Design ideas behind Bioprocess Library *for* Modelica

Jan Peter Axelsson

Vascaia AB, Sweden, jan.peter.axelsson@vascaia.se

Abstract

This paper discusses key design ideas behind the Bioprocess Library, BPL. The library facilitates modelling and simulation of bioprocesses mainly for the pharmaceutical industry. It borrows some structures from MSL Fluid and Media but differs in central design choices. A typical application consists of both configuration of standard components from the library and tailor-made Modelica code defining the application-dependent medium and bioprocess reactions. The guiding idea is that configuration of components works well for defining the setup of process equipment for a production line, while more flexibility is needed for modelling bioprocess reactions and therefore equations are used. Another central design idea is that components of equipment are centrally adapted to the medium used. One could say that the library is parameterised with the application media and reaction models. The focus of this paper is structural design aspects of the library rather than the content.

Keywords: Bioprocess, media, reactions, formal type parameters, packages, components, equations

1 Introduction

There is a growing interest in simulation also in the biopharmaceutical industry. Two major vendors of equipment Cytiva (*Bioreactor-scaling-tool* 2023) and Sartorius today offer services around their products based on using simulation. Two well-known companies are Siemens (with gPROMS) and Dassault (with 3DEXPERIENCE) who offer softwares and services in this market. The need is often a combination of mechanistic and more data-driven modelling. In the academic area the interest in simulation of biological systems has been there for decades and illustrated by the public repositories of models (*EMBL BioModels* 2023; *UC San Diego BiGG Models* 2023). So far, Modelica has had very little impact in this field, though. Important aspects of the question is discussed in (Wiechert, Noack, and Elsheikh 2010).

Developing bioprocesses requires a combination of knowledge from different fields like: reactor dynamics, gas-liquid-transfer, buffer-reactions, cell metabolism, recombinant protein expression, degradation processes of product proteins in the broth, impurities etc. Part of this knowledge is well established and can be re-used, while modeling of cell metabolism and product formation may be more unique, and less re-useable. Further, the same reactor may be operated in different ways: batch, fed-batch, continuous, perfusion etc. Various ideas of process control are also interesting to evaluate using simulation.

Bioprocess Library tries to meet these needs of flexibility and possibility to re-use code. It has been gradually developed over many years in-house for consultancy work and also teaching. Examples of applications are (Axelsson 2018; Axelsson 2019; Axelsson 2022). Examples of integrating black-box models together with the traditional mechanistic models has not been done yet, but can certainly be done in the framework. The architecture of the library was outlined in (Axelsson 2021). A key design aspect is to account for the different modelling needs for the process configuration and the actual reactions in the reactor. The engineered part is modelled by conveniently configure components of the library. The biological part requires more flexibility. Modelling of cell metabolism and other reactions in the broth is done by writing down the equations in a certain Modelica format.

This paper is organised as follows. An orientation of the library is given in section 2. Section 3 focuses on the reactor model and how library code is integrated with application code. In section 4 a simple example of fed-batch cultivation illustrates the structure of typical application code. In section 5 the technique of constraint-based modelling is briefly discussed and how such modelling can be handled with the library. In section 6 different aspects of the library design are discussed including the relation to MSL Fluid and Media. The availability of the library is outlined in section 7 and the paper ends in section 8 with concluding remarks.

2 Library structure

The library has a flat hierarchy, limiting the use of partial packages and models. In this way it is hopefully easier to understand and later to expand. A brief orientation of the library is first given. The next section describes the EquipmentLib, followed by a section on the media connectors.

2.1 Overview

The structure of Bioprocess Library is shown in Figure 1. It includes the following packages:

- `UsersGuide` - Package of brief practical information in different records, accessible from the FMU.
- `Interfaces` - Package of templates for liquid and gas, and interfaces for the `Reactor` component and its inner application models. It is all used in `EquipmentLib`.

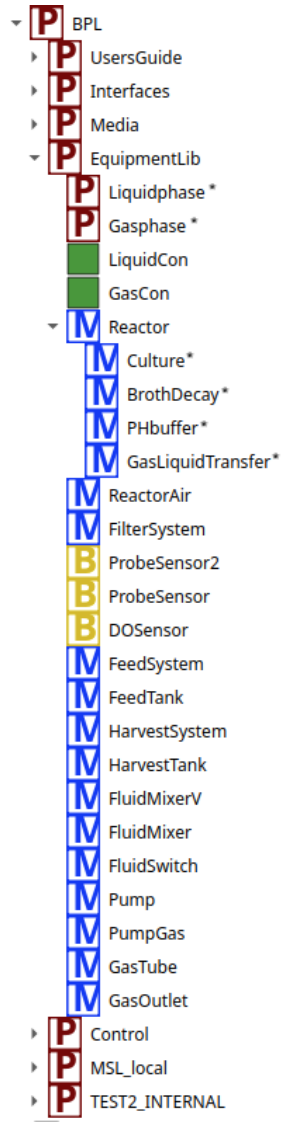

Figure 1. Structure of the Bioprocess Library. The formal (type) parameters are marked with ∗.

- `Media` - Package for various standard media.
- `EquipmentLib` - Package of models of standard equipment using a generic medium, e.g. tanks, pumps, reactors, sensors, filter, mixers etc for both liquid and gas media.
- `Control` - Package of blocks for generating electrical control signals to equipment components. They complement and adjust blocks from MSL.
- `MSL_local` - Package that contains parts of MSL for PID-control and more. In this way BPL can be used more independent of the general MSL version of the compiler. Currently the MSL version 3.2.2 build 3 is used, but to be updated to MSL 4.0.0.
- `TEST2_INTERNAL` - Package with small test applications including configuration of equipment for batch, fed-batch, chemostat and perfusion cultivation.

