Ergänzung vorhandener GUI-Modellierungswerkzeuge zur vollständig modellbasierten Entwicklung von Automotive-HMIs

Simon Gerlach

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

Abstract: Zur Realisierung moderner grafischer Benutzeroberflächen (GUIs) sind verschiedene Technologien und zugehörige Modellierungswerkzeuge am Markt verfügbar. Für eine vollständig modellgetriebene Entwicklung von Benutzerschnittstellen (HMI) im Automobil fehlen diesen jedoch Ausdrucksmittel zur Beschreibung des Dialogfluss, der internen HMI-Abläufe und ihrer Schnittstellen zu separat entwickelten Softwareteilen. Die Sprache HMISL ermöglicht die Beschreibung dieser Sachverhalte und kann daher als Ergänzung zu marktverfügbaren GUI-Modellierungswerkzeugen eingesetzt werden. Im Automobilbereich finden sich HMIs von Low-Cost Kombiinstrumenten bis hin zu Premium-Infotainmentsystemen. Die Leistungsfähigkeit der verfügbaren Hardware und die zum Einsatz kommenden GUI-Technologen sind sehr unterschiedlich. HMISL ist daher unabhängig von einer spezifischen GUI-Technologie. Zudem ist die Sprache erweiterbar gestaltet und ihre Ausdrucksmächtigkeit ist projektindividuell anpassbar. Zur Beschreibung des Verhaltens stehen hierarchischen Zustandsmaschinen und Datenbindungen zur Verfügung. Beide nutzen eine integrierte Standard-Programmiersprache, um damit auch komplexe Spezialfälle realisieren zu können. Zudem bietet die HMISL die Möglichkeit, das HMI in wiederverwendbare Module zu unterteilen und daraus unterschiedliche Softwarevarianten für verschiedene Ausstattungs-, Marken- oder Länderversionen abzuleiten.

1 Hintergrund

Am Markt ist eine Vielzahl von Technologien zur Realisierung grafischer Benutzeroberflächen (GUIs) verfügbar. Diese sind zudem einem permanenten schnellen Wandel unterworfen. Nicht alle davon sind jedoch geeignet, um damit ansprechende Benutzerschnittstellen (HMI) für einen Einsatz im Automobil zu entwickeln. Für neuartige HMI-Konzepte wie 3D-GUIs oder Augmented Reality, die in Zukunft an Bedeutung gewinnen könnten, kommen gar nur einzelne Technologien in Frage.

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

Um neuen Innovationen gegenüber offen zu sein und strategisch ungünstige Abhängigkeit zu vermeiden, können sich Automobilhersteller (OEMs) daher nicht langfristig auf eine spezifische Technologie zur Entwicklung von HMIs festlegen. Da jedoch die Konzepte der GUI-Technologien stark unterschiedlich sind, kann ihr voller Funktionsumfang jeweils nur mit speziell dafür entwickelten Modellierungswerkzeugen ausgeschöpft werden. Aus diesem Grund können die OEMs auch nicht am Einsatz eines bestimmten Modellierungwerkzeugs festhalten. Stattdessen ist es unvermeidlich, stets die nativen Werkzeuge der gewählten GUI-Technologie einzusetzen. Damit es möglich wird, größtmögliche Teile des HMIs bei einem Wechsel der Technologie bzw. des Modellierungswerkzeugs wiederverwenden zu können, müssen diese von den OEMs in einer technologieunabhängigen Weise formuliert werden können.

In der Automobilindustrie ist es üblich, System gleichzeitig von mehreren Zulieferern entwickeln und fertigen zu lassen. Obwohl sie dafür unterschiedliche Plattformen einsetzen, müssen die Systeme vor Kunde identisch wirken, was insbesondere vom HMI abhängig ist. Daher ist es sinnvoll, auf den Plattformen aller beteiligten Zulieferer dieselbe HMI-Software einzusetzen und so gleichzeitig Mehrfachentwicklungen zu vermeiden. Aufgrund ihrer besonderen Bedeutung für die Kundenwahrnehmung und Markenidentität muss sie zudem in enger Abstimmung mit dem OEM oder gar von ihm selbst entwickelt werden können. Dafür muss dies in plattformunabhängiger Weise geschehen. Die Integration der HMI-Software einschließlich einer gegebenenfalls notwendigen Abbildung auf eine spezifische Plattform ist wiederum Aufgabe des Zulieferers. Dies schließt die Bereitstellung aller benötigten Gerätefunktionen ein, auf die vom HMI zugegriffen wird.

2 Zielsetzung

Es wird somit eine Technologie benötigt, mit der OEMs die Software eines Automotive HMIs unabhängig von der einsetzten GUI-Technologie und Plattform entwickeln können. Für diesen Zweck wurde die HMI Specification Language (HMISL) entwickelt. HMISL ist eine domänenspezifische Sprache, die zusammen mit verschiedenen GUI-Modellierungswerkzeugen eingesetzt werden kann. Sie ergänzt deren Funktionsumfang, um so zusammen eine vollständige deklarative HMI-Beschreibung formulieren zu können, die auf unterschiedlichen Zielplattformen ausgeführt werden kann. Dafür stellt sie folgende Sprachmittel bereit:

- Beschreibung eines zustandsbasierten Makroverhaltens des HMIs (Menüfluss o.Ä.) und des Mikroverhaltens innerhalb dieser Zustände (für Formatter, Datenberechnungen etc.)
- Strukturierung in wiederverwendbare, abgeschlossene Module. Festlegung von Varianten, Skins und zu übersetzenden Daten
- Festlegung der Schnittstelle zu den Softwareteilen vom Plattformzulieferer[1]

[1] Die Spezifikation solcher Schnittstellen mit Hilfe domänenspezifischer Sprachen wird vom Autor auf dieser Konferenz in einem separaten Aufsatz [GW11] erläutert.

