Engineering Conferences International
ECI Digital Archives

Integrated Continuous Biomanufacturing III

Proceedings

9-18-2017

Supervisory control of integrated continuous downstream processes

Bernt Nilsson
Lund University, Sweden, bernt.nilsson@chemeng.lth.se

Niklas Andersson
Lund University, Sweden

Joaquin Gomis Fons
Lund University, Sweden

Anton Löfgren
Lund University, Sweden

Recommended Citation

Bernt Nilsson, Niklas Andersson, Joaquin Gomis Fons, and Anton Löfgren, "Supervisory control of integrated continuous downstream processes" in "Integrated Continuous Biomanufacturing III", Suzanne Farid, University College London, United Kingdom Chetan Goudar, Amgen, USA Paula Alves, IBET, Portugal Veena Warikoo, Axcella Health, Inc., USA Eds, ECI Symposium Series, (2017). http://dc.engconfintl.org/biomanufact_iii/80

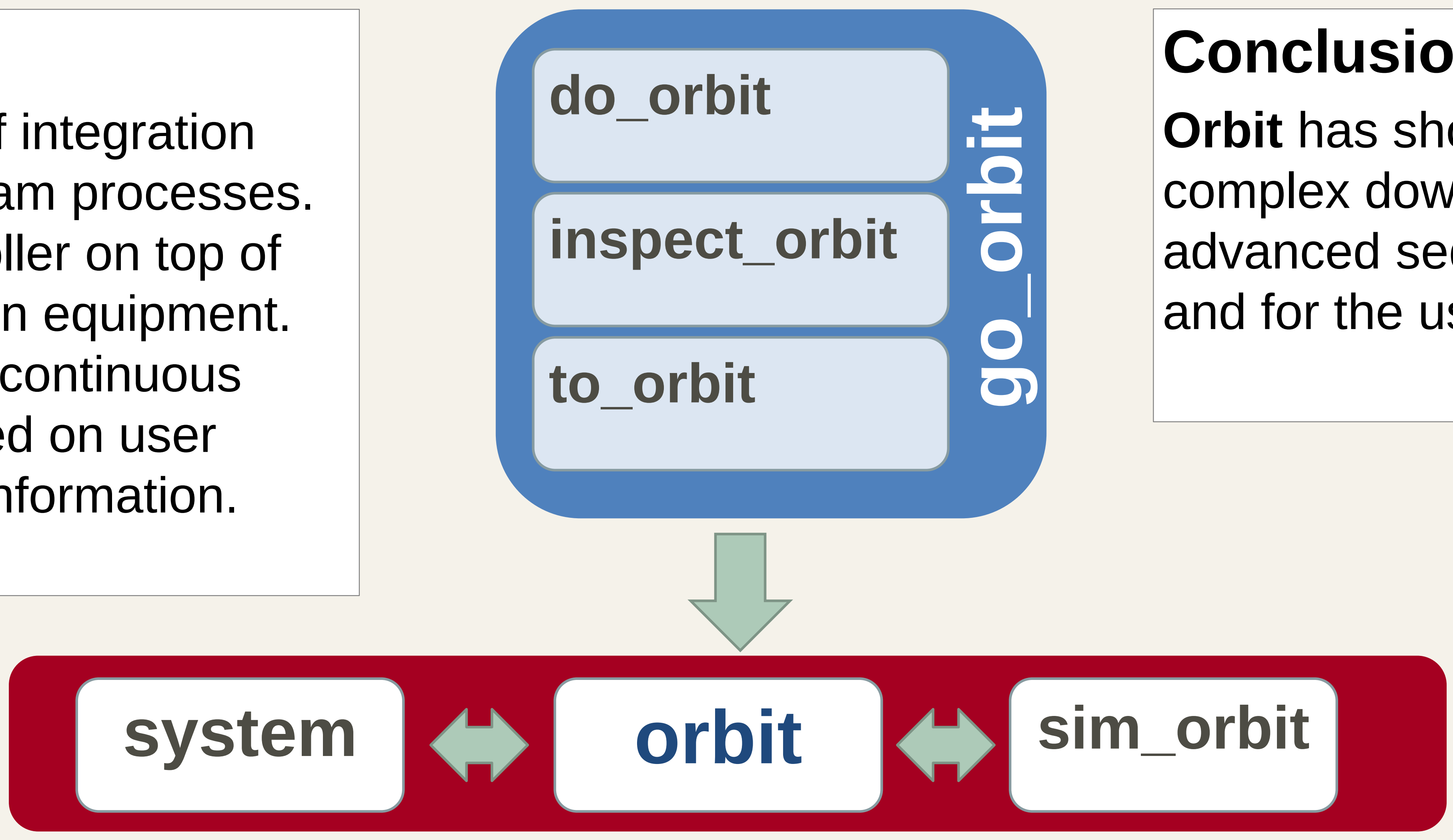
SUPERVISORY CONTROL OF INTEGRATED CONTINUOUS DOWNSTREAM PROCSSES

Bernt Nilsson[1], Anton Löfgren[1], Joaquin Gomis Fons[1], Niklas Andersson[1], and Lotta Berghard[2]

