DegradationMeister

1 Purpose of this document

The document gives an overview about the concept, the ideas and the implementation of a generic Degradation-Framework which can be used to build up fault-tolerant systems in which the capability of functions are dependent on failures and the capability of other functions.

2 Terminology

Function – A function is a well-structured element within a system that has an input/output interface.

Capability – Defines the state describing whether the function is capable to fulfill the task as defined. A function can have several capabilities

Failure – A failure will lead to a state that will lead to a deviation of the intended function in regards to the actual function. There a detectable and non-detectable failures

Failure Cause – A physical event within a physical system or part that leads to a failure.

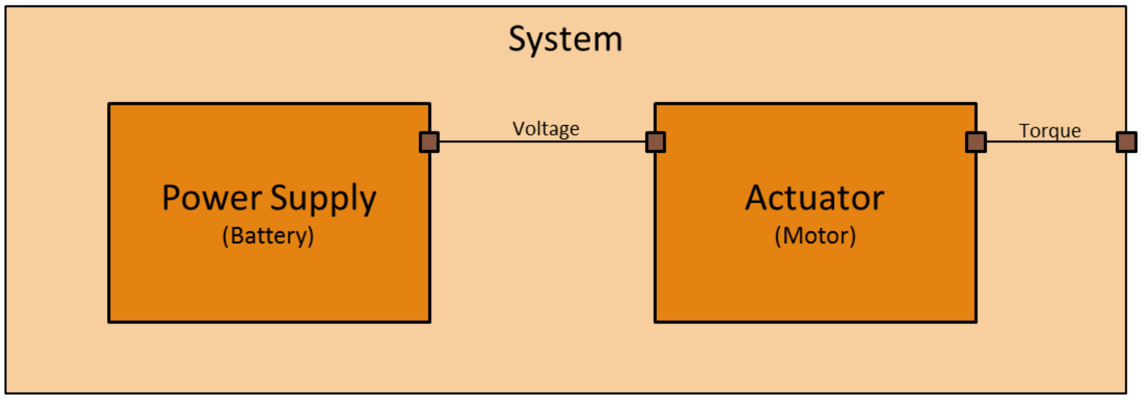
Monitor – A monitor has the task to detect a failure and report the result of the failure evaluation.

Degrader – The instance that is capable to connect the failures to capabilities and to connect capabilities to other capabilities.

3 Concept

On the example of a simple power-supply, actuator system, the concept of the DegradationMeister will be explained.

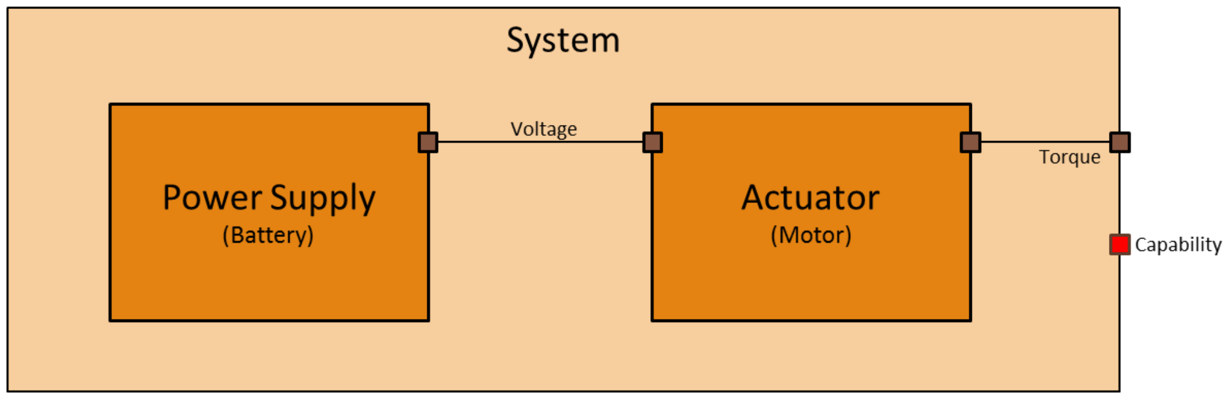
Let's consider the following system:



Picture 1 - Example system

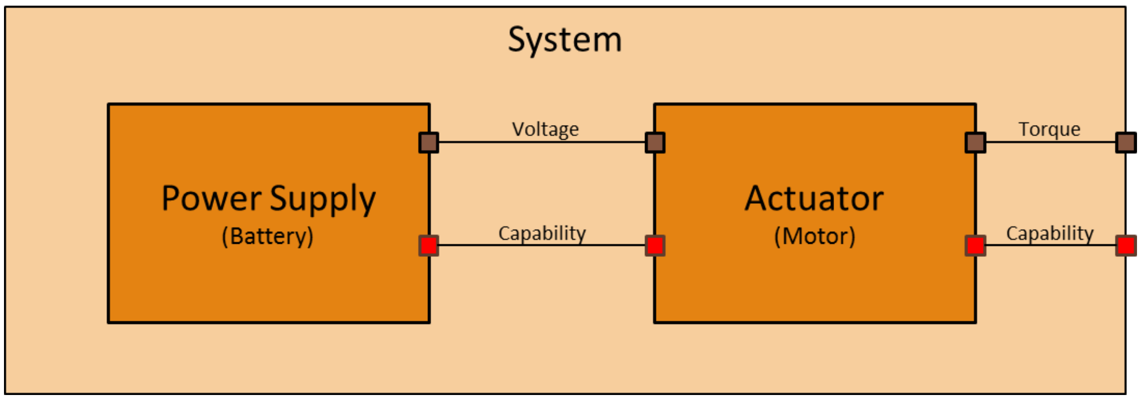
An internal power supply provides current to an actuator which converts the voltage and the resulting current into a torque which is supplied as an external interface to the user of the system.

A user of the system requests an additional information whether the system is capable to provide the torque. So, an addition of the system concept is necessary:



Picture 2 - Example system with capability information

The system has full capability, if the actuator has full capability. The actuator has full capability, if the power supply has full capability. To provide the additional information about the capability of the system, each subsystem needs to estimate its own capability. This leads to the following architecture:

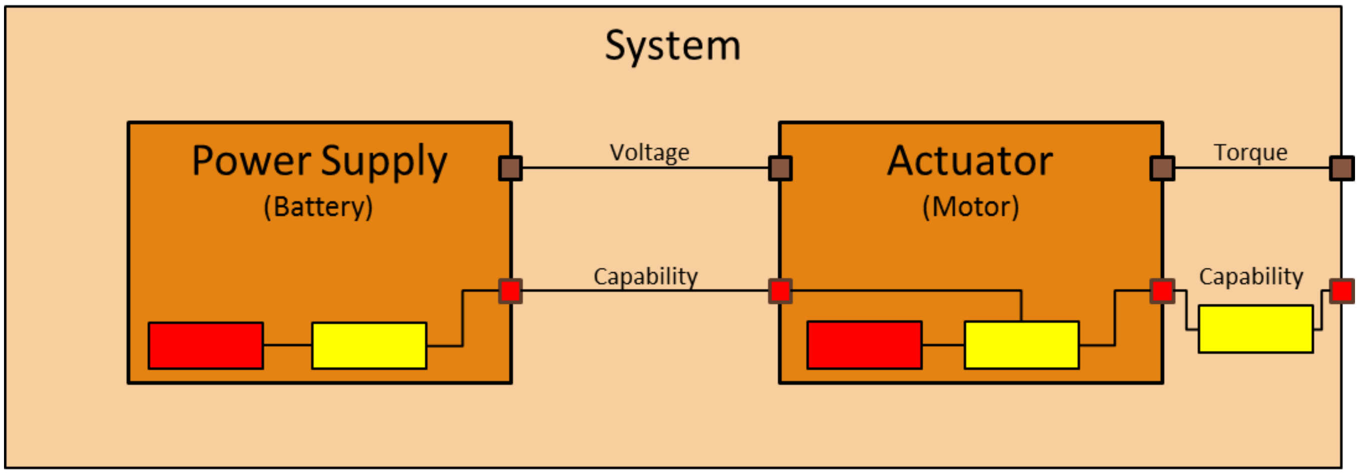


Picture 3 - Degradation via capability

To estimate whether the power supply and actuator has full capability, each subsystem includes a monitor which is capable to verify whether it has full capability in regards to its function.