- Verbindung der Module untereinander, Kopplung an das separat modellierte GUI und an die Basisfunktionen der Plattform

HMISL ist eine formale Sprache, die für eine maschinelle Weiterverarbeitung geeignet ist. Die HMISL Dokumente sollen automatisiert auf der Zielplattform zur Ausführung gebracht werden[2] (Abbildung 1). Von Wietzke wird ein typisches Framework beschrieben, wie es dabei zum Einsatz kommen könnte [WT05].

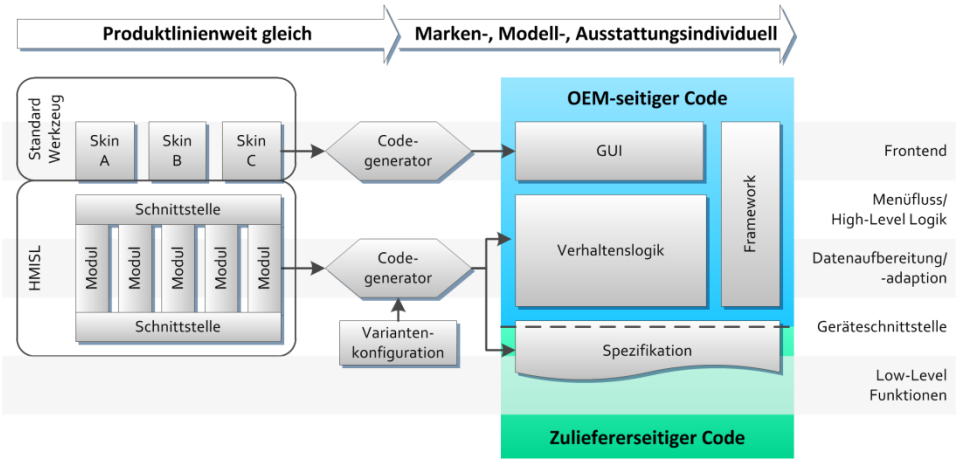

Abbildung 1 – Einsatzgebiet der HMISL und Abbildung auf die Zielplattform

3 Verwandte Arbeiten

Von Smith [SCH07] werden zur Beschreibung des HMI-Vehaltens MATLAB/StateFlow Modelle eingesetzt, wobei datenflussorientierte und zustandsbasierte Beschreibungstechniken zur Verfügung stehen. In der VOYAGER Architektur [Hö05] wird der visuelle Teil des HMIs in XML beschrieben und die Verhaltenslogik in Python codiert. Das EDONA/HMI Projekt [BV10] konzentriert sich auf sicherheitsrelevante Echtzeit-Systeme. Dort wird die GUI mit Hilfe eines SVG-Dialekts vektorbasiert beschrieben. Die darin enthaltenen Grafikprimitive können über ein vorgeschaltetes zustandsloses Datenfluss-Berechnungsmodell manipuliert werden, welches in ESTEREL formuliert ist. Alle diese Ansätze trennen somit ebenso wie die HMISL zwischen Modellen für GUI-Design und Verhalten. Von Smith wird nicht beschrieben, wie beide miteinander verbunden werden. EDONA/HMI und VOYAGER erfordern den Einsatz einer bestimmten GUI-Technologie, so dass deren Auswahl nicht mehr anhand der projektspezifischen Anforderungen erfolgen kann. Keiner der zuvor genannten Ansätze geht auf die Anforderungen von HMI-Produktfamilien ein.

[2] Aufgrund der Performancevorteile zur Laufzeit wird man dafür üblicherweise auf Codegenerierung zurückgreifen. Dennoch kann es für gewisse Anwendungsfälle wie beispielsweise Online-Dienste sinnvoll sein, die HMI-Modelle erst zur Laufzeit zu interpretieren. Die HMISL muss daher für beide Vorgehensweisen geeignet sein.

ISBN 978-3-88579-286-4

SCXML [W3C11] ist eine XML-basierte Sprache zur Beschreibung von zustandsbasiertem Verhalten. Sie bietet jedoch darüber hinaus keine Möglichkeit zur Beschreibung grafischer Oberflächen und bietet kein Variantenkonzept.

4 Notation

Moderne Automotive HMI sind sehr komplex, besitzen kurze Lebenszyklen und müssen unter hohem Kostendruck entwickelt werden. Ein wichtiges Ziel der HMISL ist es daher, eine möglichst hohe Entwicklungseffizienz zu ermöglichen. Dafür muss sie für die betroffenen Entwickler leicht erlernbar, möglichst kompakt und gut lesbar sein.

Im Gegensatz zu anderen Ansätzen wie der Infotainment Markup Language (IML) [WEA04] wurde die HMISL bewusst nicht als XML-Dialekt realisiert, weil sich dabei die Notation nicht frei anpassen lässt. Durch die spitzen Klammern und das erforderlichen Escaping gebräuchlicher Zeichen ist das Verständnis komplexer Ausdrücke und Kontrollstrukturen für Menschen erschwert. Diesem Problem wird in SCXML [W3C11] begegnet, indem nur einfachste Aktionen über XML-Sprachelemente beschrieben werden. Komplexere Abläufe und Ausdrücke werden dagegen mit Hilfe zusätzlicher Scriptsprachen ausgedrückt. Die dafür eingesetzten Sprachen sind abhängig vom SCXML-Interpreter. So akzeptiert beispielsweise die Apache Implementierung Ausdrücke wahlweise in JEXL oder JSP und kann Funktionen in separaten Java Klassen aufrufen. Diese Vorgehensweise der SCXML führt jedoch dazu, dass sich die Notation der in den XML-Dokumenten eingebetteten Scripte von den übrigen Elementen deutlich unterscheidet. Außerdem muss in den Scripten ein Escaping sowohl gemäß der gewählten Scriptsprache als auch nach XML Regeln vorgenommen werden. Dies führt zu zusätzlichen Aufwänden, erschwert die Erlernbarkeit und kann die Lesbarkeit beeinträchtigen. Weil zudem die Logik auf mehrere Sprachen und Dokumente verteilt ist, sind Verständnis und Fehlersuche darin erschwert,. HMISL soll dagegen eine geschlossene Beschreibung der vollständigen HMI-Logik liefern und dafür eine durchgängige Notation verwenden.

