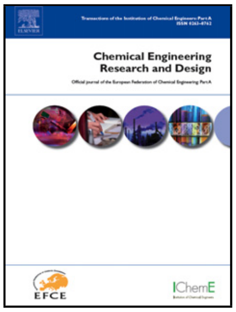


Discrete-Continuous Dynamic Simulation of Plantwide Batch Process Systems in MATLAB

Franz D. Bähner Oscar A. Prado-Rubio Jakob K. Huusom

PII: S0263-8762(20)30132-5

DOI: https://doi.org/doi:10.1016/j.cherd.2020.03.030

Reference: CHERD 4050

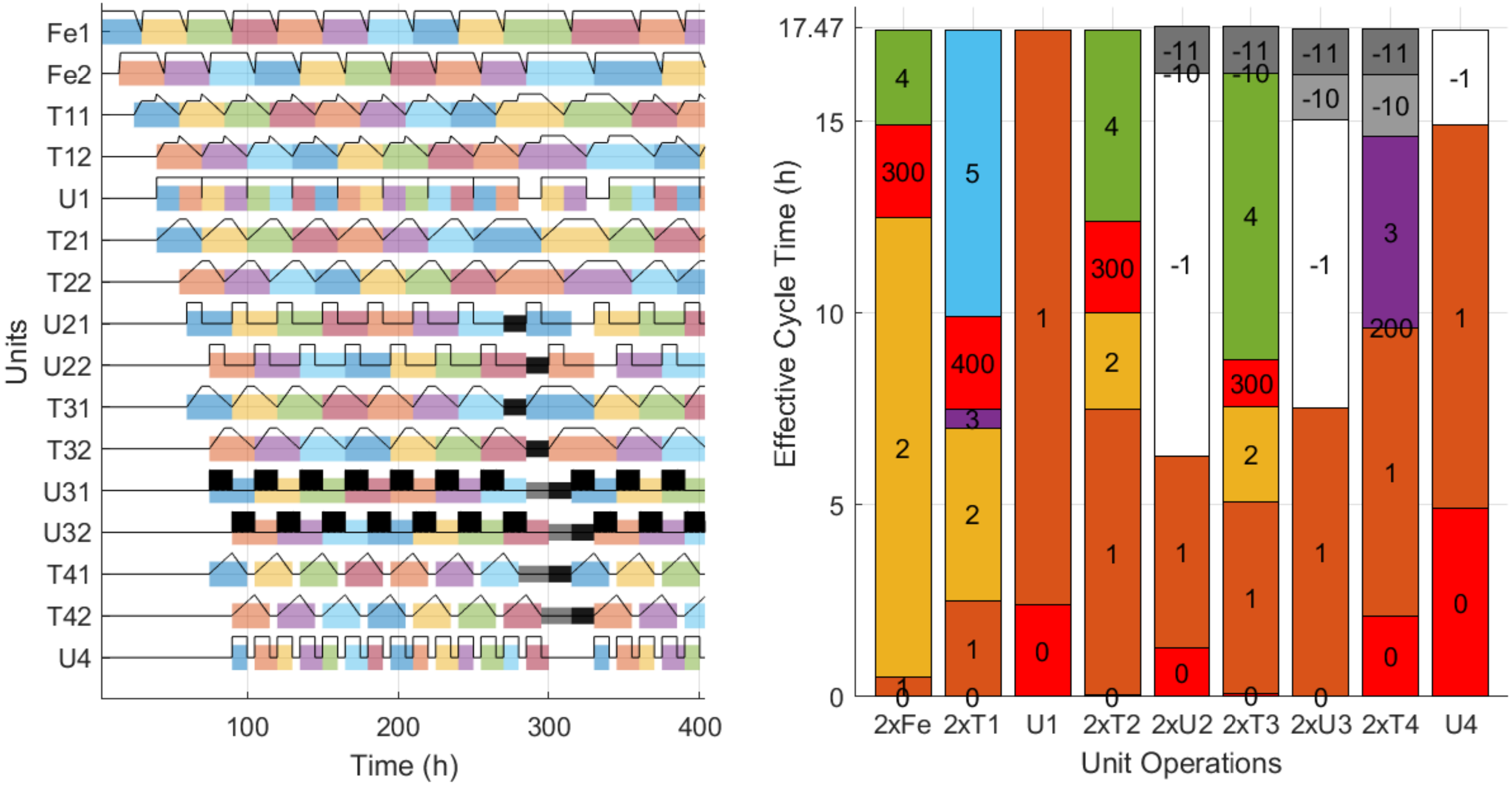
To appear in: *Chemical Engineering Research and Design*

Received Date: 16 January 2020

Revised Date: 27 March 2020

Accepted Date: 30 March 2020

Please cite this article as: Franz D. Bähner, Oscar A. Prado-Rubio, Jakob K. Huusom, Discrete-Continuous Dynamic Simulation of Plantwide Batch Process Systems in MATLAB, *<![CDATA[Chemical Engineering Research and Design]]>* (2020), doi: https://doi.org/10.1016/j.cherd.2020.03.030


Fe1
Fe2
T11
T12
U1
T21
T22
U21
U22
T31
T32
U31
U32
T41
T42
U4
Units
100
200
300
400
Time (h)
17.47
15
10
5
0
Effective Cycle Time (h)
0 1 2 300 4
0 1 2 3 400 5
0 1
0 1 2 300 4
0 1 -1 -10 -11
0 1 2 300 4 -10 -11
0 1 -1 -10 -11
0 1 200 3 -10 -11
0 1 -1
2xFe
2xT1
U1
2xT2
2xU2
2xT3
2xU3
2xT4
U4
Unit Operations

Highlights

- Modelling batch process systems w. shared resources in MATLAB/Simulink/StateFlow
- Visual cycle time analysis of different cleaning-in-place strategies
- Hybrid formalism which allows inclusion of continuous dynamics

Discrete-Continuous Dynamic Simulation of Plantwide Batch Process Systems in MATLAB

Franz D. Bähner[a], Oscar A. Prado-Rubio[b], Jakob K. Huusom[a,*]

[a] Process and Systems Engineering Center (PROSYS), Department of Chemical and Biochemical Engineering, Technical University of Denmark, 2800 Lyngby, Denmark
[b] Departamento de Ingeniería Química, Universidad Nacional de Colombia, Campus La Nubia Manizales, Caldas, Colombia

Abstract

Batch chemical and biochemical plants play an important role in the process industries. They are characterised by hybrid (continuous & discrete) dynamics as well as complex sequences and decision logic in the case of shared resources. This is challenging from a modelling and simulation perspective, both in terms of numerical algorithms as well as implementability and scalability/maintainability within software environments. In this work it is shown that it is possible to model complex plantwide batch processes at reasonably high performance, accuracy, and practicability in MATLAB/Simulink using the StateFlow toolbox. To this end, useful implementation guidelines are presented, and a complex example batch process is modelled. As focus lies on the implementability of complex batch control logic, the model is limited to mass balances. The simulation results are evaluated and carefully visualised, indicating the MATLAB's capabilities for analysis of such systems.

Keywords: Batch Process Systems, Process Modelling, Hybrid System, MATLAB Simulink StateFlow, Cycle Time Analysis

---

[*] Corresponding author
Email address: `jkh@kt.dtu.dk` (Jakob K. Huusom)

Preprint submitted to Chemical Engineering Research & Design *March 26, 2020*

1. Introduction

Batch processes play an important role in the production of speciality chemicals and pharmaceuticals (Croughan et al. (2015); Edgar (2004)). Compared to continuous processes, they suffer from a number of performance shortcomings (Teoh et al. (2016)). These include practical limitations originating from operational (scheduling) complexity which can lead to low equipment utilisation (Amaran et al. (2016)). Furthermore, energy- as well as material integration of batch processes may require advanced scheduling methods (Fernández et al. (2012)) or intermediate storages (causing additional operational and/or capital expenditure).

This complexity may be seen as a good argument for the use of process simulation in order to improve existing processes or design new ones (Foo & Elyas (2017)). Simulation of batch process systems is challenging compared to continuous processes as they ordinarily exhibit pronounced continuous and discrete dynamics. Discrete events occur as elements of the sequential batch control logic, whereas continuously integrated state space models are usually needed to describe to underlying physiochemical processes. Furthermore, Baldea & Harjunkoski (2014) point out the "inherently non-stationary nature, the non-linearity of their models, and the dynamic complexity that arises from the potential need to coordinate multiple units and stages that operate in parallel".

In industry, continuous and discrete system domains of batch process systems are ordinarily regarded in a separated approach: the scheduling domain is optimised in discrete-event manner (or an alike abstraction which focusses on material flows). Unit operation models may be based on mechanistic or data-driven continuously integrated models; dynamic plantwide phenomena are then omitted. Today, discrete-event simulation may be regarded a standard method as a number of software vendors cater to this market (for example ExtendSim®, SchedulePro®, INOSIM®, Simio®, and AnyLogic®). By means of manual optimisation or evolutionary algorithms they facilitate optimal equipment utilisation as well as efficient conduction of engineering projects.

The concurrent simulation of continuous and discrete system-elements (hybrid systems) on plantwide level is not yet a standard method. In industry, in many cases, the merits of integrating these layers into one simulation may not justify the added complexity. However, as Baldea & Harjunkoski (2014) or Costandy et al. (2018) point out, the integration of scheduling and control is an important area of research. This concerns especially processes where time constants of the continuous subsystems are so large that time-scale separation errors become relevant. Furthermore, advances have been made in data-driven modelling. This may facilitate the identification of complex reaction kinetics (Galvanauskas et al. (2018)), thereby enabling the inclusion of such detailed effects also in plantwide studies. In combination with abundant computation power - for instance through cloud solutions - this renders the computation of rigorous hybrid (here continuous-discrete) models feasible where it previously may not have been. Finally, benchmark models are invaluable as a driver of research progress and dissemination. To the best of the authors' knowledge, a rigorous continuous-discrete model of a complex batch process system in an academically accessible environment which enables the execution of advanced methods is not documented in open literature. (A good example for a continuous production plant is the re-implementation of the Tennessee Eastman Benchmark Problem by Bathelt et al. (2015)).

