Simulation Scheduling of Variable-Structure Systems in OpenModelica

Rahul Paknikar[1] Nikhil Sharma[2] Priyam Nayak[2] Kannan Moudgalya[2] Bhaskaran Raman[1]

[1]Department of Computer Science and Engineering
[2]Department of Chemical Engineering

Indian Institute of Technology Bombay, Mumbai, India
`e-mail: kannan@iitb.ac.in`

Abstract

We propose and implement a generic scheduling framework for OpenModelica to eliminate the simulation code corresponding to inactive components in a system-level model. This framework allows the model developer to auto-generate models corresponding to the discrete behavior of the underlying system, and then schedule their simulations. It also provides a `Scheduler` library in the Modelica language to help the model developer easily generate the schedule. The benefit of this approach is demonstrated with and without real-time simulations of a batch distillation system. The proposed approach also helps implement a sequential modular simulation to arrive at initial guesses for flowsheets, whose equations can then be solved simultaneously using the standard, equation-oriented, approach of Modelica.

Keywords: Schedule, OPC UA, OpenModelica, Batch distillation, Steady-state, Variable-structure modeling, Sequential modular simulation

1 Introduction

The ability to model discrete behavior is an important requirement in industrial systems, even in continuous plants, such as refineries. The reason is that these plants also need to be cold started and shut down in case of emergencies. Unless startup and shutdown procedures are clearly understood, one cannot even take the lab-scale discoveries to the plant level for manufacturing. Hence, the ability to correctly model discrete behavior is an important requirement.

System-level modeling of a huge and complex system has become a common methodology for system engineering design (Pop et al. 2019). Such modeling can include several hierarchies of subsystems and a large number of components. Most general-purpose simulators broadly fall into one of the following three categories based on the way they process the information in the system under simulation: discrete event, continuous, and hybrid, i.e., a combination of the continuous and discrete-event simulation.

The simulators treat the system-level modeling as a single unit for efficient compilation to generate the simulation code and run the simulation itself. That is, the system-level modeling design is no longer preserved, and all its hierarchy and components are compiled and simulated entirely. A similar software framework is observed as a part of the typical process of translation and execution of a system-level model in most of the Modelica tools (Fritzson 2014), including the open-source OpenModelica compiler (Pop et al. 2019). Such frameworks can lead to the compilation and simulation of inactive subsystems and components, leading to some difficulties.

In order to address the main issue of constantly running simulation code that is not required, we propose a generic scheduling framework. It describes the discrete behavior of the system to be modeled in terms of a schedule. This framework also helps schedule their simulations (Section 2). Section 3 discusses the related work within and outside the Modelica context. In Section 4, we present a prototype developed for this framework that leverages the OpenModelica Simulation Environment (Fritzson et al. 2020) along with a Modelica library called `Scheduler`. Sections 5 and 6 demonstrate the benefits of the proposed approach in a batch distillation column and process flowsheeting, respectively. The last section is devoted to the conclusion and future work. In our work, the system-level modeling primarily involves variable-structure systems.

2 Scheduling Framework

To understand the motivation behind this work, let us consider the following example. Consider the cold startup of an overflow tank connected to a pumped flow tank, as shown in Figure 1. Initially, there is no liquid in both tanks. Until the liquid level rises in the first tank to the level of the outflow line, there can be no liquid in the second tank. Without the liquid, starting the pump will damage it. In addition, suppose that we have to calculate the liquid density in the second tank. Unless suitable precautions are taken, there could be difficulties in the calculation because of the zero volume of the liquid. There are three different ways to handle this situation:

1. Noticing zero volume in the tank, do not calculate the density nor implement any calculation that requires density.

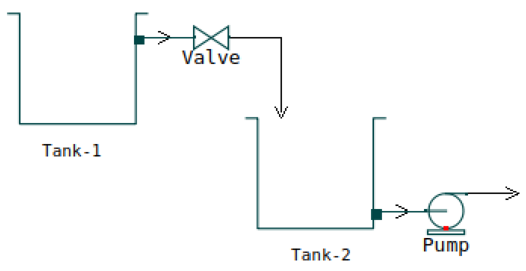


Figure 1. System-level model of two interacting tanks

2. Assume that there is always a small amount of liquid present in the second tank and calculate the density.
3. Do not even attempt to simulate the second tank until the liquid starts coming from the first tank.

The first approach increases the load on the model developer. They must anticipate all the different ways the modeling assumptions can be violated and take corrective actions. The second approach results in erroneous calculations, however small they may be.

The above two approaches necessitate solving all subsystems, including the inactive ones. This approach is not acceptable if the objective is to do minimal modeling, such as arriving at the initial guess of a large number of model equations by solving a few equations at a time, as explained in Section 5. The advantage of the minimal modeling approach is that it does not require setting up initial guesses for inactive models. The third approach does not have the difficulties mentioned above, provided we have the capability to simulate at the correct time. In this work, we attempt to create this capability to simulate only the required models.

2.1 Schedule for System-Level Modeling

The active components or subsystems of a system can be effectively determined through its discrete behavior. The same can be modeled through discrete events in terms of a schedule. Thus, the model developer constructs a schedule of dependent tasks and events for system-level modeling. An event indicates a condition based only on the currently scheduled task. It specifies the next task to be scheduled when the event triggers. I.e., the condition is satisfied. Note that the condition can involve just the simulation time so that it does not constrain the variable-structure model to generate events from its state. A task can include a set of components, subsystems, or several systems. It will be simulated only when activated based on the target of the event associated with the currently scheduled task. From the above description, the following are the observed properties of any schedule for system-level modeling:

- An event is associated with only one task and specifies only one task to be scheduled next. A task can have several events associated with it, which can schedule only one of the several next tasks depending on the order of the events being triggered.