4 Anpassbare Ausdrucksstärke

Die HMISL soll zur Entwicklung von HMIs unterschiedlicher Automotive-Systeme wie Kombiinstrumente, Infotainment-Systeme, Rear-Seat Entertainment Einheiten oder Klimabedienteile eingesetzt werden können. Weil diese verschiedene Bedienkonzepte aufweisen können (z.B. Touchscreens, Dreh-Drück-Steller), sind zur effizienten Beschreibung ihrer HMIs jeweils andere Sprachkonstrukte erforderlich. Zudem lassen sich möglicherweise aufgrund der unterschiedlichen Hardware-Plattformen einige Konstrukte in manchen Systemen nicht realisieren. Damit dennoch für alle diese Systeme die HMISL eingesetzt werden kann, muss sie projektspezifisch angepasst werden können.

ISBN 978-3-88579-286-4

In HMISL können daher individuelle Profile erstellt werden, welche die verfügbaren Sprachkonstrukte festlegen. Jedes HMISL-Dokument muss auf ein solches Profil verweisen, um zu bestimmen, mit welcher Sprachvariante sein Inhalt formuliert ist. Auf diese Weise kann die Ausdrucksmächtigkeit an die Anforderungen des jeweiligen Projekts angepasst werden, ohne dafür die Sprachdefinition verändern zu müssen. Die Profile können auch genutzt werden, um während des Entwicklungsprozesses nach und nach weitere Sprachkonstrukte freizuschalten, sobald diese vom Codegenerator bzw. Interpreter unterstützt werden.

5 Selbstbeschreibungsfähigkeit

Beim Einsatz der HMISL in größeren Softwareprojekten kann es sinnvoll sein, projektabhängig individuelle Erweiterungen des Sprachumfangs vorzunehmen. Für die Nutzung der HMISL wurden jedoch bereits eine Vielzahl an unterstützenden Werkzeugen entwickelt wie Editoren, Hilfsmittel zur automatischen Überprüfung der Modelle und Codegeneratoren. Im Hinblick auf eine möglichst einfache Anpassbarkeit der Sprache wurde sie daher selbstbeschreibend gestaltet. Das bedeutet, dass alle veränderlichen Informationen in den HMISL-Dokumenten angegeben werden müssen, anstatt sie unveränderlich in den verarbeitenden Werkzeugen zu hinterlegen. Die Modelle konfigurieren somit die Werkzeuge und beschreiben selbst, wie sie zu verarbeiten sind. Eine projektindividuelle Konfiguration der Werkzeuge braucht somit nicht separat weitergegeben zu werden, da sie über die Modelle erfolgt, die den beteiligten Entwicklern ohnehin vorliegen. Solche Selbstbeschreibungsmechanismen sind in der HMISL überall dort vorgesehen, wo projektabhängige Anpassungen oder häufige Änderungen während des Projektverlaufs erwartet werden. Sie ermöglichen es, die im Projektverlauf notwendigen Anpassungen an den sie verarbeitenden Werkzeugen zu reduzieren. Weil die gleichen Werkzeuge somit in unterschiedlichen Projekten eingesetzt werden können, ist wiederum auch die Wiederverwendbarkeit der damit erstellten Modelle erleichtert.

Ein Beispiel für dieses Selbstbeschreibungsprinzip ist die zuvor genannte Definition eines Sprachprofils mit den Mitteln eben dieser Sprache. Dies ermöglicht es, mit einem allgemein einsetzbaren Mechanismus zu prüfen, ob in dem Dokument darüber hinaus weitere Ausdrucksmittel verwendet worden sind und in einem solchen Fall entsprechende Meldungen auszugeben. Das Selbstbeschreibungsprinzip kommt darüber hinaus zum Einsatz, um die zur Modellierung verfügbaren Datentypen und GUI-Widgets der Zielplattform einzuführen. Dabei wird festgelegt, welche Konvertierungen zwischen Datentypen möglich sind und welche Signaturen ihre Konstruktoren haben. Diese Informationen können dann genutzt werden, um zum Entwicklungszeitpunkt die korrekte Verwendung dieser Typen zu überprüfen und um den Codegenerator unabhängig von konkreten Datentypen realisieren zu können. Während des Projektverlaufs können somit neue Datentypen oder GUI-Widgets eingeführt werden, ohne dass dafür Anpassungen an der Sprache oder den sie verarbeitenden Werkzeugen vorgenommen werden müssen.

ISBN 978-3-88579-286-4

6 Verhaltensbeschreibung

Das Verhalten des HMIs wird in der HMISL ereignisorientiert in Form von Reaktionen beschrieben, die beim Eintreffen von Nachrichten (Events) oder bei Änderungen nichtflüchtiger Daten (Variablen) ausgelöst werden.

Für zustandsabhängige Reaktionen kommen hierarchische Zustandsmaschinen (HSM) zum Einsatz. Weil aber viele Reaktionen wie Werteberechnung oder Formatierungen von Daten zur Anzeige zustandsunabhängig sein, stehen dafür in der HMISL zusätzlich sogenannte Bindungen zur Verfügung. In Bindungen und HSMs kommt zur Formulierung von Mikrologik wie Ausdrücken und auszuführenden Aktionen eine eingebettete Programmiersprache zum Einsatz.

