MICRO CONSULT

Das Rad nicht immer neu erfinden: Architekturmuster im Embedded-Umfeld erfolgreich einsetzen

Embedded-Projekte starten meistens sehr klein. Und oft wird im Anfangsstadium kein Gedanke an die Software-Architektur verschwendet. Die Anwendung soll bald funktionieren – und die Probleme mit der neuen Hardware sind ja auch noch zu lösen.

Doch solch ein Vorgehen rächt sich schnell. Spätestens bei der nächsten Kundenanforderung merkt man, dass ein bisschen Struktur in der Applikation die gewünschten Änderungen erreicht hätte – doch ist keine Zeit mehr dafür, denn der Kunde wartet. Das wiederholt sich dann noch ein paar Mal, und jedes Mal wird der Aufwand größer.

Das nächste Projekt wird ähnlich gestartet, und keiner hat etwas dazugelernt. Dabei ist es gar nicht schwer, ein paar solide Eckpfeiler in ein Embedded-Projekt einzubauen. Man muss nicht gleich alles neu erfinden. Die meisten Probleme sind nicht neu und wurden bereits von anderen Entwickler gelöst.

Ein paar kluge Leute haben sogar Vorlagen (Muster) entwickelt, die Sie nur noch an das eigene Projekt anpassen müssen. Solche Muster werden neudeutsch Pattern genannt.

Patterns sind in der Programmierung weit verbreitet; dort heißen sie Design Patterns. Dass es solche Muster auf für die Software-Architektur gibt, ist vielen Embedded-Entwicklern unbekannt. Wir stellen nachfolgend drei Architekturmuster vor, die für Embedded-Projekte gut einsetzbar sind.

Das Blackboard als zentrale Austauschplattform (Schwarzes Brett)

Das Blackboard-Architekturmuster kommt häufig in Expertensystemen zum Einsatz. Es lässt sich in vereinfachter Form gut in Embedded-Systemen einsetzen, um dem Chaos unzähliger globaler Variablen zu entgehen. Ausgangspunkt: Viele Komponenten eines Programmes benötigen Zugriff auf die gleichen Daten, stehen aber miteinander nicht in direktem Kontakt.

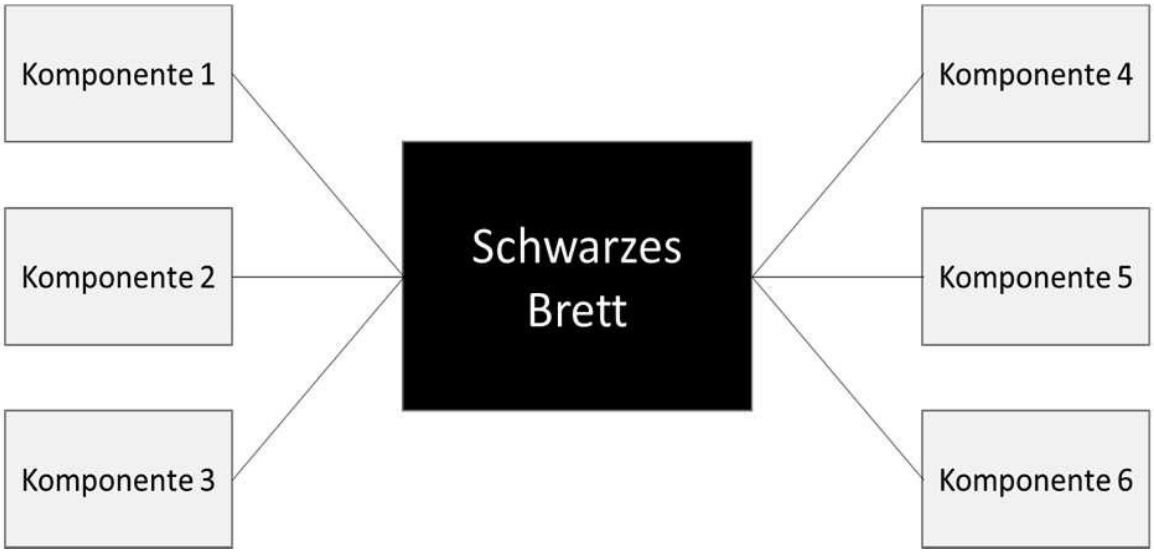

Bild 1: Prinzip Blackboard-Architektur

Das Schwarze Brett wird als zentrale Austauschplattform für gemeinsame Daten etabliert. Damit steht der Applikation eine einheitliche Datenablage zur Verfügung. Und es ist möglich, Änderungsmitteilungen an Datennutzer zu versenden oder auch Werte zu validieren und so zu vermeiden, dass Daten wild geändert werden.

Vorteile

- Einheitlicher Mechanismus für den Zugriff auf zentrale Daten
- Fehlerhafte Zugriffe auf die Daten sind einfach zu erkennen
- Keine direkten, unkontrollierten Zugriffe auf die Daten
- Änderungen sind einfacher durchzuführen

Nachteile

- Je nach Komfortgrad der Implementierung dauert der Zugriff auf ein Datum länger

Schichtenarchitektur

Die Schichtenarchitektur unterteilt die Applikation in Schichten unterschiedlicher (aufsteigender) Abstraktion. Dabei ist immer nur die obere Schicht von der unteren abhängig und nicht umgekehrt. Datenflüsse von unten nach oben werden durch Callback-Mechanismen realisiert; dies vermeidet zirkuläre Abhängikeiten.