[1] Dept of Chemical Engineering, Lund University, Lund, Sweden, [2] SOBI, Stockholm, Sweden

Introduction

There are industrial needs of integration and automation of downstream processes. **Orbit** is a supervisory controller on top of common available purification equipment. **Orbit** can control integrated continuous downstream processes based on user definition on a high level of information.

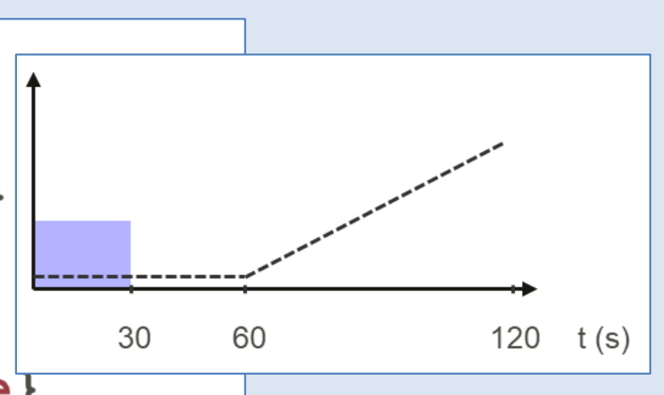


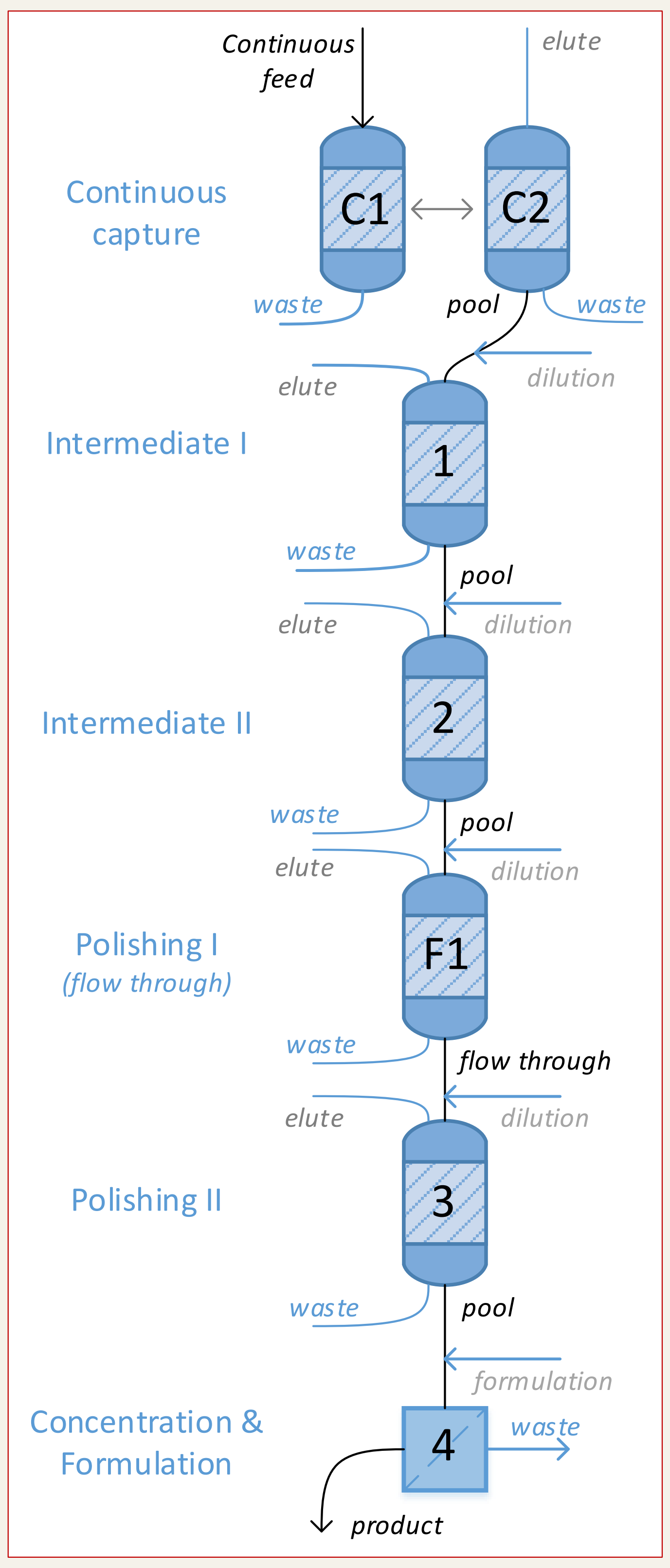
Conclusion

Orbit has shown that it is possible to define complex downstream configurations and advanced sequence structures in a modular and for the user convenient way.

