Optimizing life-cycle costs for pumps and powertrains using FMI co-simulation

Miro Eklund[1,2] Jouni Savolainen[2] Antti Lukkari[3] Tommi Karhela[2]

[1]Dept. of Information Technology, Åbo Akademi, Finland, miro.eklund@abo.fi
[2]Semantum Ltd, Finland, {miro.eklund,jouni.savolainen,tommi.karhela}@semantum.fi
[3]ABB Oy, Finland, antti.lukkari@fi.abb.com

Abstract

This paper describes a collaborative digital twin approach for equipment dimensioning and selection in industrial process plants. Dynamic process simulator (Apros) was used to model the process and its automation, including pumps, while a product specific dynamic simulator (Virtual Drive) was used to model the motor and frequency converter. This approach allows all stakeholders to design and dimension the process equipment together in a holistic and energy optimal way. Simulation can be used to reach an optimal equipment solution that prevents overdimensioning, leading to up-front and total life-cycle cost savings.

Co-simulation was made possible by implementing a prototype Functional Mock-up Interface (FMI) for both Apros 6 and Virtual Drive, allowing Apros to import Virtual Drive as a Functional Mock-up Unit (FMU). This paper shows how the FMI solution can be used for finding energy optimal selections for pumps and related powertrain products.

Keywords: co-simulation, functional mock-up interface, apros, virtual drive, optimization

1 Introduction

Over 40% of the world's electricity is currently consumed by electric motors in buildings and industrial applications, and approximately 75% of these industrial motors run pumps, fans and compressors. This is a machinery category that is highly potential for major energy efficiency improvements. Considering the huge number of industrial electric motor-frequency converter systems in operation (roughly 300 million), global electricity consumption could be reduced up to 10% if these process applications were properly optimized. Thus, significant savings can be achieved in processes when using system-level optimization and right-sized components. (Waide and Brunner 2011; *Motor-driven Equipment Research Package* 2021).

Processes, systems and industrial plants are traditionally designed and dimensioned by several stakeholders. Usually an EPC (engineering, procurement and construction company) has the main responsibility of a project by handling the design, procurement and construction work. Subcontractors, such as system integrators and OEMs (original equipment manufacturers), are used to deliver the required technical systems and equipment for an industrial plant. Processes are usually divided into sub-systems, which are designed and dimensioned separately by different stakeholders. Different process equipment such as motors, frequency converters, pumps and fans are dimensioned by OEMs based on the overall specification of the system. This traditional way of designing a process is called the waterfall model, see Figure 1 for an example of such a design process. Process design challenges and approaches has been investigated in chemical engineering and process systems engineering for decades and a lot has been written about it. We will not delve deep into the available literature on the topic, but Vega et al. (2014), Nishida, Liu, and Ichikawa (1976) and Westerberg (2004) are great starting points.

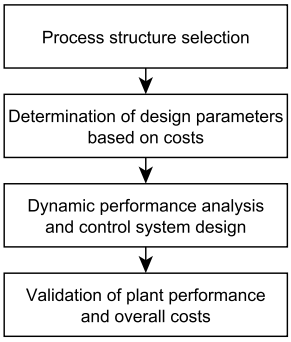

Figure 1. A traditional way of designing a process (Vega et al. 2014).

Due to separate design steps, waterfall model lacks system-level optimization of the process. In the absence of overall coordination of the process design, each stakeholder adds their own risk margins to the sizing of the design to ensure that each and every component fulfills the critical process requirements. This leads to overdimensioning of the system components. When all these separate pieces of equipment are combined into a functional process system, such as a pumping line, the whole system runs inefficiently, using too much energy with too high costs.

Our collaborative digital twin approach combines

decision-making by including all stakeholder's equipment selections in one model. A digital twin is a virtual representation of a physical phenomenon, e.g. an industrial plant. Digital twins have been investigated as a method for improving process designs and collaboration between stakeholders, e.g. by introducing modular designs (Jiapeng et al. 2019) or by using big-data (Fei et al. 2018).

In a collaborative digital twin approach, all stakeholders design and dimension the process equipment together. With the virtual model, the behaviour and performance of the whole process can be simulated before any physical implementations. Dynamic simulation and system-level optimization allows processes to be optimized and high risk margins and overdimensioning to be avoided, while still finding suitable equipment that fulfill the critical process requirements.

In our case, system-level dynamic simulation was run using *Apros 6* (2021)® process simulator and *Virtual Drive* (2021) simulator together in co-simulation. The history of Apros goes back to the commissioning of the Loviisa nuclear power plant in Finland in the 1970s. A suitable starting point for a scientific description of Apros is Lappalainen (2019) and the references therein.

Virtual Drive is ABB's commercial software product, which can be used to model the electrical behavior of actual motor and frequency converter products in a simulation environment. Functional Mock-up Interface (FMI) was used to integrate these two simulators together and co-simulation provided reliably data about how different products would run the process. Based on the simulation results, the most optimal equipment could be selected.

