Knut Salomon

(E-Mail: Knut.Salomon@proskills.de)
unterstützt seit über 20 Jahren als Berater, Trainer und Coach Unternehmen jeder Größenordnung bei der Lösung projektspezifischer Problemstellungen. Er ist Leiter des Instituts für Professional Skills.

Colin Hood

(E-Mail: Colin.Hood@HOOD-Group.com)
arbeitet seit 1977 im Engineering. 1987 gründete er die HOOD-Group, die heute mehr als 40 Systems Engineers , die „Experts in Requirements" sind, beschäftigt. Eines seiner Spezialgebiete ist es, Menschen und Organisationen bei der Überwindung von Herausforderungen, die bei Veränderungen entstehen, zu unterstützen.

Einführung einer flexiblen und robusten komponenten-basierten Software-Architektur

Die Bestandteile einer Software-Architektur können als eine Sammlung von Diensten (Services) verstanden werden, die Eingangs- in Ausgangsinformationen transformieren. Die Informationen werden geprüft und ausgehend vom Prüfungsergebnis werden angemessene Berichts- und Steuerungsmaßnahmen durchgeführt. Ist die Vorgehensweise erst einmal entwickelt worden, kann sie immer wieder verwendet werden. Dieser Artikel beschreibt die Entwicklung eines Software-Architektur-Modells, das maßgeblich auf gesundem Menschenverstand fußt. Die Architektur besteht aus Diensten und deren Schnittstellen (Service Oriented Architecture). Diese flexible und robuste komponenten-basierte Software-Architektur wird eingesetzt, um in kürzester Zeit ein komplexes und anpassbares System zu entwickeln und eine sehr erfolgreiche Überführung in die Serienproduktion zu unterstützen.

Einführung

Wir möchten Ihnen unseren Ansatz anhand eines Beispiels vorstellen. Ein Zulieferer in der Automobilindustrie ist Experte in der Herstellung und Bereitstellung von Systemen, die in der Regel aus Sensorik, Steuerungseinheiten und Aktuatorik bestehen, z. B. Lenkungssteuerung, Klimaanlage oder Geschwindigkeitsregler. Solche Systeme müssen in immer kürzerer Zeit entwickelt und produziert werden und unterscheiden sich von Kunde zu Kunde. Oftmals hat ein Kunde Änderungswünsche, die gravierende Folgen für das System haben können. Wenn dem Kunden schnell etwas geliefert werden muss, werden zu oft Änderungen gemacht, ohne die spätere Änderbarkeit zu berücksichtigen. Dadurch kann sich die Anzahl der wiederverwendeten Komponenten drastisch reduzieren, was im Nachgang zu unwirtschaftlichem Aufwand bei der Pflege der Systeme führt. Aufgrund der hochwertigen und sensiblen Systeme einerseits und der von den Kunden in immer kürzerer Zeit gewünschten individuellen Anpassungen andererseits, stellt es sich als schwierig heraus, die Balance zwischen Wirtschaftlichkeit und Kundenzufriedenheit herzustellen und zu wahren.

HOOD führte bei dem Zulieferer eine Software-Architektur ein, die aus einfachen Komponenten mit klar beschriebenen Schnittstellen besteht und es erlaubt, die Komponenten hoch flexibel zu Systemen zu kombinieren. Mit der neuen Software-Architektur können dabei Ideen in kurzer Zeit als Prototypen in Form einer Simulation umgesetzt werden. Dadurch können bestehende Anforderungen validiert und neue Anforderungen entdeckt und erhoben werden. In kurzen Iterationen wird immer mehr reale Hardware verwendet, um den Prototypen weiterzuentwickeln. Auf dessen Basis kann schnell ein erstes Muster und schließlich das produktionsbereite System entwickelt werden.

Das Beispiel, das in diesem Artikel verwendet wird, entstammt einem realen Projekt. Es wurde so verändert, dass zum einen die Anonymität des Auftraggebers gewahrt wird und zum anderen keine wirtschaftlich sensiblen Informationen preisgegeben werden. Die Herausforderung bestand darin, ein System zu entwickeln, welches an verschiedene Automobilhersteller, die unterschiedliche Sensoren und Aktuatoren einsetzen, mit geringem bis gar keinem Anpassungsaufwand vertrieben werden kann.