2.2 The EquipmentLib package

The `EquipmentLib` package has a central role in the library. Components are coded in a generic way. The package is parametrised with two general formal (package) parameters for the media:

- `Liquidphase` - the package for the liquid phase
- `Gasphase` - the package for the gas phase

and four specific formal (model) parameters for just the component `Reactor` described in section 3.

To improve readability, the code sections defining base packages and partial models are lifted out to the package `Interfaces` and imported back where needed.

2.3 The media and connectors

The flow direction of both liquid and gas media are usually well-defined in bioprocesses and reflected in the present components. The media connectors are however undirected and prepared to handle back-flow.

The `LiquidCon` connector code is shown below. The `GasCon` connector is defined similarly.

Listing 1. LiquidCon connector

```
connector LiquidCon
   stream Liquidphase.Concentration c;
   flow Real F (unit="L/h");
   Real p (unit="bar");
end LiquidCon;
```

The connector uses the flow concept for the flow F and the corresponding potential is the pressure p. The concentration vector c gets its size from the actual medium. The vector c is declared as a stream variable. The density of the liquid media is calculated when needed based on the concentration c and here is a function in the liquid media base template that takes information of molar weights from the actual application medium. Similar technique as in MSL Media.

Both the inclusion of pressure p and the use of the stream-concept for c are introduced to "future-proof" the library, and facilitate local balancing of models, see (Olsson et al. 2008; Franke et al. 2009). The pressure does not play any role in the applications so far. The pumps are ideal in the sense that a given electrical signal gives the desired flow rate immediately. There has not been any obvious need to model back-flow either, which is the major motivation of the undirected connector using the stream concept (Franke et al. 2009).

The temperature has so far not been considered important in the modelling. The temperature is usually in the range from room temperature up to 37°C and well controlled. The cell culture produces heat due to metabolism and at high cell concentrations and feed rate, the cooling capacity of the reactor sets a limit. This limit is very similar to the oxygen transfer capacity. Both set about the same limit of rate of metabolism supported by mechanical reactor design. The process is usually designed with some margin to this limit, see simulations in section 4.4.

DOI
10.3384/ecp204453

3 The reactor component

The various reactions of the process are thought to take place only in the `Reactor` component. The other equipment only transports, stores, separates media species, or mixes media from several sources. The `Reactor` combines standard and application dependent parts in a complex way. The component `ReactorAir` is an extension of the `Reactor` to also handle gas phase.

The reactions in a typical bioreactor can be divided into different parts. These parts are user-defined sub-models, i.e. `inner` components, to the `Reactor` and serve as formal (type) parameters:

- `culture` - reactions in the living cells
- `broth_decay` - degradation of substances and even of the living cells in the culture broth
- `pH_buffer` - pH-buffer reactions in the broth
- `gas_liquid_transfer` - gas-liquid-transfer between the reactor broth and gas phase

An important observation is that it is the concentration of substances that drives the rate of reactions in these different parts. The broth concentration is the "communication link" between them. Therefore, concentration is chosen as an `inner` variable in the reactor. Similarly, gas fraction is used as an `inner` variable in the extension for aerated reactor. The different parts are naturally sub-models to the reactor model. The reactor sub-models communicate back to the reactor through rate-variables. These rate-variables are normalised with respect to the biomass $m[X]$ of the reactor for reactions in the cell and with respect to liquid reactor volume V for the others.

The corresponding code-snippet for the reactor model is shown below. A key application dependent parameter is *nc*, that stands for the number of components, or we should say species, in the medium.

Listing 2. Reactor model equations

```
// Concentrations for the liquid phase:
for i in 1:Liquidphase.nc loop
   c[i] = m[i]/V;
   for j in 1:n_outlets loop
      outlet[j].c[i] = c[i];
   end for;
   for j in 1:n_ports loop
      port[j].c[i] = c[i];
   end for;
end for;

// Mass-balance for the liquid phase:
for i in 1:Liquidphase.nc loop
   der(m[i]) = culture.q[i]*m[X]
   + broth_decay.r[i]*V
   + pH_buffer.r[i]*V
   + gas_liquid_transfer.r_to_liquid[i]*V
   + sum(actualStream(inlet[j].c[i])
      *inlet[j].F for j)
   + sum(c[i]*outlet[j].F for j);
   for j in 1:n_inlets loop
      inlet[j].c[i] = c[i];
   end for;
end for;

// Liquid volume of the reactor:
der(V) = sum(inlet[i].F for i)
       + sum(outlet[i].F for i);
```

The different sub-models are all optional and Modelica provide language constructs to support use of `no_culture`, `no_broth_decay` etc. The interface standard between the reactor and the sub-models are defined in the package Interfaces in BPL.

4 Application code

A simple example will illustrate the structure of the application code, adaptation of the `EquipmentLib` to the application, and finally the configuration of the process model. The model has no gas phase or broth decay, and pH-buffer reactions are not defined either.

The mass-balance model of the example is as follows. See for instance chapter 6 in (Hu 2020).

$$\frac{d(m[S])}{dt} = -q_S(c[S]) \cdot m[X] + S_{in}F_{in}(t) \tag{1}$$

$$\frac{d(m[X])}{dt} = \mu(c[S]) \cdot m[X] \tag{2}$$

$$\frac{dV}{dt} = F_{in}(t) \tag{3}$$

where the specific cell growth rate $\mu(c[S])$ and substrate uptake $q_S(c[S])$ are directly related through the yield Y which is here a constant

$$\mu(c[S]) = Y \cdot q_S(c[S]) = Y \cdot q_S^{max}\frac{c[S]}{K_s + c[S]} \tag{4}$$

The dosage of substrate $F_{in}(t)$ is controlled from a process computer according to a pre-defined exponential scheme.

4.1 Application medium and reactions

The medium is defined by the following package.

