Embedded Wizard
A product of TARA SYSTEMS

Best Practices der toolgestützten Embedded GUI Entwicklung

Autor: Manuel Melic, Embedded Wizard Product Manager April 2015

Software Ingenieure für Embedded Systems kommen häufig zur GUI-Entwicklung wie die Jungfrau zum Kind: Schnell soll für die nächste Messe, für den nächsten (wichtigen) Kundenbesuch oder für die Präsentation der neuen Produktidee ein Prototyp implementiert werden, der schöner, schneller und besser zu bedienen ist als das aktuelle Produkt - und dabei soll der Prototyp sich nicht hinter gewohntem Design und Bedienung eines iPhones oder iPads verstecken müssen. Doch dabei ist der neu berufene HMI-Entwickler unerfahren - es ist nämlich sein erstes UI-Projekt.

Neben dem chronischen Zeitmangel und der neuen Herausforderung („wir implementieren jetzt eine graphische Benutzeroberfläche“) kommen auch noch weitere Fragestellungen hinzu: Welche Hardware wird verwendet? Ist vielleicht eine neu entworfene Hardware bereits verfügbar oder soll der Prototyp mit der bereits existierenden Hardware umgesetzt werden? Ist eventuell doch das Eval-Board das Mittel der Wahl? Welche Unterschiede erwarten den Entwickler später? Welche Performance bietet die Hardware für ein GUI? Welche Software-Lösungen und welchen Support bietet der Chip-Hersteller? Welche Lösungsansätze aus Software-Sicht gibt es überhaupt? Gibt es eine GUI-Spezifikation und wenn ja in welchem Zustand ist sie? Wie ist das Look and Feel und was soll die neue Applikation und damit das neue UI überhaupt leisten?

Nachdem man sich allmählich einen Überblick verschafft hat, ist man gezwungen, schnell mit der Implementierung anzufangen. Das heißt, dass die notwendige Zeit für Konzeptarbeiten, Proof-of-Concepts, Tests, das eigentliche Implementieren sämtlicher Features, etc. nicht zur Verfügung steht, um den gesetzten Termin halten zu können. Dies ist aber auch legitim - es handelt sich ja nur um einen Prototypen. Um schnell ans Ziel zu kommen verwendet man oft und gerne Hardware, mit der man bereits vertraut ist und die vom Chip-Hersteller dazugehörigen Software-Bibliotheken bzw. GUI-Tools. Schließlich erreicht man den Meilenstein mit Mühe und Not: Der Prototyp ist rechtzeitig fertig, das GUI-Tool unterstützte die Entwicklung und das Management ist (zunächst) zufrieden. Die Messe, der Kundenbesuch oder die Präsentation kann kommen.

Aber was passiert danach? Der nächste evolutionäre Schritt ist sicherlich die Weiterentwicklung des Prototyps zum Produkt. Damit das Endergebnis gut wird, müssen dabei aus GUI-Sicht unterschiedliche Aspekte sowie Dinge, die bei der Realisierung des Prototyps in Erfahrung gebracht wurden, berücksichtigt werden. Konkret bedeutet dies, dass vor allem die Hürde bei der optimalen Anbindung

an die Ziel-Hardware genommen werden muss. Selbstverständlich muss aber auch der eigentliche Funktionsumfang klar sein. Das zu realisierende Design und die dabei zu erwartende Performance spielen ebenso eine tragende Rolle. Stoßen im späteren Projektverlauf weitere Entwickler hinzu, sollten natürlich auch organisatorische Themen nicht außer Acht gelassen werden.

Folglich können bei der Entwicklung in den genannten fünf Themengebieten mannigfaltige Probleme auftauchen. Ein paar Beispiele:

Hardware

- Die zunächst verwendete Hardware ist zu teuer, zu alt, zu langsam oder bietet nicht die benötigte Funktionalität.

- Sollte dann ein Hardware-Wechsel anstehen (während des Projekts oder mit der nächsten Produktversion), kann es oft vorkommen, dass gerade die verwendeten Software-Komponenten, die eine zu große Abhängigkeit zur bislang eingesetzten Hardware haben, nicht wiederverwendet werden können. Folglich müsste ein großer Teil der Software neu entwickelt werden.

