Composer

A prototype multilingual model composition tool

Erik Billing and Martin Servin

And contributions by Stefan Hedman and Claude Lacoursière

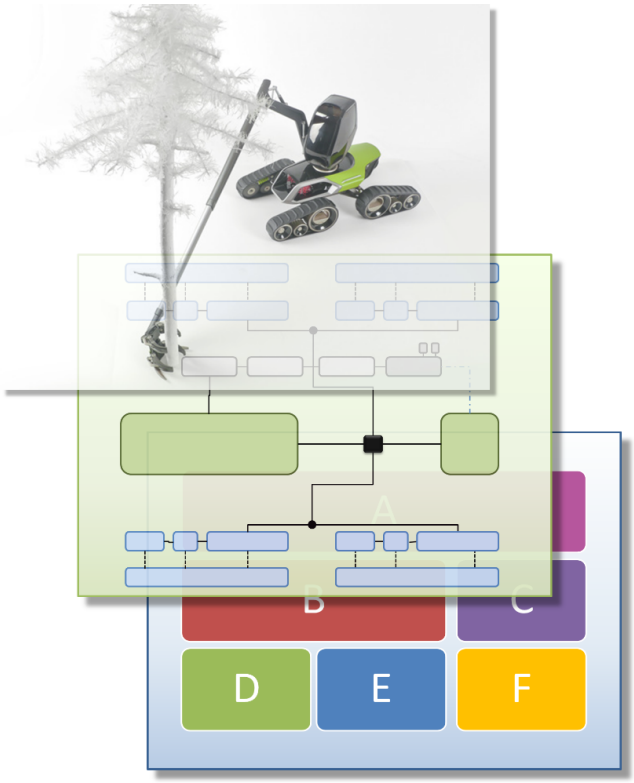
February 25, 2013

Composer

A tool for composing simulation of dynamical systems

- multilingual
- database centric
- web based

An experiment and a prototype for project *Simovate*

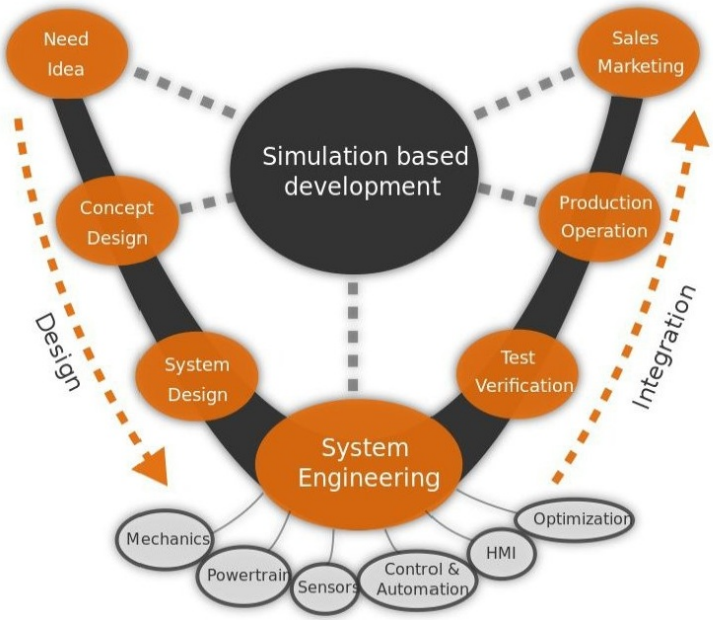


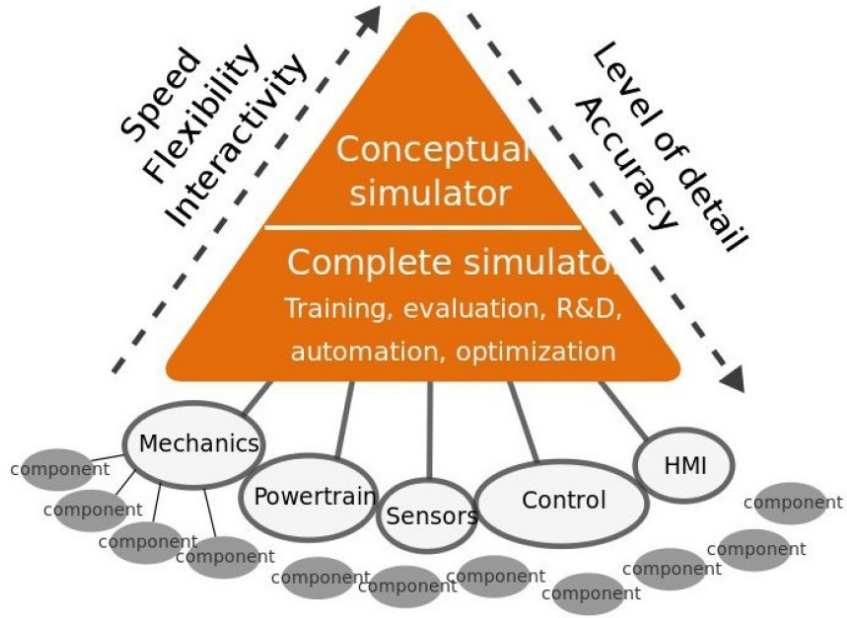
`http://imuit.cs.umu.se/composer`
`www.org.umu.se/umit/english/project-activities/simovate`

Axolotl Selective Bio-Harvester, Nick Ross, Umea Institute of Design

Innovation with simulation based development

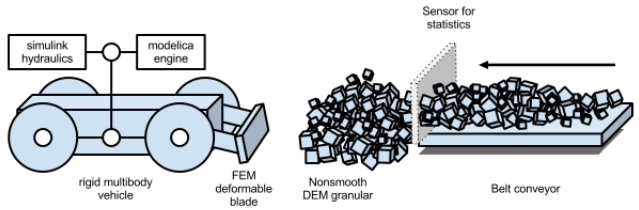
- efficient and flexible management of data and models
- concept→system engineering→marketing & training
- interoperability of different simulation software
- different level-of-detail and complexity