Listing 3. Application medium

```
package Liquidphase2
   import BPL.Interfaces.LiquidphaseBase;
   extends LiquidphaseBase
      (name="Standard components X and S",
       nc=2);
   constant Integer X=1;
   constant Integer S=2;
   constant Real[nc] mw (each unit="Da") =
      {24.6, 180.0};
end Liquidphase2;
```

The reactions in the cell culture are described by the following model, see Listing 4.

Listing 4. Application culture

```
model Culture2 "Text-book culture model."
   import BPL.Interfaces.ReactorInterface;
   extends ReactorInterface(redeclare
      package Liquidphase=Liquidphase2);
   outer Liquidphase.Concentration c;
   Liquidphase.Rate q;
   constant Integer X = Liquidphase.X;
   constant Integer S = Liquidphase.S;
   parameter Real Ks (unit="g/L") = 0.1;
   parameter Real qSmax (unit="g/(g*h)")=1;
   parameter Real Y (unit="g/g") = 0.50;
   Real mu (unit="1/h");
equation
   q[X] = mu;
   q[S] = -qSmax*c[S]/(c[S]+Ks);
   mu = -Y*q[S];
end Culture2;
```

Note that the culture model relates metabolic flow rates $q[\,]$ to reactor concentrations $c[\,]$ with a static function. Even for much more complex culture models the relation is a static function, cf equation (5) in the next section. However, if we need to introduce dynamics in the culture model this can be done in this framework too.

It is a possibility to structure the culture model in more parts. This usually improves readability and can help in the dialogue with microbiologists around this part.

4.2 Adaptation of the EquipmentLib

The adaptation of the `BPL/EquipmentLib` to the application medium and culture model is done in the few lines of code below.

Listing 5. Adaptation of `EquipmentLib` to the application

```
package Equipment
   import BPL.EquipmentLib;
   extends EquipmentLib(
      redeclare package Liquidphase =
         Liquidphase2,
      Reactor(redeclare model Culture =
         Culture2));
end Equipment;
```

Note that package `Liquidphase` and model `Culture` are formal (type) parameters to `EquipmenetLib` and get their values from the application. The concept of formal parameters are discussed in section 4.4 (Fritzson 2015).

4.3 Configuration of the application process

The application process can now easily be configured using the adapted library. Note that for the component bioreactor the medium component for cell concentration must be specified and also that an inlet to the reactor is needed. The component feedtank is an integration of a feedtank with a pump. The component dosage scheme is taken from `BPL/Control` package and has an electrical signal connector and no adaptation needed.

The application code needs just the three sections shown. It is possible in the package `Equipment` to define special tailor-made models of equipment needed for the application that is not available in the library.

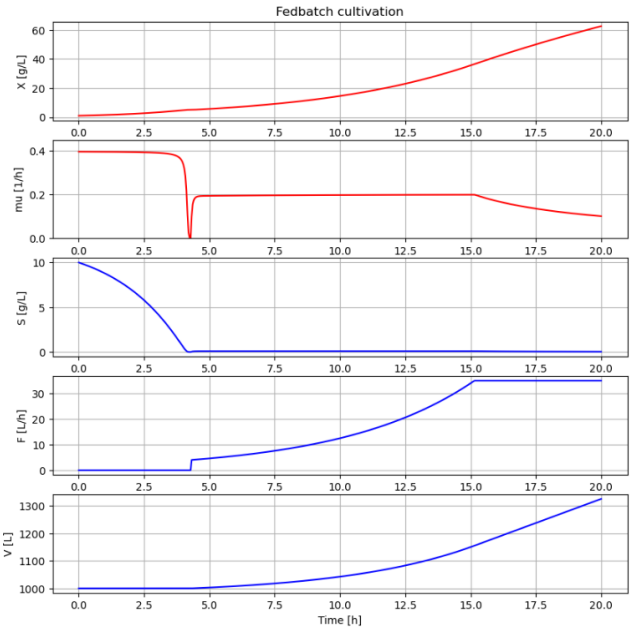

Figure 2. Simulation of fed-batch cultivation. See repository CONF_2023_10_MODELICA15 at (*BPL Applications* 2023).

Listing 6. Application configuration

```
model Fedbatch "Fedbatch cultivation"
   Liquidphase_data liquidphase;
   Equipment.Reactor bioreactor
      (X=liquidphase.X, n_inlets=1);
   Equipment.FeedSystem feedtank;
   Control.DosageSchemeExp dosagescheme;
equation
   connect(bioreactor.inlet[1], feedtank.
      outlet);
   connect(feedtank.Fsp, dosagescheme.F);
end Fedbatch;
```

4.4 Simulation results

The example above of fed-batch cultivation is simulated with quite typical parameters for yeast (*S. cerevisiae*) cultivation (by-product formation neglected), see Figure 2.

It is common to start fed-batch cultivation with a short batch phase, as we see here. When the initial substrate is consumed the substrate feeding is started at 4 h and follows a certain scheme. Here a well-designed exponential feed scheme is used. Note that the substrate level is kept low and growth rate kept constant at about half the maximal rate. From time 15 h and on, the feed rate is kept constant in order to avoid challenging the reactor capacity, but not modelled here. The capacity limit is determined by the oxygen transfer and cooling capacity. During this time with constant feed rate the culture continues to grow but at a slowly decreasing rate.

5 Note on constraint-based modelling

The culture sub-model of the reactor is in many applications a static non-linear function $f()$ relating reaction rates $q[\,]$ of cell metabolism and growth to reactor broth con-

DOI
10.3384/ecp204453

centrations $c[\,]$. There are nc species in the broth and rates $q[i](t)$ defined for each of them at time t

$$q[i](t) = f_i(c[j](t)),\ j = 1...nc \tag{5}$$

The function is derived from the underlying system of equations of reactions rates. It is quite common that the system of steady state equations is under-determined and therefore complemented with constraints that determine the system. This leads to a formulation of the function $f()$ in an implicit way, as a solution of an optimisation problem. It is a convenience of modelling that has some similarity with analytical mechanics in physics.

