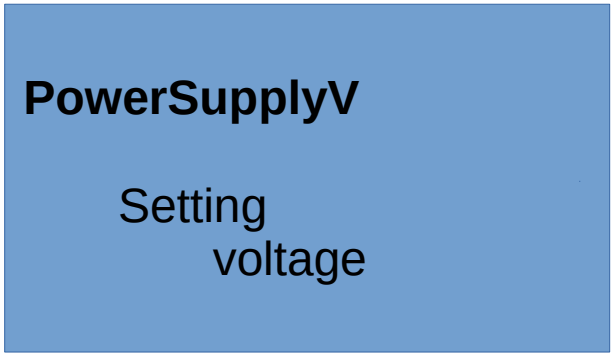
Konzept für FESA

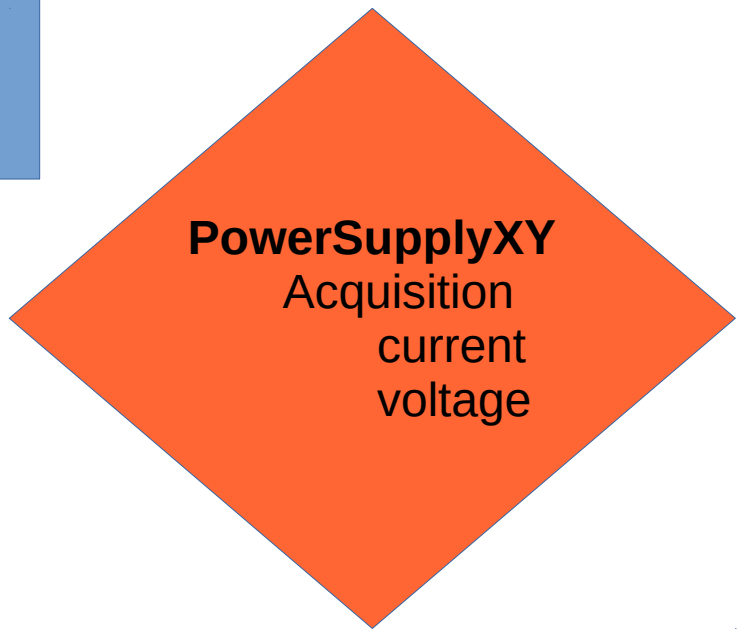
Vereinheitlichung Magnetnetzgeräte Software

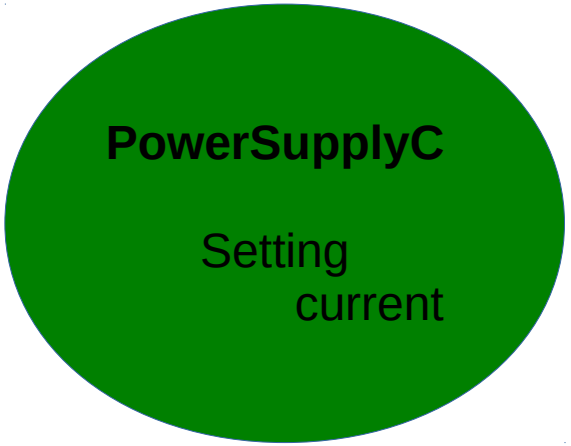
Die Gründe

- Magnetnetzgeräte unterscheiden sich tw. nur in minimalen Details
- FESA Software ist dann größtenteils identisch bis auf wenige Ausnahmen
- Software-Beispiel: Setting-Property enthält in einem Fall ein Feld für Spannung, im anderen Fall ein Feld für Stromstärke

Beispiele







- In FESA würde das jedes Mal eine eigene FESA Klasse bedeuten die nur eine leicht anders gestaltete Setting-Property enthält (=viel Klick- / Programmier- und langfristiger Pflegeaufwand)

Die Idee

- Nur eine einzelne Klasse „PowerSupply“ programmieren, die **alle jemals auftretenden Properties samt aller möglichen Werte** etc enthält
- Daraus „Subsets“ wie z.B. „PowerSupplyV“ ableiten, die eine definierbare Teilmenge der einzelnen Klasse PowerSupply enthält
- Diese Subsets dienen nur als *Interface nach oben* (als Information für die Anwendungen)
- Implementierung nur in der einzelnen Klasse „PowerSupply“, hier findet je nach Konfiguration in Instanziierung die unterschiedliche Behandlung der unterschiedlichen Typen statt