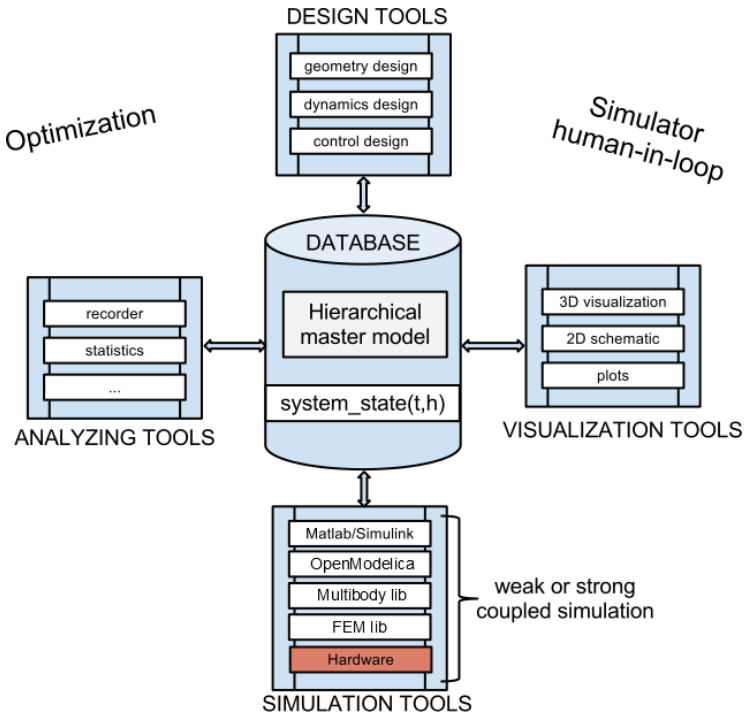




www.org.umu.se/umit/english/project-activities/simovate

Database centric approach





one system - multiple *views*

- database: model and data
- views: design, simulation, analysis, visualization

requires

- file & data formats
- communication protocols
- generic block-diagram representation
- weak and strong coupled simulation

Composer - a web based simulation editor

server-client

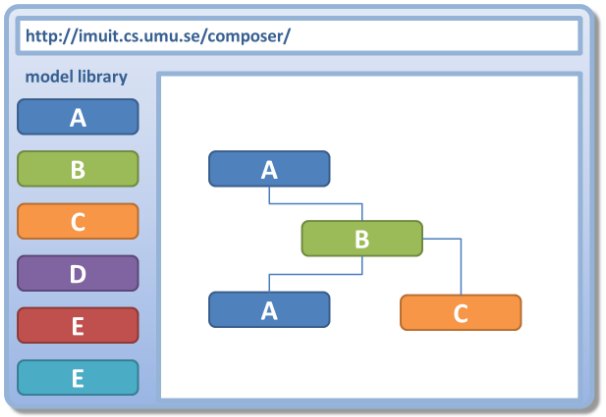
- server: models, data, compile, simulate (hpc)
- client: editor in browser

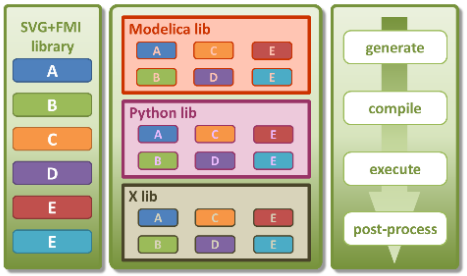
web & html5

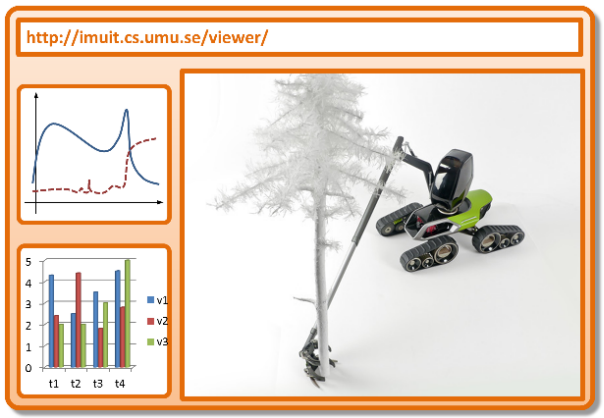
- SVG
- Javascript
- JSON
- python

generic block-diagram XML

- SVG_{FMI} = SVG + FMI metadata (Model Description)

