Embedded Software Engineering Kongress

ESE Kongress 2013

Vortragsskript:

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

Frank Listing, MicroConsult GmbH

Da embedded Projekte sehr klein starten, wird dort oft kein Gedanke an die Software-Architektur verschwendet. Etwas soll schnell funktionieren – und die Probleme mit der neuen Hardware sind ja auch noch zu lösen. Solch ein Vorgehen rächt sich schnell. Spätestens bei der nächsten Kundenanforderung merkt man, dass ein bisschen Struktur in der Applikation die Änderungen erleichtert hätte – aber wir haben ja jetzt keine Zeit mehr, etwas zu ändern, denn der Kunde wartet. So geht es dann weiter - und jedes Mal wird der Aufwand immer größer.

Das nächste Projekt wird dann ähnlich gestartet, und keiner hat etwas dazu gelernt. Dabei ist es gar nicht schwer, ein paar solide Eckpfeiler in das Projekt einzubauen. Man muss nicht einmal alles neu erfinden. Diese Probleme sind alle nicht neu und wurden schon gelöst. Ein paar kluge Leute haben sogar Vorlagen (Muster) entwickelt, die nur noch an das eigene Projekt angepasst werden müssen. Diese Muster werden neudeutsch Pattern genannt.

Solche Patterns sind vielen schon von der Programmierung her geläufig; dort heißen sie Designpattern. Dass es solche Muster auch für die SW-Architektur gibt, ist leider vielen (embedded) Entwicklern unbekannt. Deshalb werden hier ein paar Architekturmuster vorgestellt, die auch für embedded Projekte interessant sind.

Muster

Blackboard (Schwarzes Brett)

Das Blackboard-Architekturmuster wird häufig in Expertensystemen eingesetzt. Es lässt sich aber in einfacher Form auch gut in embedded Systemen einsetzen, um dem Chaos vieler globaler Variablen zu entgehen. Ausgangspunkt: Viele Komponenten eines Programmes benötigen Zugriff auf die gleichen Daten, stehen aber nicht in direktem Kontakt.

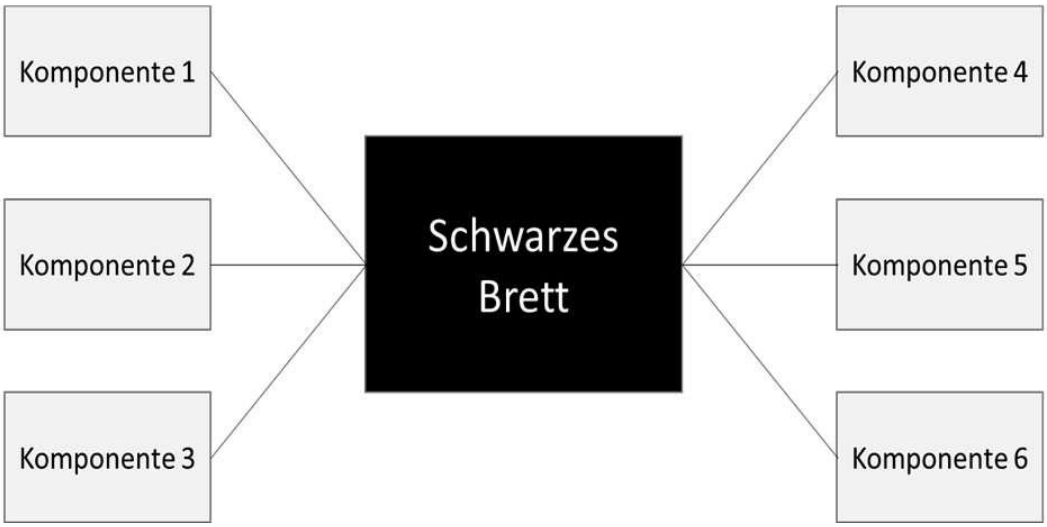

Bild 1: Prinzip Blackboard-Architektur

Das Schwarze Brett wird als zentrale Austauschplattform für gemeinsame Daten etabliert. Damit kann eine einheitliche Datenablage realisiert werden. Es ist möglich, Änderungsmitteilungen an Datennutzer zu versenden oder auch Werte zu validieren. Daten werden nicht wild geändert.

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 Abhängigkeiten.