Concluding, there are several reasons which render continuous-discrete simulation of batch process systems an important topic of research. This article specifically examines MATLAB Simulink's capabilities for simulating complex plantwide batch process systems. It promotes using the StateFlow$^{®}$ toolbox (MathWorks (2019b)) in order to graphically program state charts (Harel (1987)). The use of StateFlow in modelling hybrid systems is not new (Simeonova (2008); Sahbani & Pascal (2000)), but an implementation which is comparable in terms of complexity with the example plant presented in the article at hand is, to the best of the authors' knowledge, not documented in open literature. Creating and maintaining complex plantwide models is a challenging task and much focus in the following is dedicated to handling this challenge in the MATLAB which, unlike the discrete-event simulators listed in the above, has not been specifically designed for this. On the other hand, it offers several advantages such as flexibility as well as the inclusion of data analysis and model building into one environment - which furthermore enables facile implementation of advanced methods.

The article is structured as follows: section 2 gives an overview of important classes of dynamic models as well as computational approaches of simulating them. This is followed by section 3, which entails a step-wise modelling framework. In the course of this, the article specifically discusses how to address the structural challenges that arise when modelling complex batch process systems in MATLAB/Simulink/StateFlow. Finally, the simulation results of an exemplary process are presented in section 4. This includes careful visualisation of the simulation results. The eligibility of the software environment for these types of simulation studies, implementation and computation challenges, and finally future work are discussed in section 5. Hereafter, the work is concluded.

2. Computation Approaches to Systems with Hybrid Dynamics

This section gives an overview of the discrete-event related part of the dynamic systems classified in table 1. Here, GDEVS refers to the (*generalized discrete event specification*) framework (Giambiasi & Carmona (2006)). In hybrid systems, discrete events occur not only in the form of (predictable) timers, but as a consequence of implicit, iterative algorithms such as numerical solutions to systems of differential equations. This is computationally challenging, and especially in optimisation studies with large computational overhead it is thus essential that the modeller finds the appropriate level of abstraction.

2.1. Discrete-Event Systems

In many instances it may be sufficient to neglect the continuous elements which greatly reduces the computational burden. Examples of this are discrete-event model based material flow analyses. They fit well the ambition to *model with a purpose* (Daoutidis et al. (2018)): the evolution of a tank level between *full* and *empty* is not usually essential for optimising a plant schedule. As long as the flow itself is predictable enough to know the points-in-time of full/empty, there is no information loss if a discrete-event model is chosen.

These models have largely been developed within the operations research community and are frequently used in discrete parts manufacturing, traffic studies, or supply chain optimisation (Bangsow (2012)). There exist two computation paradigms (Law & Kelton (2000)): the *next-event time advance* and the *fixed-increment time advance* clock update. If occurrence of the next event can be predicted in advance, next-event time advance is favourable. A fixed-increment time advance algorithm not only takes unnecessary steps, it also induces discretization error that scales with the fundamental step size unless all events are strictly multiples of the chosen sampling rate - this can be somewhat amended by using a variable-step solver. Systems for which the clock value of the next event can be computed in a straightforward manner are especially systems of timed automata (Alur & Dill (1994)) as well as multi-rate timed automata (Alur et al. (2000); Geist et al. (2008)). They allow discrete approximations of systems comprising mass flows, storage tanks, and time-based operations in batch production plants.

2.1.1. Discrete-Rate Simulation

Discrete-event models have been developed outside of the process systems engineering domain, yet they are applied successfully for problems arising within it (Petrides et al. (2014); Amaran et al. (2016)). Discrete-rate simulation (DRS), available for instance within the ExtendSim® environment, aims at amending some of the shortcomings of classical discrete-event models such as the periods of stasis between events. To this end, DRS allows states (inventories) to evolve on a continuous linear envelope between events. Negligence to do this can, in some cases, lead to accruing numerical error (Damiron & Krahl (2014)).

2.2. Hybrid Systems

Continuous behaviour is best described by systems of differential equations. Analytical solutions to these systems are normally not obtainable, thus a time-advance based on a list of next events is not possible. The coexistence of continuous and discrete dynamics requires that the solver is capable of handling both, and in the following two distinctions are drawn which are likely to influence the choice of solver.

2.2.1. Hybrid Systems with Frequent Discrete Events

If system dynamics are largely dictated by discrete elements, one can try to find local approximations of the continuous system trajectory in such a way that they can be handled by a discrete-event solver. This is not element of this article, and a large base of literature around the GDEVS framework is available (Giambiasi & Carmona (2006); D'Abreu & Wainer (2003); Giambiasi et al. (2001)). Furthermore, in the form of PowerDEVS (Bergero & Kofman (2011)), a Simulink-like process simulator with user-friendly graphic implementation features is available. These solvers are necessary if discrete events occur at very high frequencies, for instance during periods of *chattering*. Compared to the function evaluations needed to describe continuous system behaviour, discrete events occur at low frequencies in batch process systems. This frees the modeller from the need to pursue such an approach.

2.2.2. Hybrid Systems w. Infrequent Discrete Events

Numerical integration is a standard method of chemical engineering to solve systems of ordinary differential equations (ODE's). Also for stiff systems, a variety of performant implicit solvers (in the case of MATLAB for instance ODE15s, ODE23s) are available. Chemical engineers are generally familiar with these methods and skilfully balance numerical error with accuracy.

The question is then how to handle the discrete-event part of the system. From a computational perspective, a split system approach is preferable (Nutaro et al. (2012); Bouchhima et al. (2007); Clune et al. (2006)). In this scheme, continuous and discrete system fractions are calculated independently, and synchronisation only occurs when an event is triggered. This is attractive both in terms of computational performance as well as numerical error control during the numerical integration scheme.

However, especially in the case of complex systems with many elements, links, and transitions, implementation of split models may be cumbersome. Thus, one can choose to embed the discrete dynamics into the continuous solver regime. A good balance between practicality and performance is indicated by using variable step solvers with discrete-event detection. The advantages in implementing and maintaining these models may outweigh the disadvantages (accuracy, performance) as computation is fairly cheap. Software environments capable of this are i.e. gPROMSs®, Modelica®/Dymola, or MATLAB/Simulink® (van Beek & Rooda (2000)).

2.3. Hybrid Systems in MATLAB/Simulink

The solver capabilities in place, several attributes render MATLAB/Simulink attractive from a modellers point of view. Firstly, as an environment apt both for data processing and modelling, it manages to integrate two tasks which are ordinarily separated if dedicated process simulators are chosen. Furthermore, through numerous libraries/toolboxes, it allows facile implementation of advanced methods either within Simulink flowsheets or in the embedding MATLAB environment.

However, neither the user interface nor currently available libraries are designed for the implementation of complex batch process systems. Notably, the SimEvents® (MathWorks (2019a); Gray (2007)) toolbox is developed specifically for systems comprising discrete events and allows the presence of continuous dynamics (Clune et al. (2006)). However, it is optimised for systems consisting of queues and entities, which is not practical for the implementation of sequential/parallel hybrid batch process systems. This can be inferred from the predefined function blocks within the toolbox (entity generators and sinks, queues, servers). While they are useful elements of a high-level abstracted discrete-event study, they are not convenient within the context of a hybrid simulation that includes continuously evolving states. SimEvents expands Simulink by useful elements connected to entity-management, which in a process systems context is necessary for batch tracking. However, this can also be implemented in the StateFlow framework with reasonable effort (shown in section 3.2.7).

3. A Framework for Modelling Batch Process Systems in MATLAB/Simulink/StateFlow

In the following, a stepwise procedure is introduced which separates the modelling task into a series of sub-tasks. This is in general anticipated to be of great help due to the complexity of the endeavour.

3.1. Limitations of Continuously Solved Flow Charts

In the chosen simulation approach, state charts have to be solved under a continuous regime, which in MATLAB R2019b has the following implications:

- Library-links are disabled.
- No state transitions through event-broadcasting.
- Outputs cannot be written during state activity.

Furthermore, the absence of model libraries renders implementation tedious and therefore error-prone. Event-broadcasting within a flowchart is convenient in synchronising resources and callers (both of which there can be multiple). However, dynamic updates of outputs (for instance set points for manipulated variables) during state execution can usually be emulated on root flowsheet level. Note also that, if the chart was a pure timed automaton (all next future events are predictable at current event), it could still be executed in event-based manner also within a continuously solved flowsheet.

3.2. Stepwise Model-Building Procedure

In the following, the most important aspects of the model building procedure in Simulink and StateFlow are elaborated. They are concisely presented in figure 1, and each step is explicated in a dedicated sub-section. It is not strictly necessary to follow this sequence, but there is a rather natural order to it. In the proposed approach, a model has two layers: the batch control system (a StateFlow state chart) and a physical process counterpart. The latter is normally a system of differential-algebraic equations, modelled using integrators or S-functions on root flowsheet level. This is conceptually visualised in figure 2. The state chart layout is representative of a StateFlow implementation.