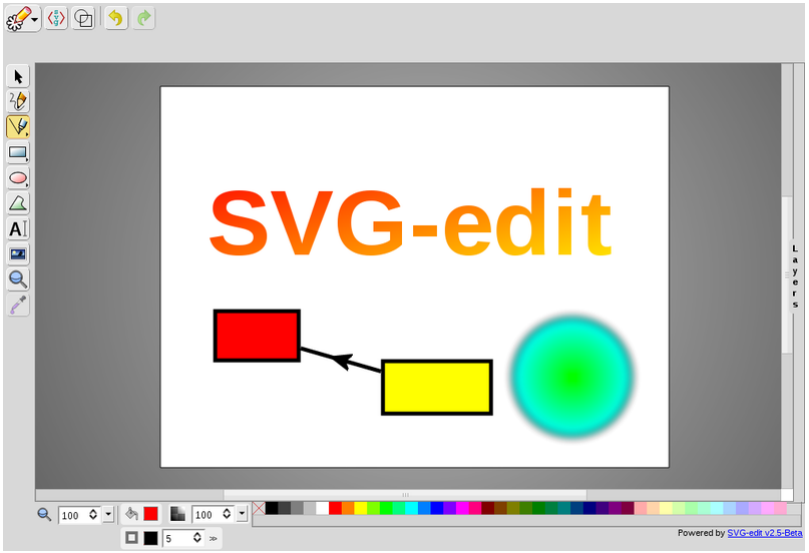


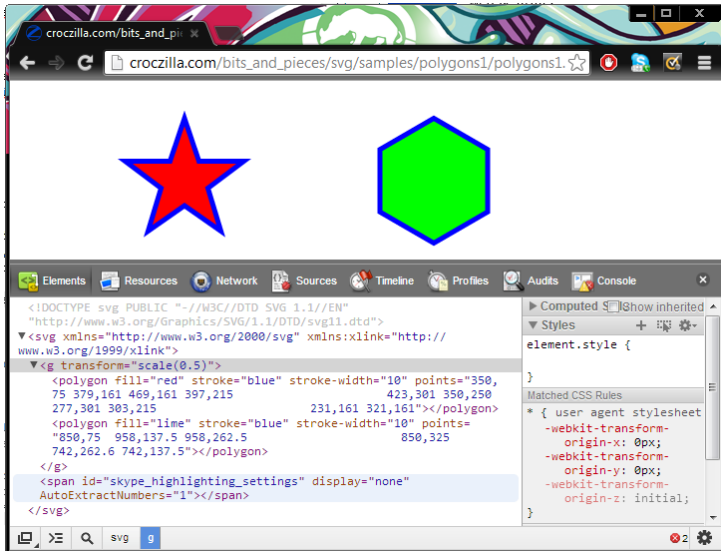




SVG - Scalable Vector Graphics

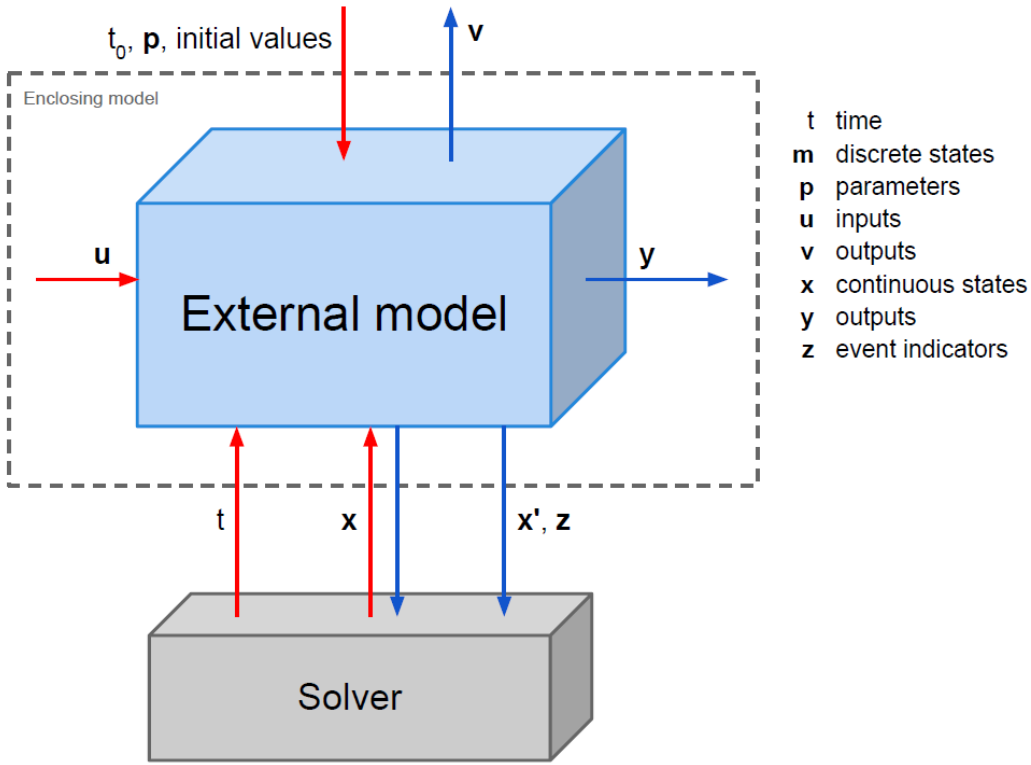
- XML-based vector image format for interactive 2D graphics
- open standard supported by web browsers
- Functionality: paths, shapes, text, colour, filters, ... , interactivity, linking, scripting, metadata





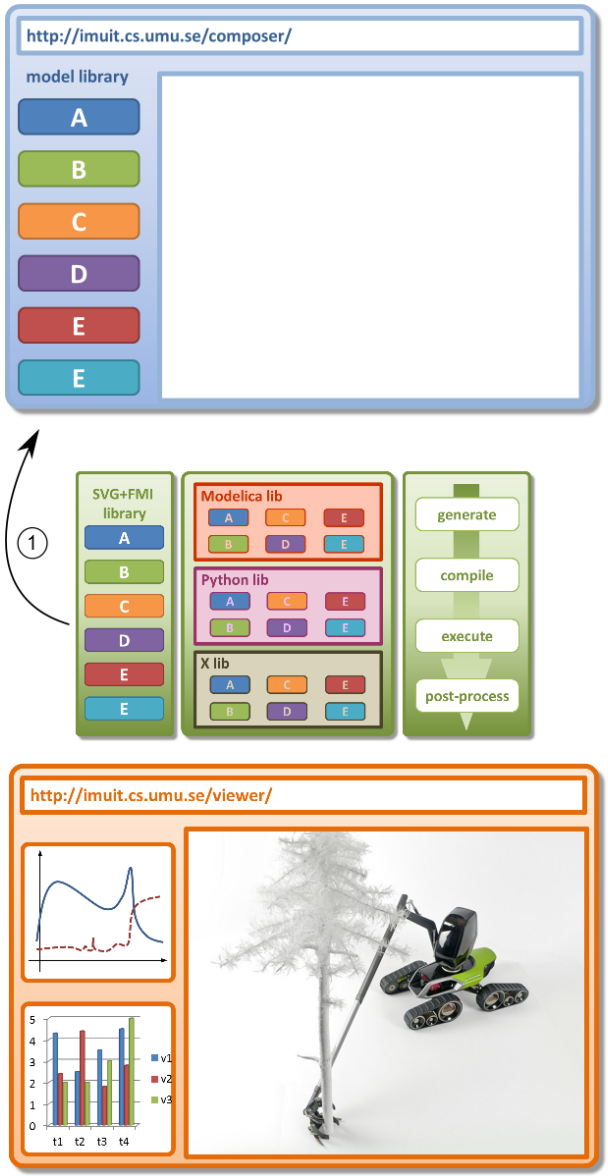
FMI - Functional Mockup Interface

- tool independent standard to support both model exchange and co-simulation of dynamic models
- standard specification, XML schema files, C-header
- an FMU a FMI implementation of a simulation model