This new way of working was first tested in a demo simulation in the spring of 2020. In collaboration with a pump OEM, an optimal pump-motor-frequency converter combination was dimensioned based on digital twin dynamic simulation. In the demo process model, water was pumped to a water tank which was located 20 meters above, see Figure 4. The water surface level inside the tank was attempted to be maintained the same at all times, and water outflow from the tank was constantly changed. Dynamic simulation of the system revealed that too small pump could not deliver enough water to the tank, but too big pump consumed too much energy. It was also noticed that the check valve fluttered during the simulation when using an optimal pump. After adding a feed forward structure to the process to measure the outflow rate and selecting smaller motor and frequency converter with an optimal pump, it was possible to maintain the water level in the setpoint level, and the energy consumption of the system was significantly decreased, compared to the oversized configuration. This demo confirmed the huge potential in the new way of working.

2 Methods

Implementing interoperability between two systems can be done by directly implementing the custom interface of one of the systems in the other. This means new code must be written whenever a new system needs to be added. An alternative approach is mentioned by Nouidui, Wetter, and Zuo (2014), which is to use existing standards such as the Functional Mock-up Interface. Implementing the interopertiblity with a standardized interface like FMI gives additional compatibility with many other tools in the FMI ecosystem. In this study, a prototype FMI importer plugin was developed for Apros, allowing it to be the coordinator and import FMU models. A prototype FMI Wrapper implementation was developed for Virtual Drive, allowing Virtual Drive to be imported as an FMU model that internally uses a proxy connection to control an existing Virtual Drive instance.

2.1 Co-simulation approaches

Meer et al. (2020) describe co-simulation as typically meaning a scenario where two or more models simulate simultaneously and periodically require inputs from eachother. Time-dependent simulations also require the models to have the same concept of time, i.e. they should advance an equal amount of time between each data exchange point. The models in co-simulation can be created with different simulators, e.g. because the simulators specialize in different fields or simply because of familiarity to the modellers. Co-simulation can be implemented with local connection, see Nouidui, Wetter, and Zuo (2014) or distributed connections, see Sadjina et al. (2018). It can be implemented with custom interfaces or standardized interfaces, such as OPC Unified Architecture, e.g. Hensel et al. (2016) or the *FMI standard* (2021).

Co-simulation between Apros 6 and Virtual Drive could feasibly have been implemented in three different ways in this study. Firstly with a custom approach specific for Virtual Drive and Apros 6, secondly using OPC Unified Architecture and thirdly using FMI.

The first approach with a custom solution was ruled out immediately, as the efforts would not allow interoperability with any other systems. The second approach was more feasible, since Apros 6 already supports OPC UA, see Miettinen (2012). Further, Virtual Drive implements OPC Data Access for reading and writing some variables. Crucially though, not all variables are available in the OPC DA interface and the simulation control is implemented only with a custom interface. Thus, implementing co-simulation would require changes directly to Virtual Drive's implementation, as well as an OPC UA-to-DA conversion, such as *OPC UA Proxy* (2021).

The third approach, FMI, would enhance Apros 6 with the capability to import FMU models. For Virtual Drive, an FMI implementation would allow similar interoperability with tools in the FMI ecosystem. Further, a client-side library implementing the entirety of the Virtual Drive's custom simulation control interface existed and using that allowed rapid implementation of the FMI interface in the form of a prototype FMI Wrapper.

DOI 10.3384/ecp21181681

2.2 Functional Mock-up Interface standard

FMI is a standard that defines a set of functions, in native C, that a coordinator and a model must implement. It is supported by many tools, see *fmi-standard tools* (2021), and allows linking different simulation software and exchange of data. One software component must take the role of a coordinator that imports other software components in the form of Functional Mock-up Units (FMU models). An FMU model is essentially an archive (i.e. a .zip file), containing a modelDescription.xml file in the root, and either binary or source files. On Windows, the binaries are Dynamic Link Libraries (.dll), while on Linux they are Shared Objects (.so). The modelDescription.xml contains a list of variables, metadata about the model, FMI versions supported by the model and more. (Blochwitz et al. 2012), (Chen, Huhn, and Fritzson 2011)

Hauf et al. (2017) describe the history of the FMI standard, as well as the differences between model exchange and co-simulation sub-standards within FMI. For the purpose of this study, co-simulation was the only approach that made sense, since Virtual Drive is a complete package that contains its own mathematical calculations, essentially a black box from the point-of-view of other systems.

Functional Mock-up Interface can suffer from some weaknesses, e.g. its floating-point representation of time (Fabio et al. 2017). However, in our study this was never a problem, as the step-sizes were larger than 0.1 seconds. For such large step sizes, floating-point representation of time was accurate.

2.3 Apros 6 FMI importer

Apros 6 (2021) contains two main parts, a solver written in Fortran and a desktop application written in Java. No version of Apros 6 that has been released to date (Apros versions 6.10.x and older) comes with built-in support for importing and simulating FMU models. However, Apros 6 can be extended using *Eclipse* (2021) based plugins. In this study, a plugin was created that allows Apros 6.9 and newer versions to import and simulate FMU models that support FMI 1.0 and FMI 2.0 co-simulation.

