Hierarchisches Degradationskonzept

In komplexen Systemen sind Funktionen von der Verfügbarkeit anderer Funktionen abhängig und können nur eingeschränkte oder gar keine Funktionalität anbieten, wenn die abhängenden Funktionen selbst degradiert sind.

Dieses Paper soll verschiedene Verfahren darlegen und ein beispielhaftes Verfahren inklusive Implementierung aufzeigen über den Funktionen ohne zentrale Instanz sinnvoll degradieren, sich aber auch wieder heilen können.

Dieses Dokument ist auf Deutsch geschrieben, um die Gedanken schneller und besser fassen zu können.

# Begriffsdefinition („English“)

**Fehlerursache** („Failure cause“) – Ein physikalisches Ereignis, das einen Fehler in einem Bauteil verursacht hat. Dies ist die kleinste Fehlergranularität, die betrachtet wird.

**Mögliche Fehlerursachenmenge** – Die Menge aller Fehlerursachen, die im Gesamtsystem auftreten können.

**Aufgetretene Fehlerursachen** – Die Menge aller Fehlerursachen, die im Gesamtsystem aufgetreten sind.

**Überwacher** („Monitor“) – Eine Funktion, der einen Fehler identifiziert und dessen Zustand im System verfügbar macht. Im Regelfall wird dieser Überwache in Software implementiert und prüft die Funktion entweder über direkte Entdeckungsmaßnahmen oder über Plausibilisierung mit Hilfe anderer Signale oder externer Regelsätze.

Die Unterscheidung, ob ein Fehlerstatus erkannt worden ist oder ob ein Monitor noch nicht erfolgreich abgeschlossen worden ist, ist wesentlich für die Ausführung sicherheitskritischer Funktionen, da diese von einer Bestätigung abhängig ist.

In der Realität ist die Wahrscheinlichkeit, dass ein Überwacher einen Fehler korrekt bestätigt oder gar ein korrektes System als fehlerhaft identifiziert nicht 0%, sondern abhängig von der Robustheit der Überwacher-Auslegung,

**Degradation** („Degradation“) – Eine bewusste und kontrollierte Einschränkung der Funktion die bis zu einem Totalverlust reichen kann. Eine Funktion ist nicht degradiert, wenn diese voll verfügbar ist. Kann eine Funktion nur mit reduzierter Leistung, reduzierter Redundanz oder gar überhaupt nicht ausgeführt werden, so gilt sie als degradiert. Eine Degradation zählt nicht als Fehler, daher ist es notwendig, dass eine Fehlfunktion zuerst über einen Überwacher erkannt wird, da ansonsten keine kontrollierte Aktion stattfinden kann. Die Degradation ist eine Funktion über die Resultate aller Überwachter.

**Befähigung** („Capability“) – Über die Schnittstellendefinition garantiert eine Funktion gewisse Eigenschaften in Bezug auf Funktion, Redundanz, Leistung und Ausfallverhalten. Kann diese Funktion diese Befähigung nicht mehr aufrechterhalten, so gilt sie als degradiert. Jede Befähigung einer Funktion kann einzeln degradiert werden.

Im Regelfall werden Funktionen nicht nur direkt über ihre Überwacher abgeschaltet, sondern auch über den Degradationsstatus anderer Funktionen. Hierzu ist es nicht nötig die internen Fehler außenstehender Komponenten zu kennen. Die obige Funktion wird erweitert:

Die Degradation einer Funktion hängt von den durch den Überwacher erkannten Fehlerursachen und über den Degradations-Status anderer Funktionen ab.

Diese Funktion wird Regelsatz einer Degradation genannt.

**Fehler** – Ein Fehler ist eine Abweichung von einem angeforderten Verhalten und ist meist über eine ungewollte Veränderung der Ausgangssignale und -energien erkennbar. Fehler müssen im Regelfall erkannt werden, so dass davon betroffene Funktionen ihr Verhalten so ändern können, dass durch den Fehler kein inakzeptabler Betriebszustand erreicht wird. Ein inakzeptabler Betriebszustand wäre zum Beispiel der Verlust der körperlichen Integrität des Fahrers oder Dritter, aber auch ein finanzieller Schaden auf Grund eines Datenverlustes.

# Möglichkeiten

## Zentrale Degradationsinstanz

In bestehenden Systemen, insbesondere im Embedded Bereich innerhalb eines Steuergerätes gibt es häufig eine zentrale Instanz, die ­alle erkannten Fehler sammelt.

Abhängig von den Fehlern werden dann bestimmte Funktionen degradiert, die wiederum zu einer Degradation anderer Funktionen führen kann.

Jede Funktion enthält eine Verbindung zu dieser Degradationsinstanz und fragt ab, ob diese degradiert ist. In Abhängigkeit des Ergebnisses entscheidet die Funktion selbst, ob sie vollständig oder eingeschränkt ausgeführt werden soll.

Hierbei ist die gesamte Degradationslogik aus den Funktionen herausgenommen und es ist zu gewährleisten, dass der vollständige Regelsatz der Degradationsinstanz die Anforderungen der gesamten Funktionen reflektiert.