  → Magnetnetzgeräte Software-Entwickler muß wissen was er tut

FESA Klassendesign

- Zuordnung von im Subset verwendeten value-items pro GSI-Setting-Property

GSI-Setting-Property	Setting
multiplexed	false
name	Setting
visibility	operational
value-item	(currentValueItem: direction=INOUT)
value-item	(voltageValueItem: direction=INOUT)
value-item	(set: direction=INOUT)
update-flag-item	(updateFlags: direction=OUT, optional=true)
cycle-name-item	(cycleName: direction=OUT, optional=true)
set-action	(partial-setting=true, transaction=true)
get-action	
GSI-subset-definition	PowerSupplyC
name	PowerSupplyC
valid-value-items	
value-item-name	(value-item-name-ref=currentValueItem)
value-item-name-ref	currentValueItem
value-item-name	(value-item-name-ref=voltageValueItem)
GSI-subset-definition	PowerSupplyV
name	PowerSupplyV
valid-value-items	
value-item-name	(value-item-name-ref=voltageValueItem)

FESA Klassendesign

- Definition verfügbarer Subsets als Konfigurationselement

data	detailedStatus_labels, detailedStatus_severity, subset
device-data	detailedStatus_labels, detailedStatus_severity, subset
configuration	detailedStatus_labels, detailedStatus_severity, subset
GSI-detailed-status-labels-field	detailedStatus_labels, type=char
GSI-detailed-status-severity-field	detailedStatus_severity, ref=DETAILED_STATUS_SEVE
GSI-available-subsets-definition	subset, type=char [3, 13]
name	subset
array2D	(type=char)
type	char
dim1	3
dim2	13
default	{PowerSupplyC, PowerSupplyV, PowerSupplyX}

FESA Instanziierung

classes	anInstance, GDEB1B2134
PS	anInstance, GDEB1B2134
rolling-buffer	(depth=3)
device-instance	(anInstance: state=development)
name	anInstance
state	development
configuration	
description	(value=)
timingDomain	(value=NONE)
accelerator	(value=NONE)
acceleratorZone	(value=NONE)
mainMuxCriterion	(value=NONE)
detailedStatus_labels	
detailedStatus_severity	
subset	
value	{PowerSupplyC}

Implementierung in FESA Klasse

```
void SettingSetAction::execute(fesa::RequestEvent* pEvt,
Device* pDev, SettingPropertyData& data) {

  ...

  boost::shared_ptr<DeviceInstantiationData> instData     =
   device->getInstantiationData();

    std::string subset = instData

      ->getInstantiationXMLElement()

      ->getFieldElement("subset")->getDefaultValue();

     ...
```

- Nun wäre String-Vergleich der jeweiligen Konfiguration möglich zur Unterscheidung im Quelltext

Umsetzung in FESA

- Möglichkeit zur Definition von verfügbaren Subsets einer Klasse im Klassen-Metamodel (FESA Metamodel: Design)
- Möglichkeit zum Festlegen was zum Subset gehört (FESA Metamodel: Design)
- Möglichkeit zum Erstellen von Subsets für Datenbank-Import (Eclipse plug-in)
- Datenbank-Import für Subsets (Eclipse plug-in)
- Möglichkeit zum Konfigurieren einer Geräteinstanz (FESA Metamodel: Instanziierung)
- Möglichkeit zum Extrahieren eines Konfigurationseintrags im C++-Code (FESA core)

Vorteile

- Nur eine einzige FESA Klasse + FESA DeployUnit zu programmieren und zu pflegen
- Realisierung von 'einheitlicher' Gerätesoftware für minimal unterschiedliche Varianten eines Gerätes

Nachteile

- Wenig Möglichkeiten vorab Fehler beim Design etc. abzufangen!
- Unübersichtlich im Design und im Quelltext
- Schwer nachvollziehbar was wo in welcher Konfiguration läuft
- Schwer nachvollziehbar was in welcher Version in die Datenbank eingetragen wurde

Auswirkungen

- Datenbank-Import der Instanziierung betroffen:
  - Es müßte der Name eines Subsets anstelle des Klassennamens in der Instanziierung eingetragen werden
- Properties in Designs die Subsets verwenden müssen partial-setting-fähig sein
-

Auch denkbar

- Subsets auf property-Ebene, nicht (nur) auf value-item-Ebene
- Fassaden: gemeinsamer Ansatz mit CERN, dort gibt es ein ähnliches Problem