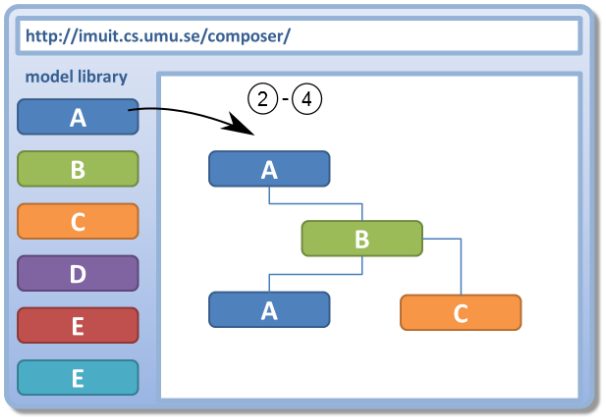
Composer - work flow

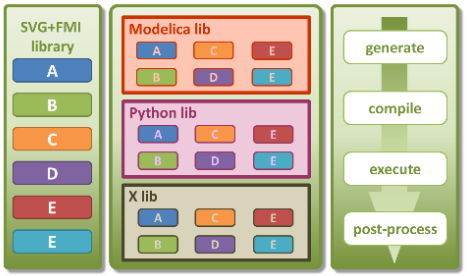
1. get SVG_{FMI} lib

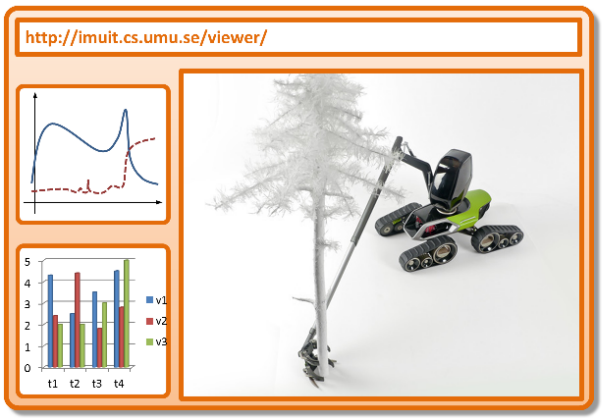


Composer - work flow

1. get SVG_{FMI} lib
2. add model A, B, C,...
3. create connections
4. edit model properties

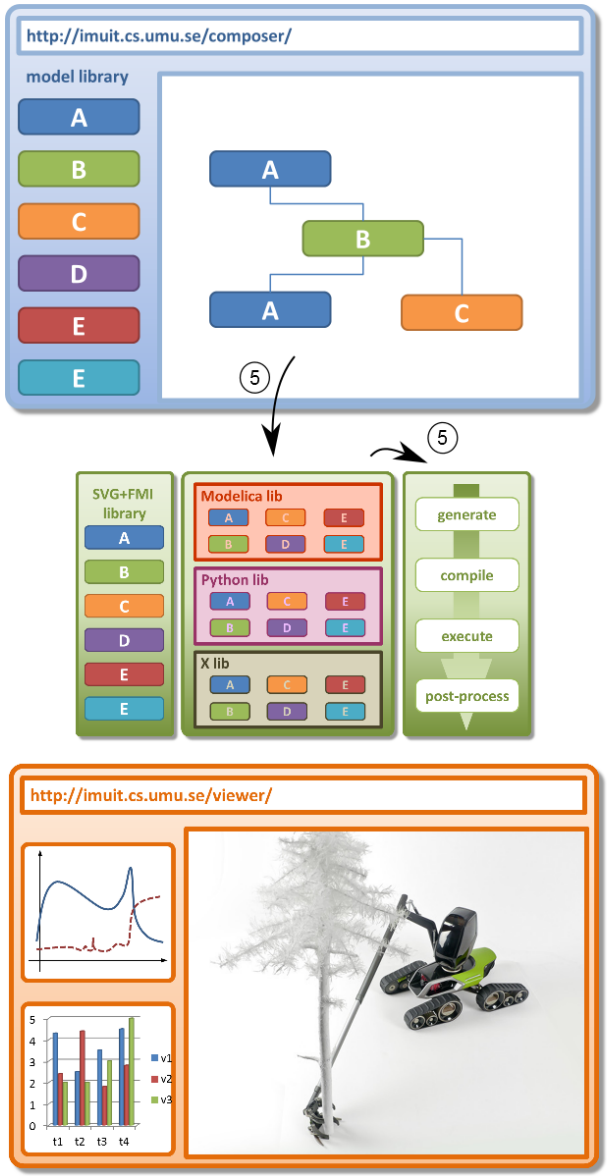






Composer - work flow

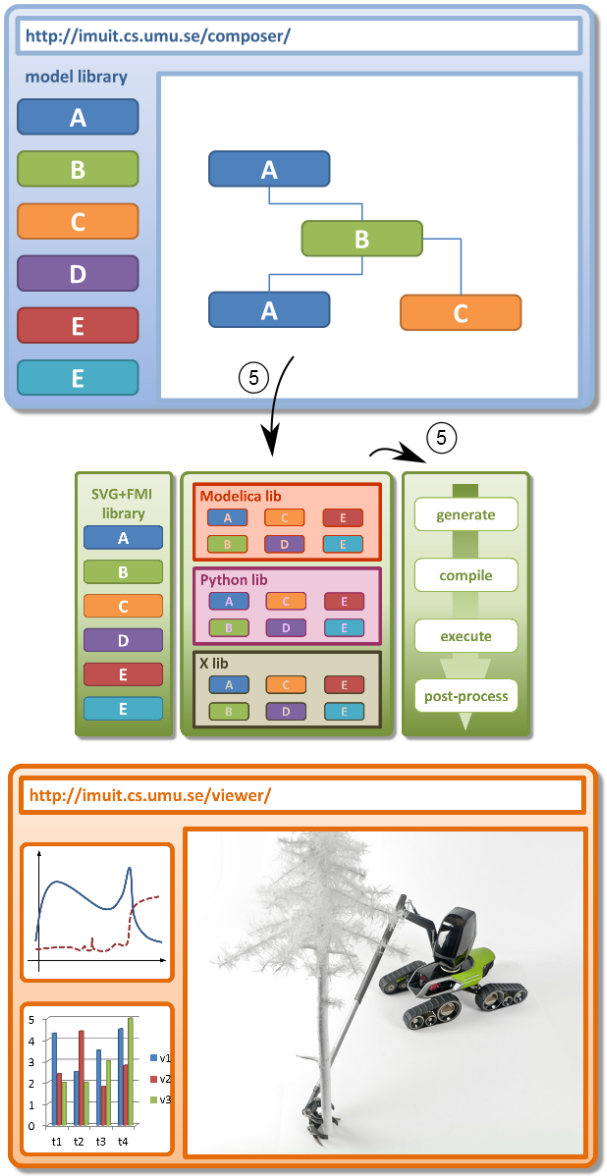
1. get SVG_{FMI} lib
2. add model A, B, C,...
3. create connections
4. edit model properties
5. launch simulation type X