6.1 High-Level Verhaltensbeschreibung mit HSMs

Eine ausschließlich zustandslose Verhaltensbeschreibung ist für komplexe Systeme nicht geeignet, da mit zunehmender Anzahl an Regeln nicht mehr erkennbar ist, wie diese zusammenwirken. Für HMIs ist es stattdessen etabliert, Statecharts zur Beschreibung der Verhaltenslogik zu verwenden [Ch06]. Die hierarchische Strukturierung der Statecharts ermöglicht eine einfache Anpassung der Bedienabläufe, die grafische Darstellung visualisiert die Zusammenhänge. Ein großer Nachteil der Statecharts ist jedoch ihr hoher Erstellungsaufwand. So ist für Aufgaben wie Ausrichtung und Justierung der grafischen Elemente viel Zeit erforderlich, obwohl dies für das HMI-Verhalten irrelevant ist. Die Arbeit mit den grafischen Editoren kann für erfahrene Benutzer hinderlich sein, da durch viele Masken geklickt werden muss. Zudem werden die grafischen Statecharts bei zunehmender Anzahl an Transitionen schnell unübersichtlich und ein Diff bzw. Merge der grafischen Modelle ist kaum möglich. Ein weiteres Problem ist es, dass in den Statecharts mehrere Transitionen definiert werden können, die im gleichen Moment auslösen. Ein solches nichtdeterministische Verhalten lässt sich jedoch in einem realen System nicht umsetzen. Stattdessen erfolgt dort immer eine Abarbeitung der Transitionen in einer bestimmten Abfolge, die in der grafischen Darstellung der Statecharts jedoch nicht erkennbar ist.

In der HMISL kann die Verhaltensbeschreibung ebenfalls zustandsbasiert mit Hilfe von HSMs erfolgen. Diese bieten die gleichen Ausdrucksmittel wie Harel'sche Statecharts [Ha87], verzichten jedoch auf AND-States, da sich diese in einigen Systemen nur schwierig abbilden lassen. Zur Vermeidung der zuvor beschriebenen Nachteile verwenden die HSMs jedoch anstatt einer grafischen eine textuelle Notation. Dies erlaubt erfahrenen Entwicklern eine effiziente Eingabe und ein Diff oder Merge ist auch ohne spezielle Hilfsmittel möglich. Zudem wird durch die lineare textuelle Notation die Abarbeitungsreihenfolge der beschriebenen Transitionen konkretisiert, ohne hierfür auf Hilfsmittel wie Prioritäten zurückgreifen zu müssen. Da keine Informationen eingegeben werden brauchen, die lediglich der grafischen Darstellung dienen, kann die Syntax zur Beschreibung der HSMs kompakt gewählt werden, so dass sich die Entwickler auf die wesentlichen Inhaltlichen Fragen konzentrieren können.

ISBN 978-3-88579-286-4

Durch die textuelle Notation der Modelle sind diese auch ohne spezielle Werkzeuge bearbeitbar. Zur Steigerung des Benutzerkomforts wurden dennoch Editoren für HMISL in der Eclipse-IDE entwickelt. Diese unterstützen die Eingabe und das Verständnis durch Hilfsmittel wie Syntax-Highlighting, Code-Vervollständigung und Hyperlinking. Für eine bessere Skalierung bei sehr großen Modellen ermöglichen sie zudem Folding oder das Verbergen von Bestandteilen, die nicht im Fokus stehen. Während der Eingabe wird eine grafische Ansicht der HSMs generiert (Abbildung 2). Diese dient hierbei jedoch lediglich zur Veranschaulichung und kann nicht direkt manipuliert werden. Sie kann von verschiedenen Details abstrahieren, um so auch zur Diskussion des Verhaltens mit nicht-technischen Projektbeteiligten verwendet werden zu können. Durch diese Kombination aus textueller Bearbeitung und grafischer Visualisierung können die Vorteile beider Notationen verbunden werden.

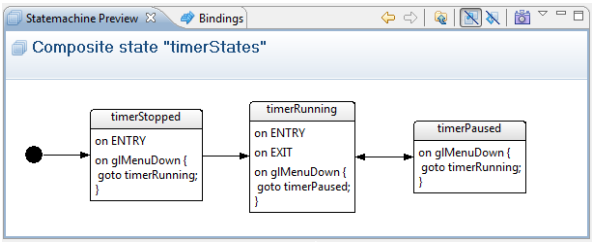

Abbildung 2 – Generierte Visualisierung für einen Zustand aus dem Codebeispiel in Kapitel 6.2

6.2 Mikroverhalten

In Statecharts können als Reaktion auf das Schalten einer Transition üblicherweise Events ausgelöst werden oder Variablen Werte zugewiesen werden. Aufwändigere Werteberechnungen oder die im HMI häufig benötigten Textoperationen sind jedoch nur mit Hilfe ergänzender, proprietärer Konzepte der Modellierungswerkzeuge möglich. Komplexere Abläufe können nicht in den Statecharts ausgedrückt werden, sondern müssen separat beschrieben werden und bei Empfang eines entsprechenden Events oder Betreten eines bestimmten Zustands ausgelöst werden. Dies führt jedoch dazu, dass das gesamte Verhalten unübersichtlich an verschiedenen Stellen verteilt beschrieben ist.

ISBN 978-3-88579-286-4

In die HMISL ist daher eine standardisierte Programmiersprache (LUA [Ie06]) an verschiedenen Stellen in die HSMs eingebettet. Mit dieser Mikrosprache können Ausdrücke und Aktionen beschrieben werden[3]. An dieser Stelle bietet die Verwendung einer echten Programmiersprache eine große Ausdrucksmächtigkeit bei guter Lesbarkeit. Die Notation der umgebenden Sprachelemente der HMISL wurde zugunsten einer einfachen Lesbarkeit an LUA angelehnt. Beide konnte zudem so nahtlos ineinander integriert werden, dass die Benutzer in der Lage sind, abhängig vom Anwendungsfall eine individuell andere Aufteilung und Verzahnung von HSMs und Mikrosprache zu wählen.