An example showing cell growth on two substrates illustrates the idea. The example is a simplified version of yeast growing on glucose G and ethanol E. In this example the important fact that yeast can produce ethanol is dropped. Both substrates can be taken up and metabolise in parallel and the cell coordinate the uptake rates to maximize its specific growth rate μ

$$\mu = Y_{Gr} \cdot q_{Gr} + Y_{Er} \cdot q_{Er} \tag{6}$$

under the constraint of the cells limited respiratory capacity q_{O2}^{lim} see (Sonnleitner and Käppeli 1986)

$$k_{og} \cdot q_{Gr} + k_{oe} \cdot q_{Er} \leq q_{O2}^{lim} \tag{7}$$

At low substrate levels the uptakes are limited by the concentrations in the broth

$$q_{Gr} \leq \alpha G \tag{8}$$
$$q_{Er} \leq \beta E \tag{9}$$

Listing 7. Calculation of "culture" $f()$ using linear programming done with Optlang in Python

```
def f(G,E):
    qGr=Variable('qGr',lb=0)
    qEr=Variable('qEr',lb=0)

    mu_max=Objective(YGr*qGr+YEr*qEr,
        direction='max'
    qO2lim=Constraint(kog*qGr+koe*qEr,
        ub=qO2max)
    qGlim=Constraint(qGr,ub=alpha*max(0,G)
    qElim=Constraint(qEr,ub=beta*max(0,E)

    f.objective=mu_max
    f.add(qO2lim)
    f.add(qGlim)
    f.add(qElim)

    f.optimize()
    return (f,objective.value,
            f.variables.qGr.primal,
            f.variables.qEr.primal)
```

Thus, the optimization problem (6)-(9) gives the static function (5) that relates metabolic rates to the broth concentrations and describes the culture, see formulation in the high level Python package Optlang (Jensen, Cardoso, and Sonnenschein 2017) in Listing 7.

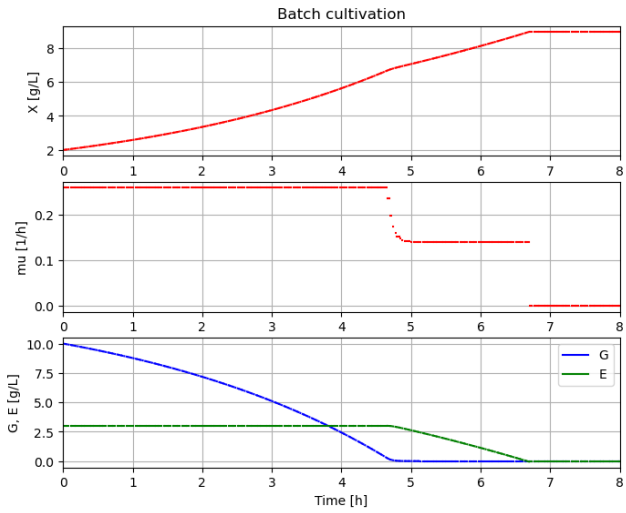

Figure 3. Simulation of batch cultivation with two substrates. CONF_2023_10_MODELICA15 at (*BPL Applications* 2023).

Listing 8. Simulation of BPL-model of bioreactor together with "culture" $f()$ done with FMU and PyFMI (or FMPy) together with Optlang in Python outlined (species index omitted)

```
n = t_final/t_delta
initialize c[0] and q[0]
c[1] = simulate(0, t_delta, c[0], q[0])
for i in 1:n:
    q[i] = f(c[i])
    c[i+1] = simulate('cont', c[i], q[i])
```

A simplified solution procedure to integrate bioreactor simulation with optimisation of culture metabolic rates at each time instant is based on the fact that here are two time scales. The concentrations $c[t]$ in the reactor broth change slowly compared to the cell culture optimisation of reaction rates $q[t]$. Thus, simulation a short step Δt of the integrated process is done with reaction rates $q[t]$ kept constant and gives concentrations $c[t+\Delta t]$. Then the culture reactions rates $q[t+\Delta t]$ are updated with an optimisation based on the concentrations $c[t+\Delta t]$ etc. The procedure is often called the "direct approach" and works well for a class of problems. Its limitations are discussed in section 2.2 and 4.4 in (Ploch, Lieres, et al. 2020). Key steps of the Python code are outlined in Listing 8.

In Figure 3 results from simulations using PyFMI (Andersson, Åkesson, and Führer 2016) are shown of batch cultivation with the two substrates G and E. For the culture to grow at maximal rate, first G is consumed and then E. Note that during a short period 4.7-4.9 hours both substrates are consumed in parallel. This occurs since levels of G is low and the constraint on oxygen allows some E to be consumed, and gradually more, and G vanishes.

In this example the model can very well instead be formulated in terms of a system of non-linear ODE as was done in the original publications (Sonnleitner and Käppeli 1986).

The advantage with constraint-based modelling is that the modelling can handle larger models and remain rel-

atively transparent. The public model repository (*EMBL BioModels* 2023) contains today about 1000 models and 20 percent of them use constraint-based modelling and the rest use mainly ODE. The public repository (*UC San Diego BiGG Models* 2023) contains about 100 large genome scale models with thousands of reactions all modelled using a constraint-based technique. The modelling software preferred is COBRApy that is based on Optlang, see (Ebrahim et al. 2013).

Thus, the framework presented here with BPL, FMU-simulation of the bioprocess setup and Optlang for handling constraint-based modelling of the culture, can integrate the two modelling techniques, see (Axelsson 2018; *BPL Applications* 2023).