Der meiste Aufwand wurde investiert, die Menschen in dem Verständnis zu unterstützen, dass die bisherige Arbeitsweise nicht länger geeignet sein wird, um in einem Markt mit immer kürzer werdenden Innovationszyklen und steigendem Wettbewerbsdruck wettbewerbsfähig zu sein. Auf der menschlichen Seite stellte sich dabei als größtes Hindernis eine für die meisten Beteiligten nicht sichtbare Notwendigkeit für Veränderungen aufgrund einer hohen Erfolgsquote der bisherigen Arbeitsweise heraus. Die Öffnung und Akzeptanz für neue Sicht- und Arbeitsweisen vollzog sich – je nach Person – im Zeitraum von ungefähr einem Jahr.

Durch Simulationen und die Verwendung von Prototypen waren wir in der Lage, schnell zu iterieren und den Weg von Anforderungen über Design-Optionen bis hin zu einer möglichen Umsetzung leicht und zügig aufzuzeigen. Auch wurden so –

getreu dem Motto „Der Appetit kommt beim Essen“ – neue Anforderungen und Design-Optionen erhoben oder bestehende verändert. Colin Hood meint dazu: „Die Benutzung von Prototypen ist eine sehr erfolgreiche Methode, um Anforderungen zu erheben. Menschen, die sehr viel Zeit damit verbringen, eine perfekte Beschreibung aller Anforderungen zu Beginn des Projekts zu erstellen, täuschen sich oft selber und verschwenden das Geld ihrer Kunden. Es ist besser, dem Kunden Ideen in einer leicht verständlichen Art und Weise, die ihm eine Visualisierung ermöglicht, zu präsentieren, um schneller und erfolgreicher Anforderungen zu erheben.“

Der Artikel beschreibt die Iterationen, die zur Entwicklung der Software-Architektur durchgeführt wurden, ausgehend von

- Scope, Träumen und Platzhalter-Anforderungen über
- die Erhebung vollständiger Anforderungen zu
- einem grundlegenden Modell mit
- identifizierten und definierten Schnittstellen über
- ein konkretes Beispiel mit Simulation der Logik bis zu
- Hardware-in-the-loop und letztendlich
- einer Probefahrt mit einem Entwicklungs-Steuergerät im Auto.

Erste Iteration: Scope, Träume und Platzhalter-Anforderungen

Bei der Entwicklung der vorgestellten Software-Architektur war es notwendig, Ideen so früh wie möglich mit allen Beteiligten zu diskutieren. Die ersten Ideen wurden mit Hilfe von Microsoft® PowerPoint® präsentiert. Dies stellte sich als hilfreich zur Erhebung und Validierung von grundlegenden Anforderungen heraus.

Die Scope-Definition bezog sich auf Funktionen, physikalische Objekte und Nutzungsszenarien. Das heißt, welche Sensorik, Aktuatorik usw. waren innerhalb des Projektes zu betrachten und welche nicht, welche Funktionalität muss gewährleistet sein und welche Verwendung soll mit dem Probegerät ermöglicht werden.

Die High-Level-Anforderungen des Auftraggebers einzufangen war für das Projekt entscheidend, da es sich dabei um deren Träume handelte: Was würden sie sich wünschen, wenn es keine physischen Grenzen gäbe? Diese Träume wurden

Abb. 1: Eingabe - Verarbeitung – Ausgabe

schriftlich festgehalten (Platzhalter-Anforderungen, zum Beispiel **„Drive-by-Wire“**), manchmal auch nur mit Stichworten wie „konfigurierbar“ oder „AUTOSAR“, um sicherzustellen, dass in der nächsten Iteration nichts vergessen wird.