3.2.1. Step 1 - Model Configuration Parameters

Aside from general solver properties (tolerances, algorithm, etc.), the number of batches to be processed during a campaign is best specified in advance. A sufficiently long simulation horizon to process all batches should

be chosen; the simulation can be terminated prematurely when the last batch has been processed on the most-downstream units (and all machines have returned to idle state after finishing the last re-initialisation).
Simulating over such large time horizons may require controlling major integration step size as the default step size is large if left on automatic selection. In general, despite of the use of StateFlow under a continuous solver regime, cases are experienced where events are not properly detected if input values change rapidly compared to step size. This will usually be identifiable by implementing a series of simulation integrity checks (section 3.4). Therefore, adjusting model configuration parameters (step 1) is iterative by nature. This is especially so if no units with inherent step size requirements are installed (for instance pulse- or sine sources). Choosing the second-largest step size (by order of decimal place) which leads to exact solutions has shown to be a robust approach with good computational performance. The fact that numerical error in the discrete-event system part may occur is undesirable and the modeller needs to be alert.

3.2.2. Step 2 - Define Structure of Process System

Liquid or gaseous material must at all times be contained in a tank or an equivalent storage unit. Hold-ups of flow processing units (centrifuges, filters, etc.) are likely negligible. Systems can however be modelled to such a high degree of fidelity if that is required. In the case of solids that can be stored more flexibly, the requirement for a containing unit is relaxed. It is no problem to extend the model by storages with room for more than one (solid) batch, but this has not been implemented in this example. In the same way, it is not a problem to combine batches in one tank or split a batch in two. It is intuitive to choose a distributed modelling approach for the continuous part of the system. Each physical entity that stores material (buffer tanks, unit operations, etc.) is represented by an integrator or S-function. A second class of physical units are those that predominantly process material downstream (or recycle it) - continuously operating units. These can often be understood as resources needed by the tank units, which also require a free recipient tank before material can be sent downstream. Flow processing units can be modelled using arbitrary (for instance algebraic) function blocks. Naturally, also a complex dynamic model - embedded within for example an S-function - can be implemented.

3.2.3. Step 3 - Batch Control System

Deciding whether or not to decompose the batch control system into separate StateFlow state charts is less straightforward. Separate state charts are generally more intuitive to understand as they comply well with the concept of recipe-driven unit procedures. As the continuous solver regime forbids the use of library functions within the state chart environment, this furthermore allows mimicking object-oriented programming: local variable names can be re-used within separate state charts, which can therefore be copied easily. On the other hand, a separation leads to more complex signal routing on root flowsheet level: each time a variable is passed between charts, this requires that a link (graphic or virtual) is drawn. Finally, the decomposition into multiple charts led to stability issues in MATLAB R2019b and previous versions. These issues can be circumvented by choosing a fixed-step solver, which is however undesirable due to performance and accuracy set-backs (section 2). A centralised implementation (one superordinate flow chart) requires that the embedding chart is solved under *parallel* (AND) decomposition. It contains an embedded sub-chart for each unit and storage tank present in the system. These are all executed at the same time and initialised as *Idle*. They represent the actual machines which can only ever be in one state and must themselves be solved under *exclusive* (OR) decomposition.

Virtual units: it is possible to model resources within the control system that have no representation in the actual flowsheet. This might be convenient if they have no inherent dynamics and there is no interrelation with the process other than an effect on the schedule. Examples of this are cleaning-in-place stations or operators.

3.2.4. Step 4 - Interface: Batch Control System - Process

As indicated in figure 1, the layout of this interface is closely interrelated with the two consecutive steps (resource handling & material routing). Firstly, the interface needs to contain ports which allow passing measurements from the process to the control system. In a scenario limited to mass flows, this specifically concerns tank volumes/levels, but it is easy to extrapolate to temperatures, pH, or any variable obtained from a direct or inferential measurement. Secondly, the bi-directional interface needs to be able to pass command signals from the batch control system to the physical process. From a simulation perspective, these signals can be directly passed to the units. Yet, if the modeller chooses to do so, it is easy to implement a regulatory control layer in-between. This also enables the introduction of *implementation errors* in the control loop (useful for diagnostic studies), and extends the model by dynamics from feedback components in the lower layer. In practical terms, the interface depends on the structure of inputs and outputs of state chart and (unit operation) sub-systems. As a centralised state chart implies that numerous variables are passed on, these should conform to an intuitive and consistent nomenclature and array sequence (channel 1 - inflow, channel 2 - outflow, ...).

Beyond the signals on unit operation level, the control system needs to be able to implement resource handling / material routing on flowsheet level. Therefore, a further variable which essentially emulates valve positions in the piping system is necessary.

3.2.5. Step 5 - Resource Handling in Control System

If a plant layout is fixed (each upstream tank feeds via a standard unit to a fixed downstream tank), an implementations in StateFlow is trivial and this step is reduced to checking whether the statically assigned units are *Idle*. However, in modelling flexible batch plants, resource handling is one of the key challenges. It is exacerbated by the limitation that state charts cannot broadcast events under a continuous solver regime (section 3.1). This precludes the immediate transition of a resource unit from *Idle* to *Busy* upon being claimed by an caller.

A workaround is possible as all sub-charts within a superordinate chart can write to- and read from the local state chart workspace. Note that, also as a consequence of continuous solving, library functions are disabled, and the list of variables can quickly grow very long. Therefore, thoughtful and consistent naming of variables is essential.

Resource handling firstly entails the check for a free process path (often both, a processing unit and a recipient tank). The check has to cover all related downstream units; introducing a dedicated state for each path (state *toTank1viaUnit2*, ...) allows choosing preferred recipient units by assigning an order to the state transitions. The moment an upstream unit starts sending material downstream, *processing resource* and *recipient tank* must no longer be called from a further upstream unit. This is best controlled by the origin tank, and here the state chart workspace comes into play. As events cannot be broadcast to the resources (causing them to leave *Idle*), a *setter variable* in the shared workspace accounts for the utilisation state of the unit. The upstream caller immediately assigns this variable a new value when a resource is called. It should now be evident that upstream units check the state of this variable to find a feasible path. As resources might be busy in a procedure which is not linked to an upstream caller (re-initialisation, cleaning, maintenance), a check for *Idle* is still necessary. Each resource can reset the setter variable once the procedure is complete and the link to the caller broken.

3.2.6. Step 6 - Material Routing on Flowsheet level

It is crucial that the flow of each unit is specified only by one control system caller. Furthermore, every outflow must eventually turn into the inflow to another storage unit. In the case of systems of multiple sources and destinations, coordination of these mass flows is necessary. The most intuitive way to solve this is by using Simulink selector blocks which contain the necessary functionality: only one input can be passed through at a time (indicated by bold lines in figure. Beyond this, only inflows from external sources are allowed in downstream tanks. (Unlike in Modelica, in Simulink the modeller is not forced to model connectors in a way that promotes mass balance consistency.) The flow resource is the counterpart to a connector, and in this way, the material balance throughout the plant is closed.

The setter variable connected to a resource (introduced in the previous section) is helpful in the material routing problem: it is efficient to use it not only to hold (*Idle*/*Busy*) information, but to contain the ID of the upstream caller. If callers are enumerated regularly, these ordinals can be used to control the path through the selector blocks. If no caller is specified, the variable is to hold zero. This index is chosen as the default feed-through of the selector blocks - if this signal is also zero, it does not affect the mass balance. (Disabling sub-systems of tanks and other resources during inactivity creates redundancy.)

3.2.7. Step 7 - Implement Batch Tracking System

The functionality to track batches through the system is not required in order to be able to execute the simulation. However, it is important in posterior validation as well as evaluation of a simulation study. A batch ID can be created either in the queue or in the most-upstream unit, in the simplest case it counts up incrementally, which is easy in StateFlow and only requires a further local variable. As a receiver knows by which unit it was called, it can take over the batch ID from upstream units. If there is a dedicated state in each unit for each caller-resource / sender-resource combination (section 3.2.5), this is implementable with ease.

Not only the batch number is required to keep track of all statistics, the machine states (*Filling*, *Waiting for ...*, ...) need to be logged as well. Also here, consistent naming is crucial to render the system as understandable as possible. In this work, machine steps have been classified according to the following keys:

- 0 idle
- 1,2,3,... standard operations (nominal processing)
- 100,200,300,... waiting for resources / recipients during processing
- -1 re-initialisation
- -10 CIP called
- -11 CIP in progress

During nominal operation, the first cypher counts up continuously: an exemplary sequence reads $1, 2, 300, 4, 500, 6, -1$ for a unit with four operations $(1, 2, 4, 6)$, two waiting steps $(300, 500)$, and a re-initialisation step (-1). There is no standard number associated with a certain type of step (filling/emptying), and it is entirely up to the modeller to find an appropriate enumeration. Similarly, the assignment of negative values for re-initialisations and CIPs is arbitrary. Here it is chosen such that it facilitates selective plotting/colouring schemes.

3.3. Re-initialisation vs. Cleaning-In-Place (CIP)

With the above, the basics for putting together a functional batch process system in Simulink using StateFlow state charts are in place. However, several functionalities necessary for realistic modelling have not yet been introduced. These are i.e. equipment re-initialisations and CIPs. The distinction is drawn as CIPs or sterilisation-in-place (SIPs) are understood as plantwide issues which require coordination with other units and a CIP system (resource), whereas unit operation re-initialisations are local. Inclusion of a virtual CIP station (free/busy) in the batch control system is sufficient, naturally a *physical entity* can be implemented on root flowsheet level if this is desirable. Either process may require operator attendance (resource); the operator model (busy/free/activity) can be implemented in the same way as a virtual CIP station. A re-initialisation or CIP may be called after each batch, after a certain time, after a certain number of batches, after a certain event-occurrence on a batch, or after a transient variable for some unit operation (for instance fouling) crosses a threshold.

3.4. Validation

As indicated by Tiwari (2002), especially for large systems with complex input patterns it is a challenging task to verify whether a state machine reacts to arbitrary input patterns in the desired way. To this end, formal verification methods exist that are not element of this work. However, validation of complex batch campaigns is an issue that needs to be addressed. The identification of faulty sequences on a unit operation is not a problem, as visual verification for several scenarios can, with relatively high certainty, confirm that the sequence is implemented properly. (Furthermore, implementing sequences on unit operation level is relatively straightforward.)