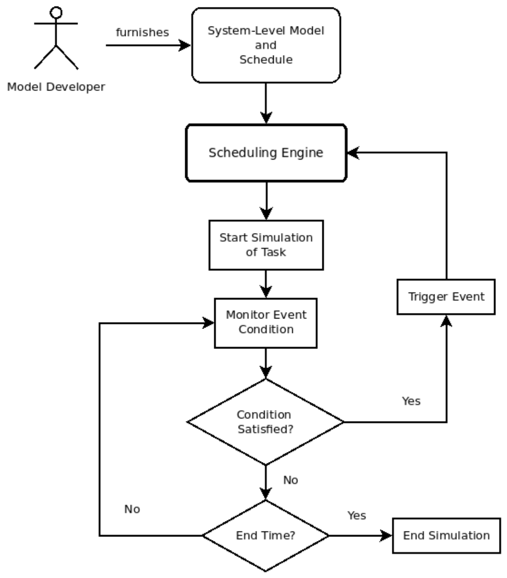


Figure 2. Generic Workflow for Simulation Scheduling

- The schedule turns out to be a directed and connected graph with an alternate sequence of tasks and events. Each node in such a graph represents the pair of a task and its associated events. The worst-case schedule can be a complete graph.
- The graph can have backward and forward edges with respect to a node. However, there is no self-loop back to a node as it has no practical significance and can be handled within the node itself.
- This graph is compliant with control workflow patterns (Russell, Van Der Aalst, and Ter Hofstede 2016) such that only one node gets activated at any point of simulation time, and the directionality between the nodes is preserved. As analyzed from the first property, the possible control patterns include unstructured loop, sequence, exclusive choice, and simple merge. These patterns are considered for implementation in Section 3.

2.2 Simulation Scheduling

Figure 2 shows the generic simulation scheduling once the model developer provides the system-level model and its schedule. The scheduling engine is an intermediate layer between the system-level modeling and the underlying simulation tool. The engine parses the model and simulates only the first task based on the schedule provided. The events associated with the task are monitored to evaluate their conditions. If any of them is satisfied, then the corresponding event gets triggered. The engine then simulates the next task in the schedule from the point in simulation time where the previous task left. The process repeats until the end of the entire simulation time.

The above framework descriptions and schedule properties divide the system-level modeling into discrete parts of the components or subsystems. Thus, it also divides the single continuous simulation into multiple but efficient smaller and faster simulations of tasks. The efficiency is achieved in terms of smaller executable code size and lesser memory requirements for each task when compared to the entire simulation done with existing frameworks. The same is noticeable from the demonstration discussed in Section 6. Otherwise, it can waste CPU cycles as the data can keep on shuffling between the CPU cache and the RAM (Pop et al. 2019). As only one task will be simulated at any given point of the simulation time, the goal is thus achieved by not simulating the rest of the inactive tasks. Therefore, the unnecessary running of the simulations for the inactive tasks is prevented, which anyway will not impact the simulation of the current task.

The framework is independent of the underlying simulation tool and the modeling language it uses. As a result, it also permits real-time and interactive simulations and does not require any rework of the existing simulation tools. Note that the interactive simulation here indicates that one can interact with the model during its simulation by monitoring and optionally modifying the state of the model as per the requirement.

3 Related Work

There is no native support, nor is there any Modelica tool that provides a generalized extension or framework of the type described here, suitable for all domains (Casella 2019; Jack 2020). Nevertheless, there are attempts at variable-structure modeling similar to the proposed simulation scheduling workflow (Briese 2018; Mehlhase et al. 2014; Stüber 2017). Stüber (2017) further discusses several implementations and their drawbacks. They use specialized forms of the proposed generic simulation scheduling workflow suitable for their applications. One approach common to all of them involves either the reimplementation of their existing models or manual work for generating models with different structures and exploiting the functionalities of proprietary tools. Also, none of these related works exhibits real-time and interactive simulation capability. Apart from these application-specific implementations, there is a need for sequential modular simulation in OpenModelica (Casella 2021). It also indicates that no such related generalized work has been done for OpenModelica.

Outside the Modelica context, the following are the prior work that attempts to have similar modeling capabilities that we have proposed but simulate as per the existing framework.

The special-purpose and open-source Ngspice circuit simulator (Vogt et al. 2021) partially avoids running simulation code that is not required. The components in the circuit are mapped to a C function through its XSPICE extension. As a result, the C function gets invoked only when the signal reaches the corresponding component. However, this behavior is applicable only for the digital components, while the analog simulation of all components is still running even though they may not be required.

A proprietary and general-purpose simulator, GoldSim, has conditional containers similar to the conditional task scheduling in our engine. However, it still keeps on constantly running the code in idle mode for those components that are not required (GoldSim Technology Group 2022). Similar functionality is observed with the general-purpose AnyLogic simulation software (The AnyLogic Company 2022) and gPROMS (Process Systems Enterprise Ltd. 2004), a special-purpose simulator focused on chemical processes. Another proprietary but discrete-event simulator, FlexSim, has conditional task functionality. However, again, the components still keep running and remain idle as described in their tutorial (FlexSim Software Products, Inc. 2022).