It would likely be better to handle the constraint-based modelling within Modelica. With the Modelica extension Optimica at least the basic example above can be simulated with ease (Axelsson 2018). Here is also a recently developed Modelica toolbox for Differential Algebraic Embedded Optimization (DAEO) which address constraint-based models and seems very interesting (Ploch, Zhao, et al. 2019; Ploch, Lieres, et al. 2020). The drawback with a Modelica-solution is that models from the public repositories needs to be translated to Modelica, but can perhaps be automated.

6 Reflections on the library design

Here the BPL approach is compared with techniques of MSL Fluid and Media and some other libraries. The first sub-section addresses the scope of the library more clearly. In the later half of the section more broader questions are briefly discussed.

6.1 Limitations of the library

The focus of the Bioprocess Library is after all towards bio-pharmaceutical processes, often involving recombinant protein expression. This means that processes are usually operated in the temperature range $20-37°C$. Further the pressure is usually just above room pressure. The viscosity of liquids is like water. The reactor volumes are often up to 1000 L but occasionally 10000 L. Thus, several properties of media accounted for in MSL Fluid and Media are not that relevant.

If we look at biotechnology processes more broadly and include antibiotics, enzyme-production, baker's yeast production, breweries, or biorefinery industry, then the reactor scale may go up to 100 m^3, and reactor media are more complex and may be viscous. In this scenario MSL Fluid and Media can be more of help. One interesting application from biorefinery industry is (Ploch, Zhao, et al. 2019) where Modelica is used together with MSL Fluid and Media in combination with tailor-made modelling of bioreactor with a microbial culture.

6.2 Central vs local definition of media

The BPL has a structure where the medium is centrally defined for all components in `EquipmentLib`. This is in contrast to MSL Fluid where each component has an individual local definition of medium. There are certainly pros and cons with the two approaches.

There has been an expressed wish to in some way automate the process of choice of medium to simplify and ensure correctness of code. In the conclusion of (Franke et al. 2009) it is stated: *"The used medium has currently to be defined for every component. It would be nicer if the medium was defined at one source and the medium definition would then be propagated through the connection structure."*

The "propagation" of medium definition is in BPL done using "adaptation" of package using formal (type) parameters, as shown in the example in section 4.2. The idea behind using application package media as well as sub-models of the reactor as "type formal parameters" for the `EquipmentLib` package came from material in chapter 4.4 and an odd example in chapter 10.4 in (Fritzson 2015), combined with an urge for simplicity.

6.3 Equations vs components

The BPL defines reactions with equations rather than components. The reactions are grouped in four categories as outlined in section 3, and seen as sub-models to the Reactor component. The sub-models can be given an internal structure of interacting sub-sub-models to enhance readability and simplify modifications. Especially relevant for culture models. It is up to the user who writes the application code.

To introduce components for say the culture model is difficult. Different approaches to model biochemical networks are discussed in (Wiechert, Noack, and Elsheikh 2010). There is a public Modelica library, BioChem (Brugård et al. 2009), that brings component modelling to biochemical reaction network. The possibility to integrate BPL with BioChem has not been investigated, so far.

The focus of MSL Fluid and Media is on thermo-fluid modelling as is clearly stated in the MSL documentation. Thus, the focus of the library is not really on chemical reaction processes as pointed out earlier (Baharev and Neumaier 2012).

6.4 Code in consultant-customer relation

In my experience of consultancy work the customer is focused on results of using simulation rather than the simulation tool itself. In this context it is important to show and document the application code outlined in section 4. Provided the configuration of the process using standard components is clear and tested enough, there is little interest for the details of the library. The customer is usually satisfied to own the application code, while the consultant owns the library. If there is an interest, the consultant can deliver a compiled FMU and Jupyter notebooks that generate the results presented in the project.

DOI
10.3384/ecp204453

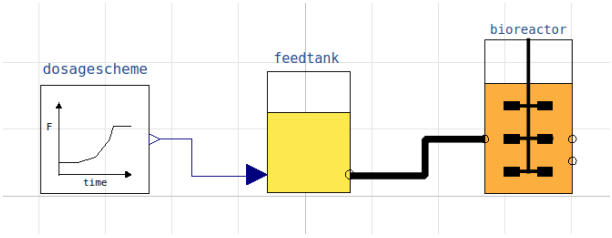

Figure 4. Graphical configuration of fed-batch cultivation in OpenModelica, see Listing 9. The formal (type) parameters Liquidphase and Culture of the library are not shown.

6.5 Challenges for the GUI

The library has been developed with little concern about a graphical user interface. One reason is that the (deprecated) software JModelica (Åkesson et al. 2010) has been used which lacks a GUI. Another reason is that there are differences between different vendors design of the GUI although the Modelica Standard Library sets an informal standard of what GUI facilities are implemented. However, the Modelica language is richer.

The challenges to configure BPL applications graphically are the following:

1. The package `BPL/EquipmentLib` has two formal (type) parameters for media: `Liquidphase` and `Gasphase`.
2. The `BPL/EquipmentLib/Reactor` has four formal (type) parameters for sub-models: `culture`, `broth_decay`, `pH_buffer`, and `gas_liquid_transfer`.
3. The `BPL/EquipmentLib/Reactor` use inner/outer implicit connection to the sub-models listed in the previous item. There is also a specific connection for each sub-model back to the `Reactor`.
4. The `BPL/EquipmentLib/Reactor` has also variable number of inlets and outlets and affect the icon of the component.

A first attempt to address GUI configuration in OpenModelica is shown in Figure 4, cf Listing 9. We see the model of fed-batch cultivation in section 4. Note that the electrical connector is represented by a thin line and the liquid phase connector with a thick filled line. For gas phase a thick open line is used, not shown.