Resource handling and material routing are substantially more error-prone. It would greatly upset the fidelity of a simulation if a batch could be lost or created 'out of nowhere' in the middle of the downstream line. (Or, in the worst case, both - which renders detection difficult.) Keeping track of the total number of processed batches allows to deduce whether simulation integrity is maintained or not. Visual or automatic checking of cycle times and volume profiles under different solver options gives the modeller a quick feeling for both implementation and numerical issues.

4. Implementation of an Example Plant

In the following, the implementation of a reference batch process system is presented. It is inspired by the industrial case study documented in Bähner & Huusom (2019). Beyond modelling, focus is put on posterior graphic evaluation to give the reader an intuitive understanding of the type of process which has been simulated. In terms of operational complexity (i.e. dynamic selection of units and cleaning operations) the model does not stand back from batch or hybrid processes documented in open literature (Montes et al. (2018); Alshekhli et al. (2010); Monroy & Vallejo (2013); Sharda & Bury (2010); Toumi et al. (2010); Noguera & Watson (2004). The generic example line is fed by two fermenters; before each set of unit operations, two holding tank are installed. Some process steps contain parallel machines, others only one. A schematic overview is given in figure 3.

4.1. Model Specifications

A description of the operational procedures in terms of constraints and rates is given in table 2. A simple campaign is visualised in Gantt chart notation in figure 4. Here, a campaign of four batches (colour-code) is shown. Scaled volumes and flow profiles are plotted for the sake of understandability. A re-initialisation occurs when a unit is still *coloured* due to attribution to a batch while no flows are processed (units U21/22, U31/32, and U4). Flows on units U31/32 are chosen to alternate frequently to show the possibility. Rapid changes in flow rates may require step size control (section 3.2.1) due to numerical error. Cycle times on the machines are designed such that, normalised for the number of available units, all process steps take 15 hours per batch. That is with one exception: due to the irregular re-initialisation schedule on unit 4 (after every second batch), there is minor theoretical overcapacity. This stems from the instances in which the machine is idle while no re-initialisation occurs, as the timeslot needs to be reserved if a fixed schedule is to be implemented.

4.1.1. Example CIP Procedure

To study complex schedules arising from cleaning-in-place events, the model is extended by CIP routines. This firstly requires the introduction of a CIP station in the batch control system (here reduced to a virtual resource, section 3.2.3). CIP stations are often shared between production lines and may block more than one machine at a time, for instance when a tank is needed in order to CIP a unit. Therefore, all machines that are subject to a CIP need to be extended by the related states, these are i.e. *CIP called* and *CIP in progress*.

The scenario is designed such that a CIP covers units 21/22 up to tanks 41/42, as indicated in figure 3. It is called every 8.75 days, thus fitting exactly into the schedule. It follows a rigid procedure which can be seen in figure 5. In the standard CIP sequence, firstly unit 21 and tank 31 are cleaned congruently. Upon completion, unit 31 and tank 41 are blocked to enable a consecutive CIP (grey bar). Once unit 22 and tank 32 have processed the last batch, they proceed to active CIP. Blocking the downstream units from further processing guarantees a coherent CIP barrier between the pre- and post-CIP batches. When the CIP on unit 22 and tank 32 is completed, unit 31 and tank 41 can proceed to active CIP, and as in the above, unit 32 and tank 42 are blocked from processing a batch to prevent cross-contamination until they are cleaned in the final CIP routine. Each cleaning of a unit/tank group lasts exactly 15 hours - the constraining cycle time in the system.

A second procedure follows a different pattern: after cleaning U21/T31, U31/T41 are subjected to a CIP. Consecutively U22/T32, and finally U32/T42 are cleaned. This is shown in figure 6, and visual assessment reveals that less waiting is experienced in this scenario.

4.1.2. Effect of CIPs on Equipment Efficiencies

It can be seen that the first introduced CIP procedure in figure 5 leads to a significant amount of blockage due to units being taken out of operation in anticipation of a CIP. As the CIP duration is designed such that it actually fits into the schedule, this is suboptimal and leads to long cycle times induced by the step *waiting for CIP* (step -10), visualised in figure 7-a. Cycle times of the second (improved) procedure are presented in figure 7-b. They lie notably below those in the previous CIP design and lead to a 5.4% capacity increase.

The schedule indicated in figure 6 (dashed rectangle) exhibits an interesting property, namely the coincidental starting and finishing of the filling procedure in tanks 41/42. In the designed case it does not matter in which order the tanks are processed on unit 4, as it leads to equal waiting periods. It is however a good example of complex scheduling decisions which are not trivial to make without support through technological tools, as it would require the proper course of action if capacities were leveraged.

4.2. Behaviour Under Stochastic Uncertainty

Randomised waiting can easily be added in within the control system by introducing a random timer. This is representative of manual control, as operators may react delayed. Beyond that, the physical system on Simulink flowsheet level can easily be extended by random effects. As a simple example, in the following the flow rates on U21/U22 unit are subjected to Gaussian noise (flow rate values are kept constant during a cycle). To this end, the inlet flow rate of $1.5m^3/h$ is superposed a random term $\Delta_{F,i}$ with

$$\mathrm{var}\left(\Delta_{F,U21}\right) = 0.05 \tag{1}$$

$$\mathrm{var}\left(\Delta_{F,U21}\right) = 0.075 \tag{2}$$

As equipment capacities in the process are relatively even, system performance does not benefit from short cycles, but prolongations propagate up- and downstream. An overview of the affected step durations on tanks 21/22 is shown in figure 8, and it becomes evident that variability is not only experienced in the affected material transfer step, but also periods of idle time are experienced due to short cycles and upstream delays. The according cycle times are shown in figure 9.

4.3. Computational Performance

Due to the great number of additional function evaluations under a continuous solver regime - compared to a pure discrete-event system - performance differences are present. The overall computation workload is strongly linked to the differential equation solver, the nature of the continuous system, and the allowed for error tolerances. A rigorous hybrid simulation of a campaign of several batches may thus result in substantial execution times and should be kept in mind.

In the example simulation, the system is reduced to piecewise linear mass balances between time- and state discrete events. The system is solved with ODE15s on an Intel® Core™ i5-5300U CPU which is rated at 2.3 GHz. Execution time scales linearly with the number of batches in a campaign and the Simulink flowsheet for a duration of 500 batches executes in less than 25 seconds.

In this calculation, a maximum step size of 0.1 - which in the given system corresponds to hours - and the standard absolute and relative ODE15 error tolerances are chosen. The maximum step size does not equate incurred error on event detection; the identification of these events is accurate and in the case of intrinsic timers exact. However, if this tolerance is left unchecked, the solver tends to miss chains of events entirely as the linearity of the continuous system may trigger excessively large integration step sizes. These can result in missed zero-crossings of the event detection system. Increasing maximum step size to 1 (hour) reduces the execution time to under 20 seconds; all events are still identified properly.

Performance after the inclusion of complex continuous dynamics remains to be investigated, but at least it is indicated that the execution of the discrete part of the system can be included in the holistic modelling approach without inhibiting performance drastically.

5. Discussion

It can be concluded that the MATLAB/Simulink/StateFlow environment is apt for modelling and simulating batch process systems. This is expected, not least due to the fact that it has been used for applications of this kind before. However, the implementation which has been presented in the article at hand surpasses them in complexity, and guidelines have been introduced which aid in structuring a complex model building process. Overall, there are only few environments which can handle true continuous-discrete models, especially when it comes to systems with numerous elements and complex sequential procedures (such as batch process plants).

The Simulink/StateFlow environment allows facile study of the interplay between continuous and discrete systems. An exemplary phenomenon of interest would be the effect of proportional-integral controller tuning on time-scale separation error. Another example would be the quantification of the gains from being able to terminate a fermentation subject to biological variability based on process analytical technology rather than a fixed schedule. In general, if dynamic phenomena connected to product quality or yield are in need of quantification, this calls for an environment which can handle continuous system elements (Costandy et al. (2018)). While the models in Simulink/StateFlow are not accessible to mixed-integer solvers, black-box optimisation algorithms can be tried. Furthermore, the models can be used for the sake of validating an abstracted optimisation model (Vieira et al. (2019)).

The proposed framework allows the generation of hybrid data sets based on mechanistic models which resemble those of real production sites to a very high degree. Here, the Simulink flowsheet environment gives the modeller intuitive control of the inputs and thus the occurring effects. The modelled behaviour can exceed mere random uncertainties, which is expectedly an Achilles' heel of many machine learning algorithms. Therefore, this framework might be seen as a first steps toward creating a sandbox environment for facile testing and validation of data-driven algorithms before they are tried in real production environments. (Here, typically a large number of unknowns and uncertainties are beyond the analyst's control). The value of accepted benchmark models such as the Tennessee Eastman Proces (Downs & Vogel (1993)), or the Benchmark simulation model no 2 (Jeppsson et al. (2007)) in disseminating maturity and aptitude of technologies between academic but also industrial researchers has been pointed out many times (see for instance Huusom (2015); Downs (2012)). Unfortunately, numerical accuracy of the simulations needs to be asserted through integrity checks which are likely to require some manual evaluation. On the other hand, despite of Simulink's well understood capabilities for solving continuous systems (Klee & Allen (2016)), it is not unusual that simulation accuracy and performance are balanced by means of iterative tuning. Therefore, this should not be considered a disadvantage compared to other software environments. Simulation studies of campaigns consisting of several batches are likely to require lengthy computation, therefore this approach is not apt for real-time hard tasks.