Another application of variable-structure modeling, which has received great attention, is the simulation of the reconfigurable manufacturing system (RMS) that implies a change in the factory structure. K et al. (2019) and Herps et al. (2022) demonstrate the simulation of their proposed manufacturing processes using FlexSim and AnyLogic respectively. However, as mentioned earlier, both software keeps executing the conditional elements even if they are redundant. Kahloul, Bourekkache, and Djouani (2016) use reconfigurable object Petri nets to model and simulate RMS. It has system-level nets that involve fire and transform transitions and a set of morphisms. It allows changing between the object net markings and structures corresponding to each configuration. Although this modeling is similar to the work described here, their entire RMS representation is simulated as a single model. It thus loses the benefit of an already discretized model and the scope to avoid running redundant simulation code.

The FMI standard in terms of System Structure and Parameterization (Modelica Association Project SSP 2019) and Distributed Co-Simulation Protocol (Modelica Association Project DCP 2019) may possibly be exploited to achieve the goal of our work. These co-simulation standards, which are out of the scope of this work, shift efforts from simulation run-time (scheduling framework) to simulation configuration time. However, as the discrete behavior of the system can be known at run-time, one has to additionally anticipate different scenarios with variable structures and generate them only at configuration time.

4 Implementation of Scheduling Framework for OpenModelica

The primary goal of our work is to run only the simulation code that is required. In the simulation context and to the model developer, it indicates that the subsystems or components are to be simulated only when activated, i.e., lazily simulated. As mentioned earlier, our implementation is based on the Modelica language. The systems are modeled using equations in this language and require all equations to be solved simultaneously during the entire simulation. The goal, hence, boils down to preventing unnecessary solving of the equations governing the behavior

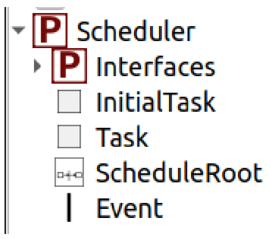
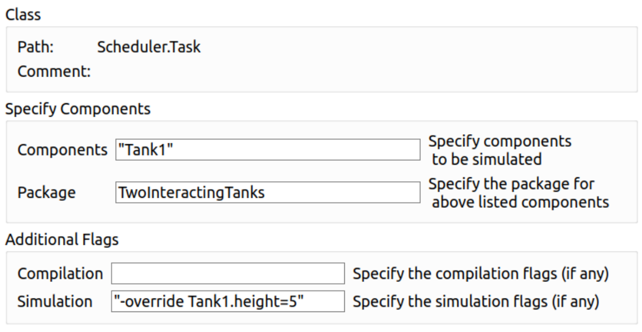


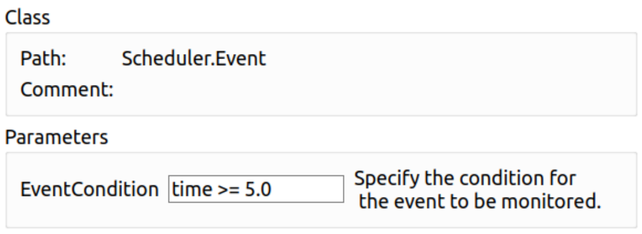
Figure 3. Structure of the `Scheduler` library shown in OMEdit - OpenModelica Connection Editor


Class
Path: Scheduler.Task
Comment:

Specify Components
Components "Tank1" Specify components to be simulated
Package TwoInteractingTanks Specify the package for above listed components

Additional Flags
Compilation Specify the compilation flags (if any)
Simulation "-override Tank1.height=5" Specify the simulation flags (if any)

(a) Task Interface


Class
Path: Scheduler.Event
Comment:

Parameters
EventCondition time >= 5.0 Specify the condition for the event to be monitored.

(b) Event Interface

Figure 4. User interfaces for `Scheduler` library's important blocks as shown in OMEdit - OpenModelica Connection Editor

of inactive components or subsystems.

The overall prototype implementation has two parts. The first one is only the creation of the schedule for the variable-structure model using the `Scheduler` library without any need to re-implement the model. The second one is the simulation scheduling using an engine that leverages the OpenModelica Simulation Environment.

4.1 `Scheduler` - Modelica Library

For the model developer to construct the schedule consisting of tasks and events, we have developed a library, called `Scheduler`, in the Modelica language itself. Figure 3 shows its structure in OMEdit, the OpenModelica's connection editor. The `Task` and `Event` classes correspond to the schedule's task and event discussed in Section 2, and their user interfaces are shown in Figure 4. The `InitialTask` class is the same as the `Task` class, except that it indicates the start of the schedule. The `ScheduleRoot` class requires the model developer to specify the top-level model in the hierarchy of system-level modeling and the general simulation parameters applicable to all the tasks in the schedule. The schedule must have an instance of this class. The `Interfaces` sub-package provides the desired abstraction to the above-mentioned classes and is not meant to be used directly.

The classes in the `Scheduler` library themselves are available as components, which the model developer can drag and drop in OMEdit, and also write the Modelica code to connect the tasks and events. As shown in Figure 4, the model developer has to provide a list of components or subsystems for each task along with the package name to find these components. There is also a provision to specify compilation and simulation flags in addition to those mentioned in the instance of `ScheduleRoot` class. These flags will be applicable only to that task. The event condition needs to be specified for all the events in the schedule. The schedule can be created within the top-level model, i.e., in the same file as the top-level model or outside it as a separate file having the name as `Schedule` within the same package.