Im folgenden Codebeispiel wird der Menüfluss einer Stoppuhr unter Nutzung von HSMs und der Mikrosprache (grau hinterlegt) realisiert.

```
module HmiClockController ... {
        statemachine clockSM startswith notShown {
                state notShown {
                        on entry showScene(FpkMain);
                        on glMenuUp() goto(timerStates);
                }

                state timerStates startswith timerStopped {
                        on entry showScene(Clock);
                        on glMenuUp() goto(notShown);

                        state timerStopped {
                                on entry do
                                        log("Reset counter");
                                        counter = 0;
                                end;
                                on glMenuDown() goto(timerRunning);
                        }
                        state timerRunning {
                                on entry do
                                        log("Start timer");
                                        startTimer(ClockTimer);
                                end;

                                on exit do
                                        log("Stop timer");
                                        stopTimer(ClockTimer);
                                end;

                                on glMenuDown() [allowPause] goto(timerPaused);
                        }
                        state timerPaused [allowPause] {
                                on glMenuDown() goto(timerRunning);
                        }

                }
        }
        ...
}
```

[3] Grönniger integriert ebenso eine Standard-Programmiersprache in textuell modellierte Statecharts [GKR06] und übernimmt die in den Modellen formulierten Codefragmente unverändert bei der Codegenerierung. Auf diese Weise soll ein Zwischenschritt auf dem Weg zu einer vollständig modellbasierten Entwicklung eingeführt werden.

Der LUA Interpreter ist sehr kompakt und stellt nur geringe Systemanforderungen. Zudem steht er für unterschiedlichste Plattformen zur Verfügung (u.A. in einer C/C++-Implementierung für die CLR und auch in der JVM), in denen er mit umgebenen Code Daten austauschen kann. Die Programmiersprache LUA wäre daher geeignet, um darin die HMI-Software zu realisieren. In diesem Fall gestaltet sich die Codegenerierung aus den HMISL Modellen besonders einfach, da hierbei die darin enthaltenen Codefragmente unverändert übernommen werden können. Durch die Wahl einer interpretierten Sprache wäre zudem bei Veränderungen daran keine erneute Compilierung notwendig, wodurch insbesondere in großen Projekten die Zeit bis zum Erleben der Veränderung im laufenden System reduziert werden kann. Dennoch ist es ebenso möglich, die LUA Codefragmenten bei der Codegenerierung in eine andere Programmiersprache zu wandeln.

6.3 Bindungen

Neben den HSMs können auch zustandslose Verhaltensweisen in der HMISL modelliert werden. Diese sogenannten Bindungen können dazu genutzt werden, um Datenwerte für die Anzeige umzurechnen oder zu formatieren. Sie werden außerdem verwendet, um die Zustandsmaschinen untereinander, mit dem umgebenden System und dem extern modellierten GUI zu verbinden.

Bindungen werden stets zu bestimmten Zeitpunkten ausgelöst. Auslöser können Wertänderungen von Variablen, der Empfang bestimmter Events oder das Ablaufen von Timern sein. Die Reaktion darauf wird ebenso wie in den HSMs mit der Mikrosprache formuliert. Auf diese Weise können Werteberechnungen durchgeführt werden, andere Events erzeugt werden, Views aufgeschaltet oder Animationen gestartet werden.

Das folgende Codebeispiel ergänzt das vorherige. Es zeigt die Implementierung der Stoppuhr-Funktion mit Hilfe zweier Bindungen. Die wird von einem zyklischen Timer ausgelöst und erhöht eine Zählvariable. Eine zweite Bindung berechnet bei Wertänderung dieser Variable die Winkelstellung der Zeiger einer Analoguhr.

```
module HmiClockController ... {
          ...
          private integer counter = 0;
          every 1000ms timer ClockTimer counter += 1;

          private real hoursHandAngle = 0;
          private real minutesHandAngle = 0;
          private real secondsHandAngle = 0;

          onchange counter do
                    local seconds = (counter)%60;
                    local minutes = (counter/60)%60;
                    local hours   = (counter/60/60)%24;
                    log("Time is "..hours..":"..minutes..":".. seconds);

                    secondsHandAngle = 360*(seconds/60);
                    minutesHandAngle = 360*(minutes/60);
                    hoursHandAngle   = 360*(hours/12+minutes/60/12)%360;
          end;
}
```

Um die mit steigender Größe des Systems zunehmend komplexen Abhängigkeiten überblicken zu können, wurde ein Analysewerkzeug für Bindungen entwickelt. Abbildung 3 zeigt dessen Einsatz zur Ermittlung der Auslöser einer Wertänderung der Variablen „counter“ und der daraufhin ausgelösten Reaktionen.

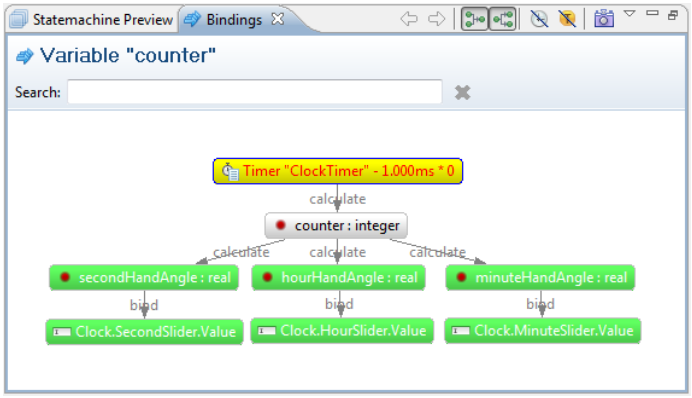

Abbildung 3 – Ausgabe des Analysewerkzeugs für das vorhergehende Beispiel

7 Variantenbildung

Um unter Wiederverwendung größtmöglicher Modellteile effizient eine Vielzahl an Marken-, Länder und Ausstattungsvarianten eines HMIs zu erzeugen, kommen in der HMISL mehrere Variabilitätskonzepte zum Einsatz.