While the chosen environment allows modelling batch process systems of some complexity, limitations arise which one needs to be aware of. Generally, it is not advisable to model complex

- Multi-purpose plants (plants without a fixed topology)
- Multi-product plants with severely different recipes (differing not only in parameter values, but recipe sequences)

To some extent, this is a consequence of the restriction which arise in continuously solved state charts (such as limitations in object-oriented modelling practice and event broadcasting). Therefore, the model-building process is likely too tedious and a recipe-based discrete-event simulator is a much more appropriate environment. Not least, it is unlikely that detailed reaction kinetics and unit operation models for a wide variety of products are available.

On the other hand, for plants with fixed layouts and moderate product diversity (or dedicated to one product) it is possible to build models in a straightforward, graphically supported way. Here, the benefits of the integrated MATLAB environment (data pre-analysis and parameter identification, modelling & simulation, posterior data analysis and optimisation) can be exploited - while offering facile inclusion and study of continuous effects. Furthermore, simple manual scheduling studies (Georgiadis et al. (2019)) which are ordinarily conducted in discrete-event simulators can be executed effectively. MATLAB could be assessed with respect to its abilities for pure discrete-event system studies of complex batch process systems. While the modelling effort is still going to exceed that in simulators dedicated to the cause, recipes can be implemented in a straightforward way using StateFlow, especially as the limitations from section 3.1 are mediated if a pure discrete-event solver is chosen. Furthermore, the SimEvents® toolbox offers (strongly abstracted) standard blocks which might be useful for such models. MATLAB's integrated functionalities and widespread availability (i.e. due to its academic licensing scheme) would enable effective method development, for instance related to automatic derivation or validation of discrete-event models based on batch process data.

6. Conclusion

In this work, applied guidelines have been presented that support constructing sequential/parallel hybrid batch process system models in

MATLAB. An example plant has been simulated, and the capabilities for posterior data visualisation and analysis have been shown. Model-building in MATLAB entails some challenges which arise on the one hand from the lack of standardised functionalities, and secondly from several limitations in StateFlow as a consequence of a continuous solver regime. These difficulties render the environment unattractive for industrial applicants who need quickly-implementable solutions. Furthermore, it is inapt for systems with high combinatorial complexity; still, it is shown that the simulation environment allows the creation of holistic, non-linear, continuous-discrete plantwide models of reasonably complex systems. Data sets can be generated which closely resemble those of real batch process systems - with full and intuitive control of the modelled phenomena and especially disturbances. In the future, an implementation of a batch process system benchmark model in MATLAB would enable easy access throughout the academic community as well as facile testing and development of new methods.

Acknowledgements

The authors would like to acknowledge the Technical University of Denmark (DTU) and BIOPRO2. The project received financial support from Innovation Fund Denmark through the BIOPRO2 strategic research center (Grant number 4105-00020B).

References

Alshekhli, O., Foo, D. C., Hii, C. L., & Law, C. L. (2010). Process simulation and debottlenecking for an industrial cocoa manufacturing process. *Food and Bioproducts Processing*, *89*, 528–536. URL: `http://dx.doi.org/10.1016/j.fbp.2010.09.013`. doi:`10.1016/j.fbp.2010.09.013`.

Alur, R., & Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, *126*, 183–235. doi:`10.1016/0304-3975(94)90010-8`.

Alur, R., Henzinger, T. A., Lafferriere, G., & Pappas, G. J. (2000). Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, *88*, 971–984. doi:`10.1109/5.871304`.

Amaran, S., Sharda, B., & Bury, S. J. (2016). Targeted Incremental Debottlenecking of Batch Process Plants. In T. M. K. . Roeder, P. I. . Frazier, R. Szechtman, T. . Huschka E. Zhou, & S. E. Chick (Eds.), *Proceedings of the 2016 Winter Simulation Conference* (pp. 2924– 2934).

Bähner, F. D., & Huusom, J. K. (2019). A Debottlenecking Study of an Industrial Pharmaceutical Batch Plant. *Industrial & Engineering Chemistry Research*, *58*, 20003–20013.

Baldea, M., & Harjunkoski, I. (2014). Integrated production scheduling and process control: A systematic review. *Computers and Chemical Engineering*, *71*, 377–390. URL: `http://dx.doi.org/10.1016/j.compchemeng.2014.09.002`. doi:`10.1016/j.compchemeng.2014.09.002`.

Bangsow, S. (2012). *Use Cases of Discrete Event Simulation*. doi:`10.1007/978-3-642-28777-0`.

Bathelt, A., Ricker, N. L., & Jelali, M. (2015). Revision of the Tennessee eastman process model. *IFAC-PapersOnLine*, *28*, 309–314. URL: `http://dx.doi.org/10.1016/j.ifacol.2015.08.199`. doi:`10.1016/j.ifacol.2015.08.199`.

van Beek, D., & Rooda, J. (2000). Languages and applications in hybrid modelling and simulation: Positioning of Chi. *Control Engineering Practice*, *8*, 81–91. doi:`10.1016/s0967-0661(99)00137-9`.

Bergero, F., & Kofman, E. (2011). PowerDEVS: A tool for hybrid system modeling and real-time simulation. *Simulation*, *87*, 113–132. doi:10.1177/0037549710368029.

Bouchhima, F., Brière, M., Nicolescu, G., Abid, M., & Aboulhamid, E. M. (2007). A SystemC/Simulink co-simulation framework for continuous/discrete-events simulation. In *BMAS 2006 - Proceedings of the 2006 IEEE International Behavioral Modeling and Simulation Workshop*. doi:10.1109/BMAS.2006.283461.

Clune, M. I., Mosterman, P. J., & Cassandras, C. G. (2006). Discrete Event and Hybrid System Simulation with SimEvents. In *Proceedings of the 8th International Workshop on Discrete Event Systems* (pp. 386–387). doi:10.1109/wodes.2006.382398.

Costandy, J. G., Edgar, T. F., & Baldea, M. (2018). A scheduling perspective on the monetary value of improving process control. *Computers and Chemical Engineering*, *112*, 121–131. URL: https://doi.org/10.1016/j.compchemeng.2018.01.019. doi:10.1016/j.compchemeng.2018.01.019.

Croughan, M. S., Konstantinov, K. B., & Cooney, C. (2015). The future of industrial bioprocessing: Batch or continuous? *Biotechnology and Bioengineering*, *112*, 648–651. doi:10.1002/bit.25529.

D'Abreu, M., & Wainer, G. (2003). Models for continuous and hybrid system simulation. In *Proceedings of the 2003 Winter Simulation Conference*. New Orleans, LA, USA: IEEE. doi:10.1109/wsc.2003.1261479.

Damiron, C., & Krahl, D. (2014). A Global Approach for Discrete-Rate Simulation. In *Winter Simulation Conference* (pp. 2600–2608). doi:10.1016/j.copbio.2004.09.001.

Daoutidis, P., Lee, J. H., Harjunkoski, I., Skogestad, S., Baldea, M., & Georgakis, C. (2018). Integrating operations and control: A perspective and roadmap for future research. *Computers and Chemical Engineering*, *115*, 179–184. URL: `https://doi.org/10.1016/j.compchemeng.2018.04.011`. doi:`10.1016/j.compchemeng.2018.04.011`.

Downs, J. J. (2012). Industrial Perspective on Plantwide Control. In G. P. Rangaiah (Ed.), *Plantwide Control: Recent Developments and Applications*. doi:`10.1002/9781119968962.ch2`.

Downs, J. J., & Vogel, E. F. (1993). A plant-wide industrial process control problem. *Computers and Chemical Engineering*, *17*, 245–255. doi:`10.1016/0098-1354(93)80018-I`. `arXiv:1722`.

Edgar, T. F. (2004). Control and operations: When does controllability equal profitability? *Computers and Chemical Engineering*, *29*, 41–49. doi:`10.1016/j.compchemeng.2004.07.013`.

Fernández, I., Renedo, C. J., Pérez, S. F., Ortiz, A., & Mañana, M. (2012). A review: Energy recovery in batch processes. *Renewable and Sustainable Energy Reviews*, *16*, 2260–2277. doi:`10.1016/j.rser.2012.01.017`.

Foo, D. C., & Elyas, R. (2017). *Introduction to Process Simulation*. Elsevier Inc. URL: `http://dx.doi.org/10.1016/B978-0-12-803782-9.00001-7`. doi:`10.1016/B978-0-12-803782-9.00001-7`.

Galvanauskas, V., Simutis, R., & Lübbert, A. (2018). Hybrid modeling of biochemical processes. In J. Glassey, & M. von Stosch (Eds.), *Hybrid Modeling in Process Industries* chapter 5. (1st ed.).

Geist, S., Gromov, D., & Raisch, J. (2008). Timed discrete event control of parallel production lines with continuous outputs. *Discrete Event Dynamic Systems: Theory and Applications*, *18*, 241–262. doi:`10.1007/s10626-007-0023-2`.

Georgiadis, G. P., Elekidis, A. P., & Georgiadis, M. C. (2019). Optimization-Based Scheduling for the Process Industries : From Theory to Real-Life. *Processes*, *7*, 438.

Giambiasi, N., & Carmona, J. C. (2006). Generalized discrete event abstraction of continuous systems: GDEVS formalism. *Simulation Modelling Practice and Theory*, *14*, 47–70. doi:10.1016/j.simpat.2005.02.009.

Giambiasi, N., Escude, B., & Ghosh, S. (2001). GDEVS: A generalized discrete event specification for accurate modeling of dynamic systems. In *Proceedings - 5th International Symposium on Autonomous Decentralized Systems, ISADS 2001*. doi:10.1109/ISADS.2001.917452.

Gray, M. A. (2007). Discrete Event Simulation: A Review of SimEvents. *Computing in Science and Engineering*, *9*, 62 – 66. doi:10.1109/MCSE.2007.112.

Harel, D. (1987). Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, *8*, 231 – 274. doi:10.1016/0167-6423(87)90035-9.