Composer - work flow

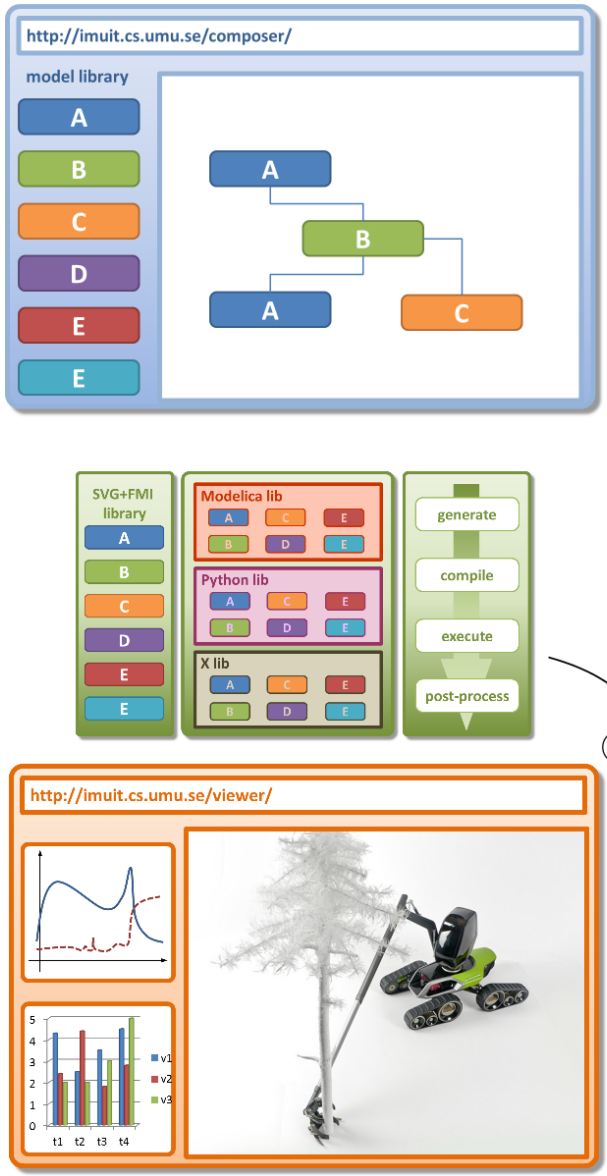
5. launch simulation type X

- send SVG_{FMI} to server
- code generation from lib X
- compile
- execute
- create output ($\leftarrow \text{SVG}_{\text{FMI}}$)



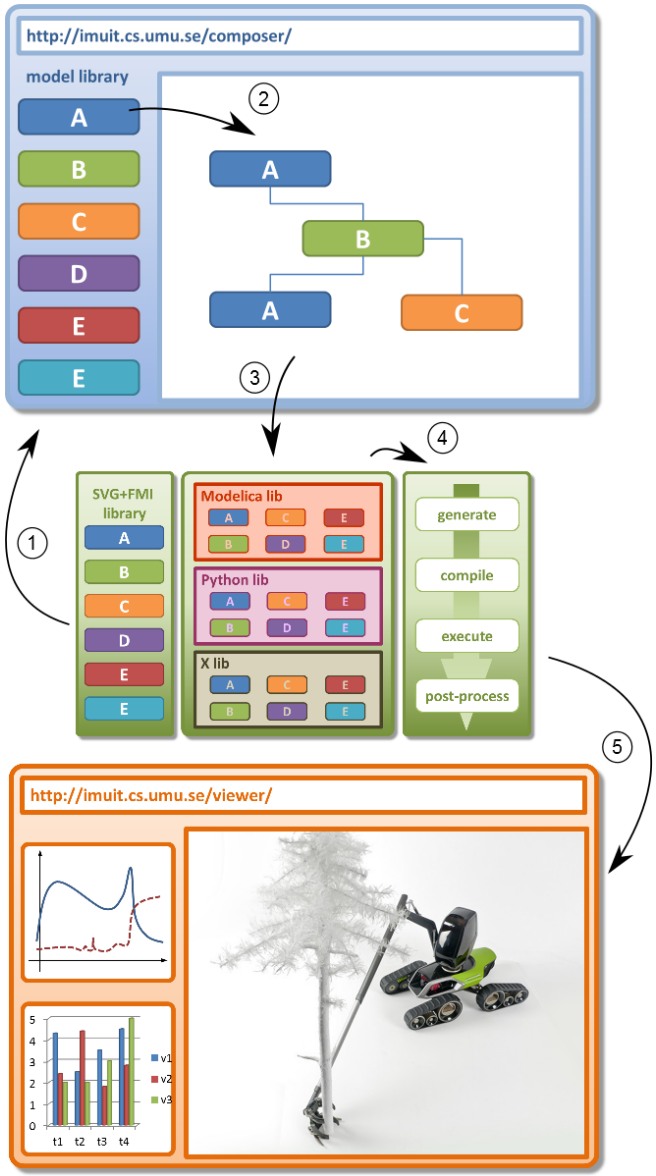
Composer - work flow

1. get SVG_{FMI} lib
2. add model A, B, C,...
3. create connections
4. edit model properties
5. launch simulation type X
6. receive output