4.2 Scheduling Engine

As discussed earlier, there is no native support in Modelica for scheduling. So, the implementation of the scheduling engine needs to be outside the Modelica context, which we have done in Python language. The scheduling engine requires only the system-level model and the schedule to be provided by the model developer. It is an additional layer between the user's system-level model and OpenModelica. That is, it runs on top of the OpenModelica compiler (OMC) in an interactive mode and is loosely coupled to OMC. It leverages OMC's ZeroMQ communication interface through OMPython to take over the typical compilation and simulation process in OpenModelica. OMPython is a part of the OpenModelica Simulation Environment and acts as a Python interface to communicate with OMC. The provided schedule is parsed independently of OpenModelica per the scheduling framework. It then distributes the system-level model into several sets of active subsystems or components preserving the connections between them, where each set corresponds to a task in the schedule. As a result, it automatically generates all the models corresponding to each task on behalf of the model developer.

The scheduling technique used by the engine is similar to the next-event scheduling used commonly in the discrete-event simulation. That is, when each event is set up, it creates (schedules) the next procedure (task and event). Following this scheme, only the initial task to be run first is compiled and then simulated. The events associated with the initial task are monitored to evaluate the corresponding conditions. The next task in the schedule is activated when one of these events gets triggered. If the event condition gets satisfied immediately upon the start of task simulation, then the task simulation is terminated immediately without consuming any further simulation steps and the next task gets scheduled. The next task is compiled and then simulated by transferring the end results of the previous task to the next. This transfer of end results ensures the continuity of the state from where the previous task left. It acts as initialization conditions for the next task, by overriding the default ones provided by the model developer within the model itself. If any other additional information is not present in the previous or the next task, then the model developer can still provide it using suitable OpenModelica simulation flags applicable to that task. An example of specifying additional information to initialize the tank height with 5 units is shown in Figure 4 (a). The above entire process is repeated until the end of the entire simulation time.

The monitoring of events and evaluation of the associated conditions is achieved through the OPC UA server implementation in OpenModelica (henceforth referred to as the server). OPC UA is an interoperability standard in industrial applications. The server is suitable for interactive simulation in real-time as well (Kumar et al. 2021). The next section also illustrates the same using the scheduling framework. Note that the interactive simulation through OPC UA does not involve any kind of visual interface or animation, and is out of the scope of the work described here. The scheduling engine acts as the OPC UA client and simulates each task with an embedded server. The event monitoring process leverages the publish-subscribe model of the server. The process variables (PV) and the manipulated variables (MV) in the event conditions are subscribed for notifications by the scheduling engine. As a result, simulation time is saved by not polling continuously for the changes in PV and MV over the OPC UA client-server configuration.

As observed from the above engine implementation, the compilation and simulation of each task are done just in time. That is, the tasks are compiled and simulated only when they are required. Note that there are no repeated compilations of the same tasks. The very first compiled tasks are reused again with a different context whenever required. It is possible to manually create the model corresponding to each task, compile them and then use them for manual scheduling. However, it is not feasible and error-prone when the complexity of the system and schedule scales up. Thus, the automatic distribution of the system-level model and simulation scheduling is more efficient than the manual work for the model developer. The importance of the problem attempted in this work and the usage of the proposed framework through the above implementation is illustrated through two engineering examples, to be presented next.

5 Batch Distillation System

In this section, we explain how the proposed framework helps reduce the computations in the operation of the batch distillation column (Figure 5) studied by Kumar et al. (2021) using an OpenModelica OPC UA client-server configuration. In this example, a feed stream containing three chemicals is separated into pure components. Sharma, Moudgalya, and Shah (2021) operate the opening and closing of the valves through the StateGraph library

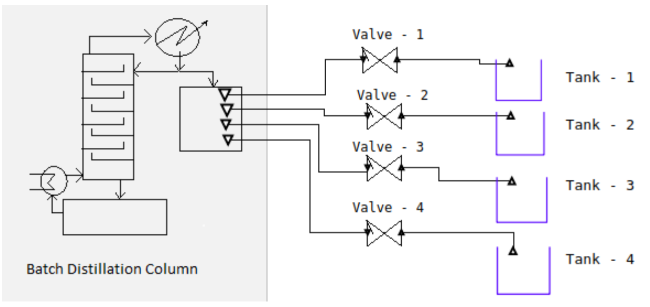


Figure 5. Batch distillation system for ternary mixture with product and slop cuts

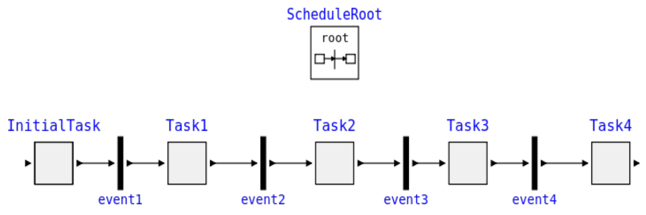
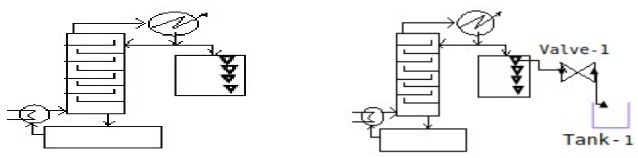
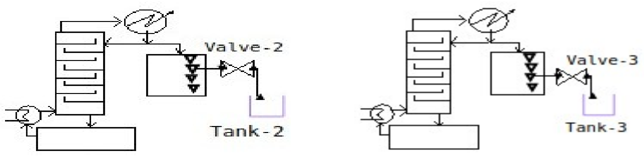


Figure 6. Schedule of the batch distillation system using `Scheduler` library



(a) `InitialTask`: Batch distillation column with no distillate withdrawal

(b) `Task1`: Batch distillation system with Product-1 collected in Tank-1



(c) `Task2`: Batch distillation system with Slop-1 collected in Tank-2

(d) `Task3`: Batch distillation system with Product-2 collected in Tank-3

Figure 7. Task-wise sequence of active components of batch distillation system simulated through the scheduling framework

in OpenModelica, which obviates the need for event constructs such as `if-else` and `when` statements. All components of the batch distillation system are simulated simultaneously in both approaches, irrespective of whether they are active or not.