Huusom, J. K. (2015). Challenges and opportunities in integration of design and control. *Computers & Chemical Engineering*, *81*, 138–146. doi:10.1016/j.compchemeng.2015.03.019.

Jeppsson, U., Pons, M. N., Nopens, I., Alex, J., Copp, J. B., Gernaey, K. V., Rosen, C., Steyer, J. P., & Vanrolleghem, P. A. (2007). Benchmark simulation model no 2: General protocol and exploratory case studies. *Water Science and Technology*, *56*, 67–78. doi:10.2166/wst.2007.604.

Klee, H., & Allen, R. (2016). *Simulation of dynamic systems with MATLAB and simulink, second edition*.

Law, A. M., & Kelton, W. D. (2000). *Simulation modeling and analysis*. (3rd ed.). McGraw-Hill Education. doi:10.1145/1667072.1667074.

MathWorks (2019a). SimEvents. URL: https://se.mathworks.com/products/simevents.html.

MathWorks (2019b). Stateflow. URL: https://se.mathworks.com/products/stateflow.html.

Monroy, D. F. Z., & Vallejo, C. C. R. (2013). Production planning and resource scheduling of a brewery with plant simulation. In *Use Cases of Discrete Event Simulation: Appliance and Research*. doi:10.1007/978-3-642-28777-0_15.

Montes, F. C., Gernaey, K., & Sin, G. (2018). Dynamic Plantwide Modeling, Uncertainty, and Sensitivity Analysis of a Pharmaceutical Upstream Synthesis: Ibuprofen Case Study. *Industrial and Engineering Chemistry Research*, *57*, 10026–10037. doi:10.1021/acs.iecr.8b00465.

Noguera, J. H., & Watson, E. F. (2004). Analyzing throughput and capacity of multiproduct batch processes. *Journal of Manufacturing Systems*, *23*, 215–228. doi:10.1016/S0278-6125(04)80035-9.

Nutaro, J., Kuruganti, P. T., Protopopescu, V., & Shankar, M. (2012). The split system approach to managing time in simulations of hybrid systems having continuous and discrete event components. *Simulation*, *88*, 281–298. doi:10.1177/0037549711401000.

Petrides, D., Carmichael, D., Siletti, C., & Koulouris, A. (2014). Biopharmaceutical Process Optimization with Simulation and Scheduling Tools. *Bioengineering*, *1*, 154–187. URL: http://www.mdpi.com/2306-5354/1/4/154/. doi:10.3390/bioengineering1040154.

Sahbani, A., & Pascal, J. C. (2000). Simulation of Hybrid Systems Using Stateflow. *In 14th European Simulation Multiconference (ESM'2000)*, (pp. 271–275).

Sharda, B., & Bury, S. J. (2010). Bottleneck analysis of a chemical plant using discrete event simulation. In *Proceedings - Winter Simulation Conference*. doi:`10.1109/WSC.2010.5678916`.

Simeonova, I. (2008). *On-line periodic scheduling of hybrid chemical plants with parallel production lines and shared resources*. Doctoral thesis Universite catholique de Louvain.

Teoh, S. K., Rathi, C., & Sharratt, P. (2016). Practical Assessment Methodology for Converting Fine Chemicals Processes from Batch to Continuous. *Organic Process Research and Development*, *20*, 414–431. doi:`10.1021/acs.oprd.5b00001`.

Tiwari, A. (2002). Formal semantics and analysis methods for Simulink Stateflow models. *Unpublished report, SRI International*, . URL: `http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Formal+Semantics+and+Analysis+Methods+for+Simulink+Stateflow+Models{#}0`.

Toumi, A., Jürgens, C., Jungo, C., Maier, B. A., Papavasileiou, V., & Petrides, D. P. (2010). Design and optimization of a large scale biopharmaceutical facility using process simulation and scheduling tools. *Pharmaceutical Engineering*, *30*, 1–9.

Vieira, M., Moniz, S., Gonçalves, B., Pinto-Varela, T., & Barbosa-Povoa, A. P. (2019). Integrating Simulation and Optimization for Process Planning and Scheduling Problems. In *29th European Symposium on Computer Aided Process Engineering* (pp. 1441–1447).

<table>
<tr><th>Type</th><th>Linear</th><th>Non-linear</th></tr>
<tr><td>Continuous Systems</td><td><i>Continuous LTI models (Discrete LTI approximations*)</i></td><td><i>Differential (P)D(A)E Systems</i></td></tr>
<tr><td rowspan="2">Discrete-Event Systems</td><td><i>Multi-Rate Timed Automata</i></td><td><i>Automata, Petri Nets</i></td></tr>
<tr><td colspan="2" rowspan="2"><i>Frequent Events: GDEVS Formalism</i></td></tr>
<tr><td rowspan="2">Hybrid Systems</td></tr>
<tr><td><i>Scarce Events: Discrete-Rate Simulation**</i></td><td><i>Scarce Events: (Complete Batch Process Systems)</i></td></tr>
</table>

Table (1) Dynamic system types and common mathematical model expressions.
*Discrete computation of linear time-invariant (LTI) systems is trivial as the sampling rate is constant.
**Continuous evolution of volumes between events is considered in this otherwise discrete modelling framework (section 2.1.1).

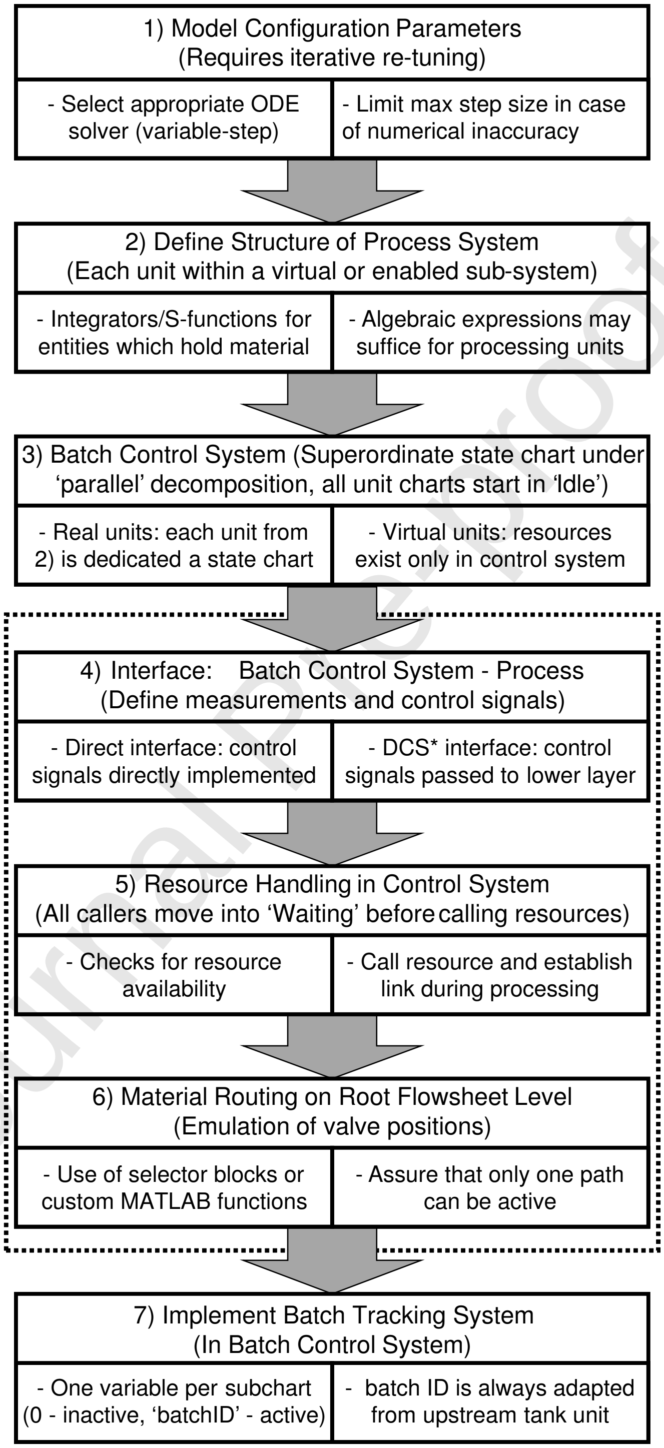


Figure (1) List of the most important steps concerned with building a batch process system model in MATLAB/Simulink/StateFlow. The steps are delineated in detail in the subsections within section 3.2.
(* DCS: Distributed Control System)