*Example of **orbit** code*

```
Load = {'Time': 30,
        'FlowRate':1
        'Inject': True}

Wash = {'Time': 30,
        'Inject': False}

Elu  = {'Time':  60,
        'Gradient': 1}

Chrom = [Load,Wash,Elu]

s.runFunc(Chrom)
```





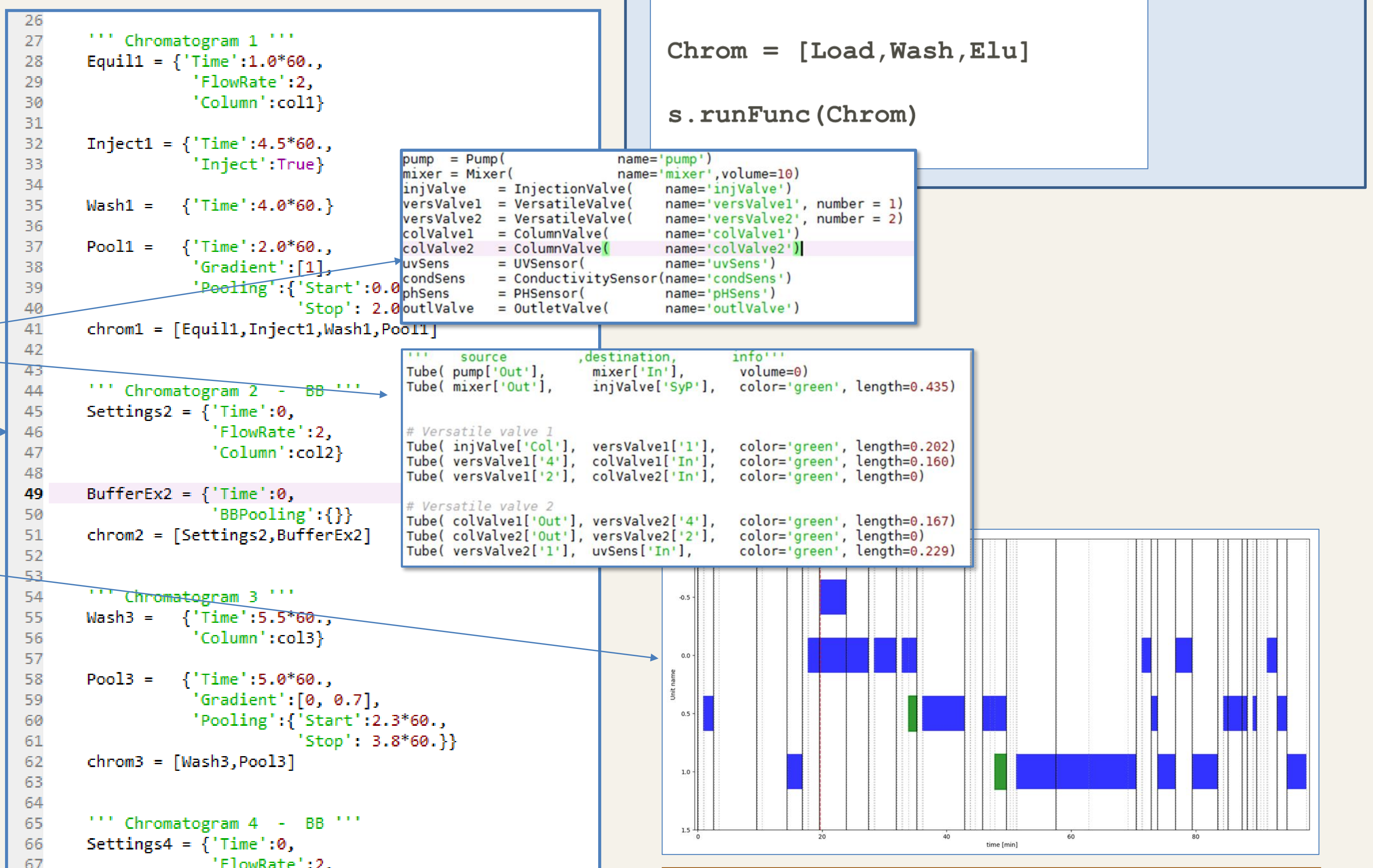
go_orbit

The **orbit** controller is automatically generated based on the design defined in **do_orbit** in two parts:

- Configuration and connections
- Overall sequence

inspect_orbit do analysis of the controller to find unfeasible operations.