The batch distillation system with the product and slop scheduling in different tanks is modeled through the `Scheduler` library. The inlet valves of the product and slop tanks are controlled according to the purity levels obtained in the distillate. This depends on the mole fraction of components in the distillate and reboiler, respectively. Figure 6 shows the sequence of events and tasks to be simulated on the occurrence of a particular time or state event or both. This schedule is created in OMEdit by drag and drop of five instances of `Task` class, four instances of `Event` class, and one mandatory instance of

`ScheduleRoot` class. It is defined outside the top-level model of the batch distillation system (Figure 5) in a separate file but within the same package. Figure 7 shows the sequence of subsystems corresponding to the simulated tasks as per the schedule shown in Figure 6.

Listing 1. Illustrative Modelica code for top-level model of batch distillation system

```
model BatchDistillation
  DistillationColumn Column;
  Valve Valve1, Valve2, Valve3, Valve4;
  Tank Tank1, Tank2, Tank3, Tank4;
equation
  connect(Column.outflowProduct1, Valve1.
      inflow);
  connect(Valve1.outflow, Tank1.inflow);
  connect(Column.outflowSlop1, Valve2.
      inflow);
  connect(Valve2.outflow, Tank2.inflow);
  connect(Column.outflowProduct2, Valve3.
      inflow);
  connect(Valve3.outflow, Tank3.inflow);
  connect(Column.outflowSlop2, Valve4.
      inflow);
  connect(Valve4.outflow, Tank4.inflow);
end BatchDistillation;
```

Listing 2. Illustrative Modelica code for `InitialTask`

```
model BatchDistillation
  DistillationColumn Column;
end BatchDistillation;
```

Initially, the batch distillation system is operated at total reflux with no distillate taken out of the system. In this condition, the valves and tanks connected to the batch distillation column are idle. So, the initial task is operated only with the batch distillation column present, and all other components are excluded as shown in Figure 7 (a). This corresponds to the `InitialTask` of the schedule in row 1 of Table 1. The scheduling engine parses this task by removing all of the components and their connect equations, except the batch distillation column, from the Modelica code shown in Listing 1. It communicates with OMC interactively through OMPython to achieve the same. Listing 2 shows the illustrative Modelica code for the resultant model after parsing `InitialTask`.

The communication involves sending commands, that invoke the appropriate scripting API available in OpenModelica, and receiving the status of the command. As mentioned earlier, the communication is done in the form of client-server configuration over the ZeroMQ interface, where the scheduling engine acts as the client and OMC as the server. In this way, the scheduling engine automatically generates the model for `InitialTask` without any manual intervention of the model developer. This model is kept in memory until it is compiled, but it can also be dumped into a separate file for the model developer's reference.

Once the model corresponding to `InitialTask` is generated, it is compiled and its simulation is started by embedding an OPC UA server with its executable. The same is already demonstrated by Kumar et al. (2021). This embedded OPC UA server allows the scheduling engine to monitor the first event (`event1`). As soon as the event condition being satisfied is detected, the scheduling engine sends another command to gracefully terminate the simulation. It, in turn, saves `InitialTask`'s current state in the form of a result file. It can be visualized in OMEdit by selecting the desired variables in its interface. The end results from this result file act as the initialization condition for `Task1`. The model generation, compilation, and simulation for the rest of the tasks are done in a similar manner as `InitialTask`.

When the desired purity of the lighter component is achieved in the distillate, Valve-1 is opened, and the product is collected in Tank-1. Accordingly, the next task is scheduled with the corresponding event condition that activates the Valve-1 and Tank-1 and is shown in Figure 7 (b). This corresponds to `Task1` in Table 1. The initialization of this task is done with the transfer of end results (state) from `InitialTask`. Note that the initialization condition for the distillation column gets overridden here as only its state information is present in the simulation of `InitialTask`. Thus, the initialization condition for Tank-1 and Valve-1 falls back to the default one already described within the model itself.

In the next task of the schedule, as the purity of the first component decreases below the desired level in the distillate, Valve-1 is closed. So, both Valve-1 and Tank-1 no longer need to be solved in the simulation and hence are removed. Simultaneously, Valve-2 is opened, and distillate goes to Tank-2 as slop cut, an undesirable product, and is shown in Figure 7 (c). This corresponds to `Task2` in Table 1. Similar to the previous case, the initialization for this task is done only for the distillation column with the state from `Task1`.

When the desired purity of the second component is achieved in the distillate, it is collected in Tank-3. Correspondingly, Valve-3 is opened, and Valve-2 and Tank-2 are removed from the simulation as shown in Figure 7 (c). This corresponds to `Task3` in Table 1. Similar to the previous cases, the initialization for this task is done only for the distillation column with state from `Task2`.

When the purity level of the second component decreases below the desired level in the distillate, the next step is collecting the slop in Tank-4 until the third component reaches the desired purity in the reboiler. As seen in Figure 9 the purity of the third component reaches desired level well before the distillate is collected in Tank-4. Hence, Valve-4 is never opened, and so `Task4` is never compiled and simulated.