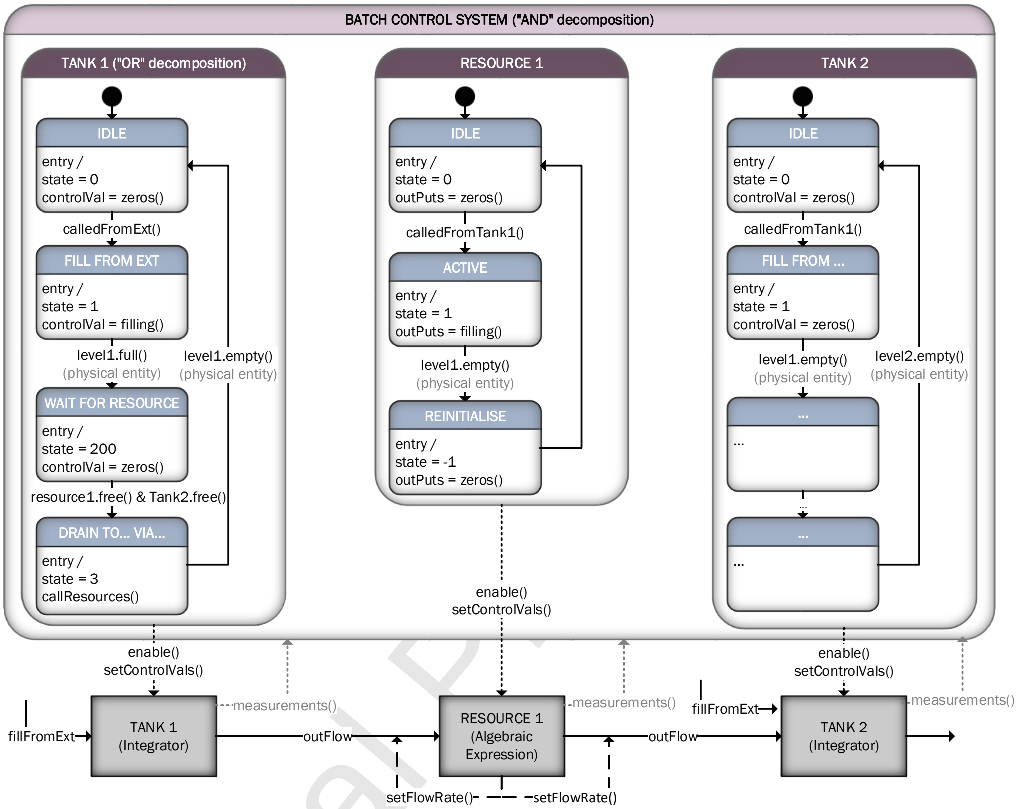


Figure (2) Exemplary architecture if one superstate with parallel decomposition of sub-states (machine states) and continuous elements on root flowsheet level is chosen.

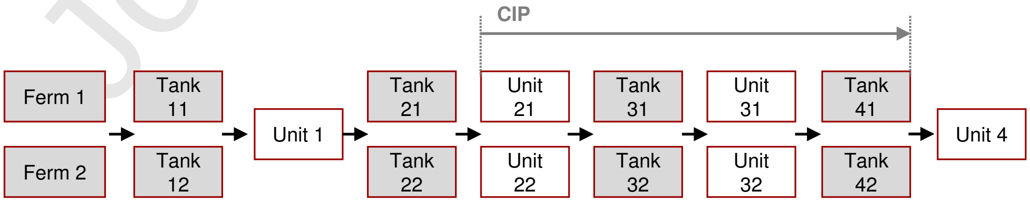


Figure (3) Overview of units in production line to be modelled.

Unit	**Constraint**			
Steps (excl. idle)	Value	Rate	ID	Dur. (h)
Ferm 1/2				
Fill from ext.	10 m^3	10 m^3/h	1	1
Ferment	24 h		2	24
Wait for R.			300	
Drain	0 m^3	2 m^3/h	4	5
Tank 11/12				
Fill	10 m^3	2 m^3/h	1	5
Hold	9 h		2	9
Fill from ext.	+5 $m^{3\star}$	5 m^3/h	3	1
Wait for R.			400	
Drain (via Unit 1)	0 m^3	1 m^3/h	5	15
Unit 1				
Processing	15 m^3	1 m^3/h	1	15
Tank 21/22				
Fill	15 m^3	1 m^3/h	1	15
Hold	5 h		2	5
Wait for R.			300	
Drain (via U21/22)	0 m^3	1.5 m^3/h	4	10
Unit 21/22				
Processing	15 m^3	1.5 m^3/h	1	10
Reinitialise	15 h		-1	20
Tank 31/32				
Fill	15 m^3	1.5 m^3/h	1	10
Hold	5 h		2	5
Wait for R.			300	
Drain (via U31/32)	0 m^3/h	1 m^3/h	4	15
Unit 31/32				
Processing	15 m^3	1 m^3/h	1	15
Reinitialise	15 h			15
Tank 41/42				
Fill	15 m^3	1 m^3/h	1	15
Wait for R.			200	
Drain (via Unit 4)	0 m^3	1.5 m^3/h	3	10
Unit 4				
Processing	15 m^3	1.5 m^3/h	1	10
Reinitialise	5 $h^{\star\star}$		-1	5

---

*Amount of material added relative to current fill level.

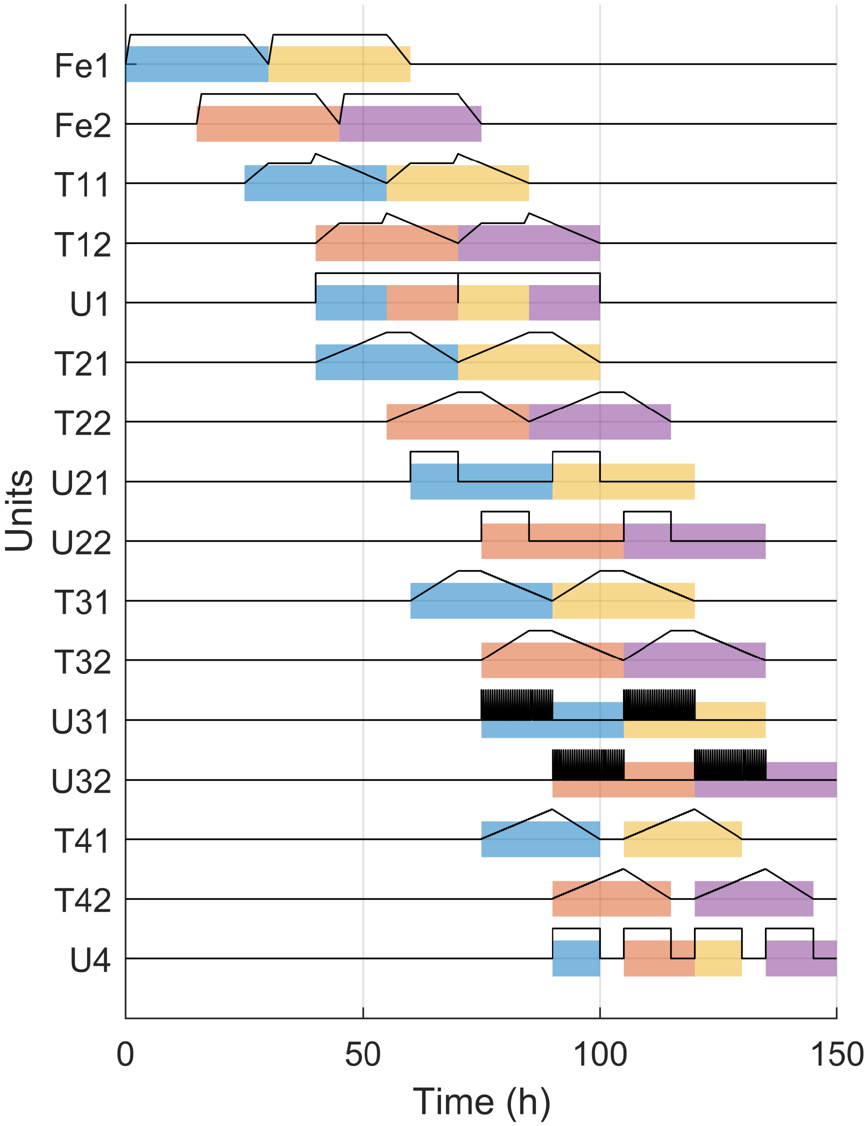


Figure (4) Exemplary campaign of four batches (colour-code).

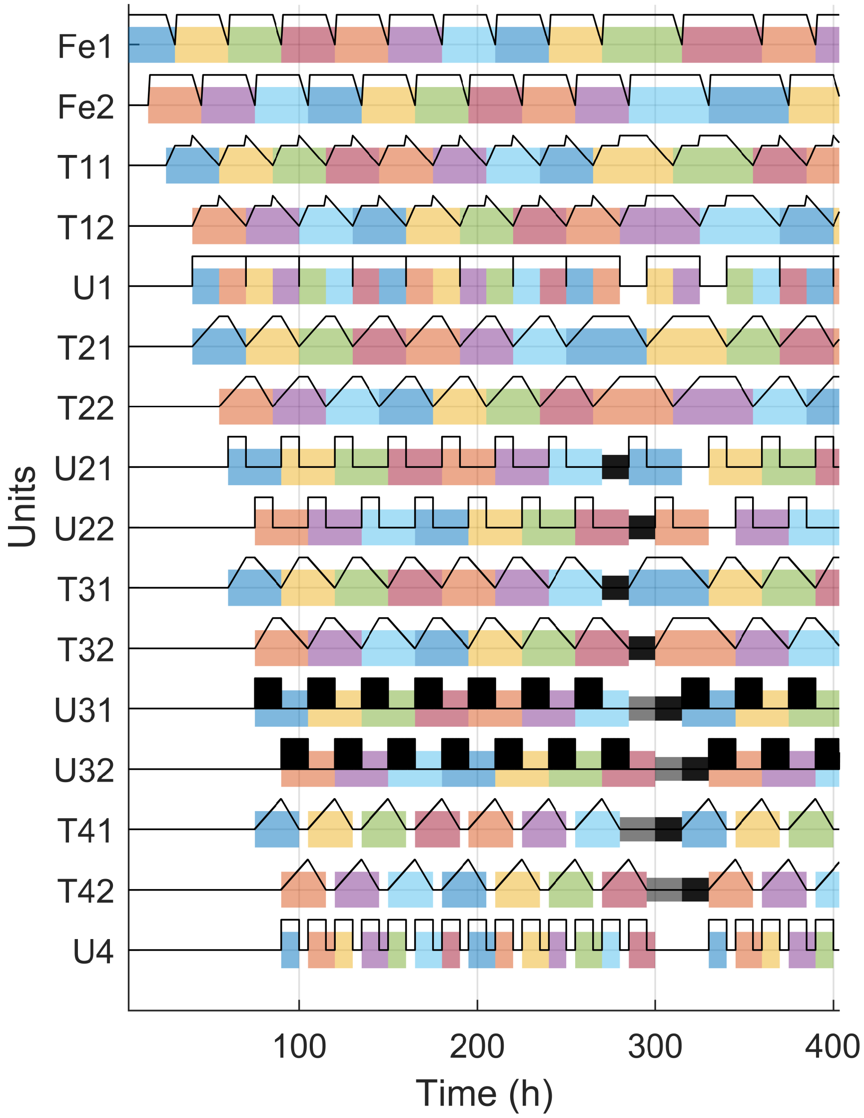


Figure (5) CIPs on units 21/22 & 31/32, tanks 31/32 & 41/42. Gray bar: unit blocked, CIP system busy. Black centred bar: CIP.