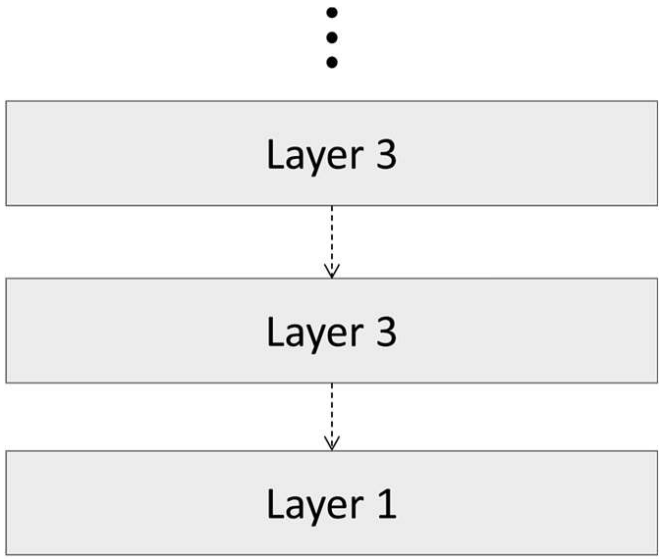

Bild 2: Schichtenarchitektur

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

Nachteile
- Eventuell Effizienzverringerung durch Datentransformationen beim Durchreichen in höhere Schichten

Bekannter Vertreter der Schichtenarchitektur ist das OSI-7-Schichten-Modell.

Pipes und Filter

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

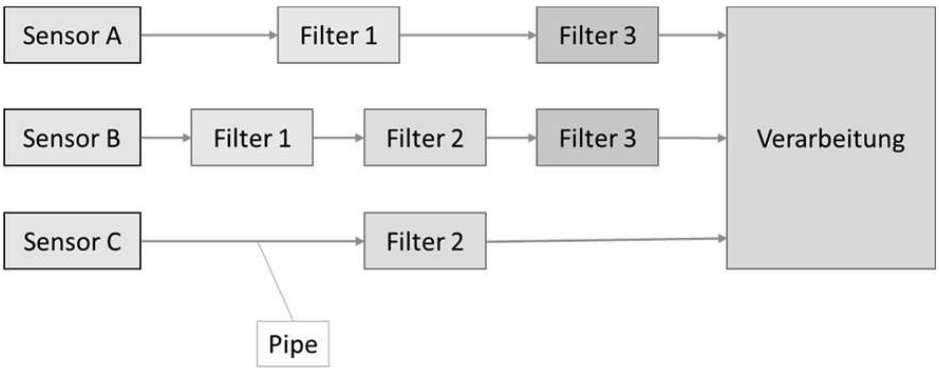

Bild 3: Pipes und Filter

Vorteile

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

Nachteile

- Eventuell Effizienzverluste durch Datentransformationen in das Datenformat der Pipe

Ereignisgesteuertes System

Ereignisgesteuerte Systeme lassen sich gut in Anwendungen verwenden, 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.

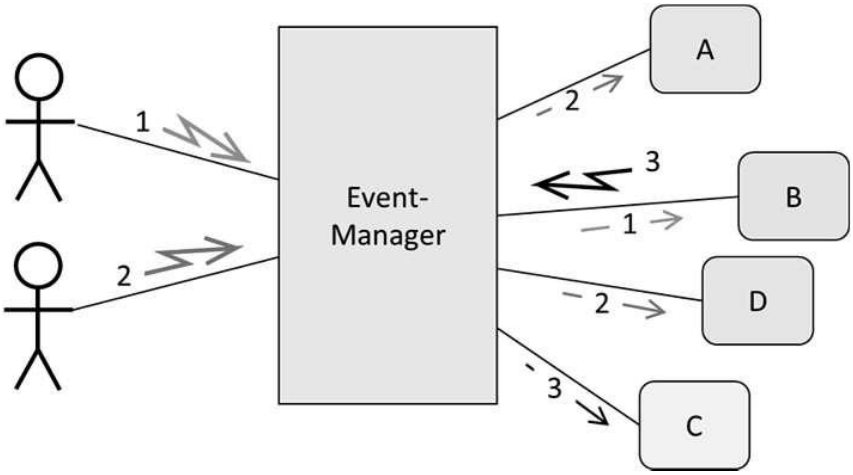

Bild 4: Ereignisgesteuertes System

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 SW-Architektur zu verzichten. Abhängig vom Projekt wird eine oder eine Kombination mehrerer Architekturvorlagen 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.

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

Autor

Dipl.-Ing. Frank Listing ist seit 2002 Trainer und Projektcoach bei der MicroConsult GmbH mit dem Schwerpunkt Microsoft-Plattformen, objektorientierte Programmierung und Testen von embedded Systemen und u.a. fachlich für das Thema .NET verantwortlich. Sein Wissen gibt er immer wieder auch in Publikationen und Fachvorträgen weiter.