Figure 8 shows the moles of product and slops in the batch distillation system. Figure 9 shows the mole fractions of component-1 and component-2 in distillate, and component-3 in reboiler. These results obtained using the scheduling framework are the same when performed with and without real-time simulations. The setup for the real-time simulation and the above results are identical to

Table 1. Task-wise distribution of units for the batch distillation system simulated through the scheduling framework

Task (as per the schedule)	**Active Units** (simulated)	**Inactive Units** (not simulated)	**Corres-ponding Figure**
Initial Task	Distillation column	All valves and tanks	Figure 7 (a)
Task1	Distillation column, Valve-1 and Tank-1	Valve-2,3,4 and Tank-2,3,4	Figure 7 (b)
Task2	Distillation column, Valve-2 and Tank-2	Valve-1,3,4 and Tank-1,3,4	Figure 7 (c)
Task3	Distillation column, Valve-3 and Tank-3	Valve-1,2,4 and Tank-1,2,4	Figure 7 (d)

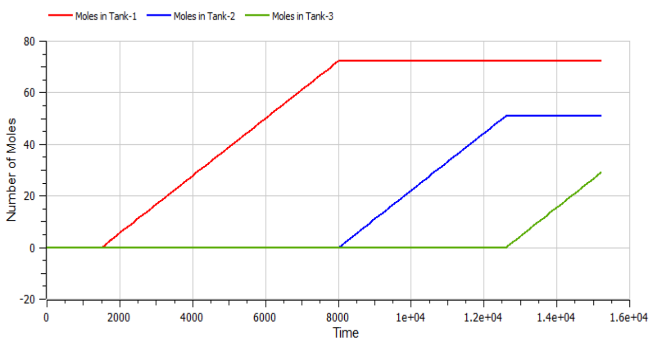


Figure 8. Moles of distillate collected in tanks

those described by Kumar et al. (2021) using Raspberry Pi performing real-time simulation in OpenModelica. The only difference in the setup is that the controller here is within the batch distillation model instead of Raspberry Pi. Furthermore, the results here are in agreement with the simulation results done using the StateGraph library by Sharma, Moudgalya, and Shah (2021).

Table 2 compares the number of equations solved by the previous methods with the current scheduling framework for each task. Instead of solving the entire flowsheet containing the distillation column, four valves, and four tanks resulting in 440 equations, the scheduling framework solves the active units at a particular event and removes the inactive units from the simulation of the batch distillation system. All the modeling equations are solved using DASSL (Brenan, Campbell, and Petzold 1996) in OpenModelica.

Table 2. Number of equations solved per task (as given by OpenModelica) compared with previous methods

Variable-Structure	**Number of Equations Solved**		
Models	**OPC UA**	**StateGraph library**	**Scheduling framework**
InitialTask	440	440	**260**
Task1	440	440	**268**
Task2	440	440	**268**
Task3	440	440	**268**
Entire Batch Distillation System	440	440	440

6 Steady-state solution of a flowsheet through a sequence of calculations

This application is concerned with finding the steady-state solution to chemical engineering flowsheets described by a large number of equations. OpenModelica has the capability to collect the equations from different parts of a flowsheet and solve them simultaneously. It is an important capability, as design problems can be solved easily in this framework. It is also suitable for dynamic simulations. Unfortunately, having to solve a large number of equations gives rise to some difficulties. As these are generally nonlinear equations, initial guesses are required to solve them. Setting up the initial guess itself is difficult for a large number of equations, let alone converging to the steady-state solution.

Let us consider the Methanation flowsheet (Reklaitis 1983) given in Figure 10. In this system, the synthesis gas, which is a mixture of CO, H_2, and a small amount of CH_4, is converted to a higher content of Methane. The reaction taking place is:

$$CO + 3H_2 \rightarrow CH_4 + H_2O$$

The feed stream at 93.3°C and the recycle stream are fed to the mixer, followed by an adiabatic reactor with an

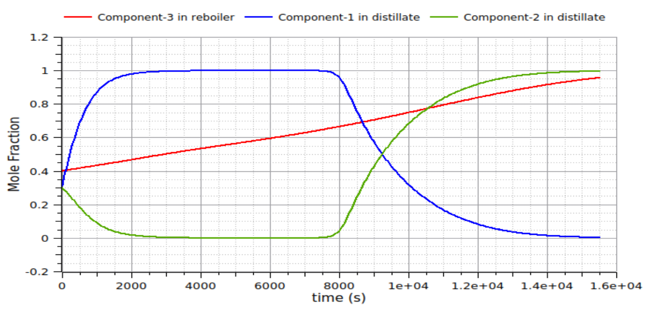


Figure 9. Mole fractions of desired components in distillate

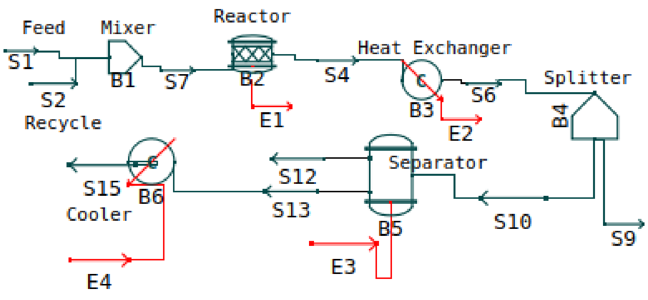


Figure 10. Recyle process for the Methanation flowsheet

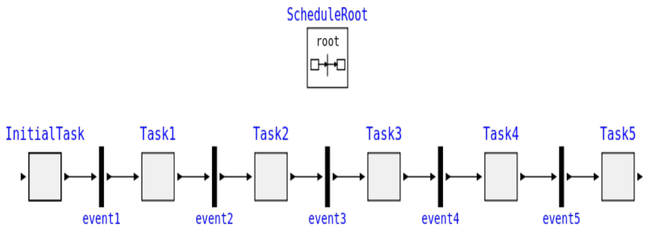


Figure 11. Schedule of the Methanation flowsheet using the `Scheduler` library

outlet temperature of 537.7°C. The effluent is then cooled to 260°C in a heat exchanger. The effluent is split into Methane rich stream containing 50% Methane at 93.3°C, and the other stream is fed to a separator to remove water.