7.1 Ausstattungsvarianten durch Kombination konfigurierbarer Module

Grundsätzlich werden alle in der HMISL beschriebenen Sachverhalte in Modulen strukturiert. Diese Module können unterschiedlich kombiniert werden. Sie sind in sich abgeschlossene Einheiten, können darüber hinaus aber auch auf öffentliche Elemente anderer Module zugreifen, wenn Abhängigkeiten zu diesen Modulen formuliert worden sind. Umgekehrt können Module Teile ihrer internen Events und Variablen veröffentlichen, so dass darauf von anderen Modulen aus zugegriffen werden kann. Zur Verbesserung der Wiederverwendbarkeit der Module können Abhängigkeiten auch gegenüber abstrakten Modulschnittstellen definiert werden, welche in den verschiedenen Softwarevarianten von unterschiedlichen Modulen angeboten werden können.

Um eine Softwarevariante zu definieren, werden Sätze an Modulen zusammengestellt (Variantendefinition). Diese Auswahl muss derart erfolgen, dass sich darin alle Abhängigkeiten auflösen lassen. Zudem kann dabei jedes genutzte Modul variantenspezifisch konfiguriert werden (Abbildung 4).

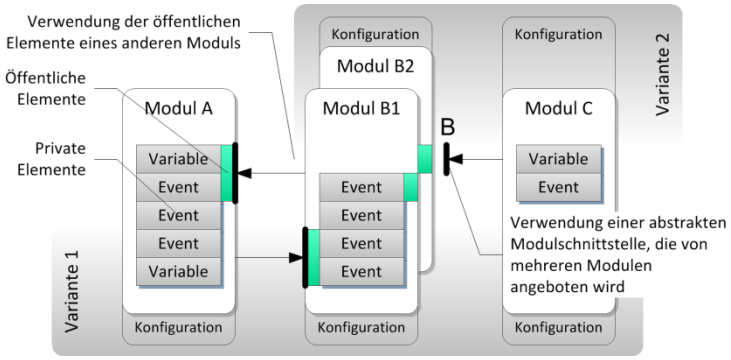

Abbildung 4 – Definition von Softwarevarianten durch Auswahl von Modulen und deren individueller Konfiguration

Insbesondere wenn Infotainmentsysteme mit gleichem Funktionsumfang in unterschiedlichen Zielmärkten angeboten werden sollen, muss dafür eine sehr hohe Zahl von HMI-Varianten erzeugt werden, die sich jedoch nur geringfügig voneinander unterscheiden. Die zugehörigen Variantendefinitionen sind somit ebenfalls sehr ähnlich. Um den Erstellungs- und Pflegeaufwand zu reduzieren, können sie daher voneinander erben, wobei ausschließlich die Abweichungen zu der übergeordneten Variantenkonfiguration festgelegt werden brauchen.

In einer Variantendefinition muss zudem angegeben werden, welches GUI-Design zu verwenden ist. Dazu wird den in der HMISL verwendeten abstrakten GUI-Elementen mittels eines Skin ein konkretes Design zugeordnet. Solche Skins können in mehreren Sprachen vorliegen, von denen in der Variantendefinition die Untermenge ausgewählt wird, die vom System unterstützt werden soll. Der Skinning-Mechanismus wird im folgenden Kapitel erläutert.

7.2 Skinning für Designvarianten

Um dieselbe Software für unterschiedliche Fahrzeugmarken verwenden zu können, muss jeweils das Look-and-Feel ihres GUIs an die spezifische Designsprache angepasst werden. In der HMISL ist daher ein Skinning-Mechanismus vorgesehen, der es ermöglicht, die gleiche Verhaltenslogik zusammen mit unterschiedlich gestalteten GUIs zu verwenden, die zudem mittels verschiedener Technologien realisiert worden sein können.

In der HMISL wird daher statt gegen ein konkretes GUI ausschließlich gegen abstrakte GUI-Schnittstellen programmiert. Diese führen zwar die Bestandteile ein, aus denen die GUI zusammengesetzt ist, sind jedoch unabhängig von einem konkreten Look-And-Feel und Realisierungstechnologien.

Der Aufbau der GUI-Schnittstellen wird ebenfalls in der HMISL definiert. Sie sind jeweils in mehrere Szenen unterteilt, die unabhängig voneinander aktiv und sichtbar sein können. Hierbei wird zwischen zwei verschiedenen Szenetypen unterschieden, die sich in ihrem Verhalten unterscheiden. Einerseits existieren normale Szenen, von denen stets nur jeweils eine aktiv sein kann. Sie können mittels eines speziellen Kommandos in der Mikrosprache sichtbar geschaltet werden, wobei gleichzeitig die vorher gezeigte Szene ausgeblendet wird. Darüber hinaus existieren Popups, von denen mehrere gleichzeitig aktiv sein können und die einander überlagern können. Popups können daher per Kommando sowohl aktiviert als auch deaktiviert werden, wobei jeweils eine Priorität angegeben wird. Popups mit höherer Priorität überlagern diejenigen mit niedrigerer. Die Schnittstelle einer Szene besteht aus den Properties der darin enthaltenen Widgets. Wird eine Szene in einem Modul der HMISL verwendet, so können diese Properties dort wie Variablen angesprochen werden. Darüber hinaus enthält die GUI-Schnittstelle Animationen, die ebenfalls über Kommandos gestartet und gestoppt werden können.