6.6 Library modification for different GUI

Not all Modelica implementations have a GUI that supports parametrisation of packages, although the compiler does. This affects the design of the package `EquipmentLib` and the structure of the applications code. One approach to solve this dilemma is to introduce a parallel package `EquipmentLib2` where the formal (package) parameters for media as well as connector code, are moved into each component instead. Then adaptation to the application media is done at instantiation of the component instead. Thus Listing 5 and Listing 6 are then combined as shown in Listing 9, done by the GUI.

Listing 9. Application configuration when parametrisation of packages are not supported by the GUI

```
model Fedbatch "Fedbatch cultivation"
   Liquidphase_data liquidphase;
   EquipmentLib2.Reactor bioreactor(
      redeclare package Liquidphase =
          Liquidphase2,
      redeclare model Culture = Culture2,
      X = Liquidphase.X, n_inlets=1);
   EquipmentLib2.Feedsystem feedtank(
      redeclare package Liquidphase =
          Liquidphase2);
   Control.DosageSchemeExp dosagescheme;
equation
   connect(bioreactor.inlet[1], feedtank.
       outlet);
   connect(feedtank.Fsp, dosagescheme.F);
end Fedbatch;
```

The solution in Listing 9 has a structure similar to the MSL Fluid and Media. The drawback with this structure is that you need to redeclare media packages for each component. In OpenModelica this is done manually by the user. In the software Impact from Modelon there is also a possibility to automate this tedious and error-prone work in the GUI, see (*Modelon* 2023).

An alternative approach to keep redeclaration of media to one central place for components taken from package `EquipmentLib2`, is to introduce a global formal type parameter for the application code that the components media redeclaration refer to, see Listing 10 below.

Listing 10. Application configuration when parametrisation of packages are not supported by the GUI and global parameter for package `Liquidphase` introduced to centralize user interaction

```
model Fedbatch "Fedbatch cultivation"
   Liquidphase_data liquidphase;
   replaceable package Liquidphase =
       Liquidphase2;
   EquipmentLib2.Reactor bioreactor(
      redeclare package Liquidphase =
          Liquidphase,
      ...
```

The approach with a global formal type parameter can be found in some MSL Fluid examples, e.g. `ThreeTanks`, and the technique is in fact generally widely used. The drawback with this method is that it requires editing of the code manually, and can not be done with just using the (current) GUIs.

6.7 An idea for GUI improvement

The different Modelica GUI are similar and simplified to focus on components and their interconnections. The awareness of which package a certain component comes from is important at the time of configuration but later of little use. In the code view the programmer is naturally more aware of the package a component belong to.

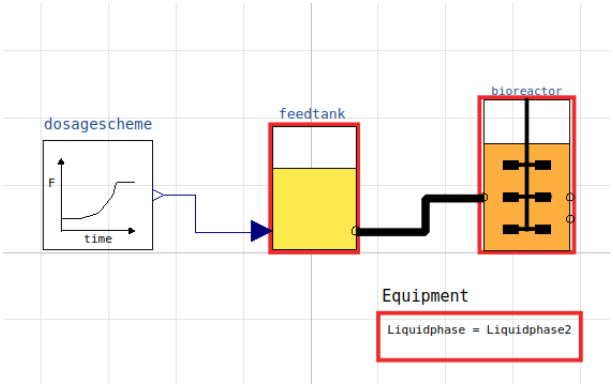

Figure 5. An idea to improve GUI by including access to package parameters for Equipment.

One way to make package information accessible in the GUI is shown in Figure 5. The package `Equipment` is represented by the box in the lower right corner. Clicking on this box shows general (type) parameters for the package and in this case the package `Liquidphase` and what that package is, and the user can change it. The code is then re-translated. Note that clicking on the box highlight the components in the configuration that belong to `Equipment` and affected by a change. A complement could be to in each component introduce a symbol for what package the component come from and when clicked the box with information of the package is shown and the other components in the configuration from the same package are high-lighted.

In MSL there are libraries where the need for expressing common properties for several components in a central way are addressed. The technique used is to define a central component that other components connect implicitly with using inner/outer variable mechanism. In the Fluid library the central component is called `system` and in the Multibody library it is called `world`. When the central component is defined on the top level it provides application wide access and can be seen as global parameters (*SystemComponent* 2023). This should be compared with the idea presented in this paper, namely package parametrisation using mechanism of inheritance. It make use of the package structure and is not global parameters. The pros and cons of the different approaches needs further analysis.

6.8 Two-level system configuration with GUI

In practice it is common to first settle the process setup and then explore various strategies for operation and control. However, in some contexts it can be fruitful to investigate the interplay of process and control design.

The process setup can effectively be coded using the procedure in section 4 and shown in Listing 11. The configuration provides an openness of how operation and control should be done. The model `Fedbatch_base` combines the feed tank with the bioreactor, but we do not include any dosage scheme. Further we equip the reactor with an on-line sensor for measurement of substrate concentration.

The operation and control of the process is then configured at a second level. Here we can let the process be operated by a fixed dosage scheme of the feed rate as before, or for example investigate the use of feedback control of the feed rate around the dosage scheme based on the on-line substrate measurement signal.

Listing 11. Application configuration fed-batch - level 1

```
model Fedbatch_base "Fedbatch cultivation"
   Liquidphase_data liquidphase;
   EquipmentLib.Reactor bioreactor(
      X=liquidphase.X, n_inlets=1,
         n_ports=1);
   EquipmentLib.ProbeSensor sensor(
       component=liquidphase.S);
   EquipmentLib.Feedsystem feedtank;
   Interfaces.RealInput Fsp;
   Interfaces.RealOutput S_measured;
equation
   connect(Fsp, feedtank.Fsp);
   connect(feedtank.outlet,
       bioreactor.inlet[1]);
   connect(bioreactor.port[1],
       sensor.probe);
   connect(sensor.probe.out, S_measured);
end Fedbatch_base;
```

This second-level configuration of the control system can be done using the GUI, as shown in Figure 6. Since we on this level only deal with electrical signals here is little difference between the GUI of different vendors, in this respect. Further, the advantage of exploratory GUI-configuration is more obvious than for the process setup.