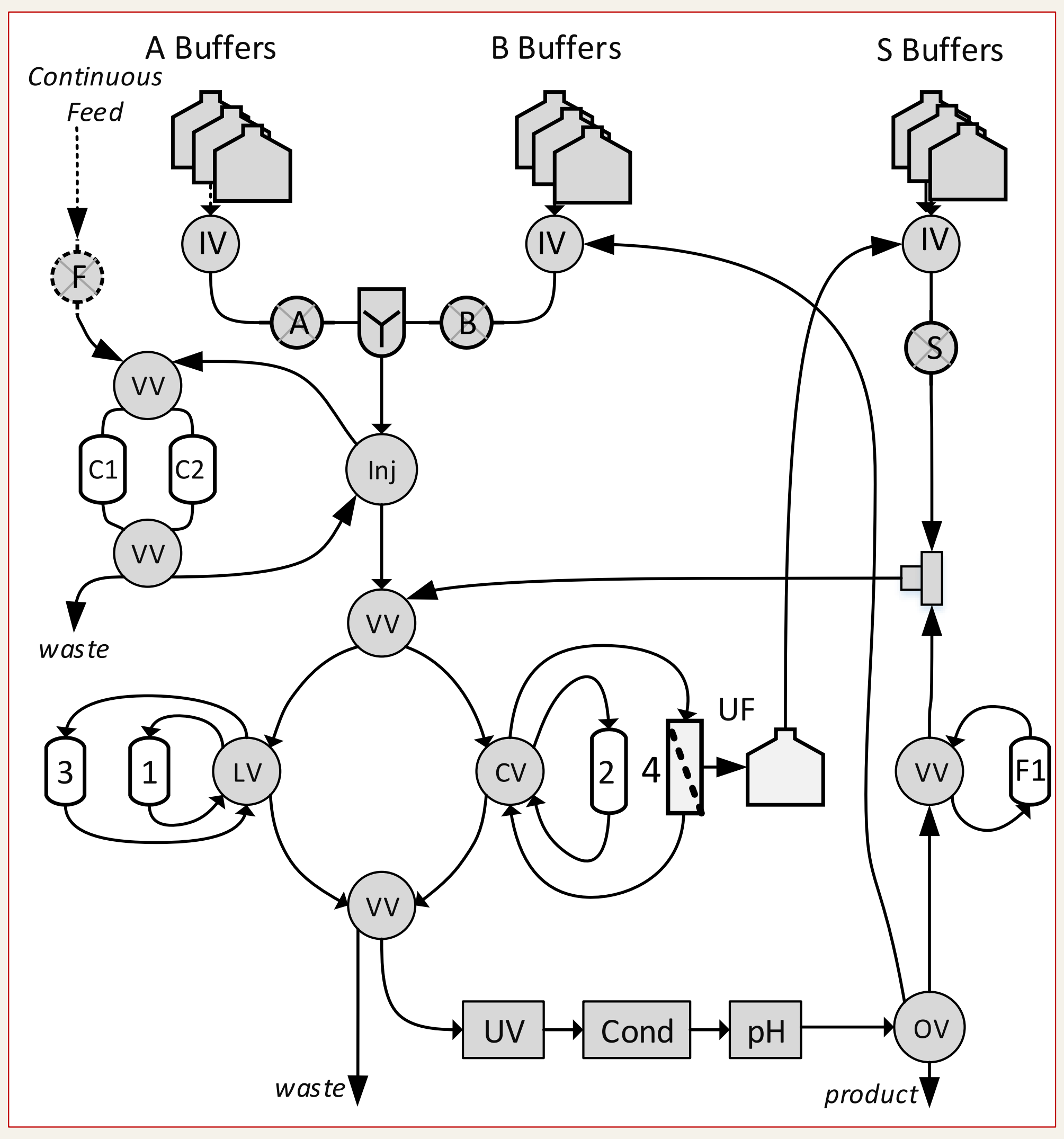
to_orbit takes the design and automatically generates the orbit controller code for sequential control based on event-action logic.



```
''' Chromatogram 1 '''
Equil1 = {'Time':1.0*60.,
          'FlowRate':2,
          'Column':col1}

Inject1 = {'Time':4.5*60.,
           'Inject':True}

Wash1 =   {'Time':4.0*60.}

Pool1 =   {'Time':2.0*60.,
           'Gradient':[1],
           'Pooling':{'Start':0.0
                      'Stop': 2.0
chrom1 = [Equil1,Inject1,Wash1,Pool1]


''' Chromatogram 2  -  BB '''
Settings2 = {'Time':0,
             'FlowRate':2,
             'Column':col2}

BufferEx2 = {'Time':0,
             'BBPooling':{}}
chrom2 = [Settings2,BufferEx2]


''' Chromatogram 3 '''
Wash3 =   {'Time':5.5*60.,
           'Column':col3}

Pool3 =   {'Time':5.0*60.,
           'Gradient':[0, 0.7],
           'Pooling':{'Start':2.3*60.,
                      'Stop': 3.8*60.}}
chrom3 = [Wash3,Pool3]


''' Chromatogram 4  -  BB '''
Settings4 = {'Time':0,
             'FlowRate':2,
             'Column':col7}

BufferEx4 = {'Time':0,
             'BBPooling':{'Next':'Waste'}}
chrom4 = [Settings4,BufferEx4]


chroms = [chrom1,chrom2,chrom3,chrom4]
guiController(s,chroms)
```

```
pump  = Pump(                 name='pump')
mixer = Mixer(                name='mixer',volume=10)
injValve    = InjectionValve(     name='injValve')
versValve1  = VersatileValve(     name='versValve1', number = 1)
versValve2  = VersatileValve(     name='versValve2', number = 2)
colValve1   = ColumnValve(        name='colValve1')
colValve2   = ColumnValve(        name='colValve2')
uvSens      = UVSensor(           name='uvSens')
condSens    = ConductivitySensor(name='condSens')
phSens      = PHSensor(           name='pHSens')
outlValve   = OutletValve(        name='outlValve')
```

```
'''     source          ,destination,       info'''
Tube( pump['Out'],        mixer['In'],        volume=0)
Tube( mixer['Out'],       injValve['SyP'],    color='green', length=0.435)


# Versatile valve 1
Tube( injValve['Col'],  versValve1['1'],    color='green', length=0.202)
Tube( versValve1['4'],  colValve1['In'],    color='green', length=0.160)
Tube( versValve1['2'],  colValve2['In'],    color='green', length=0)

# Versatile valve 2
Tube( colValve1['Out'], versValve2['4'],    color='green', length=0.167)
Tube( colValve2['Out'], versValve2['2'],    color='green', length=0)
Tube( versValve2['1'],  uvSens['In'],       color='green', length=0.229)
```