Die Verhaltenslogik wird in der HMISL ausschließlich gegen diese abstrakten GUI-Schnittstellen programmiert. Ein Skin verweist auf ein GUI-Modell, das eine dazu kompatible Schnittstelle vorweisen muss. Solche GUI-Modelle können mit Hilfe der dafür üblicher Modellierungswerkzeuge wie beispielsweise Fujitsu CGI Studio, Elektrobit GUIDE oder Expression Blend erstellt werden. Der Codegenerator erzeugt aus den HMISL-Dokumenten automatisch für jede Designvariante den spezifischen Code zum Ansprechen der GUI-Elemente.

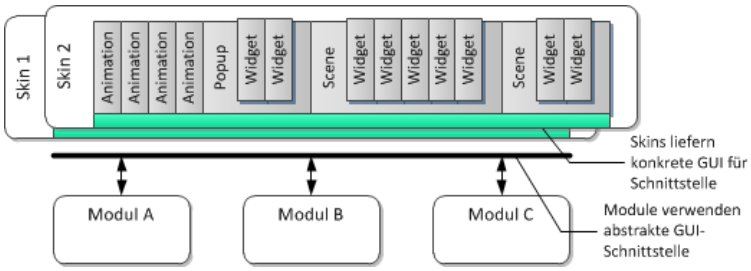

Abbildung 5 – GUI-Schnittstelle und Skins

7.3 Sprach- und Ländervarianten

Die mittels Skins erzeugten verschiedenen Designvarianten eines HMIs müssen in verschiedenen Sprachen angeboten werden. Dazu kann eine landesspezifische Konfiguration vorhandener Module vorgenommen werden. Denjenigen Variablen dieser Module, die als sprachabhängig ausgezeichnet wurden, weist sie dazu einen übersetzten Wert zu. Ebenso wie die Variantendefinitionen können auch diese sprachspezifischen Konfigurationen voneinander erben, um so bei Familien ähnlicher Sprachen Mehrfachübersetzungen zu vermeiden. Da bestimmte Sprachen unterschiedliche Schreibrichtungen verwenden, muss für eine optisch ansprechende Darstellung möglicherweise auch das GUI-Design angepasst werden. Daher kann sprachabhängig auch auf andere GUI-Modelle verwiesen werden.

ISBN 978-3-88579-286-4

7.4 Beispiel

Der folgende Code ergänzt die beiden vorherigen Beispiele. Er legt die abstrakte GUI-Schnittstelle fest und führt dafür einen Skin ein. Außerdem bindet er die Variablen eines Moduls an Properties einer Szene. Dabei verwendet er auch eine sprachabhängige Variable, für die Übersetzungen bereitgestellt werden müssen. Schließlich definiert er zwei Softwarevarianten aus den vorhandenen Modulen, Skins und Übersetzungen.

```
module HmiClockController requires ClockLanguageInterface, WpfDemoFpk, VehicleInput {
          ...
          public variant boolean allowPause;
          ...
          scene Clock {
                    HoursHand {Angle=hoursHandAngle;}
                    MinutesHand {Angle=minutesHandAngle;}
                    SecondsHand {Angle=secondsHandAngle;}
                    WelcomeLabel {Caption=welcomeMessage;}
          }
}

module VehicleInput {
          public glMenuUp();
          public glMenuDown();
          ...
}

interface gui WpfDemoFpk engine WPF {
          widgets {
                    Label {
                              string Caption;
                    }
                    Hand {
                              real Angle;
                    }
                    ...
          }
          animation StartAnimation;
          scene FpkMain {...}
          scene Clock {
                    Hand HoursHand;
                    Hand MinutesHand;
                    Hand SecondsHand;
                    Label WelcomeLabel;
          }
}

interface language ClockLanguageInterface {
          translate string welcomeMessage;
}

skin wpfFpkWide offers myLanguageInterface, WpfDemoFpk {
          language deDE {
                    model = "/WpfApplication/WPF_FPK";
                    welcomeMessage = "Willkommen";
          }
          language deAU {...}
          language enGB {...}
          language enUS {...}
}

variant ClockExample {
          modules {WpfDemoFpk, HmiClockController, VehicleInput}
          configuration {
                    allowPause = false;
          }
          skin wpfFpkWide with enUS, deDE;
}
variant PausableClockExample extends ClockExample {
          configuration {
                    allowPause = true;
          }
}
```

8 Zusammenfassung

HMISL ist eine formale Sprache um die Verhaltenslogik eines HMIs inklusive dem Dialogfluss zu beschreiben. Sie kann gemeinsam mit verschiedenen GUI-Technologien zum Einsatz kommen. Durch diese Trennung der Verhaltenslogik von den designrelevanten Teilen konnte die HMISL für eine effiziente Nutzung durch erfahrene Softwareentwickler optimiert werden. Die Modellierung erfolgt daher ohne grafische Hilfsmittel direkt in der textuellen Notation der Sprache. Um diese möglichst menschenlesbar zu gestalten, wurde sie bewusst nicht als XML-Dialekt entworfen.

Die Beschreibung der Verhaltenslogik erfolgt in der HMISL ereignisorientiert mit Hilfe zustandsbehafteter und zustandsloser Techniken, in denen jeweils eine eingebettete allgemeine Programmiersprache zur Verfügung steht. Dies ermöglicht eine kompakte Formulierung und flexible Einsetzbarkeit, ohne eigene Sprachelemente für jeden Spezialfall vorhalten zu müssen. Die HMISL bietet zudem Modularität und Variabilitätskonzepte, um unterschiedliche HMI-Varianten aus den gleichen Modellen abzuleiten und Teile dieser Modelle für Nachfolgeprojekte wiederverwenden zu können. Ein Skinning-Konzept ermöglicht es, dieselbe Verhaltenslogik mit unterschiedlichen GUI-Designs zu verwenden.