Representing an FMU model in a simulation tool like Apros was straight forward with user components (UC). User components in Apros 6 are re-usable blocks that support scripting using the *SCL* (2021) language. Input and output signals on a UC allows data-flow between the FMU model and other components in the Apros model, while FMU parameters can be represented using UC properties. The FMI importer plugin uses *Simantics FMIL* (2021) as a Java implementation of the FMILibrary and the user component's SCL scripts import these Java functions. Due to the technical implementation of the Simantics FMIL, only x64-bit FMU models are supported in Apros. The component call flow and data flow of the plugin's auto-generated user components can be seen in Figure 2. Figure 3 shows an auto-generated Apros 6 user component from an import FMU model, as it appears in the Apros 6 model browser and diagram.

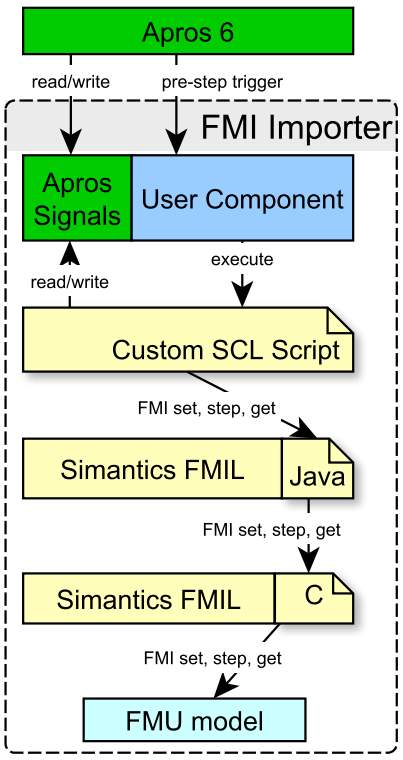

Figure 2. Component call- and dataflow for the Apros 6 FMI importer plugin. Apros 6 triggers user component's SCL scripts each step. These scripts internally use the Simantics FMIL implementation, included in the plugin. Apros signals are written to FMU models as inputs and FMU model outputs are written to Apros signals

2.4 Virtual Drive's FMI Wrapper

ABB's Virtual Drive is a Windows x86 executable that can be configured using *Drive Composer Pro* (2021) desktop application. Drive Composer Pro requires the add-on "ABB Virtual Drive" to allow creation, configuration and simulation of Virtual Drives. *DriveSize* (2021) is a product catalogue look-up tool that shows the size and specifications of ABB's drive products. These physical products can then be manually created as virtual simulation models using Drive Composer Pro and Virtual Drive. In this study, Virtual Drives were created with the specifications of a pump OEM, using DriveSize tool to find suitable equipment sizes to represent underdimensioned, overdimensioned and optimal equipment.

Virtual Drive uses the same software as physical drive equipment. By using Virtual Drive in simulation, rather than a simple modelica model of a drive and motor, the models can achieve results that reflect the real-world behaviour of drives more accurately.

FMI was possible to implement with two approaches, either directly or as a proxy. A direct approach would've

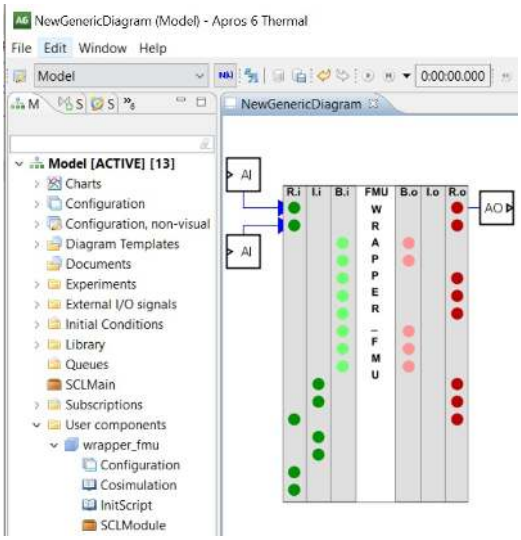

Figure 3. An auto-generated user component when importing an FMU model file using the Apros 6 FMI importer

meant packaging Virtual Drive itself as an FMU model, while a proxy approach would use an FMU model that internally uses a remote interface for communication with the Virtual Drive. The proxy approach was further divided into two sub-approaches: Tool-coupling or standalone. In a tool-coupling approach, Virtual Drives would be configured and started outside of the FMU models context and the FMU model would only connect to existing drives. A standalone approach would've meant packaging the virtual drive executable inside the FMU model and having it automatically started by the main FMU model.

The approach that was chosen was a tool-coupling proxy approach. Virtual Drive could only be compiled to x86 architecture, due to limitations in its dependencies. Thus, the direct approach would also only support x86 architecture and thus be incompatible with the Apros FMI importer plugin. A proxy approach would allow the creation of an FMU model that supports x64 and x86 architectures. Additionally, a proxy approach made sense because Virtual Drive already implemented a custom remote interface for simulation control, supported by e.g. Drive Composer Pro. A .NET client-side library implementing all of these remote function calls had been developed by ABB prior to this study, written in C#. A tool-coupling approach was chosen, as this would allow Drive Composer Pro to still be used in the configuration and monitoring of Virtual Drives. Version management would also be separated from the FMI Wrapper, allowing it to stay up-to-date with future releases of Virtual Drive and Drive Composer Pro, as long as the custom remote interface for controlling Virtual Drive does not changed.

2.5 Apros model

The process under study was modelled using the Apros simulator as it is intended for system-wide fluid process modelling. The modelled process is show schematically in Figure 4.