- Auch kann es vorkommen, dass die finale Hardware zum Start der Software-Entwicklung noch nicht verfügbar ist, da jene selbst noch entwickelt werden muss. Allerdings ist es zwingend notwendig, den Entwicklungsprozess zu parallelisieren, um den gerne mal straffen Zeitplan einhalten zu können.

- Eventuell sollen nicht nur eine Plattform, sondern zwei oder mehrere Plattformen mit der gleichen Software versorgt werden. Zum Beispiel möchte der Vertriebsaußendienst dem potentiellen Kunden nicht die tonnenschwere Maschine beim Besuch auf den Tisch legen - vielmehr sollte ein handliches Tablet oder Smartphone die Bedienbarkeit und Funktionalität der Maschine demonstrieren.

Durch plattformunabhängige Software Komponenten können die o.g. Probleme von vornherein vermieden werden. Doch wie kann Plattformunabhängigkeit erreicht werden? Beim toolbasierten Entwicklungsprozess sollte das Tool selbst die Möglichkeit bieten, GUIs unabhängig von der späteren Zielplattform zu entwickeln. Mit Hilfe einer abstrakten Zwischenschicht kann dies gelöst werden.

Funktionsumfang

- Es liegt auf der Hand, dass der Softwarestand des realisierten Prototyps nur einen Ausschnitt der späteren Funktionalität des Endprodukts zeigt. Soll nun weitere Funktionalität eingearbeitet werden, die erst im späteren Verlauf des Projekts spezifiziert wird, kann es passieren, dass die eingangs gewählte Architektur zu einer schlecht erweiterbaren, nur schwer pflegbaren und zu komplexen Software führt.

- Mangels Zeit könnte es auch passiert sein, dass der Prototyp nur als Mock-Up bzw. Click-Dummy implementiert wurde. Folglich wurde keine Anbindung an die Hardware und/ oder Middleware durchgeführt - doch wie genau soll dies nun umgesetzt werden?

- Ein ähnliches Problem wird dem Entwickler früher oder später bewusst, wenn man den Ort der implementierten Logik ausfindig macht: Gibt es eine klare Trennung von GUI und Logik oder befindet sich ein Großteil der Logik innerhalb der graphischen Benutzeroberfläche?

- Selbstverständlich können aber auch nicht-funktionale Anforderungen Hindernisse bei der GUI-Entwicklung darstellen. Für das Produkt soll zum Beispiel ein neuer Markt erschlossen werden und dieser erfordert eine zusätzliche Spracherweiterung. Doch wie lässt sich dies am einfachsten mit der bestehenden Software-Struktur realisieren?

Die oben genannten Hindernisse, die während oder nach der Entwicklung auf den Software-Entwickler zukommen, können gut eliminiert werden, sofern bereits zu Beginn eines Projekts ausreichend Zeit für die Konzeptarbeit eingeräumt wird. Die Architektur und Struktur der neu zu implementierenden Software sollte klar sein, bevor mit der eigentlichen Implementierung begonnen wird. Dazu ist es notwendig, Dinge, die sich logisch voneinander trennen lassen, zu identifizieren und entsprechend zu separieren: Das Aussehen, die Geschäftslogik, die einzelnen UI Komponenten (Buttons, Listen, Menüs, Dialoge, ...), String-Literale (für die Unterstützung von mehreren Sprachen), etc.

Design und Interaktion

- Gute Software zu entwickeln ist eine Kunst für sich und Kunst kommt bekanntlich vom Können. Und selbstverständlich ist gutes Design auch eine Kunst für sich. Nun gibt es mit Sicherheit auch gute Software-Entwickler, die etwas von gutem Design verstehen - aber das ist eher die Ausnahme. Demzufolge sehen UIs, die von guten Software Entwicklern angefertigt werden, eher wie technische Schalttafeln aus - denn gutes Design können andere möglicherweise besser.