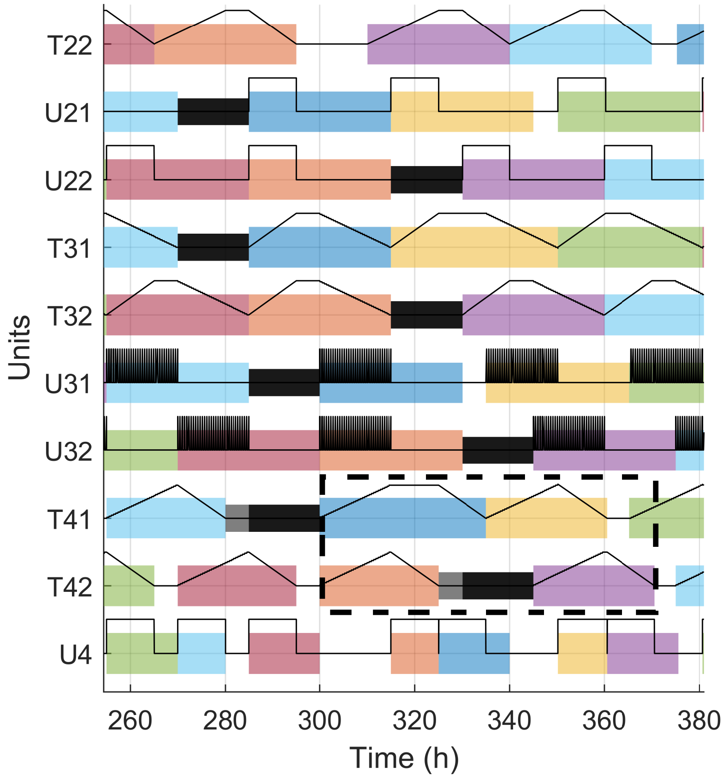
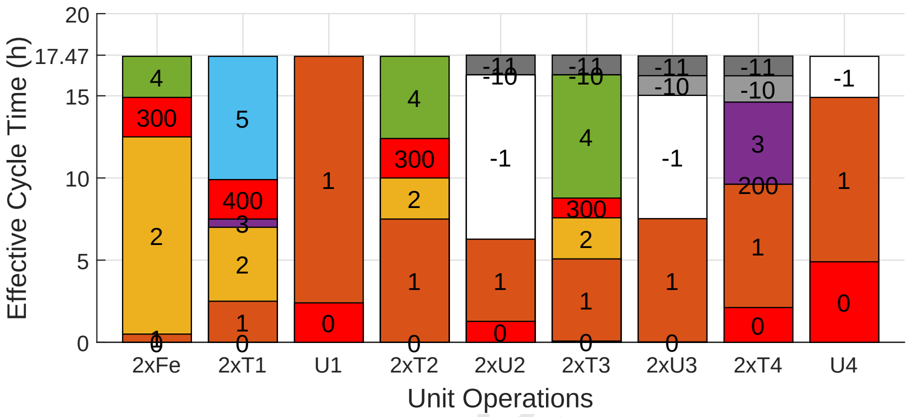
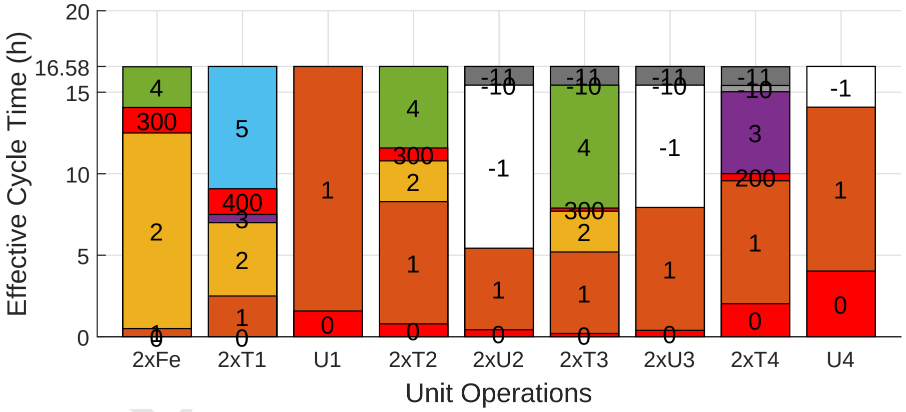


Figure (6) Excerpt of improved CIP schedule with reduced waiting time.



(a) Line capacity under standard CIP policy.



(b) Line capacity under improved CIP schedule.

Figure (7) Effective cycle times for a campaign of 300 batches. Step nomenclature: 0:idle, 1,2,3...:processing, 200,300,...:waiting, -1:reinitialisation, -10:blocked by CIP-call, -11:CIP in progress

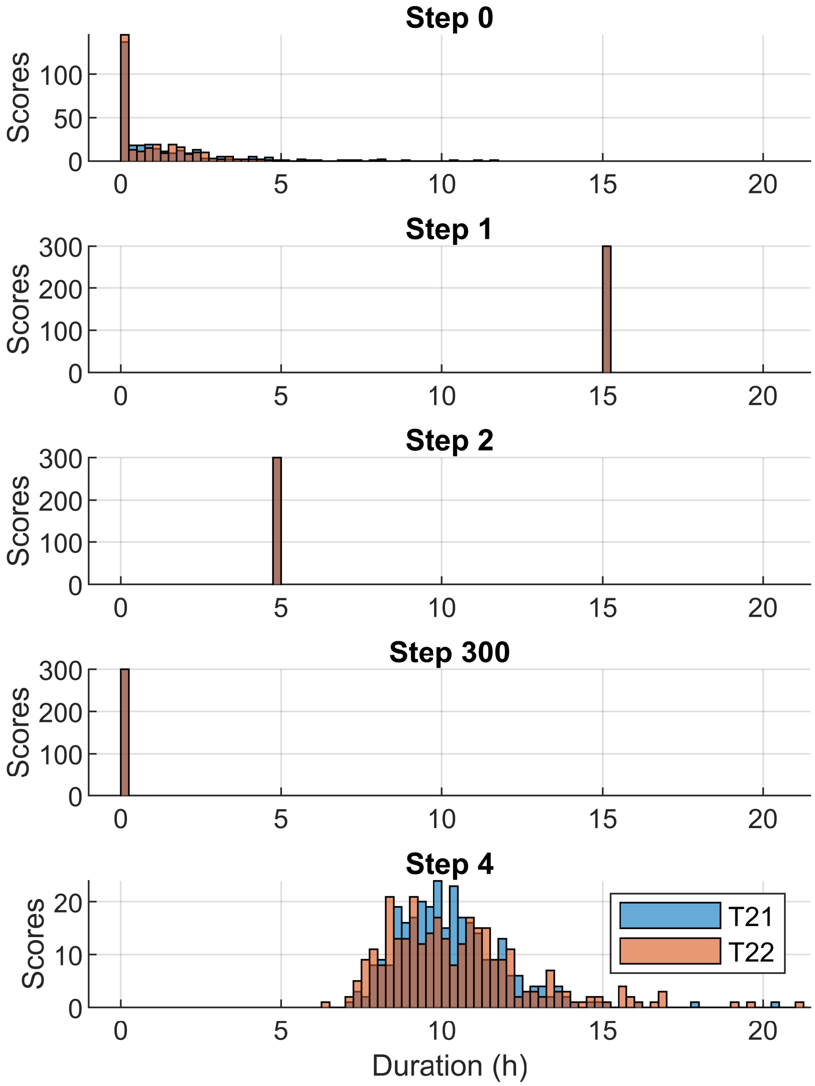


Figure (8) Durations of operations on tanks 21/22 as a consequence of the randomised flow rates on the downstream processing units.

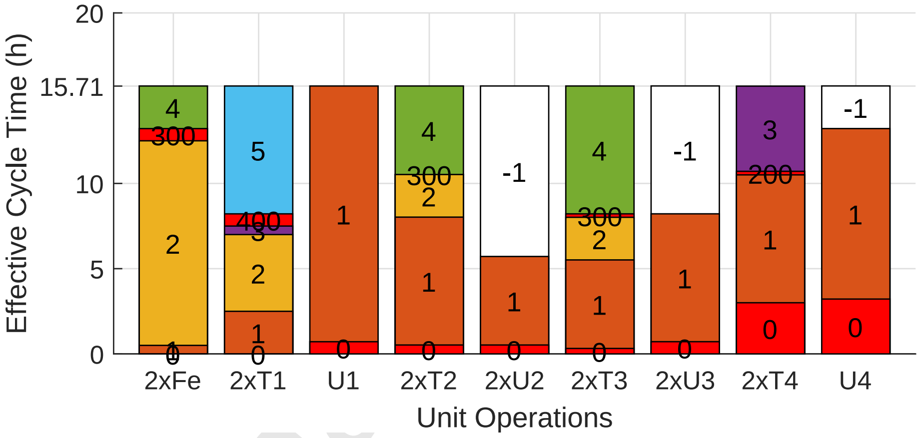


Figure (9) The equipment utilisation throughout the plant as a consequence of variability on the flow rates on U21/U22.