Zweite Iteration: Funktionale Anforderungen, und nicht-funktionale Anforderungen (Qualitätsanforderungen und Randbedingungen)

Vage Ideen aus der ersten Iteration wurden als Basis verwendet, um eine Liste kompletter Anforderungen zu erstellen, hauptsächlich in textueller Form (zum Beispiel *„Wenn das Lenkrad nach links gedreht wurde, dann lenke nach links.“*). Teilweise wurden Skizzen zur Unterstützung der Verständlichkeit hinzugefügt. Anfangs fühlte sich der Auftraggeber am wohlsten damit, Anforderungen in Form vorhandener Lösungen zu spezifizieren. Das wurde akzeptiert, um Barrieren während der Anforderungserhebung zu verringern, in späteren Iterationen wurde dies jedoch verbessert.

Dritte Iteration: Eine grundlegende Architektur

Die verwendete grundlegende Architektur wurde sehr einfach gehalten.

a) Die Architektur sollte eine einfache Konfiguration der einzelnen Komponenten und damit des Gesamtsystems,
b) einen iterativen Übergang von einer reinen Simulation zu realer Hardware und
c) eine von der Hardware unabhängige Wiederverwendung der Algorithmen ermöglichen.

Der Ausgang der Überlegungen zum Aufbau der Basis der Software-Architektur bildet ein in Deutschland als EVA (Eingabe - Verarbeitung - Ausgabe) bekanntes und sehr einfaches Modell (s. Abbildung 1).

Die Eingabe erlaubt uns, aus verschiedenen Arten von Sensoren (zum Beispiel Lenkrad) Informationen der Verarbeitung zur Verfügung zu stellen. Der Fahrtrichtungswunsch des Fahrers wird in Daten umgesetzt, die das System verarbeiten kann. Die Verarbeitung bildet den Kern der Übersetzung der Anforderungen. Die Ein- und Ausgabe erlaubt es, den Verarbeitungsalgorithmus hardware-unabhängig zu gestalten. Die Ausgabe erlaubt uns, verschiedenen Arten von Aktuatoren (zum Beispiel Elektromotor) Informationen aus der Verarbeitung zur Verfügung zu stellen. Die verarbeiteten Daten werden in Signale umgesetzt um die Aktuatoren zu steuern.

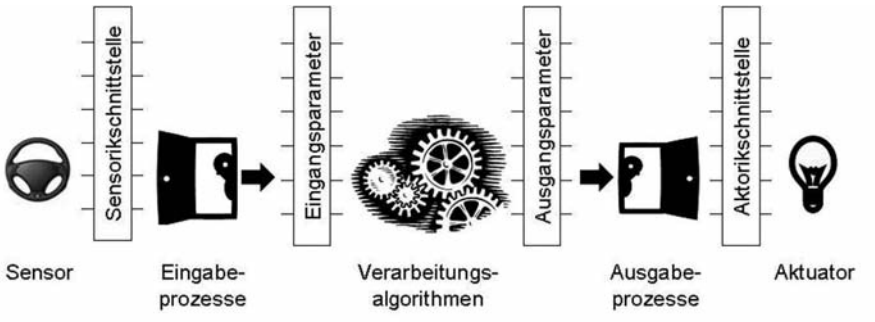

Abb. 2: Grundlegendes Software-Architekturmodell im Systemkontext

Vierte Iteration: Schnittstellen identifiziert

Das Modell aus der dritten Iteration wurde weiter verfeinert, um Komponenten zu identifizieren, die sowohl in Abhängigkeit vom Kunden als auch zwischen einzelnen Systemfamilien im Endprodukt variieren können. Abbildung 2 zeigt, wie Komponenten auf Ein- und Ausgabeseite über zusätzliche Schnittstellen an das EVA-Modell angebunden werden. Die kundenspezifische Sensorik, Aktuatorik und Schnittstellen sowie deren Eingabe- und Ausgabeprozesse werden ausgewählt und zu einem gültigen System konfiguriert.

Für die Lenkungssteuerung kann auf der Eingabeseite die Aufforderung nach links oder rechts abzubiegen oder einfach geradeaus zu fahren aus verschiedenen Quellen kommen:

d) Über ein Lenkrad, das von einem Fahrer bedient wird.
e) Über einen Joystick, der von einem Fahrer bedient wird.
f) Über automatische Anpassungen, die z. B. von einem Bremssystem ausgehen.
g) Ferngesteuert.