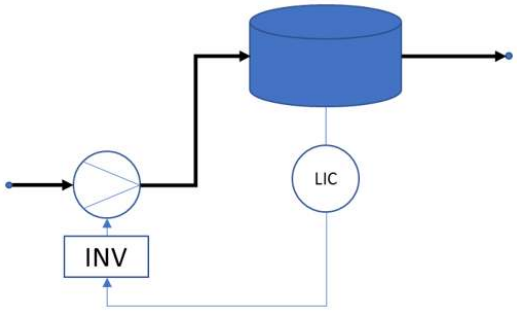

Figure 4. Schematic of the process under study, showing main process components and the liquid level control loop.

In the process we pump water to a tank, whose outlet flow varies. The goal of the level control loop (LIC) is to keep the liquid level within an allowable range of its setpoint. In the basic setting the level controller's output is the pump rotation speed setpoint. This is given to the drive+motor (INV) model who in turn returns the pump shaft torque. The process simulator then uses this value to determine the pump rotation speed and from that its head. This head value then goes to the pressure-flow solver of the simulator. In this case the fluid pressure-flow behaviour was modelled as 1D homogenous two-phase flow using dynamic conservation equations for mass, energy and momentum. The modelling was done with a graphical user interface, with no need to explicitly write the governing equations. In addition to the fluid flow components, i.e. pipes, valves, pumps and tanks, the model also included automation component. Namely, in the model we implemented a liquid level control loop. In later stages of the investigation we extended the control loop to include a feedforward term from the tank outflow measurement. To test the different drivetrain dimensionings and control structures, a simulation test sequence was used, see Figure 5.

3 Results

3.1 FMI Wrapper related results

The prototype FMI Wrapper created during this study implements the FMI 2.0 co-simulation interface. The existing client-side library, with remote function calls for Virtual Drive, had been implemented in C#, thus it was deemed the simplest solution to also write the FMI Wrapper in C# and use the existing library directly. A .NET C# project that compiles into both x86 and x64 Dynamic Link Libraries (.dll) was created, utilizing FMI c-headers

DOI 10.3384/ecp21181681

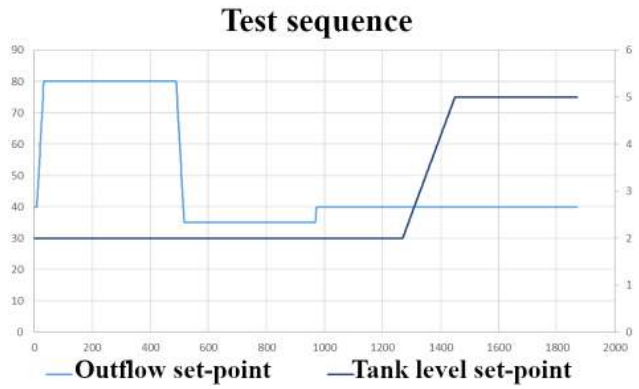

Figure 5. Simulation test sequence: Part 1: outflow variables 40 kg/s → 80 kg/s → 35 kg/s → 40 kg/s (light blue curve, primary y-axis), Part 2: tank level setpoint ramped from 2m to 5m (dark blue curve, secondary y-axis).

and the existing client-side library. Functions needed to be explicitly configured using *DllExport* (2021) for .NET like Listing 1, in order to make them compatible with native C, i.e. the FMI specifications.

Listing 1. A generic .NET DllExport usage example. All .NET projects that need to be compiled as .dll and need compatibility with native C can use this approach.

```
[DllExport("fmi2DoStep",
    CallingConvention =
        CallingConvention.Cdecl)]
public static fmi2Status fmi2DoStep(
    IntPtr c,
    double currentCommunicationPoint,
    double communicationStepSize,
    [MarshalAs(UnmanagedType.Bool) bool
        noSetFMUStatePriorToCurrentPoint])
{
    ModelInstance m = ((GCHandle.c).Target
        as ModelInstance);
}
```

Virtual Drives are given a unique local identifier when created using Drive Composer Pro. This identifier represents the local channel used to communicate with that drive instance. When importing the FMU model of the FMI Wrapper, this identifier must be set using the FMI integer-type parameter VirtualDriveLocalNodeID and it must match the identifier of an existing Virtual Drive instance that is running on the same machine.

The FMI Wrapper was tested in various simulators to confirm its implementation of the FMI 2.0 co-simulation interface was correct. The Apros 6 FMI importer plugin created during this study, FMU compliance checker, *FMI Toolbox import* (2021) and *Simulink FMI import* (2021) were successfully able to import and simulate Virtual Drive through the FMI Wrapper.

With the FMI Wrapper, Virtual Drive gained interoperability with multiple simulation tools, including Apros 6. It comes in a format that is familiar to those who already use Virtual Drives, since Drive Composer Pro is still used for configuring Virtual Drives.

3.2 Apros FMI importer related results

The Apros FMI importer went through two phases of development: In the first phase, a user component was manually created to represent the Virtual Drive's FMI Wrapper and its parameters, inputs and outputs. This was created only for the FMI Wrapper and would not have worked for any other FMU model. In the second phase, the importer was generalized to allow importing of any x64-bit FMU models implementing either the FMI 1.0 or 2.0 co-simulation interface.