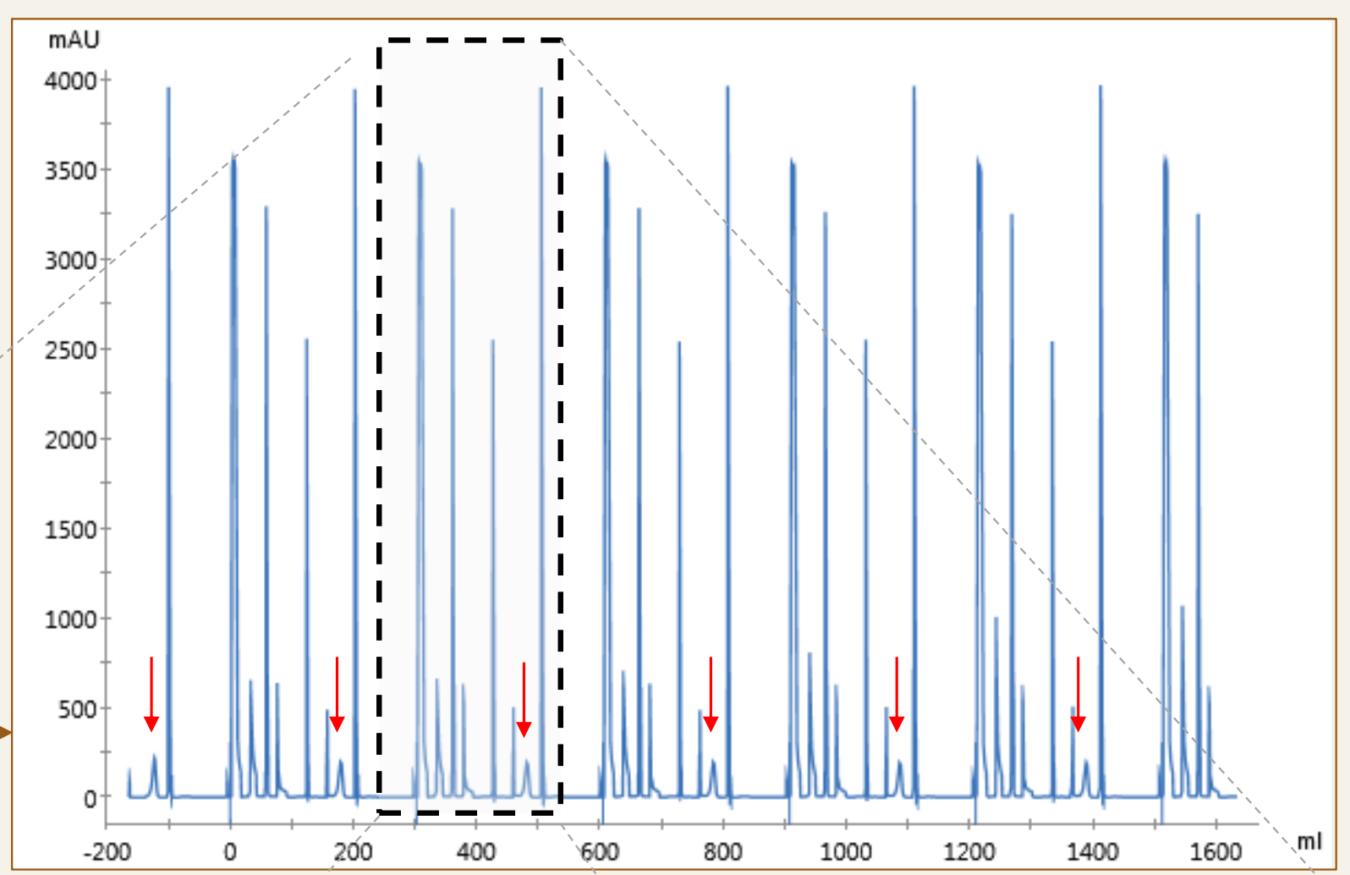
orbit

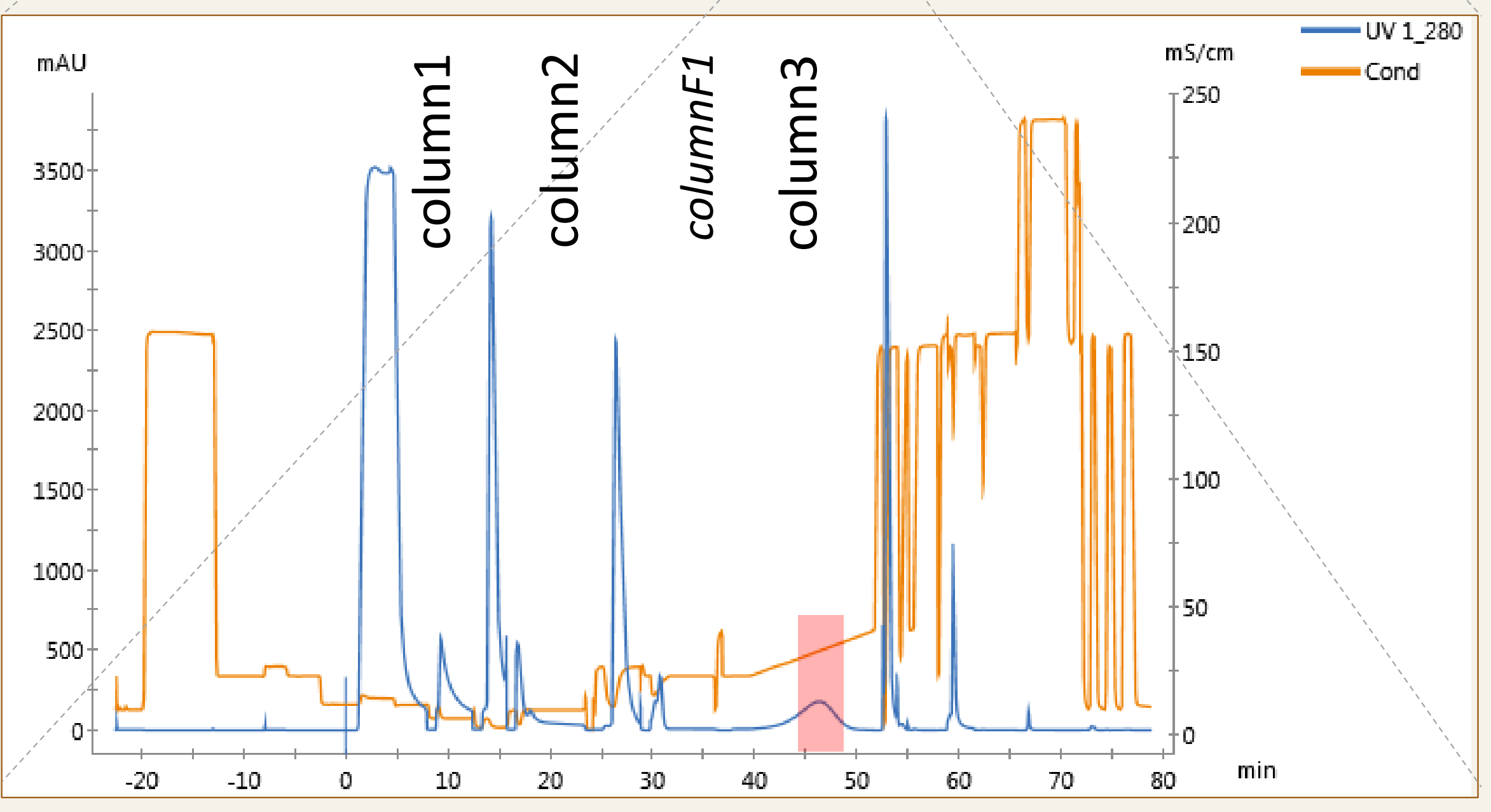
This is the actual sequential controller that executes actions based on events that is automatically generated from go_orbit.