- Gutes Design zeichnet sich nicht durch “schön” oder “ansprechend” aus. Viel wichtiger ist nämlich die gute Bedienbarkeit, die Usability. Jeder kennt den Begriff. Usability ist wichtig, um die Bedienung sicher, d.h. gefahrenfrei, dazu effizient und somit produktivitätssteigernd aber auch einfach, um die Einarbeitungszeit zu verkürzen, zu machen.

- Hat man einen guten Look gefunden, bedeutet dies jedoch nicht zwangsläufig, dass das Feel auch gut ist. Die Übergänge der einzelnen Ansichten - die Animationen - müssen ebenfalls konzipiert, implementiert und auf Performanz getestet werden. Und erst die Kombination aus gutem Design, Usability und einer flüssigen Bedienung ermöglichen ein gutes Nutzererlebnis.

- Schließlich möchte das Management, der Vertrieb oder die Marketingabteilung schon während der Entwicklung frühzeitig ein Gefühl vom fertigen Endprodukt bekommen. Doch wie können Nicht-Entwickler sich schnell einen Überblick über den aktuellen Entwicklungsstand verschaffen, ohne dass erneut, zum Teil kostspielige Vorab-Prototypen aus dem Boden gestampft werden müssen?

Analog zum Software-Entwurf sollte auch beim Entwurf der graphischen Einheiten und Interaktionen, also dem User Experience Design, in Komponenten gedacht werden. Nur so kann eine Verzahnung beider, oftmals paralleler, Entwicklungen erfolgen. Altbekannte Entwicklungsmodelle wie das Wasserfall- oder das V-Modell könnten hierbei grundsätzlich auch zur Anwendung kommen, allerdings würde dann die Verzahnung in zu späten Stadien erfolgen. Von daher wird empfohlen, agile Entwicklungsmodelle, wie Scrum oder Kanban, sowohl für das Design- als auch das Software-Projekt

einzusetzen. Dadurch ist es möglich, schnellere Iterationen mit öfteren Feedback-Schleifen zwischen Designern, Software-Entwicklern und Produktverantwortlichen zu etablieren. Demzufolge sollte das verwendete GUI-Tool auch diese Entwicklungsmethoden unterstützen. Zum Beispiel könnte das das Bereitstellen des resultierenden GUIs ohne Abhängigkeiten zur Zielplattform, aus dem aktuellen Stand der Software, bedeuten. Dies kann in Form einer PC oder Mac-Anwendung sein oder als im Browser lauffähige Version.

Performance und Ressourcen

- Die gewohnte Bedienbarkeit von Smartphones und Tablets soll natürlich auch bei dem neu zu realisierenden HMI zur Anwendung kommen: Es soll ähnlich aussehen, soll sich ähnlich verhalten und soll selbstverständlich genauso schnell auf Benutzereingaben reagieren.
- Aber wie findet der Software-Ingenieur heraus, ob das implementierte GUI auch auf der Zielplattform performant läuft? Denn bereits im Prototyp stockte die eine oder andere Animation merklich.
- Anders als bei PC-Entwicklungen ist der zur Verfügung stehende Speicher bei Embedded Systems begrenzt. Folglich muss der Entwickler von Anfang an sowohl mit dem verfügbaren RAM- als auch dem ROM-Speicher gut haushalten.

Um Performance-Probleme in den Griff zu bekommen, ist es wichtig, die gesamte Kette zu beherrschen. Nur so kann ein möglicher Flaschenhals auch richtig identifiziert und lokalisiert werden: Sind die Performance-Einbußen der Applikation, dem GUI-Tool bzw. der GUI-Bibliothek oder dem Graphik-Treiber zuzuschreiben oder liegt eventuell sogar ein Hardware-Problem vor? So kann es durchaus von Vorteil sein, wenn das GUI-Tool in der Lage ist, unterschiedliche Graphic-APIs zu bedienen, um das Performance-Problem näher einkreisen zu können. Gilt es jedoch den Verbrauch von Flash-Speicher zu optimieren, sollte das GUI-Tool die Möglichkeit bieten, Ressourcen - wie Bilder und Schriften - sowohl statisch (zur Compile-Zeit) als auch dynamisch (zur Laufzeit) einzubinden. Um den RAM-Speicherverbrauch (nachträglich) zu minimieren gibt es die einfache Möglichkeit, das resultierende Farbformat zu reduzieren. Folglich ist es wünschenswert, wenn diesen Vorgang das verwendete GUI-Tool automatisch unterstützt.