Fünfte Iteration: Schnittstellen definiert

Die Schnittstellen wurden konform zu AUTOSAR (Automotive Open System Architecture) modelliert und entwickelt. AUTOSAR wird immer mehr zu einem Standard für Software-System-Architekturen in der Automobilindustrie. Als Betriebssystem wurde OSEK-OS (Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug - Operating System) verwendet. Auf Basis dieses Betriebssystems sind die Implementierung quasi paralleler Prozesse und die Wahrung eines de facto-Standards in der Automobilindustrie möglich.

Es war notwendig, die Ingenieure bei der Spezifizierung der Anforderungen zu unterstützen. Sie sollten nicht das Design des Systems einschränken, sondern beschreiben, was wirklich erreicht werden soll. Ein Verarbeitungsalgorithmus darf beispielsweise nicht die Beschreibung der zu verwendenden Eingangs- und Ausgangskomponenten beinhalten. Der Verarbeitungsalgorithmus, der priorisiert, welche Lenkbewegung durchgeführt werden soll, muss nicht wissen, von welchem Sensor die Lenkaufforderung kommt. Zum Beispiel soll es für den Verarbeitungsalgorithmus nicht relevant sein, ob der Fahrer seinen Lenkwunsch über ein Lenkrad oder einen Joystick äußert.

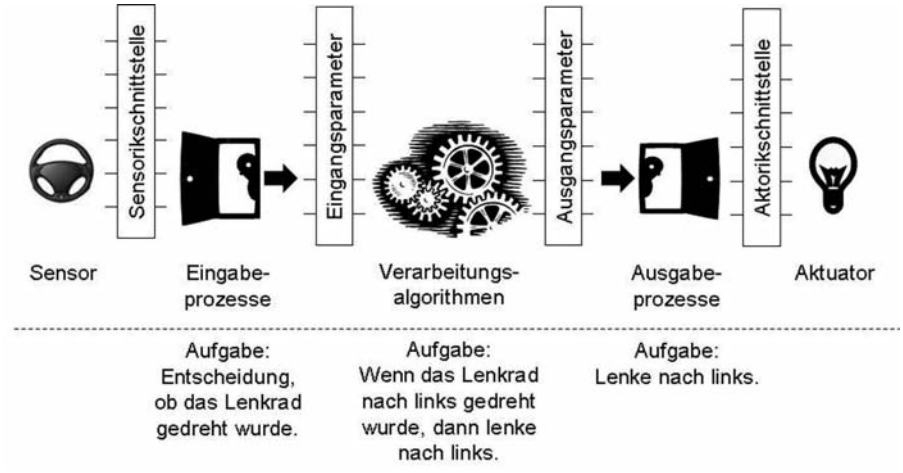

Abb. 3: Prototypisches Beispiel (Verarbeitungsalgorithmus noch sensorabhängig)

Die Aufgabe der Eingabeprozesse ist es, die Information von der Sensorschnittstelle in Eingangsparameter zu übersetzen. Der Eingangsparameter für den Lenkalgorithmus könnte der gewünschte Winkel, die Änderungsgeschwindigkeit oder die Art der Änderung sein. Die Eingabeprozesse werden aufgeteilt in weitere Unterprozesse oder Schritte. Diese wandeln die aktuellen Eingangsgrößen der Sensoren in die gewünschten Eingangsparameter für den Verarbeitungsalgorithmus um. Diese Eingabeinformationen müssen zusammen mit den endgültigen Sensoren und deren Schnittstellen konfiguriert werden, um ein gültiges System herzustellen. In einem ersten Prototyp können Aufgaben der Eingabeprozesse und des Verarbeitungsalgorithmus unterteilt werden wie in Abbildung 3 beispielhaft dargestellt.

