwicklungsprozess die Schnittstellenspezifikation jedoch bereits abgeschlossen war, konnte bisher zu den Veränderungen der Spezifikations- und Abstimmzeiten durch die Einführung der DSISL noch keine Vergleichsmessung durchgeführt werden.

6 Zusammenfassung und Ausblick

In diesem Aufsatz wurde die Beschreibungssprache DSISL vorgestellt, die eine technologieunabhängige kompakte Beschreibung der Schnittstelle zwischen HMI und den darun-

terliegenden Softwareschichten ermöglicht.

Die formale Darstellung der Struktur und Nutzung der Schnittstelle ermöglicht die automatisierte Umwandlung in vielfältige Formate. So konnte daraus sowohl der Code für das verwendete Schnittstellen-Framework erzeugt werden als auch eine Dokumentation als Grundlage für Fremdvergabe und Metriken für statistische Auswertungszwecke.

Da lediglich der Codegenerator angepasst werden braucht, kann nun ein Wechsel der Schnittstellentechnologie oder Realisierungssprache mit geringem Aufwand vollzogen werden. Dies kann z.B. durch die Zusammenarbeit mit anderen Zulieferern erforderlich sein. Da zu erwarten ist, dass die Schnittstelle in Nachfolgeprojekten evolutionär weiterentwickelt wird, ergibt sich durch die technologieunabhängige Beschreibung ein weiterer Vorteil durch die verbesserte Wiederverwendungsmöglichkeit.

Im weiteren Projektverlauf sind weitere Codegeneratoren geplant, die nun mit geringen Aufwand erstellt werden können. Da es sich um eine Schnittstelle zwischen Softwareteilen zweier unterschiedlicher Firmen handelt, muss das spezifikationsgemäße Verhalten der dahinterliegenden Softwareteile besonders intensiv getestet werden. Es könnten daher zukünftig Stubs generiert werden, um mit diesen Testcases zu erstellen oder aus den in der DSISL beschriebenen Kommunikationssequenzen automatisch Testcases synthetisiert werden. Außerdem könnten Logger-Komponenten erzeugt werden können, die Traces analysieren oder in Echtzeit im Gerät an der Schnittstelle mithören und Verletzungen der spezifizierten Kommunikationssequenzen erkennen. In früheren Projektphasen können weitere Codegeneratoren zudem nützlich sein, um für neue Funktionen automatisiert Mock-Ups zu erzeugen, mit denen die Entwickler ihren Softwareteil testen können, solange ihnen hierfür noch keine Implementierung der Gegenseite vorliegt.

Zukünftig sollen weitere Arbeitsschritte in den Arbeitsprozessen durch spezialisierte Werkzeuge automatisiert werden, beispielsweise durch einen Mechanismus zur Notifikation der betroffenen Entwickler nach erfolgter Abstimmung. Außerdem ist geplant, durch eine Web-basierte Lösung den Zulieferer frühzeitiger in den Abstimmprozess einbinden zu können.

Literatur

[Bri04] Björn Briel. Softwaretechnologien und Entwicklungsprozess für das HMI: VDE-Kongress 18.-20.10.2004 Berlin, 2004.

[Fue10] Simon Fuerst. Innovationstreiber Autosar und GENIVI. *Automobil-Elektronik*, (Februar):40–41, 2010.

[Ham07] Werner Hamberger. Einsatz einer automatisierten Toolkette in der Bedienkonzeptentwicklung: Auswirkungen auf die Rollenverteilung zwischen OEM und Zulieferindustrie. In Internationaler Kongress Elektronik im Kraftfahrzeug und Gesellschaft Fahrzeug- und Verkehrstechnik, Hrsg., *Elektronik im Kraftfahrzeug*, Seiten 357–366, Düsseldorf, 2007. VDI-Verlag.

[HB08] Mathias Hüttenrauch und Markus Baum. *Effiziente Vielfalt: Die dritte Revolution in der Automobilindustrie*. Springer, Berlin, 2008.

ISBN 978-3-88579-286-4

[HSS07] Andreas Hildisch, Jürgen Steurer und Reinhard Stolle. HMI generation for plug-in services from semantic descriptors. In IEEE, Hrsg., *Fourth International Workshop on Software Engineering for Automotive Systems*, Piscataway, NJ, 2007. IEEE.

[OK07] Stefan E. Ortmann und Torsten Klein. Funktionsorientierte Entwicklung an der Schnittstelle OEM/Lieferant, 08.05.2007.

[PBKS07] Alexander Pretschner, Manfred Broy, Ingolf Kruger und Thomas Stauner. Software Engineering for Automotive Systems: A Roadmap. In Lionel C. Briand und Alexander L. Wolf, Hrsg., *Future of software engineering, 2007*, Seiten 55–71, Los Alamitos, Calif., 2007. IEEE Computer Society.

[SBK] Reinhard Stolle, Thomas Benedek und Christian Knuechel. Model-based Test Automation for Automotive Human Machine Interfaces.

[Ver04] Verband der Automobilindustrie. *Future Automotive Industry Structure (FAST) 2015: Die neue Arbeitsteilung in der Automobilindustrie*, Jgg. 32 of *Materialien zur Automobilindustrie*. Henrich Druck + Medien GmbH, Frankfurt, 2004.

[W3C04] W3C. OWL Web Ontology Language, 10.02.2004.

[W3C07] W3C. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, 26.06.2007.

[W3C09] W3C. OWL 2 Web Ontology Language, 27.10.2009.

ISBN 978-3-88579-286-4