Composer - work flow

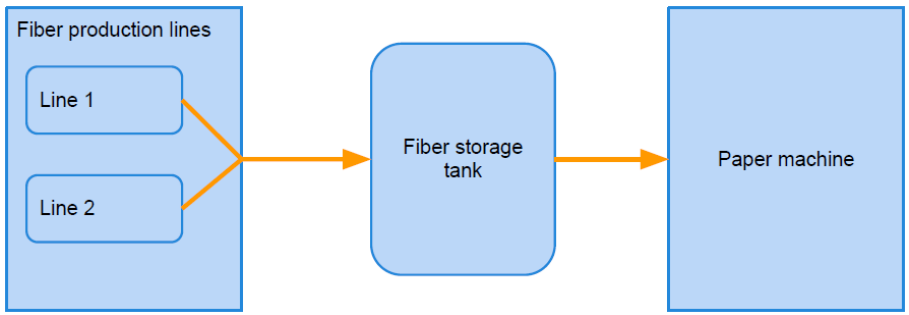
1. get SVG_{FMI} lib
2. add model A, B, C,...
3. create connections
4. edit model properties
5. launch simulation type X
6. receive output
7. modify or done

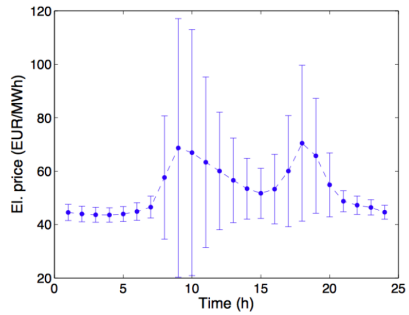


Example: fiber production line

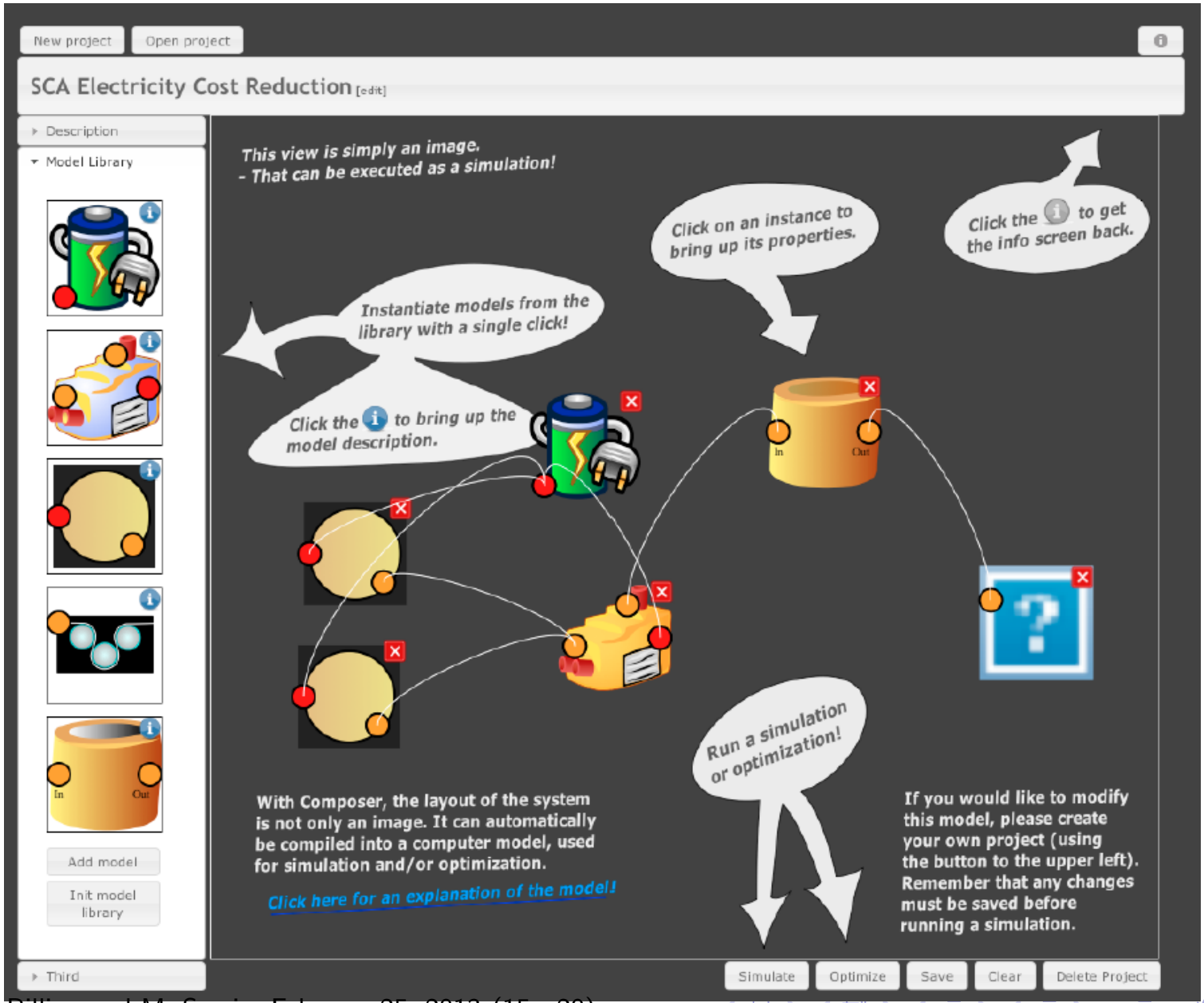
Scheduling strategy to reduce electricity cost in a paper plant

- dynamical simulation with modelica
- $\rightarrow$ discrete event simulation with Python
- $\rightarrow$ mixed integer optimization problem