Organisation

- Es kann durchaus sein, dass das HMI-Projekt bei Projektstart bezüglich der benötigten Entwickler-Ressourcen unterschätzt wurde. Um den Einführungstermin dennoch halten zu können, wurde beschlossen, weitere Entwickler für das Projekt abzustellen. Wie sieht dann eine effiziente Einarbeitung aus?
- Wenn das Team größer wird, ist es unausweichlich, dass der Source Code entsprechend verwaltet und versioniert wird. Können die eingesetzten Software Komponenten und Tools dies unterstützen?

Eine ordentliche Dokumentation innerhalb einer Software hilft die Struktur, das Verhalten und die gefundenen Lösungen auch über Projektgrenzen hinweg festzuhalten. Aber auch bereits während des Projekts kann sie neuen Mitarbeitern eine gute Startbasis bieten, um sich schnell in die Architektur

einzuarbeiten. Das GUI-Tool sollte folglich auch Dokumentationsmöglichkeiten (sog. inline-Dokumentation) für neu entstandene Software bieten, so dass eine zusätzliche, evtl. mit einem anderen Tool erstellte, Beschreibung nicht notwendig ist. In größeren UI-Projekten werden oftmals Prozesse etabliert, die eine kontinuierliche Integration (Continuous Integration) und dadurch eine kontinuierliche Lieferung (Continuous Delivery) an den Kunden ermöglichen. Demzufolge sollten GUI-Tools sich auch in diese automatisierten Build-Umgebungen integrieren lassen, um derartige Entwicklungsprozesse zu unterstützen.

Die oben aufgeführten Ursachen und Probleme lassen sich durch gezielte Maßnahmen - mit Hilfe von geeigneten GUI-Tools und -Bibliotheken - in den Griff bekommen. Die nachfolgenden Ratschläge sollen darüber hinaus helfen, die Umsetzung von GUIs für Embedded Systems zu erleichtern:

1) Plattformunabhängigkeit

Wie oben bereits erwähnt sollte das GUI-Tool auf einer Zwischenschicht aufsetzen. Diese abstrahiert die Schnittstellen zur Hardware, zum verwendeten Betriebssystem, zur benötigten Auflösung und zum gewählten Farbformat sowie zum eigentlichen graphischen Subsystem. So lässt sich die Hardware einfach austauschen, indem nur die Abstraktionsschicht auf die neue Plattform portiert wird.

2) Trennung von Middleware und GUI

Eine klare Trennung der Geschäftslogik von der graphischen Benutzerschnittstelle hilft, die Fachlichkeiten besser zu strukturieren. Dadurch ist es zum einen möglich, parallele Entwicklungen zu betreiben. Zum anderen können durch geeignete APIs mehrere, unterschiedliche GUIs auf die gleiche Instanz der Geschäftslogik (Middleware) zugreifen. Dies wird zum Beispiel bei der Implementierung von sog. Companion-Screens interessant, wenn es darum geht, zur eigentlichen, stationären Bedieneinheit eine weitere, mobile Anzeige hinzuzufügen.

3) Nutzen von bekannten Best-Practices: OOP mit MVC und MVVM

Da in der Regel GUI-Projekte durchaus schnell komplex werden können, ist es empfehlenswert, ein Objekt-orientiertes Programmier-Paradigma zu wählen: Durch Kapselung, Wiederverwendbarkeit von Code und Vererbungen lässt sich ein GUI-Projekt in einzelne Teile zerlegen und macht damit das Projekt beherrschbarer. Des Weiteren können gängige Entwicklungsmuster - wie das Model-View-Controller- oder das Model-View-ViewModel-Muster - helfen, eine klare Struktur in dem Projekt zu schaffen. Demzufolge sollte auch das verwendete GUI-Tool solche Entwurfsmuster von sich aus unterstützen.