Die Sprache bietet Möglichkeiten zur Anpassung der verfügbaren Ausdrucksmächtigkeit, ohne dafür die Grammatik der Sprache verändern zu müssen. Die zur Modellierung zur Verfügung stehenden Datentypen und GUI-Widgets werden mit Hilfe der Sprache selbst eingeführt. Dies ermöglicht es, die HMISL in unterschiedlichen Automotive-HMI-Projekten einzusetzen, ohne die verarbeitenden Werkzeuge verändern zu müssen.

9 Bewertung und Ausblick

Mit Hilfe der HMISL wurden bereits einige HMI-Prototypen entwickelt, an denen die grundsätzlichen Sprachkonzepte gezeigt werden können. Derzeit wird zudem mit Hilfe der HMISL ein Teil der HMI-Software von einem in der Entwicklung befindlichen Infotainmentsystem reimplementiert. Dabei wird eine Zeitmessung durchgeführt, um so einen Vergleich der Aufwände gegenüber der bisherigen Vorgehensweise vorzunehmen.

Während der Entwicklung erster Prototypen war jedoch bereits erkennbar, dass die Verwendung der dynamisch typisierten Sprache LUA innerhalb der statisch typisierten HMISL zu einer aufwändigen Fehlersuche führen kann. Aus diesem Grund wird derzeit nach einer anderen geeigneten Programmiersprache mit statischer Typisierung gesucht, um sie anstelle von LUA als Mikrosprache einzusetzen.

Zudem wurde begonnen, den Codegenerator so zu erweitern, dass er bereits Ausdrücke variantenspezifisch auswertet und gegebenenfalls vereinfacht, um so die Laufzeitperformance des Generats zu verbessern. Zukünftig sollen von ihm auch ungenutzte Bindungen und unerreichbare Zustände erkannt und automatisch entfernt werden.

ISBN 978-3-88579-286-4

Um die mittels Codegenerator aus einer HMISL-Beschreibung erzeugte HMI-Software an separat erstellte Gerätedienste oder den Fahrzeugbus anzubinden, ist bisher noch manuelle Codierung notwendig. Gleiches gilt für die Verarbeitung von Eingabeereignissen aus der GUI. Zukünftig sollten daher weitere Sprachmittel in der HMISL vorgesehen werden, mit denen diese Anbindung modelliert und der entsprechende Code somit ebenfalls generiert werden kann. Um in der HMISL auch Sprachdialoge beschreiben zu können sind weitere Ergänzungen des Funktionsumfangs geplant. Darüber hinaus sollen für eine einfache Verwendung komplexer Datentypen wie Listen, Strukturen, Bitfelder und Enumerationen zusätzliche Sprachkonstrukte bereitgestellt werden.

Literaturverzeichnis

[Bo10] Boisgérault, S.; Vecchié, E.; Meunier, O.; Temmos, J.: EDONA/HMI - Modelling of Advanced Automotive Interfaces. In (Société des Ingénieurs de l'Automobile): Proceedings of Embedded Real Time Software and Systems (ERTS2), 2010.

[Ch06] Chlebek, P.: User Interface-orientierte Softwarearchitektur. Friedrich Vieweg & Sohn Verlag/GWV Fachverlage GmbH, Wiesbaden, 2006.

[GKR06] Grönniger, H.; Krahn, H.; Rumpe, B.; Schindler, M.: Integration von Modellen in einen codebasierten Softwareentwicklungsprozess. In (Mayr, H. C.; Breu, R. Hrsg.): Proc. Modellierung 2006, Innsbruck, Austria, GI, 2006; S. 67-81.

[GW11] Gerlach, S.; Widegreen, A.: Spezifikationstechniken für Software-Schnittstellen in Fahrzeug-Infotainmentsystemen. (unveröffentlicht)

[Hö05] Höwing, F.: Impact of the Software Architecture on the Development Process in Distributed Systems. In (Gesamtzentrum für Verkehr): AAET 2005, GZVB, Braunschweig, 2005; S. 174–190.

[Ha87] Harel, D.: Statecharts: A visual formalism for complex systems. In: Science of Computer Programming, North-Holland, 8/1987, S. 231-274

[Ie06] Ierusalimschy, R.: Programming in Lua (second edition). Lua.org, 2006

[SCH07] Smith, P. F.; Chen, J.; Hu, H.: Model-Based Design Study and Evaluation of New HMI Concepts for Vehicle Multimedia, Climate Control and Navigation Systems. In: Safety test methodology, 2007; Society of Automotive Engineers, Warrendale, Pa., 2007.

[W3C11] World Wide Web Consortium: State Chart XML (SCXML). State Machine Notation for Control Abstraction. Working Draft 26 April 2011. http://www.w3.org/TR/2011/WD-scxml-20110426/

[WEA04] Wegner, G.; Endt, P.; Angelski, C.: Das elektronische Lastenheft als Mittel zur Kostenreduktion bei der Entwicklung der Mensch-Maschine-Schnittstelle von Infotainment-Systemen im Fahrzeug. In (C. Müller-Bagehl; P. Endt Hrsg.): Infotainment, Telematik im Fahrzeug; expert-Verlag, Renningen, 2004; S. 38-45.

[WT05] Wietzke, J.; Tran, M. T.: Automotive Embedded Systeme. Effizientes Framework - Vom Design zur Implementierung; Springer, Berlin, Heidelberg, 2005.

ISBN 978-3-88579-286-4