Methanation flowsheet is modeled and simulated sequentially using the scheduling framework. The schedule is shown in Figure 11 and created in a similar manner as that for the batch distillation system. Again, similar to the batch distillation system, the initialization of only those components of subsequent tasks is done for which the state information is present in the respective previous task. The initialization of the rest of the components falls back to the default one described within the model itself. The following describes the tasks and their scheduling:

- In the first task, a pure feed stream ($S1$) and the recycle stream ($S2$) are mixed in the mixer ($B1$). The B1 output is sent to another stream, $S7$. So, the subsystems (units) $S1$, $S2$, $B1$, and $S7$ are active during `InitialTask`, and all other units do not participate in the simulation.
- In the second task, the mixed stream ($S7$), which was the output of the `InitialTask`, acts as an input for this task. It is taken to a reactor ($B2$) for the Methanation reaction. Hence the units $S7$, $B2$, $S4$, and $E1$ are active during `Task1`.
- In the third task, the product stream from the reactor acts as input for the heat exchanger. The units $S4$, $B3$, $S6$, and $E2$ are active when `Task2` is scheduled.
- During `Task3`, the cooled stream ($S6$) from the heat exchanger is taken to a splitter ($B4$) to split the stream into two material streams. During this task, the units $S6$, $B4$, $S9$, and $S10$ are active, and other units do not participate in the simulation.
- In the fifth task, the $S10$ output from the previous task is taken to the separator unit ($B5$) and separated to give two output material streams. The units $S19$, $B5$, $S12$, $S13$, and $E3$ are active during `Task4`.
- In the final task, i.e., `Task5`, one of the output streams from the separator unit ($S13$) is further cooled using the cooler unit (B6). In this task, the units $S13$, $B6$, $S12$, and $E4$ are active, and other units do not participate in the simulation.

The results obtained using the scheduling framework are identical to those shown by Reklaitis (1983). The number of equations solved and the simulation efficiency in terms of executable code size and memory requirements for each task are provided in Table 3. Since multiple models are now compiled through the tasks, one may perceive that the sum total of code size and the memory requirements increase as compared to the entire flowsheet. However, only one of the tasks is simulated at any given point in time. Thus, one has to consider only the resources corresponding to a single task's code size and memory requirements. Hence, as mentioned earlier in Section 2, the efficiency here is determined with respect to a given task only. It is nearly a three-fourth average reduction in the number of equations solved and code size, and a two-third average reduction in memory requirement compared to the simulation of the entire flowsheet. Here also, all the modeling equations are solved using DASSL in OpenModelica.

Table 3. Number of equations solved and simulation efficiency per task (as given by OpenModelica) compared to the entire Methanation flowsheet

Variable-Structure Models	**Number of Equations Solved**	**Executable Code Size (in KB)**	**Memory Requirement (in MB)**
InitialTask	648	1126	17.6
Task1	442	842	14.9
Task2	434	833	14.4
Task3	646	1126	16.9
Task4	675	1228	14.3
Task5	434	828	17.1
Entire Methanation Flowsheet	2339	3600	48.4
Average reduction per task compared to entire Methanation flowsheet	76.63 %	72.30 %	67.22 %

7 Conclusion and Future Work

An attempt has been made to tackle the problem of excluding the simulation code corresponding to the inactive components or subsystems while simulating a system. This approach generally leads to more correct results. In some cases, this may be the only way to achieve the end goals of a simulation. It is achieved by allowing the model developer to model the discrete behavior of their system through a schedule.

Construction of a schedule is made possible through a Modelica library `Scheduler`, developed in this work. The scheduling engine is implemented as a layer between the user model and the OpenModelica simulation environment. This approach is validated by applying it to the operation of a batch distillation column that separates a mixture, along with its real-time and interactive simulation. An example involving the steady-state solution to a chemical engineering flowsheet through a sequential modular simulation is also presented. In both cases, the results are identical to those reported in the literature, obtained through other approaches.

The future work would involve the performance and optimization aspects of the framework and its prototype. As there is a dependency between any two given tasks, their simulations cannot be done in parallel. But, their compilations can definitely be done in parallel, a feature available even in standard multi-core laptops. OPC UA, being a protocol on the network, can be a significant bottleneck (in terms of time and resources) for some models, and can lead to accuracy issues due to the inherent nature of the network. So, another method of simulation scheduling without OPC UA is desired. The usage of either of the two methods can be left to the model developer to decide as per their simulation requirements. Another direction to explore would be to extend the schedule's workflow pattern to include parallel routing. It would enable independent tasks to be simulated simultaneously and possibly in a distributed manner.

Acknowledgments

This work is partly funded by the National Mission on Education through ICT of the Ministry of Education, Govt. of India, through the FOSSEE project. The scheduling framework implementation for OpenModelica will be released as open source. In the meantime, those who are interested in accessing it are invited to contact the authors.