Simulation during start-up of the process is shown in Figure 7. We see mainly that the control system is stable and keeps the substrate level at a low level close to the set-point during the exponential growth phase. To evaluate the advantage of substrate control compared to fixed dosage schemes requires a number of simulations taking into account variation in the initial cell concentration as well as variation in the culture parameters, and is outside the scope of this paper.

A two-level configuration process is natural in many situations and combine the strength of package parametrisa-

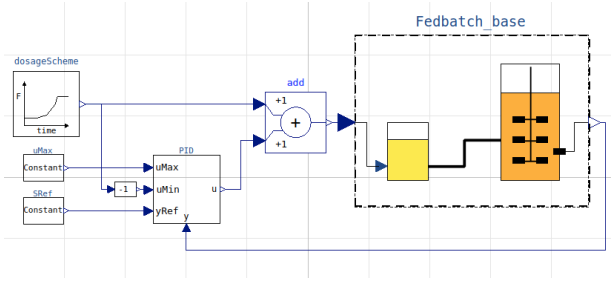

Figure 6. Application configuration fed-batch with feedback PID-control of feed rate from substrate measurement - level 2. The process setup is in the dash-lined box, see Listing 11.

DOI
10.3384/ecp204453

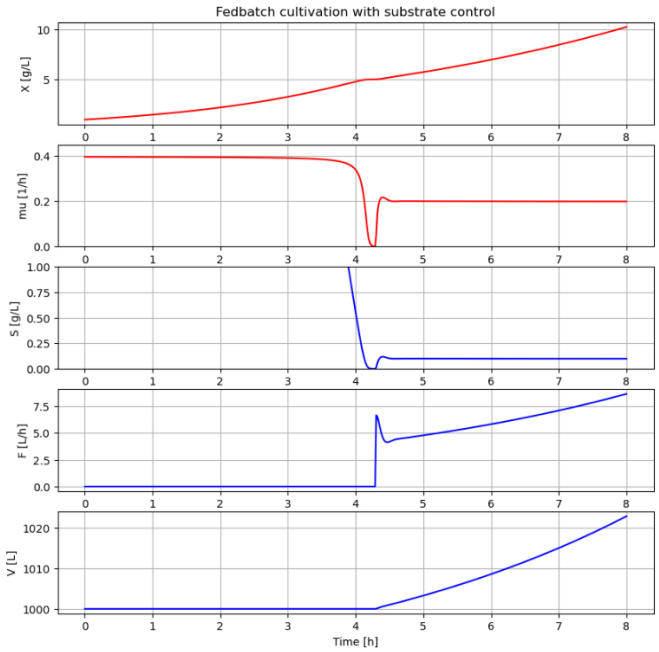

Figure 7. Simulation of fed-batch cultivation with substrate control from on-line substrate measurement and adjustments of the feed rate around the fixed dosage scheme. Focus on the start-up.

tion at the process level and the flexibility of GUI at the control system level. If you prefer, the first level configuration can also be done graphically. And if your Modelica software does not support package parametrisation you can use the procedure described in the previous section.

6.9 The importance of FMU for the library

For bioprocess simulations it is important to let it be part of the wider context of data analysis and detailed cellular modelling. Therefore compilation to FMU of high quality for further integration to Python (or Julia, Matlab) is a very important part.

One practical aspect is that information about media and culture etc in the Modelica model is good to have accessible from the FMU. This information is often declared as constants. Today vendors differ about whether constant information is available in the FMU, and OpenModelica has not implemented this facility, just yet.

7 Availability of Bioprocess Library

The library has been used in a few industrial projects over the years where I have been involved as consultant. It has also been used for teaching operators. These interactions have to some extent set the priorities made in the library development. The name Bioprocess Library is a registered trademark and owned by me.

I am now interested in that more people use the library and hope to get ideas back for further development. One possibility is to make it publicly available at GitHub. Information will be given at the GitHub-page, see (*BPL Applications* 2023).

At GitHub I already provide FMU-compiled demo examples with Jupyter notebooks that can be downloaded, or run from the web-browser using Google Colab virtual machines. The focus is here on further developing usage of simulation and post-processing using Python, by the GitHub-community. The examples include both the textbook culture model here, as well as models of microbial yeast (*S. cerevisiae*) and mammalian cell culture (CHO) for recombinant protein production. The cultures are run as batch, fed-batch, continuous and perfusion. The examples include dynamics of operation, model calibration, sensitivity analysis, design space calculation, scale-down, regulator tuning and process optimisation.

8 Concluding remarks

The presentation has focused on structural design aspects of the Bioprocess Library, and little on the content. Design aspects of more general interests are:

- Part of the user configuration is done by equations in a certain format as described in Section 3 and exemplified in Listing 4. The other part of the user configuration is done traditionally using components, as described in Section 4 and exemplified in Listing 6.
- The parametrisation of components with media are done on the package level instead of on the individual component level, as described in Section 4 and exemplified in Listing 5 and 6.
- The GUI varies between different vendors and not all supports parametrisation on the package level. Even for those vendors that allow GUI configuration as in Listing 6, it is not clear how to interact with the formal parameters on the package level using the GUI.

Hopefully, the library will lower the threshold to use Modelica for simulation of bioprocesses, and the unorthodox design appreciated. Suggestions for including important new components to facilitate usage are welcome! There is also a public place for developing and sharing Jupyter notebooks addressing bioprocess questions using simulation in combination with other tools.