Die Degradation wird von außen definiert und der komplette Regelsatz einer jeden Degradation kann potentiell abhängig von allen Überwacher und anderen Degradationszuständen sein.

## Dezentrales, hierarchisches Konzept

In dem von mir favorisierten Konzept gibt es diese zentrale Degradationsinstanz nicht mehr.

Jede Funktion und jede Subfunktion definiert eine wohldefinierte Degradations-Schnittstelle, die dann von abhängigen Funktionen eingelesen werden kann. In Abhängigkeiten dieser Schnittstelle kann sich die Funktion dann selbst degradieren.

Hierzu sind eine klare Hierarchie-Ebene des Systems und eine Gliederung und Zuordnung möglicher Fehlerursachen und Überwacher in jedes System-Element erforderlich. Fehlerursachen und dessen Überwachen müssen lokal zueinander sein. Sind diese nicht lokal zueinander, so muss die Befähigungsinformation über einen Kanal übertragen werden. Die Schnittstelle eines jeden Elements ist eine Befähigungs-Schnittstelle, die von anderen Elementen wieder eingelesen werden kann.

### Das Element

Mit obiger Definition besitzt jedes Element, welches nun als Funktion definiert wird,

* Klar zugeordnete Fehlerursachen
* Überwacher, die diese erkennen
* Befähigungen und dessen Degradationen

Reine Sensorfunktionen sind befähigt physikalische Größen in reale Größen umzuwandeln. Hierzu fallen spontan folgende Degradationsmöglichkeiten ein:

* Signal ist ungenau
* Signal ist nicht verfügbar

Reine Aktorfunktionen sind befähigt einen Aktor gemäß den geforderten Stellgrößen zu stellen. Hierzu fallen spontan folgende Degradationsmöglichkeiten ein:

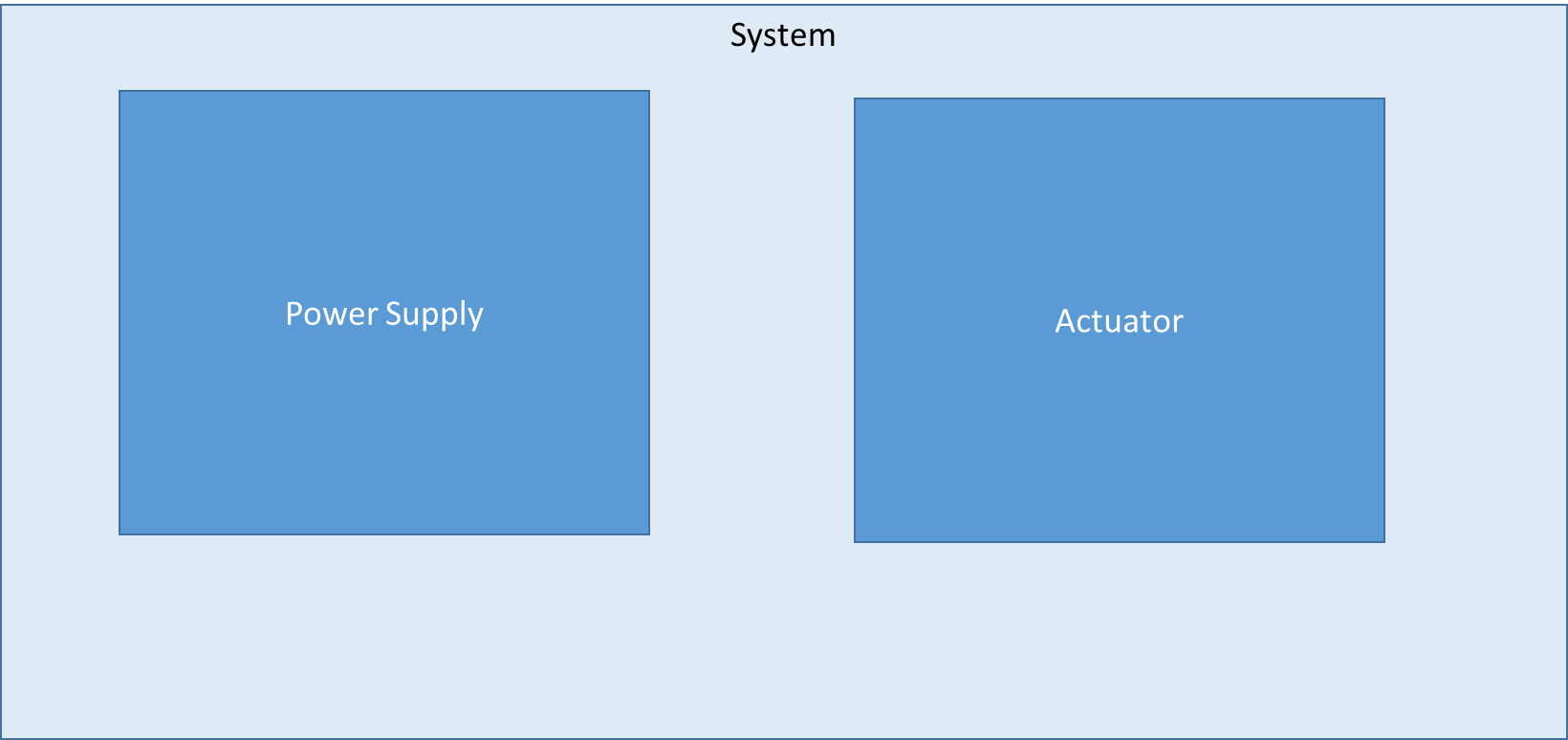
* Aktor ist ausgefallen
* Aktor besitzt einen eingeschränkten Operationsbereich