A copy of the imported FMU file is stored in the auto-generated user component, which will be copied to a well-known location within the plugin's filesystem before an FMU instance is started. Exporting an Apros model that contains an auto-generated FMU user component is possible and a copy of the FMU model will be stored in the exported Apros model. Importing such a model to an Apros version without the FMI importer plugin will succeed, but simulation will fail, as the user components' SCL scripts no longer find the required Java implementation.

With the new plugin, modellers can easily import FMU models to Apros and use them as familiar user components, saving modellers' time and bringing new interoperability options to Apros.

3.3 Process engineering related results

As was describe earlier, the study consisted of simulating three pumps, each with two drivetrains. In Figure 6 we present the liquid level behaviour of three pump-motor-drive combinations.

In the figure we show three drivetrains: the green line is the so called optimal dimensioning, the yellow line is an intentionally undersized dimensioning and the brown line is an oversized case. In the chart we can also see the liquid level setpoint (blue dotted line) as well as the acceptable range for the liquid level around it (dotted yellow line and dotted grey line). The thick lines depicts the liquid level behaviour during the simulation.

The optimally dimensioned drivetrain keeps the liquid level in its range, except for a short while during the setpoint change. The underdimensioned drivetrain fails immediately and the liquid level falls to nearly zero. Finally, the overdimensioned drivetrain keeps the level closer to the setpoint than the others. This is achieved with a larger energy consumption. More specifically, the specific energy consumptions were, respectively: 0.101 kWh/t, 0.278 kWh/t and 0.13 kWh/t. These numbers reflect the utter failure of the underdimensioned drivetrain: while it is unable to achive its target, it also uses a lot of energy in doing so.

In addition, we experimented with two alternative control structures. The alternative was to add a feedforward term from the tank outlet flow measurement. This was investigated only with the optimally dimensioned case, see Figure 7.

We see that, as expected, the feedforward control nearly perfectly compensates the outflow changes. In combina-

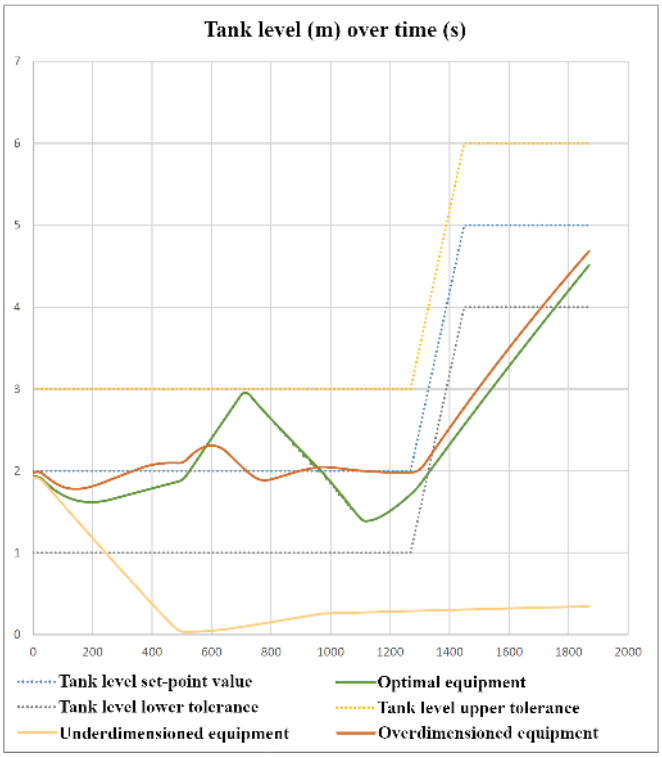

Figure 6. Liquid level behaviour with "optimal" (green line), "undersized" (yellow line) and "oversized" (brown line) drivetrains.

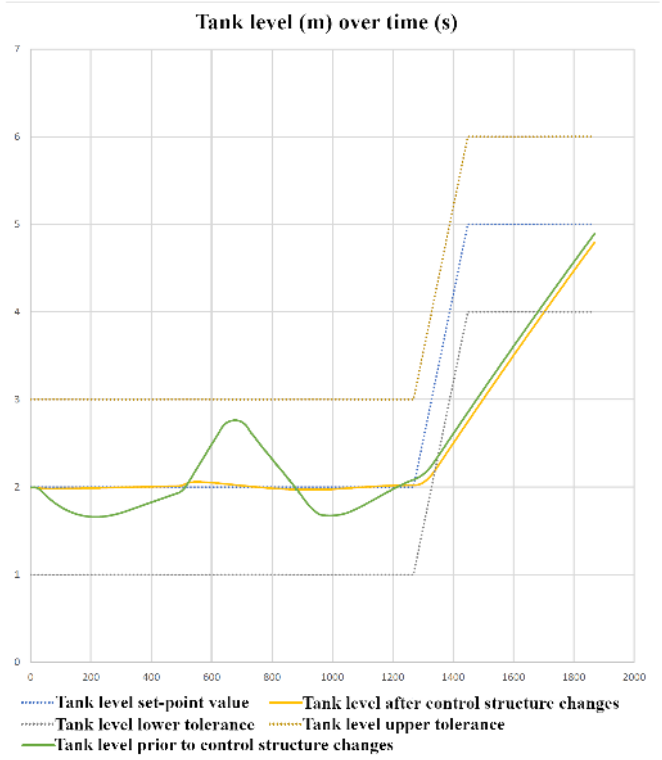

Figure 7. Liquid level behaviour with "optimal" drivetrain, using the two control structures. Green line = before changes, Yellow line = after changes.

tion with a smaller motor and frequency converter, it is able to reduce energy consumption from 0.101 kWh/t to 0.095 kWh/t, a 6% reduction. This is a significant cost and energy consumption reduction when applied to all drivetrains in a large process.