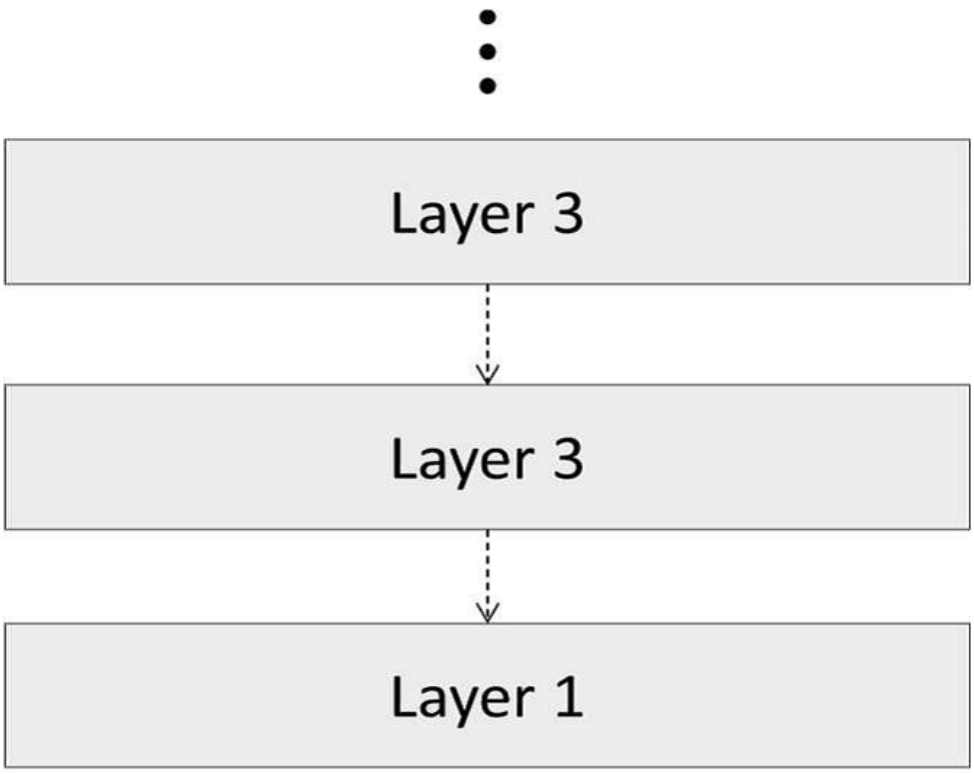

Bild 2: Schichtenarchitektur

Vorteile

- Reduzierung von Abhängigkeiten
- Definierte Schnittstellen zwischen den Schichten
- Änderungen wirken sich nur selten auf andere Schichten aus
- Wiederverwendung und Austausch einer Schicht sind möglich

Nachteile

- Eventuell geringere Effizienz aufgrund von Datentransformationen beim Durchreichen in höhere Schichten

Bekannter Vertreter der Schichtenarchitektur ist das OSI-7-Schichtenmodell.

Hohe Flexibilität durch Pipes und Filter

Dieses Muster eignet sich für Systeme, in denen Daten aufbereitet und verarbeitet werden müssen, wie bei der Verarbeitung von Sensordaten. Dabei wird die Aufbereitung der Daten in einzelne Teilschritte zerlegt und über Pipes miteinander verknüpft. Eine Pipe dient lediglich dem Datentransport (evtl. mit Zwischenpufferung); ein Filter bereitet Daten auf und übergibt sie einer Pipe.