- The power supply needs to have the capability to provide voltage.
- The actuator needs to have the capability to provide torque.
- The system needs to have the capability to provide torque, which is completely dependent on the actuator.

The estimation will be done by internal monitors which detect failures and are mapped through the degrader to the capability.



🟥	Monitor
🟨	Degrader

Picture 4 - Degradation via degrader

Each degradation of the capability of power supply propagates through the actuator to the system capability. The full implementation of this concept is given in the Github-Project: https://github.com/mbrenn/degradationmaster

The example above is shown within the file: https://github.com/mbrenn/degradationmaster/blob/develop/src/DegradationMeisterTest/Tests.cs

```
var degraderPowerSupply = new Degrader("Power Supply");
var monitorPowerSupply = new PowerSupplyMonitor(degraderPowerSupply);
_capabilityPowerSupply = new Capability(degraderPowerSupply, "Power Supply");

var degraderActuator = new Degrader("Actuator");
var monitorActuator = new ActuatorMonitor(degraderActuator);
_capabilityActuator = new Capability(degraderActuator, "Actuator");

var degraderSystem = new Degrader("System");
_capabilitySystem = new Capability(degraderSystem, "System");

degraderPowerSupply.AddDefaultRules(monitorPowerSupply.TotalFailure, _capabilityPowerSupply);
degraderPowerSupply.AddRuleForUnknown(monitorPowerSupply.LimitedCurrent, _capabilityPowerSupply);
degraderPowerSupply.AddRule(monitorPowerSupply.LimitedCurrent,
    MonitoringResult.NOK,
    _capabilityPowerSupply,
    Capabilities.Limited);


degraderActuator.AddDefaultRules(monitorActuator.TotalFailure, _capabilityActuator);
degraderActuator.AddDefaultRules(_capabilityPowerSupply, _capabilityActuator);
degraderActuator.AddRule(_capabilityPowerSupply, Capabilities.Limited, _capabilityActuator,
    Capabilities.Limited);

degraderSystem.AddRule(_capabilityActuator, Capabilities.Failed, _capabilitySystem, Capabilities.Failed);
degraderSystem.AddRule(_capabilityActuator, Capabilities.Limited, _capabilitySystem, Capabilities.Limited);

degraderSystem.AddTrigger(_capabilitySystem,
    x => Console.WriteLine($"- System: {Capabilities.Convert(x.Current)}"));
```

As you can see in the sourcecode above, the power supply also includes a failure, leading to a limited power supply. This will result in a limited actuator behavior and therefor to a limited system behavior. By changing the rules, it would also be possible to assess the system as failed in case of a limited power supply.

3.1 MONITOR

Each monitor can contain several failures and is assigned to a specific degrader. Whenever the monitor detects that a failure occurred, the degrader is informed about the change.

The degrader will then update all dependent capabilities and recursively its dependencies.

Each failure can have one of the following failure states:

- Unknown: It is not known whether the failure has occurred
- OK: It is known, that the failure is currently not occurring
- NOK: It is known, that the failure is currently occurring

3.2 CAPABILITY

Each capability has a certain degradation. There are the following default degradations:

- Unknown
- Full
- Failed
- Limited

If a certain capability needs additional degradation, these could be easily added since the framework is using a regular integer number as the state for degradation.

3.3 DEGRADER

The degrader will be triggered by the monitor each time a dependent failure or capability has been changed. It will then update all dependent capabilities and inform other degraders, if there is a dependency to their associated capabilities.

The generic form for a general rule looks like the following ones:

If no rule is applicable to a capability, the capability will be set to 'full'. The first matching rule for each capability will be chosen in the current version of DegradationMeister.

3.3.1 Failure Rule

If the monitor has estimated a certain failure state *FS* for a failure *F*, the associated capability *C* will be set to degradation *DEG*.

3.3.2 Capability Rule

If the degrader has estimated that a certain capability *CInput* has reached a degradation DEGInput, the associated capability C will be set to degradation DEG.

4 CONCLUSION

By using the DegraderMeister, you have a predeveloped degradation and capability manager which works on failures and resolves the dependencies between capabilities and failures.

The application is currently based on .Net 4.6 and is planned to be rebased to .Net Standard 1.6.