References

Brenan, K. E., S. C. Campbell, and L. R. Petzold (1996). *Numerical Solution of Initial-Value Problems in Differential Algebraic Equations*. Philadelphia: SIAM.

Briese, Lâle Evrim (2018-05). "Mission-Dependent Sequential Simulation for Modeling and Trajectory Visualization of Reusable Launch Vehicles". In: *Proceedings of the 2nd Japanese Modelica Conference*. Tokyo, Japan, pp. 230–239. DOI: http://dx.doi.org/10.3384/ecp18148230.

Casella, Francesco (2019-09). *Are variable-structure models allowed in Modelica? #2411*. URL: https://github.com/modelica/ModelicaSpecification/issues/2411 (visited on 2022-05-11).

Casella, Francesco (2021-02). "Is OpenModelica Finally Coming of Age?" In: *OpenModelica Annual Workshop 2021*. Linköping University. URL: https://openmodelica.org/images/M_images/OpenModelicaWorkshop_2021/0900_ComingOfAge.pdf.

FlexSim Software Products, Inc. (2022). *Tutorial 3 - Conditional Tasks*. URL: https://docs.flexsim.com/en/22.1/Tutorials/TaskLogic/Tutorial3ConditionalTasks/ConditionalTasksOverview/ConditionalTasksOverview.html (visited on 2022-08-24).

Fritzson, Peter (2014-11). *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Second Edition. Wiley-IEEE Press. ISBN: 978-1-118-85912-4.

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

GoldSim Technology Group (2022). *Behavior of Elements in Conditional Containers*. URL: https://help.goldsim.com/#!Modules/5/behaviorofelementsinconditionalcontainers.htm (visited on 2022-08-24).

Herps, Koen et al. (2022). "A simulation-based approach to design an automated high-mix low-volume manufacturing system". In: *Journal of Manufacturing Systems* 64, pp. 1–18. ISSN: 0278-6125. DOI: https://doi.org/10.1016/j.jmsy.2022.05.013. URL: https://www.sciencedirect.com/science/article/pii/S0278612522000838.

Jack (2020-12). *An error of using multi-mode DAEs in Dymola*. URL: https://stackoverflow.com/questions/65199823 (visited on 2022-05-11).

K, Raja et al. (2019). "Implementation of Reconfigurable Manufacturing Systems in the Manufacturing of Turbo Charger Turbine Housing". In: *SAE Technical Paper Series* October. ISSN: 01487191. DOI: 10.4271/2019-28-0135.

Kahloul, Laid, Samir Bourekkache, and Karim Djouani (2016). "Designing reconfigurable manufacturing systems using reconfigurable object Petri nets". In: *International Journal of Computer Integrated Manufacturing* 29.8, pp. 889–906.

Kumar, Sudhakar et al. (2021). "OpenModelica OPC UA framework for control applications". In: *2021 9th International Conference on Control, Mechatronics and Automation (ICCMA)*, pp. 78–83. DOI: 10.1109/ICCMA54375.2021.9646198.

Mehlhase, Alexandra et al. (2014-03). "An example of beneficial use of variable-structure modeling to enhance an existing rocket model". In: *Proceedings of the 10th International Modelica Conference*. Lund, Sweden, pp. 707–713. DOI: http://dx.doi.org/10.3384/ECP14096707.

Modelica Association Project DCP (2019). *DCP Specification Document, Version 1.0*. Modelica Association. Linköping, Sweden. URL: http://www.dcp-standard.org.

Modelica Association Project SSP (2019). *SSP Specification Document, Version 1.0*. Modelica Association. Linköping, Sweden. URL: https://ssp-standard.org/.

Pop, Adrian et al. (2019-03). "A New OpenModelica Compiler High Performance Frontend". In: *Proceedings of the 13th International Modelica Conference*. Regensburg, Germany, pp. 689–698. DOI: http://dx.doi.org/10.3384/ecp19157689.

Process Systems Enterprise Ltd. (2004). *gPROMS Introductory User Guide*. URL: https://dokumen.tips/documents/gproms-introductory-guide.html (visited on 2022-08-24).

Reklaitis, G.V (1983). "Introduction to Material and Energy Balances". In: pp. 602–610.

Russell, Nick, Wil Mp Van Der Aalst, and Arthur HM Ter Hofstede (2016-02). *Workflow Patterns: The Definitive Guide*. MIT Press. ISBN: 9780262029827.

Sharma, Nikhil, Kannan Moudgalya, and Sunil Shah (2021-02). "Development of Modelica Library for Batch Distillation". In: *OpenModelica Annual Workshop 2021*. Linköping University. URL: https://openmodelica.org/images/M_images/OpenModelicaWorkshop_2021/1430_Nikhil_OpenModelica_Workshop2021.pdf.

Stüber, Moritz (2017-05). "Simulating a Variable-structure Model of an Electric Vehicle for Battery Life Estimation Using Modelica/Dymola and Python". In: *Proceedings of the 12th International Modelica Conference*. Prague, Czech Republic, pp. 291–298. DOI: http://dx.doi.org/10.3384/ecp17132291.

The AnyLogic Company (2022). *Controlling the model execution*. URL: https://anylogic.help/anylogic/running/controlling-model-execution.html (visited on 2022-08-24).

Vogt, Holger et al. (2021). *Ngspice User's Manual Version 35 (ngspice release version)*. URL: http://ngspice.sourceforge.net/docs/ngspice-35-manual.pdf (visited on 2022-08-24).