4) Asynchrone statt synchrone APIs verwenden

Sobald das GUI mit der Middleware verknüpft wird, passiert es oft, dass nach Benutzereingaben Animationen nicht mehr flüssig dargestellt werden oder sogar stocken und keine weiteren Eingaben mehr möglich sind. Dies liegt häufig daran, dass die Benutzereingaben synchron mit der Verarbeitung innerhalb der Middleware gekoppelt sind: Das GUI wartet, bis die Middleware die Eingabe verarbeitet hat und kann keine weiteren Aktionen mehr durchführen - es blockiert. Um dies zu vermeiden sollten alle rechenintensiven Aufrufe der Middleware asynchron, also ohne aktives Warten der GUI, ausgeführt werden. Die Ergebnisse des Aufrufs werden dann zu einem späteren Zeitpunkt dem GUI wieder zurückgeführt. Dies ist vor allem bei Animationen im GUI unumgänglich.

5) User Experience

Um eine gute User Experience, die Kombination aus ansprechendem Design und guter Bedienbarkeit zu erreichen, bedarf es eine Menge gedanklicher Vorarbeit: Damit die einzelnen Ansichten leicht zu erfassen sind und die touchbaren Elemente groß genug sein können, müssen die Funktionen der Benutzeroberfläche auf mehrere Ansichten aufgeteilt werden. Diese Aufteilung richtet sich nach Anwendungszenarien und logischer Zusammengehörigkeit der Funktionen, wobei jeder Screen übersichtlich und in sich logisch gegliedert sein muss. Die einzelnen Elemente müssen dann nach Wichtigkeit richtig gewichtet werden. Dies geschieht durch Größe, Farbe und Form. Des Weiteren muss das GUI in sich konsistent sein: Die an einer Stelle im GUI gelernte Bedienung muss sich unbedingt an allen anderen ähnlichen Stellen gleich verhalten. Dafür ist es notwendig, das GUI-Konzept in Komponenten bzw. Templates zu strukturieren. Letzteres ist eine wichtige Vorarbeit für die objektorientierte Programmierung. Letztlich bedarf es eines gestalterischen Gespürs und Erfahrung um aus einer Kombination aus Farben, Linien, Symbolen und Schriften ein harmonisches und attraktives Ganzes werden zu lassen. Für diesen Bereich gibt es Spezialisten, die entsprechende Methoden für Anforderungserhebung, Konzeption und Design parat haben.

6) Instant-Prototyping

Wurde die Middleware vom eigentlichen UI getrennt und nutzt dieses eine Abstraktionsschicht zur Zielplattform, ist es mit wenig Aufwand möglich, Prototypen zu realisieren. Es ist hilfreich, wenn für die Prototyping-Plattform das Host-System (zum Beispiel der Windows PC) verwendet wird. So können schnell Vorabversionen der Benutzeroberfläche an Nicht-Entwickler wie Designer, Produktmanager oder Mitarbeiter der Marketing-Abteilung bereitgestellt werden. Änderungen im GUI können damit schneller kommuniziert und greifbar gemacht werden.

7) RAM Optimierung

Den Verbrauch von Arbeitsspeicher bei GUI-Applikationen bestimmen hauptsächlich die gewählte Auflösung, das verwendete Farbformat (True Color, High Color, Index8, etc.) und die Anzahl und Art der benutzten Ressourcen wie Schriftarten oder Bitmaps. Kann man die Auflösung nicht reduzieren und hat man bereits die Auswahl der Schriftarten und Bitmaps optimiert, so ist die Reduktion des Farbformats oft das Mittel der Wahl, um signifikant den Verbrauch von Arbeitsspeicher zu minimieren. Demzufolge sollte ein GUI-Tool bzw. eine GUI-Bibliothek die Möglichkeit bieten, das Farbformat ohne große Einbußen beim Design zu ändern.