4 Analysis

4.1 FMI simulation speed

Speed was not critical during the creation of the Apros FMI importer nor for the prototype FMI Wrapper for Virtual Drive. However, Apros FMI importer plugin's speed has been compared to *Simulink 2020b* (2021) and *FMU compliance checker* (2021). The simulation speeds were tested with an example *ControlledTemperature.fmu* (2021) model, as well as the created FMI Wrapper with and without a connection to Virtual Drive. Without a connection, each FMI 2.0 function in the wrapper will simply return immediately, thus allowing us to analyse the speed loss caused purely by the Apros FMI importer. We simulated the FMU models in Apros multiple times in one-minute tests, with a step size of 0.1 and calculated the average real-time factor (RTF) for the models, with various numbers of simultaneous FMU models present in the model. In Simulink and FMU Compliance Checker, we specified 100 000.0 seconds as the target simulation time for the fastest cases, and 100.0 seconds for the slowest cases, with fixed step-size set to 0.1 seconds, and measured how long it took to simulate. The average of five executions per case was used.

The results for different simulator programs and different number of FMU instances can be seen in Table 1. The time spent loading and initializing the FMU models was not taken into account when calculating the total time, but was observed to be less than 1 second even when using 8 ControlTemperature.fmu models. This should affect the RTF factor seen in the tables positively by at most 1.7%, since $1/60 \approx 1.7$, and even then only for cases with 8 simultaneous models.

Clearly, the Apros FMI importer plugin is much slower that the other simulators, when using the ControlledTemperature.fmu. However, it should still be much faster with the unconnected FMI Wrapper than ControlledTemperature.fmu. Instead, it is much slower. The root cause for the slow speeds was not identified in this study and requires further investigation. An initial analysis of the speed issue would suggest that the problem is not in the Apros user component's themselves, i.e. the speed is not lost due to Apros simulation pre-step hooks triggering SCL scripts in the user components and updating signal values, but rather in the SCL functions and the underlying Java and C implementation of Simantics FMIL. Similar simulation speed issue could be seen by executing SCL scripts independently, without Apros.

As expected, connecting the Virtual Drive lead to slower simulation speeds for all simulators. In fact, the overhead introduced by the custom interface between FMI

DOI 10.3384/ecp21181681

Wrapper and Virtual Drive was so large that all simulators had at best RTF of around 3.0 for a single Virtual Drive instance, with step-size 0.1.

Table 1. Average simulation speed, as a real-time-factor (RTF), with varying number of simultaneous FMU model instances, step-size 0.1 seconds. RTF of 1.0 correspond to as fast as real-time, 2.0 means twice as fast as real-time, etc. Simulink 2020b FMU import, Apros FMI importer plugin and FMU Compliance Checker were tested. CT = ControlledTemperature.fmu, FW = FMI Wrapper without a connected Virtual Drive. FW+VD = FMI Wrapper with a connected Virtual Drive.

Simulator	Instances	CT	FW	FW+VD
Simulink	1	333	35000	3.0
-||-	2	298	25000	2.9
-||-	4	230	16000	2.5
-||-	8	136	12000	1.4
Apros	1	132	17.4	3.0
-||-	2	92.3	15.1	1.7
-||-	4	53.9	11.3	0.88
-||-	8	27.8	7.5	0.44
Checker	1	384	46000	3.2

The demonstration Apros model in this study had a different step-size than we used in the simulation speed tests. Additionally, variable step-sizes are taken, depending on what the model is currently calculating. This variable step-size is matched by the FMU instances in the model, which in turn directly affects the simulation speed. E.g. small step-sizes were taken by the Apros model when the check valve fluttered, while larger step-sizes were taken in steadier states. In the simulation speed tests, we forced the step-size to always be 0.1 seconds and the model contained only the FMU models.

4.2 Apros FMI importer limitations

Due to the usage of Simantics FMIL as the base implementation for the FMI, x86 FMU models could not be imported in x64 desktop environments, e.g. in Apros 6 desktop. Further limitations inherited from this base implementation were a lack of model exchange support. However, both FMI 1.0 and 2.0 co-simulation standards were implemented.

Unimplemented functions for FMI 2.0 were all model exchange functions, as well as:

- fmi2SetRealInputDerivatives
- fmi2GetRealOutputDerivatives
- fmi2CancelStep
- fmi2GetDirectionalDerivative
- fmi2DeSerializeFMUstate
- fmi2SerializeFMUstate
- fmi2SerializedFMUstateSize
- fmi2FreeFMUstate
- fmi2SetFMUstate
- fmi2GetFMUstate
- fmi2Reset

4.3 FMI Wrapper limitations

The prototype FMI Wrapper for Virtual Drive was implemented for both x86 and x64 architectures. The wrapper's FMU model is only available for Windows operating systems, just like Virtual Drive. The Virtual Drive and the FMI Wrapper must also physically be located on the same machine, as the remote interface requires local visibility. It is a remote interface only in the sense that the Virtual Drive executable can be started by another program than the program that imports the FMI Wrapper's FMU model.

