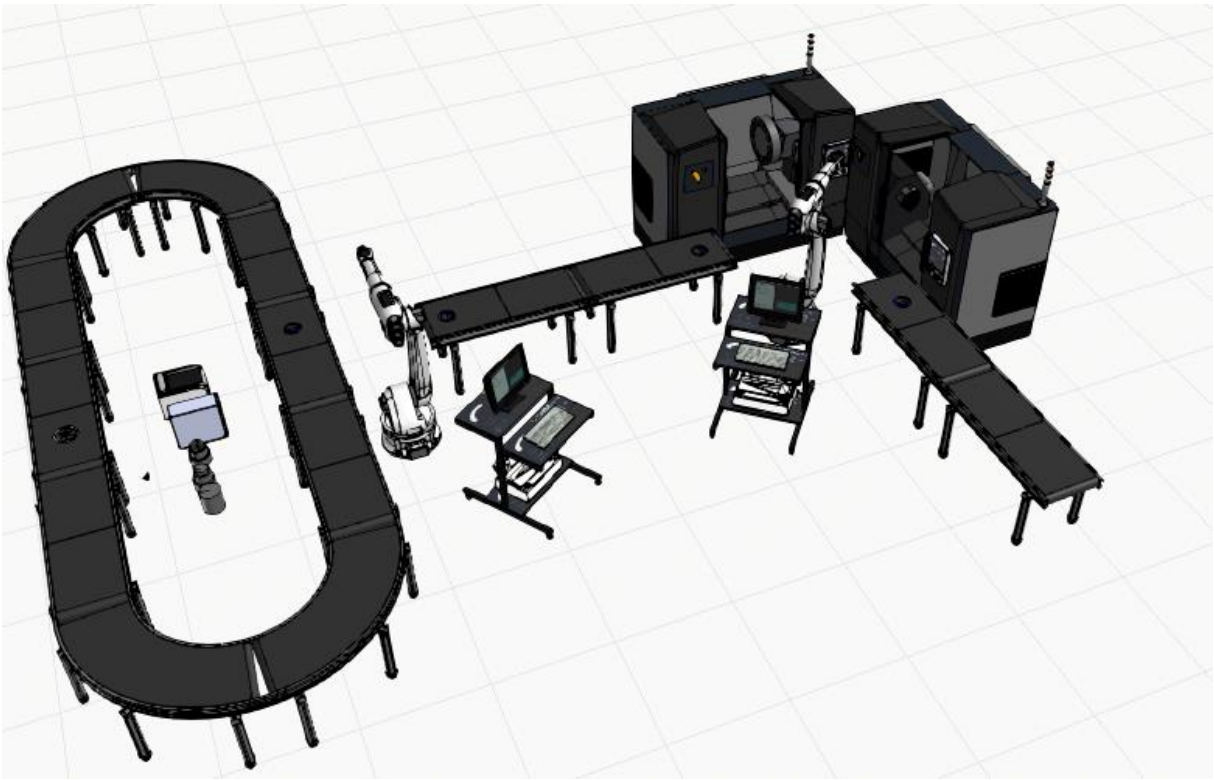
Schnittstelle zwischen realer und virtueller Anlage in Visual Components



Semesterarbeit

von Mungan Baris

Ruhr-Universität Bochum

Inhaltsverzeichnis

1. Anforderung

Die Aufgabe der Semesterarbeit war die Entwicklung einer Schnittstelle zwischen einer realen und virtuellen Anlage in der Software Visual Components mit der Programmiersprache Python.

Ziel war es, mithilfe von Python Informationen außerhalb der Software zu lesen und anhand dieser Informationen bestimmte Funktionen wie die Positionierung von Komponenten oder die Steuerung von Robotergelenken auszuführen.

2. Lösungsansatz

Zum Austausch von Informationen zwischen einer realen und virtuellen Anlage wird eine Datenbank genutzt. Informationen aus der realen Anlage werden in der Datenbank gespeichert, welche wiederum von der virtuelle Anlagen gelesen werden.