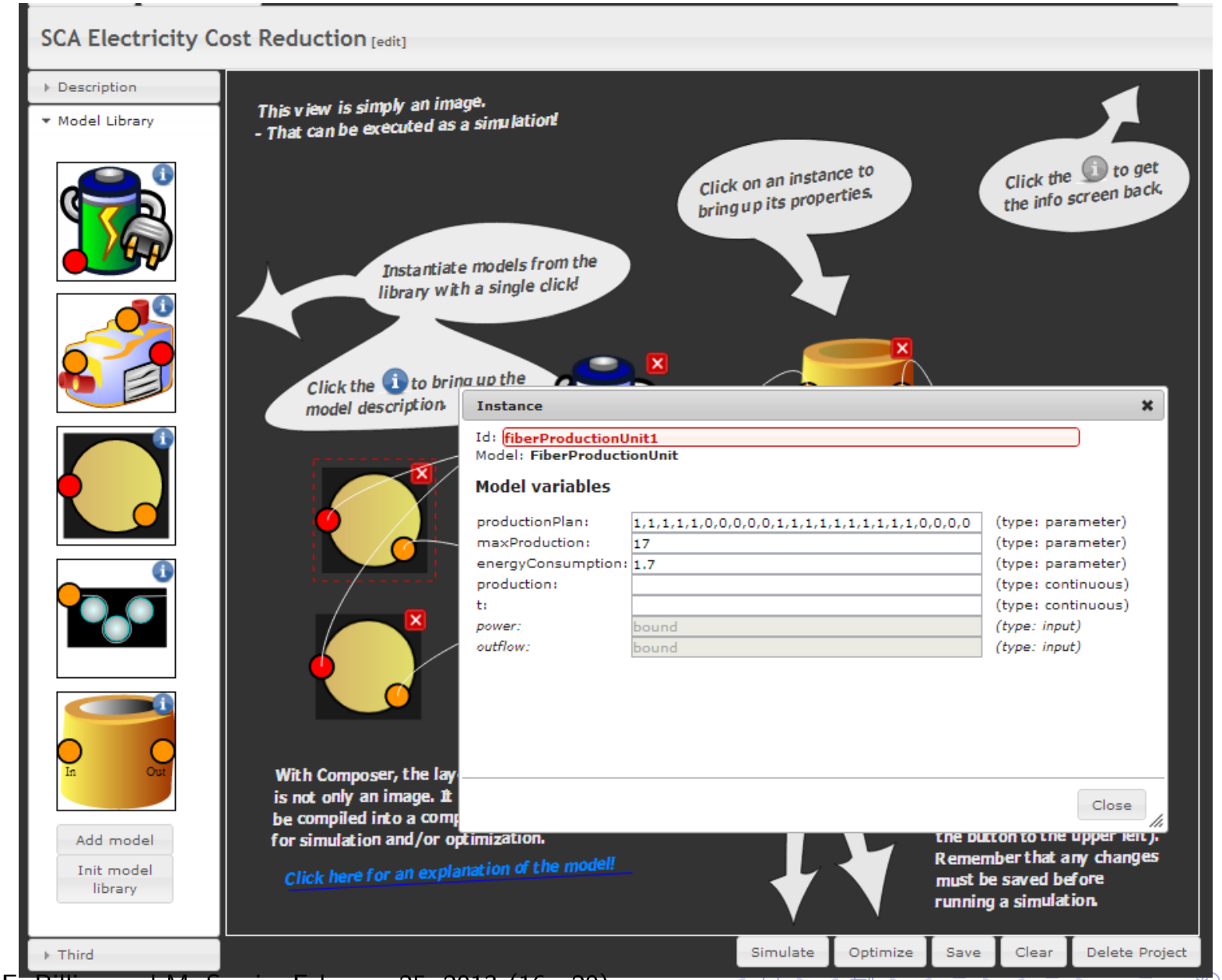




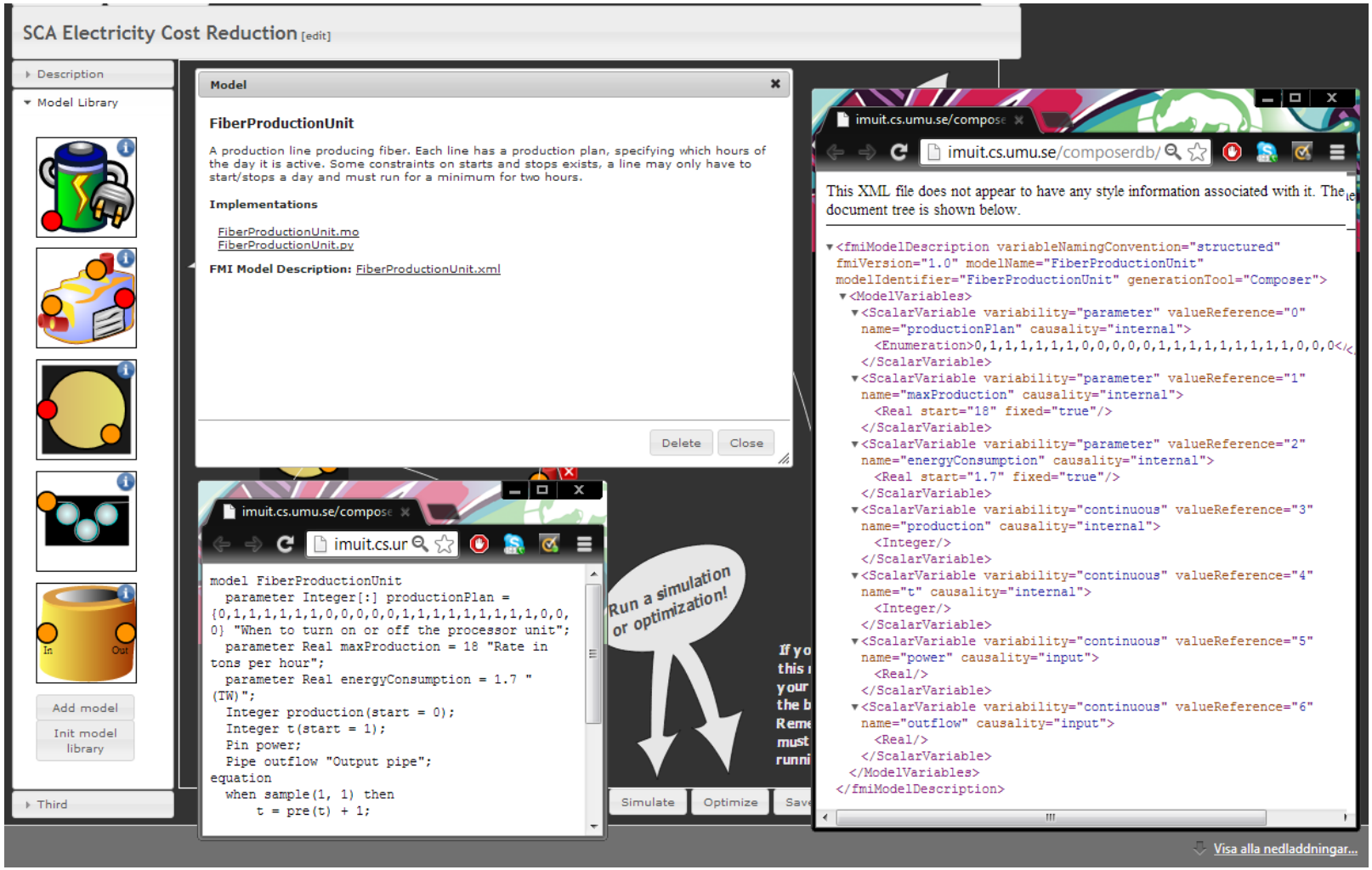
Composer prototype: fiber production line