Most configuration of Virtual Drives must still be performed using Drive Composer Pro's user interface, e.g. selection of motor and parametrization of the drive's size. Most output variables can also only be monitored using Drive Composer Pro. The prototype FMI Wrapper only exposes a small sub-selection of variables, such as torque as output, and motor speed as input. Thus, the FMI wrapper in its current state is far from standalone, since users are unable to define new Virtual Drive instances exclusively using the FMU model and the variables exposed by it.

Maximum simulation speed continues to be a challenge, due to time spent by functions in Virtual Drive and time spent by the custom remote interface between FMI Wrapper and Virtual Drive. FMI Wrapper was only implemented for the FMI 2.0 co-simulation interface, but not all functions were implemented. Virtual Drive did not support saving or loading the internal state, thus state-related functions could not be implemented.

Unimplemented functions for FMI 2.0 were all model exchange functions, as well as:

- fmi2SetRealInputDerivatives
- fmi2GetRealOutputDerivatives
- fmi2CancelStep
- fmi2GetDirectionalDerivative
- fmi2DeSerializeFMUstate
- fmi2SerializeFMUstate
- fmi2SerializedFMUstateSize
- fmi2FreeFMUstate
- fmi2SetFMUstate
- fmi2GetFMUstate

5 Discussion

Implementing the prototype FMI Wrapper was remarkably simple for Virtual Drive, since there already existed a complete client-side implementation of Virtual Drive's custom simulation interface in a C# library. An initial

challenge had to be overcome, namely the C#-to-native-C function exports, but mechanisms for dealing with that were found.

The FMI Wrapper inherited the limitations of the existing Virtual Drive remote control interface, e.g. a lack of storing the internal state of Virtual Drive in a serializable blob and only having visibility to a small sub-selection of variables. The prototype FMI Wrapper developed requires Drive Composer Pro to be used for the creation and configuration of Virtual Drives.

5.1 Future Work

Apros FMI importer and the base Simantics FMIL will need a thorough investigation to reduce the overhead from using the Java FMI implementation. Additionally, parallel execution of FMU models within an Apros model, rather than serial, will reduce simulation speed losses caused by simulating multiple simultaneous FMU models.

The interoperability of Apros 6 and Virtual Drive was only tested with a simple demonstration model during this study. Future work should include validation of this approach using a real customer case. A collaborative digital twin, using Apros 6 dynamic process simulation in co-simulation with Virtual Drive, should be created to find the optimal equipment selection of a real-life process. Other ABB software and pump OEMs equipment can be used for a more detailed collaborative digital twin, assuming their simulation software is compatible with the Functional Mock-up Interface. Implementing FMI compatibility from scratch to these equipment simulators will be needed as part of future real customer cases, if they do not already implement it.

5.2 Conclusion

Traditional process design uses a waterfall model for equipment selection, where multiple sub-contractors add too high risk margins to the sizing of their equipment in order to ensure components fulfill critical requirements. In this paper we presented the first steps towards a collaborative digital twin approach as an alternative way of designing a process. This approach aims to remove overdimensioning, leading to up-front cost savings and total life-cycle cost savings for the whole process.

The approach was demonstrated with a Virtual Drive instance connected to a simple Apros model that had a few equipment selection options. However, the approach should be possible to utilize when optimizing the equipment selection in an entire industrial process plant. The approach was made possible with interoperability between simulators and we have presented Functional Mock-up Interface as a candidate for achieving this interoperability.

Acknowledgements

This activity has received funding from the European Institute of Innovation and Technology (EIT). This body of the European Union receives support from the European Union's Horizon 2020 research and innovation programme.

The authors would like to thank Reino Ruusu for their support implementing the prototype Apros FMI importer. The authors would also like to thank Daniel Rogoz and Roman Okolovich for their support implementing the prototype Virtual Drive FMI Wrapper.

References

Apros 6 (2021). URL: https://www.apros.fi/ (visited on 2021-04-23).

Blochwitz, T. et al. (2012-09). "Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models". In: DOI: 10.3384/ecp12076173.

Chen, Wuzhu, Michaela Huhn, and Peter Fritzson (2011). "A Generic FMU Interface for Modelica". In:

ControlledTemperature.fmu (2021). URL: https://github.com/modelica/fmi-cross-check/blob/master/fmus/2.0/cs/win64/Dymola/2017/ControlledTemperature/ControlledTemperature.fmu (visited on 2021-04-28).

DllExport (2021). URL: https://github.com/3F/DllExport (visited on 2021-04-23).

Drive Composer Pro (2021). URL: https://new.abb.com/drives/software-tools/drive-composer (visited on 2021-04-23).

DriveSize (2021). URL: https://new.abb.com/drives/software-tools/drivesize (visited on 2021-04-23).