Das integrierende System gibt die benötigten Fähigkeiten und dessen Degradationszustände über einen Top-Down Entwicklungsprozess vor. Sollen existierende Subsysteme mit dessen schon vorhandenen Befähigungen integriert werden, so ist ein Schnittstellenabgleich durchzuführen.

Wie die Befähigungen gebildet werden und welche Fehlerursachen maßgeblich damit verbunden sind, ist für das integrierende System ohne Relevanz.

### Beispiel:

Sei ein System aus Stromversorgung und Aktor gegeben. Das System hat die Funktionen einen Aktor in regelmäßigen Abständen von links nach rechts zu bewegen, es sind daher keine Eingangsinformationen notwendig. Die Stromversorgung gewährleistet die Energieversorgung für den Aktor.



Die Befähigung des Gesamtsystems ist über Cap\_Bewegen definiert.

Die Degradation dieser Befähigung für Cap\_Bewegen lauten:

* OK (OK)
* Eingeschränkte Geschwindigkeit (Limited)
* Komplett ausgefallen (NOK)

Ein äußeres System kann nun die Befähigung einlesen und nun selbst Degradationen durchführen ohne den internen Betriebszustand des Systems zu kennen. Beispielhaft kann ein Bediener eine Warninformation erhalten, wenn die volle Funktionsfähigkeit des Systems nicht gegeben ist.

Das System besteht nun aus den zwei genannten Einzelelementen:

Die Stromversorgung besitzt folgende Degradationen für seine Hauptfunktion der Energielieferung:

* OK
* Eingeschränkter Strom (Limited)
* Komplett ausgefallen (Es sei hier zu vernachlässigen, dass in diesem Falle der Prozess vollständig zum Erliegen kommt, hierzu später ein Beispiel).

Hierzu gibt es einen Überwacher „Power Supply Monitoring“ und „High Current Power Supply Monitoring“.

Der Aktor besitzt folgende Befähigung für seine Funktion der Bewegung:

* OK
* Eingeschränkte Geschwindigkeit
* Komplett ausgefallen

Er ist abhängig von der Befähigung der Stromversorgung des eigenen Überwachers „Actuator Monitoring“.

Cap\_Actuator = OK, wenn Cap\_PowerSupply == OK & „Actuator Monitoring” == OK

Cap\_Actuator = Limited, wenn Cap\_PowerSupply == Limited & “Actuator Monitoring” == OK

Cap\_Actuator = Failed, wenn Cap\_PowerSupply == NOK || “Actuator Monitoring” == NOK

In der Implementierung würde man den Status „OK“ standardmäßig setzen und erst im Falle des Zutreffens einer Fehlerbedingung die Befähigung heruntersetzen.

Das System wird in seinem Außenverhalten nur durch den Aktor beschrieben. Daher ist diese nur abhängig von der Befähigung des Aktors. Mit diesem Konzept ist der Aktor von den inneren Eigenschaften der Power Supply getrennt, aber auch der Power Supply hat keinerlei Informationen über den Aktor. Es ist allerdings wichtig, dass die Schnittstelle zwischen Aktor und Stromversorgung abgeglichen sind. Sollen zum Beispiel weitere Aktoren oder Energiekonsumenten dem System zugefügt werden, so ist zu prüfen, ob die Degradationen der Befähigungen in seinen Eigenschaften und Parametern stimmen.

Der Aktor fordert Befähigungen der Power Supply an, prüft diese und ermittelt seine eigenen Befähigungen aus diesen. Hierbei ist eine hohe Informationskapselung und damit auch die Umsetzung der Systemeigenschaften zum jeweiligen Systemelement zugeordnet.

Das System ist in seiner Schnittstelle ebenso unabhängig von den inneren Eigenschaften!

# Implementatorische Grundlagen

Systeme und dessen Elemente werden häufiger die Befähigungen der abhängigen Elemente abfragen und es wird erwartet, dass ein Wechsel der Degradationszustände ein seltenes Ereignis ist.

Daher soll der Degradationsstatus der Befähigungen lokal im Speicher vorliegen, während eine Änderung eines Überwacher-Resultats bzw. die Änderung einer Befähigung sofort durch alle abhängigen Funktionen durchgeleitet werden sollen.

Als erste Testimplementierung wird die Programmiersprache C# genutzt und es werden zusätzlich auch Anforderungen auf den Heap durchgeführt. Als Zielimplementierung soll eine nackte C-Implementierung ohne Zugriff auf dynamischen Speicher erfolgen.