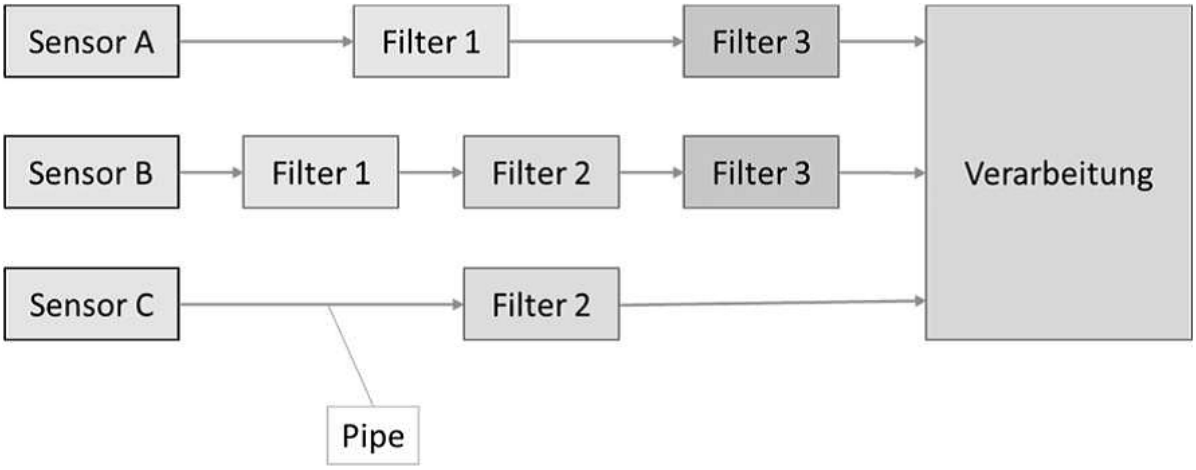

Bild 3: Pipes und Filter

Vorteile

- Hohe Flexibilität – Filter können ohne großen Aufwand ausgetauscht werden
- Wiederverwendung einzelner Filter

Nachteile

- Eventuell Effizienzverluste durch Datentransformation in das Datenformat der Pipe

Ereignisgesteuertes System

Ereignisgesteuerte Systeme lassen sich gut in Anwendungen integrieren, in denen viele Ereignisse in nicht vorbestimmter Reihenfolge auftreten. Alle Ereignisse werden von einem Event-Manager entgegengenommen. Dieser leitet ein Ereignis dann an die Komponenten weiter, die sich bei ihm für dieses Ereignis registriert haben.

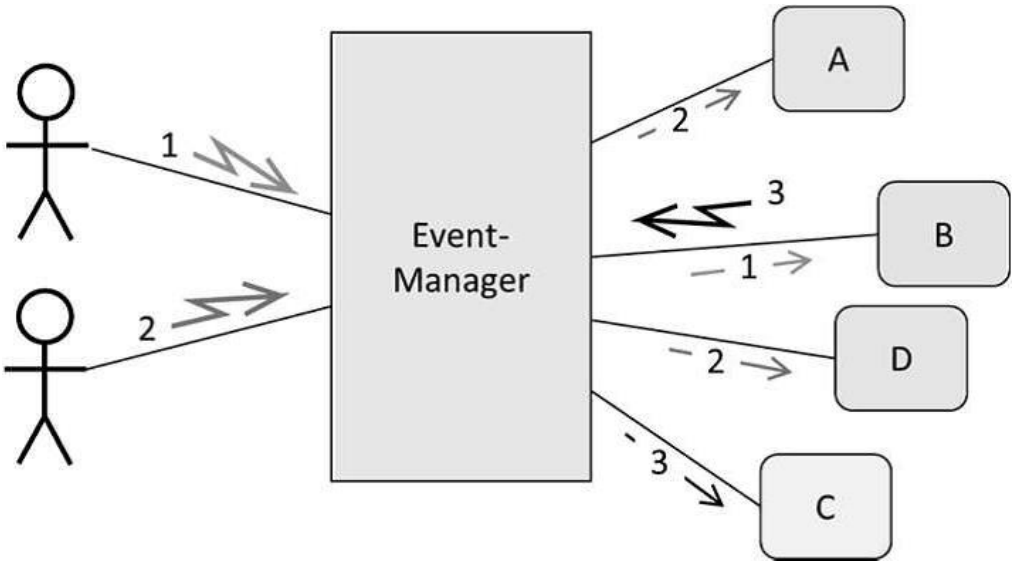

Bild 4: Ereignisgesteuertes System

Vorteile

- Starke Entkopplung der Komponenten
- Eine Komponente ist leicht austauschbar

Nachteile

- Keine Garantie, dass ein Ereignis bearbeitet wird
- Falls sich mehrere Komponenten für ein Ereignis registriert haben, ist die Reihenfolge der Ausführung nicht bekannt

Fazit

Es gibt keinen Grund, auf eine gute Software-Architektur zu verzichten. Abhängig vom Projekt wird eine Architekturvorlage oder eine Kombination aus mehreren ausgewählt und angepasst. Die (kleine) Mühe amortisiert sich sehr schnell. Änderungen werden einfacher (der Kunde hat immer noch eine Idee kurz vor der Auslieferung), und auch Test und Fehlersuche werden beschleunigt.

Autor: Frank Listing

Seit 2002 ist Frank Listing Trainer und Projektcoach bei MicroConsult; seine Schwerpunkte sind Microsoft-Plattformen, Software-Architekturen, objektorientierte Programmierung und Testen von Embedded-Systemen. Außerdem ist er Spezialist für die Themen C++, C#, Finite State Machines, Clean Code und .NET. Sein Wissen gibt er regelmäßig in Publikationen und Flachvorträgen z.B. auf dem Embedded Software Engineering (ESE) Kongress weiter.

Weiterführende Informationen

MicroConsult Training & Coaching zum Thema Softwarequalität

MicroConsult Fachwissen zum Thema Softwarequalität

MicroConsult Training & Coaching zu Embedded-Programmierung

MicroConsult Fachwissen Embedded-Softwareentwicklung

Quellen

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, “A System of Patterns – Pattern Oriented Software Architecture“, Wiley

Bruce Powel Douglass, “Real Time Design Patterns: Robust Scalable Architecture for Real-time Systems”, Addison-Wesley