Die gespeicherten Einträge beinhalten den Namen der Komponente, dessen Typ, die auszuführende Funktion und zusätzliche Informationen, falls erforderlich, für die Funktion.

Die Informationen werden von einer bestimmten Komponente (Schnittstelle) gelesen und über ein StringSignal-Verhalten namens ‚UpdateSignal' der jeweiligen Komponente übergeben.

Da keine reale Anlage zur Verfügung steht, wurde stattdessen eine weitere Simulation als ‚reale Anlage' genutzt, welche periodisch Informationen abhängig von den Sensoren an die Datenbank sendet.

Zum Erstellen und Verwalten der Datenbank wurde das Modul SQLite von Python gewählt.

3. Projektaufbau

3.1 Überblick

Im Projektverzeichnis sind folgende Ordner und Dateien enthalten:

- ***datenbanken***: enthält die Datenbanken
- ***komponenten***: enthält die Code-Dateien, die von den Komponenten importiert werden
- ***steuerung_gui***: enthält die Code-Dateien, die von der Steuerung importiert werden
- ***real.vcmx***: Simulation, die Informationen in die (virtuelle) Datenbank schreibt und von der (realen) Datenbank liest
- ***virtuell.vcmx***: Simulation, die Informationen von der (virtuellen) Datenbank liest
- ***steuerung.pyw***: GUI zum Steuern der realen Anlage (*real.vcmx*); speichert Informationen in der (realen) Datenbank für die reale Anlage
- ***vccode.py***: importiert die ganzen Klassen und stellt diese den Komponenten in Visual Components zur Verfügung; muss sich im Ordner *Visual Components\Visual Components Premium 4.2\Python\lib\site-packages* befinden
- ***konst.py***: enthält Informationen über die Datenbanken, wie Pfad, Tabellenname und Felder

3.2 Ordner *komponenten*

3.2.1 Importieren der Dateien

Zum Importieren der Dateien muss im Pythonscript der Komponente in der Regel Folgendes stehen (Ausnahme *roboter.py* und *rob_schnittstelle.py*):

```
import vcScript
from vccode import <Klasse>

komptyp = <Klasse>(vcScript)

def OnStart():
   komptyp.OnStart()

def OnSignal(signal):
   komptyp.OnSignal(signal)

def OnRun():
   komptyp.OnRun()
```

3.2.2 *creator.py*

Klasse: *Creator*

Komponente: egal

Beschreibung: Stellt ComponentCreator zur Verfügung, die vor allem vom Modul *transport.py* genutzt werden, um abh. von Signale Komponenten zur Erstellen.

3.2.3 *datenbank.py*

Klasse: *Datenbank*

Komponente: keine

Beschreibung: Stellt Methoden für den Zugriff auf die Komponente für andere Module bereit.

3.2.4 *feeder.py*

Klasse: *Feeder*

Komponente: Process Feeder (in Process Flow Components)

Beschreibung: Wird in der realen Anlage benutzt, um periodisch oder manuell Komponenten herzustellen.

3.2.5 *maschine.py*

Klasse: *Maschine*

Komponente: Maschinen von Visual Components (in Machines)

Beschreibung: Ermöglicht die Steuerung der Maschine wie das Öffnen und Schließen der Tür und das Starten des Bearbeitungsprozesses.

3.2.6 *rob_schnittstelle.py*

Klasse: *RobSchnittstelle*

Komponente: egal

Beschreibung: In der realen Anlage werden periodisch die Gelenkwerte der ausgewählten Roboter (in den Eigenschaften festgelegt) gelesen und in einer, speziell für die einzelnen Roboter, Datenbank gespeichert.

In der virtuellen Anlage werden die Roboter entsprechend den gelesenen Gelenkwerten aus den jeweiligen Datenbanken gesteuert.