Um Menschen dabei zu helfen, Verarbeitungsalgorithmen unabhängig von Hardware zu beschreiben, kann ein Sensor (z. B. Lenkrad) in diesem Beispiel als eine Datenquelle und ein Aktuator (z. B. Lenkaktuator) als eine Datensenke betrachtet werden. Im Hinblick auf mögliche kundenspezifische Anpassungs- und Konfigurationswünsche der Sensorik und Aktuatorik muss also die Beschreibung der Verarbeitungsprozesse aus Abbildung 3 verändert werden. Die in Abbildung 3 dargestellte Verarbeitungsbeschreibung wird in Abbildung 4 unabhängig von einer spezifischen Sensorik und Aktuatorik beschrieben.

Um Sensoren und Aktuatoren austauschbar zu machen, müssen die Eingabe- und

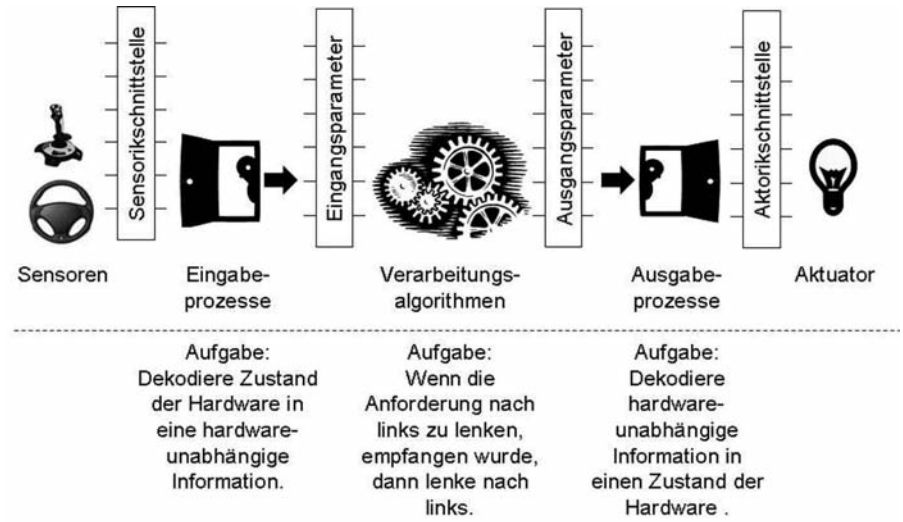

Abb. 4: Prototypisches Beispiel (Verarbeitungsbeschreibung ohne spezifische Sensorik- oder Aktuatorikinformation)

Ausgabeprozesse zusammen mit deren Schnittstellen konfigurierbar sein. Durch die Verwendung von konfigurierbaren Schnittstellen, die von der aktuell verwendeten Hardware unabhängig sind, kann das System leicht an unterschiedliche Hardware angepasst werden. Dabei muss der Kern der Verarbeitungsalgorithmen nicht verändert werden. Auch kann identische Hardware bei verschiedenen Automobilherstellern, die ein unterschiedliches Systemverhalten wünschen, eingesetzt werden. Nur der Verarbeitungsalgorithmus muss ausgetauscht werden. Während der Konzeption, des Designs und der Entwicklung wurde ein CASE-Tool eingesetzt, um die Konsistenz und Machbarkeit der Algorithmen und Prozesse zu überprüfen.

Die Kommunikation zwischen den einzelnen Komponenten (die aus Diensten und deren Schnittstellen bestehen) der Software-Architektur sollte so konzipiert werden, dass die technische Umsetzung der Kommunikation vor den Komponenten verborgen wird. In einem idealen System könnte man sich vorstellen, dass zum Beispiel der Austausch eines Sensors durch einen Sensor mit einer anderen technologischen Basis keinen Einfluss auf die übrigen Komponenten der Software-Architektur hat. Realistisch betrachtet wird nicht jeder Sensor über die gleiche Schnittstelle verfügen. So wird vermutlich die Komponente des Systems, die die Schnittstelle des Sensors anspricht, bei einem Austausch des Sensors angepasst werden müssen. Das bedeutet, dass die Eingabeprozesse in den **Abbildungen 3 und 4** in mehrere konfigurierbare Schichten zerlegt sind.