8) ROM Optimierung

Ähnlich wie bei der Optimierung des RAM-Verbrauchs, bestimmen im Wesentlichen die verwendeten Ressourcen und der eigentliche Programmcode die Größe der resultierenden Applikation im Flash-Speicher. Da man am Programmcode bei einer guten Software-Architektur selten signifikante Änderungen durchführen kann, die einen Einfluss auf den ROM-Footprint haben, lohnt es sich, bei den verwendeten Ressourcen zu optimieren. Dabei ist es entscheidend, ob diese statisch (zur Compile-Zeit) oder dynamisch (zur Laufzeit) verwendet werden. So kann beispielsweise der Wechsel von der statischen zur dynamischen Verwendung große Auswirkungen haben. Sollen zum Beispiel nur wenige Schriftzeichen in einer bestimmten Größe dargestellt werden, sollte das verwendete GUI-Tool die

Option anbieten, auch nur diese Selektion statisch im Programmcode einzubetten. Damit muss nicht die ganze Schriftart eingebunden werden und der Speicherverbrauch wird gesenkt.

9) Performance-Steigerung

Gilt es nicht den Speicherverbrauch, sondern die Performance von Benutzeroberflächen zu verbessern, sollten zunächst die eigentlichen Zeichenoperationen optimiert werden. Bessere GUI-Tools sind deshalb zum Beispiel so ausgelegt, dass nur sich ändernde Bereiche, sog. Dirty Areas, zur Darstellung gebracht werden. Dies spart Rechenzeit und die Anzahl der Frames pro Sekunde (FPS) steigt. Eine weitere Möglichkeit, die FPS bei vollflächigen Animationen zu verbessern, besteht darin, ein intelligentes Caching zu nutzen: Einmal zusammengesetzte UI-Elemente wie Buttons, Sliders, Text-Views, etc. werden vor der Animation eingefroren, dupliziert und zu einem, sich nicht mehr änderbaren Abbild zusammengefügt. Die vollflächige Animation wird dann mit dem temporären Abbild durchgeführt und dieses danach wieder aus dem System entfernt. Dadurch muss die Animation nur noch auf einem zwischengespeicherten Objekt anstatt auf vielen kleineren Objekten durchgeführt werden. Das spart kostbare Rechenzeit, kostet aber zusätzlichen Speicher.

10) Integration

Während der Projektlaufzeit gibt es normalerweise mehrere Sitzungen, bei denen die neu entwickelten Software-Komponenten zusammengebracht und auf der Zielplattform integriert werden. Stellt man dabei fest, dass bestimmte Animationen oder Übergänge nicht zufriedenstellend dargestellt werden, ist es vorteilhaft, die gesamte Kette zu beherrschen. Folglich sollte man keine Unbekannte in seinem System vorfinden: Von der Applikation selbst über das GUI-Tool bzw. der GUI-Bibliothek bis hin zum graphischen Subsystem sollte der Entwickler Bescheid wissen.

Wer in dem Bereich der Benutzeroberflächenentwicklung ein Neuling ist und sich in der eingangs erwähnten Situation befindet, kann durch die Berücksichtigung der genannten Best Practices viel Zeit, Kosten und auch Nerven einsparen. Was am Anfang zunächst als Mehraufwand erscheint - genaue Konzeption, Architektur, Plattformunabhängigkeit, etc. - zahlt sich hinterher aus, da oftmals doppelte Entwicklungen vermieden werden können.

Natürlich erwarten Marketing und Management, dass der Prototyp schon in der ersten Version die gesamte Funktionspalette des zu bedienenden Gerätes abbildet. Jedoch sollte gerade zu Beginn der Funktionsumfang reduziert sein, damit verstärkt auf die Softwarequalität geachtet werden kann. Die genannten Aspekte können somit auch als Argumentationshilfe dienen, einen effektiven, vernünftigen und pragmatischen Weg einzuschlagen. So kann es gelingen, ein HMI-Projekt benutzerfreundlich, termingerecht und nachhaltig umzusetzen.

Further information and contact

Visit our website:
www.embedded-wizard.de
or write an email:
emwi@tara-systems.de

TARA Systems GmbH
Gmunder Str. 53
81379 München
Phone: +49 89 74 71 21-0