Code zum Importieren:

```
import vcScript
import vcHelpers.Robot2 as vcRobot
from vccode import RobSchnittstelle

komptyp = RobSchnittstelle(vcScript, vcRobot)

def OnStart():
   komptyp.OnStart()

   def OnRun():
   komptyp.OnRun()
```

3.2.7 *roboter.py*

Klasse: *Roboter*

Komponente: Generic Articulated Roboter (in Robots)

Beschreibung: Wird zum Steuern der Roboter benutzt wie greifen, platzieren und bewegen der Gelenke.

Code zum Importieren:

```
import vcScript
import vcHelpers.Robot2 as vcRobot
from vccode import Roboter

komptyp = Roboter(vcScript, vcRobot)

def OnStart():
   komptyp.OnStart()

def OnSignal(signal):
   komptyp.OnSignal(signal)

def OnRun():
   komptyp.OnRun()
```

3.2.8 *schnittstelle.py*

Klasse: *Schnittstelle*

Komponente: egal

Beschreibung: Stellt fest, ob es sich um eine reale oder virtuelle Anlage handelt (Eigenschaft) und liest abh. davon Einträge von der jeweiligen Datenbank. Die erhaktenen Informationen werden an die verschiedenen Komponenten weitergegeben.

3.2.9 *transport.py*

Klasse: Transport

Komponente: Conveyor, Sensor Conveyor, Curve Conveyor und andere (in Conveyors)

Beschreibung: In der realen Anlage wird der Signalwert des Sensors in der Datenbank gespeichert und in der virtuellen der reale Signalwert gelesen und mit der virtuellen verglichen. Abhängig vom Ergebnis wird eine Komponente erstellt, entfernt oder nichts gemacht.

In der realen Anlage werden, falls Komponenten umgeleitet werden, diese entfernt und an der gewünschten Stelle neu erstellt und nicht neu positioniert, da dies in dynamischen Containern nicht möglich ist.

3.2.10 *transportstelle.py*

Klasse: *Transportstelle*

Komponente: Bundler Point (in Conveyor Utilities)

Beschreibung: Legt die Position fest, von der Komponenten vom Roboter gegriffen werden sollen oder nicht und ob Komponenten auf dem Förderband gestoppt werden sollen oder nicht (Eigenschaft).

Außerdem werden wie bei der Klasse *Transport* auch reale und virtuelle Signale miteinander verglichen und dem entsprechend Komponenten erstellt, entfernt oder nichts gemacht.

3.2.11 *upvia.py*

Klasse: *Upvia*

Komponente: keine

Beschreibung: *Up*date *vi*rtuelle *A*nlage. Diese Klasse wird von den Klassen *transport.py* und *transportstelle.py* geerbt. Wird zum speichern und vergleichen der realen und virtuellen Signale genutzt.

3.3 Ordner *steuerung_gui*

3.3.1 Die Steuerung

Über die GUI *steuerung.pyw* werden Funktionen zum Steuern der realen Anlage zur Verfügung gestellt.

Für jeden Reiter wurde ein eigenes Modul erstellt.

3.3.2 *feeder_tab.py*

Reiter: Feeder

Funktionen:

- Erstelle eine Komponente
- Erstelle periodisch Komponenten nach einem bestimmten Intervall

3.3.3 *maschine_tab.py*

Reiter: Maschine

Funktionen:

- Schalte zwischen Automatisch und Manuell um
- Öffne und Schließe die Türe
- Starte den Prozess
- Lege die Prozesszeit fest

3.3.4 *roboter_tab.py*

Reiter: Roboter

Funktionen:

- Schalte zwischen Automatisch und Manuell um
- Greife die Komponente an Station x
- Platziere die Komponente in Station y
- Lege die Gelenkwerte fest

3.3.5 *transport_tab.py*

Reiter: Transport

Funktion:

- Leite die Komponente von Förderband x zu Förderband y um

3.4 Order datenbanken

Es gibt mindestens drei Datenbanken (Felder in Klammern), welche automatisch erstellt werden:

- *komponenten.db*: (Name, Typ) Hier sind alle Komponenten in der realen Anlage gespeichert, die einen Typ haben. Wird von der GUI *steuerung.pyw* genutzt.
- *signale_real.db*: (Name, Typ, Funktion, Info) Hier wird festgelegt, welche Komponente welche Funktion in der realen Anlage ausführen soll. Welche Funktion ausgeführt werden soll wird über die GUI *steuerung.pyw* festgelegt.
- *signale_virtuelle.db*: (Name, Typ, Funktion, Info) Hier wird festgelegt, welche Komponente welche Funktion in der virtuellen Anlage ausführen soll. Die auszuführende Funktion wird über die reale Anlage festgelegt.
- <*robotername*>*.db*: (Gelenkwerte) Die Gelenkwerte des jeweiligen Roboters werden hier periodisch gespeichert und gelesen. Die Anzahl dieser Datenbanken hängt ab von der Anzahl an Roboter im Layout.

4 Projektanleitung

4.1 Pfad zum Sqlite-Modul in der Umgebungsvariable speichern

In Visual Components 4.2 muss der Pfad zum sqlite-Modul in der Umgebungsvariable PATH gespeichert werden, damit diese importiert werden kann. Dieser befindet sich im Ordner *DLLS*, welches (abh. vom Installationsort) folgenden Pfad hat: *„C:\Program Files\Visual Components\Visual Components Premium 4.2\Python\DLLs“.*

Pfad der Umgebungsvariable hinzufügen (Win10):

1. Im Suchfeld der Taskleiste (Start) „Umgebungsvariable“ suchen und auf „Umgebungsvariablen für dieses Konto bearbeiten“ klicken.
2. Auf „Umgebungsvariablen“ klicken.
3. Oben in der Liste „Path“ wählen und auf „Bearbeiten“ klicken.
4. Auf „Neu“ klicken, den Pfad hinzufügen und auf „OK“ klicken

4.2 Download des Projekts von GitHub

Der Projektordner sollte entweder über git (git pull) oder über „Code>Download Zip“ runtergeladen und ggf. extrahiert werden.

4.3 Pfade festlegen

Der Pfad zum Projektordner (z.B. *r'C:\Users\Baris\Desktop\Semesterarbeit'*) sollte in den Dateien vccode.py und konst.py der jeweiligen Variable zugewiesen werden.

vccode.py: pfad_zum_code = *r'C:\Users\Baris\Desktop\Semesterarbeit'*

konst.py: PFAD = *'C:\\Users\\Baris\\Desktop\\Semesterarbeit\\datenbanken\\'*

Bei *konst.py* ist es nicht möglich die r-Notation (s.o.) zu Nutzen, da am Ende des Strings auch ein Schrägstrich vorkommt.

4.4 Komponente zum Importieren bereitstellen

Damit die Code-Dateien auch von Visual Components genutzt werden können, müssen diese im Ordner *site-packages* (*„C:\Program Files\Visual Components\Visual Components Premium 4.2\Python\lib\site-packages“*) gespeichert werden.

Da die Klassen schon vom Modul *vccode.py* importiert werden, muss nur diese Datei im Ordner *site-packages* gespeichert werden.

4.5 Projekt starten

Zum Starten des Projekts müssen folgende Dateien geöffnet werden: *real.vcmx*, *virtuell.vcmx* und *steuerung.pyw*. Die Fenster von *real.vcmx* und *virtuell.vcmx* sollten nebeneinander gelegt werden, damit diese miteinander verglichen werden können.

Über *steuerung.pyw* können Befehle an die reale Anlage gesendet werden. Die virtuelle Anlage sollte sich der realen Anlage anpassen. Dies kann man sehen, wenn man einer der Anlagen für kurze Zeit pausiert und dann wieder startet.