Dies führte zu einer weiteren Herausforderung. Um Anpassungen am Systemverhalten einfach, konsistent und annähernd fehlerfrei zu ermöglichen, musste die Software-Architektur, eine automatisierte Konfiguration der verwendeten Hardware, der Schnittstellen und Algorithmen erlauben. Dazu wurde eine Konfigurations-Methode in die Software-Architektur integriert. Dies ermöglicht dem Anwender nur die Übernahme von validen Konfigurationen in das System.

Sechste Iteration: Ein konkretes Beispiel mit Simulation der Logik

Die nächste Stufe bestand darin, das Systemverhalten in einem CASE-Tool zu simulieren. Dies ermöglichte es, die Umsetzbarkeit der Ideen früh zu verifizieren. Insbesondere der Informationsfluss zwischen den Komponenten konnte auf diese Weise sehr schnell und effektvoll aufgezeigt, überprüft und optimiert werden. Durch die Simulationen wurde das Systemverhalten für die Ingenieure und deren Kunden in einem frühen Entwicklungsstadium erlebbar. Für die Lenkungssteuerung wurden verschiedene Fahrsituationen simuliert, zum Beispiel Stadtverkehr, Landstraße und Autobahn.

Siebte Iteration: Hardware-in-the-loop

Nachdem die ersten Simulationen den gesamten Komponentenbereich (vom Sensor bis zum Aktuator) umfassten, wurden reale Aktuatoren an das System angeschlossen. Dabei diente eine Laborumgebung als Ersatz für die reale Steuerungseinheit. Dies war vor allem für das Management wichtig. Es konnte gezeigt werden, dass ein Projektfortschritt vorhanden war und alle Beteiligten ein gemeinsames Verständnis der Aufgabenstellung und des Lösungsansatzes hatten. Bei der schrittweisen Einführung von Hardware wurden zuerst ein echtes Lenkradsystem eingebunden und dann die realen Steuerungsmotoren verwendet.

Achte Iteration: Probefahrt mit Entwicklungs-Steuergerät im Auto

Die Modelle im CASE-Tool wurden automatisiert in ausführbaren Quellcode übersetzt. Der nächste konsequente Schritt war, die Software auf eine reale Steuerungseinheit zu übertragen und diese in einem Fahrzeug zu verbauen. Dann konnte das System unter realen Bedingungen getestet werden.

Erfolgsmessung

Die Funktionsfähigkeit der neuen Software-Architektur wurde unter anderem durch den Austausch von Sensoren und Aktuatoren überprüft. Die Verarbeitungsalgorithmen wurden dabei nicht verändert. Das System zeigte ein identisches Verhalten vor und nach dem Austausch der Sensoren und Aktuatoren. Eine weitere Überprüfung bestand darin, die Sensoren und Aktuatoren zu belassen und die Algorithmen durch andere Implementierungen zu ersetzen, um ein verändertes Systemverhalten zu erzielen. Dies verlief ebenfalls erfolgreich. Die automatisierte Konfigurations-Methode wurde durch ein separates System unterstützt. Dessen Funktionstüchtigkeit wurde eigenständig verifiziert.

Fazit

Die Geschwindigkeit des Projektfortschritts kann durch die Verwendung von kurzen Iterationszyklen, unterschiedlichen Arten von Prototypen und die Konzentration auf das zu erzielende Ergebnis stark gesteigert werden. Dabei muss das Design der vorherigen Systeme nicht notwendigerweise die zu erfüllenden Anforderungen des neu zu entwickelnden Systems widerspiegeln. So können gedankliche Begrenzungen offenbart werden und die Entwickler fühlen sich freier, radikale Verbesserungen vorzuschlagen. Wenn Veränderung als ein normaler Bestandteil in der Systementwicklung angesehen wird, kann ein Design, das Spielraum für künftige Veränderungen lässt und gleichzeitig erlaubt, diese kostengünstig durchzuführen, entworfen werden. Mithilfe einer Software-Architektur aus einer Anzahl von Komponenten, die wiederum aus Diensten und Schnittstellen bestehen, können diese Herausforderungen gemeistert werden. ■

www.HOOD-Group.com
www.ProSkills.de