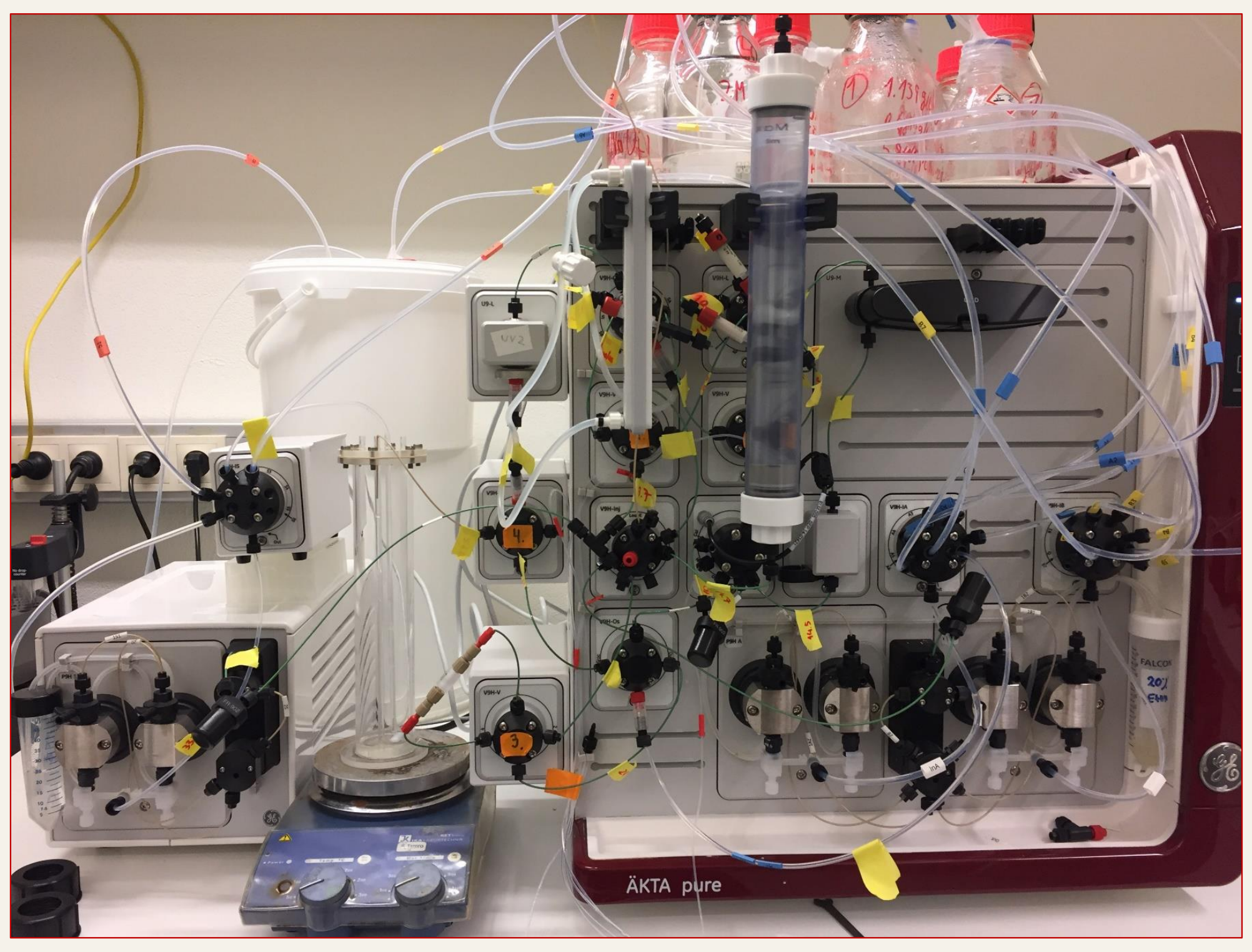
Actions are set flow rates, switch valve positions, get sensor values, start peak integration,… etc