Eclipse (2021). URL: https://www.eclipse.org/ide/ (visited on 2021-04-23).

Fabio, Cremona et al. (2017). "Hybrid co-simulation: it's about time". In: *Software and Systems Modeling* 18, pp. 1655–1679. DOI: https://doi.org/10.1007/s10270-017-0633-6.

Fei, Tao et al. (2018). "Digital twin-driven product design, manufacturing and service with big data". In: *The International Journal of Advanced Manufacturing Technology* 94, pp. 3563–3576. DOI: https://doi.org/10.1007/s00170-017-0233-1.

FMI standard (2021). URL: https://fmi-standard.org/ (visited on 2021-04-27).

FMI Toolbox import (2021). URL: https://www.modelon.com/products-services/modelon-deployment-suite/fmi-toolbox/ (visited on 2021-04-29).

fmi-standard tools (2021). URL: https://fmi-standard.org/tools/ (visited on 2021-04-27).

FMU compliance checker (2021). URL: https://github.com/modelica-tools/FMUComplianceChecker (visited on 2021-04-26).

Hauf, Dominik et al. (2017). "Multifunctional use of functional mock-up units for application in production engineering". In: *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pp. 1090–1095. DOI: 10.1109/INDIN.2017.8104925.

Hensel, Stephan et al. (2016). "Co-simulation with OPC UA". In: *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, pp. 20–25. DOI: 10.1109/INDIN.2016.7819127.

Jiapeng, Guo et al. (2019). "Modular based flexible digital twin for factory design". In: *Ambient Intelligence and Humanized Computing* 10, pp. 1189–1200. DOI: https://doi.org/10.1007/s12652-018-0953-6.

Lappalainen, Jari (2019). "Extending mechanistic thermal-hydraulic modellling and dynamic simulation for new industrial applications". Doctoral dissertation. Aalto University,

Department of Chemical Engineering. DOI: http://urn.fi/URN:ISBN:978-952-60-8759-7.

Meer, A. A. van der et al. (2020). "Simulation-Based Assessment Methods". In: *European Guide to Power System Testing: The ERIGrid Holistic Approach for Evaluating Complex Smart Grid Configurations*. Ed. by Thomas I. Strasser, Erik C. W. de Jong, and Maria Sosnina. Cham: Springer International Publishing, pp. 35–50. ISBN: 978-3-030-42274-5. DOI: 10.1007/978-3-030-42274-5_3. URL: https://doi.org/10.1007/978-3-030-42274-5_3.

Miettinen, T. (2012). "Synchronized Cooperative Simulation: OPC UA Based Approach". Master's thesis. Aalto University School of Electrical Engineering. URL: http://lib.tkk.fi/Dipl/2012/urn100571.pdf.

Motor-driven Equipment Research Package (2021). URL: https://omdia.tech.informa.com/-/media/tech/omdia/brochures/electric-motor-systems/fans-blowers-database---2020.aspx (visited on 2021-05-04).

Nishida, N., Y. A. Liu, and A. Ichikawa (1976). "Studies in chemical process design and synthesis II. Optimal synthesis of dynamic process systems with uncertainty". In: *AIChE Journal* 22.3, pp. 539–549. DOI: https://doi.org/10.1002/aic.690220318.

Nouidui, Thierry, Michael Wetter, and Wangda Zuo (2014). "Functional mock-up unit for co-simulation import in EnergyPlus". In: *Journal of Building Performance Simulation* 7.3, pp. 192–202. DOI: 10.1080/19401493.2013.808265. eprint: https://doi.org/10.1080/19401493.2013.808265. URL: https://doi.org/10.1080/19401493.2013.808265.

OPC UA Proxy (2021). URL: https://opcfoundation.org/products/view/opc-ua-proxy-1 (visited on 2021-04-23).

Sadjina, Severin et al. (2018-09). "Distributed Co-Simulation of Maritime Systems and Operations". In: DOI: 10.1115/1.4040473.

SCL (2021). URL: https://dev.simantics.org/index.php?title=SCL_Tutorial (visited on 2021-04-23).

Simantics FMIL (2021). URL: https://gitlab.simantics.org/simantics/fmil (visited on 2021-04-23).

Simulink 2020b (2021). URL: https://se.mathworks.com/downloads/ (visited on 2021-04-28).

Simulink FMI import (2021). URL: https://se.mathworks.com/help/simulink/ug/work-with-fmi-in-simulink.html (visited on 2021-04-29).

Vega, P. et al. (2014). "Integrated design and control of chemical processes – Part I: Revision and classification". In: *Computers and Chemical Engineering* 71, pp. 602–617. DOI: doi:10.1016/j.compchemeng.2014.05.010.

Virtual Drive (2021). URL: https://new.abb.com/drives/software-tools/drive-composer (visited on 2021-04-23).

Waide, Paul and Conrad Brunner (2011-01). "Energy-Efficiency Policy Opportunities for Electric Motor-Driven Systems". In: URL: https://www.iea.org/reports/energy-efficiency-policy-opportunities-for-electric-motor-driven-systems.

Westerberg, Arthur (2004). "A retrospective on design and process synthesis". In: *Computers and Chemical Engineering* 28, pp. 447–458. DOI: http://dx.doi.org/10.1016/j.compchemeng.2003.09.029.