Acknowledgements

I thank Stéphane Velut at Modelon (now at Emulate Energy) for encouraging my work on the library and especially for giving me an idea of what concepts to borrow from MSL Fluid and Media, as well as improvements of parts of the code. Thanks to John Batteh (Modelon) for review and thoughts around GUI. I also thank Adrian Pop at Linköping University for giving me feedback on how to simplify the adaptation of EquipmentLib to the actual application. I am also grateful for the Modelica Stackoverflow community for concrete help as well as ideas.

References

Åkesson, Johan et al. (2010). "Modelling and optimization with Optimica and JModelica.org - languages and tools for solving large-scale dynamic optimization problems". In: *Computers & Chemical Engineering* 34.11, pp. 1737–1749.

Andersson, Christian, Johan Åkesson, and Claus Führer (2016). *PyFMI: A Python package for simulation of coupled dynamic models with Functional Mock-up Interfaces*. Tech. rep. Lund University: Centre for Mathematical Science. URL: https://lucris.lub.lu.se/ws/portalfiles/portal/7201641/pyfmi_tech.pdf.

Axelsson, Jan Peter (2018-01). "Integrating microbial genome-scale flux balance models with JModelica and the Bioprocess Library for Modelica". In: *Proceedings of the 21st Nordic Process Control Workshop*. Åbo Akademi University, pp. 126–127.

Axelsson, Jan Peter (2019-08). "Simplified model of CHO-cultivation in Bioprocess Library for Modelica – some experience". In: *Proceedings of the 22st Nordic Process Control Workshop*. Technical University of Denmark, Lyngby, pp. 56–63. ISBN: 978-87-93054-88-2.

Axelsson, Jan Peter (2021-02). "Design aspects of Bioprocess Library for Modelica". In: *OpenModelica Workshop*. Linköping University.

Axelsson, Jan Peter (2022-03). "Interpretation of responses to bolus-feeding during CHO-fedbatch cultivation using an extended bottleneck model and simulation with Bioprocess Library for Modelica". In: *Proceedings of the 23st Nordic Process Control Workshop*. Luleå University of Technology, p. 34.

Baharev, Ali and Arnold Neumaier (2012-09). "Chemical process modelling in Modelica". In: *Proceedings of the 9th International Modelica Confererence*.

Bioreactor-scaling-tool (2023). URL: https://www.cytivalifesciences.com/en/us/solutions/bioprocessing/knowledge-center/simplify-bioreactor-scale-up-and-scale-down (visited on 2023-08-15).

BPL Applications (2023). URL: https://github.com/janpeter19 (visited on 2023-08-15).

Brugård, Jan et al. (2009-09). "Creating a bridge between Modelica and systems biology community". In: *Proceedings of the 7th International Modelica Confererence*. DOI: 10.3384/ecp09430016.

Ebrahim, Ali et al. (2013). "COBRApy: Constraint-based reconstruction and analysis for Python". In: *BMC Systems Biology* 7, pp. 74–79. DOI: 10.1186/1752-0509-7-74.

EMBL BioModels (2023). URL: https://www.ebi.ac.uk/biomodels/ (visited on 2023-08-15).

Franke, Rüdiger et al. (2009-09). "Stream connectors - an extension of Modelica for device-oriented modelling of convective transport phenomena". In: *Proceedings of the 7th International Modelica Confererence*. DOI: 10.3384/ecp09430078.

Fritzson, Peter (2015). *Principles of Object Oriented Modeling and Simulation with Modelica 3.3*. 2nd ed. Wiley. ISBN: 9781118859124.

Hu, Wei-Shou (2020). *Cell Culture Bioproces Engineering*. 2nd ed. CRC Press. ISBN: 9781498762861.

Jensen, Kristian, Joao G. R. Cardoso, and Nikolaus Sonnenschein (2017). "Optlang: An algebraic modeling language for mathematical optimization". In: *Journal of Open Source Software* 2.(9), p. 139. DOI: 10.21105/joss.00139.

Modelon (2023). URL: https://help.modelon.com/latest/guides/propagate_class/?h=propaga (visited on 2023-08-15).

Olsson, Hans et al. (2008-03). "Balanced models in Modelica 3.0 for increased model quality". In: *Proceedings of the 6th International Modelica Confererence*., pp. 21–33.

Ploch, Tobias, Eric von Lieres, et al. (2020). "Simulation of differential-algebraic equation systems with optimization criteria embedded in Modelica". In: *Computer & Chemical Engineering* 140. DOI: https://doi.org/10.1016/j.compchemeng.2020.106920.

Ploch, Tobias, Xiao Zhao, et al. (2019). "Multiscale dynamic modeling and simulation of a biorefinery". In: *Biotech. Bioeng.* 116, pp. 2561–2574. DOI: 10.1002/bit.27099.

Sonnleitner, Bernhard and Othmas Käppeli (1986). "Growth of *Saccharomyces cerevisiae* is controlled by its limited respiratory capacity: formulation and verification of a hypothesis". In: *Biotech. Bioeng.* 28, pp. 927–937.

SystemComponent (2023). URL: https://doc.modelica.org/Modelica%204.0.0/Resources/helpWSM/Modelica/Modelica.Fluid.UsersGuide.BuildingSystemModels.SystemComponent.html (visited on 2023-10-28).

UC San Diego BiGG Models (2023). URL: http://bigg.ucsd.edu (visited on 2023-08-15).

Wiechert, Wolfgang, Stephan Noack, and Atya Elsheikh (2010). "Modeling language for biochemical network simulation: reactions vs equation based approaches". In: *Adv Biochem Engin/Biotechnol* 121, pp. 109–138. DOI: 10.1007/10_2009_64.