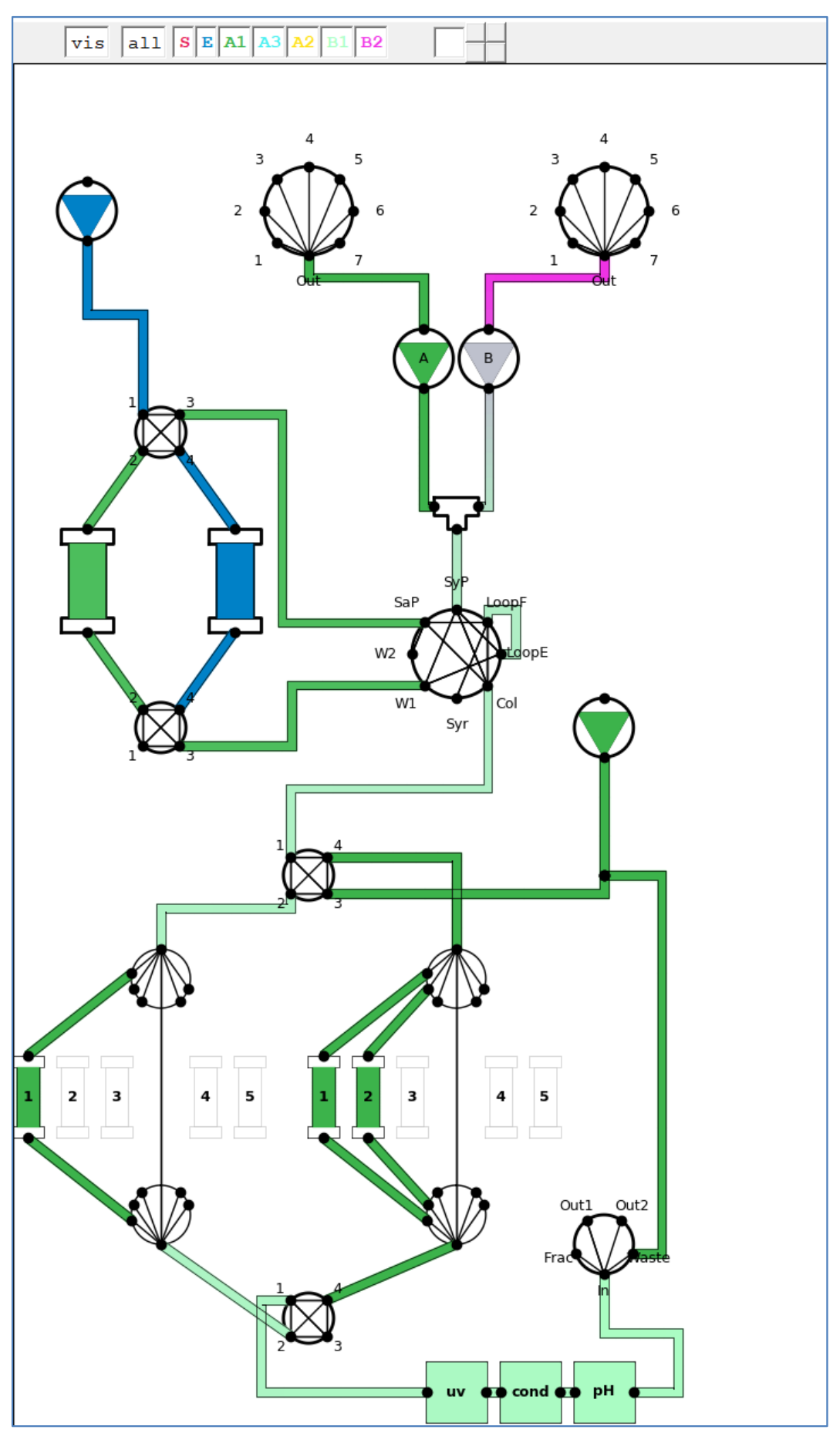
Events are often a given time. Other events are sensor values or peak integration value.

The orbit controller run autonomous.









sim_orbit

A system simulator based on automatically generated information from orbit: i) the configuration, ii) control sequence and iii) a model library of all units available.

The simulator makes it possible to estimate the performance of the sequence:

- Buffer flow path and consumption
- Cleaning of complete configuration
- Check of complete cycle

Acknowledgement

novo nordisk

Modelon

VINNOVA

PiiA Process Industrial IT and Automation

PIC Process Industry Centre

Bernt Nilsson, Dept. of Chemical Engineering, Lund University, Box 124, 221 00 Lund, Sweden | +46462228088 | bernt.nilsson@chemeng.lth.se