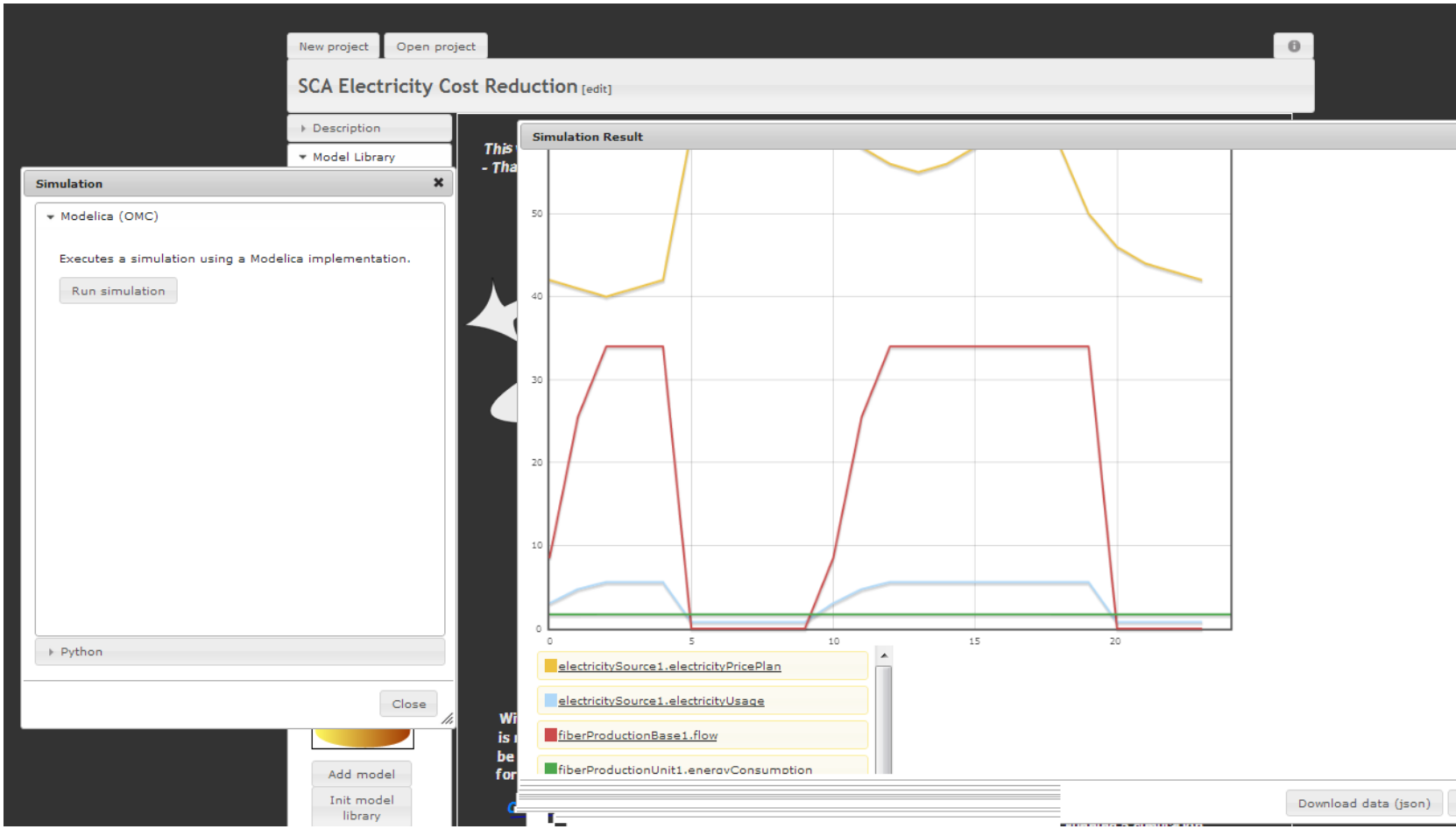
Composer prototype: fiber production line



Composer prototype: fiber production line



Example: fiber production line



Conclusions

SVG_{FMI} can be used for

- generic box-diagram representation of dynamical systems
- web based simulation editor
- managing separation between a models and its multiple implementations

but building SVG_{FMI} entities for many models and tools requires large effort.

Future work

Continuation in project *Simovate*

- coupled simulation of mechatronic multibody systems
- 3D models and CAD software
- exploit SVG_{FMI} on the viewer side

`http://imuit.cs.umu.se/composer`

`www.org.umu.se/umit/english/project-activities/simovate`