Visual Modelling Environment for CBD's

Final project for Model Driven Engineering, 2014-2015

Michaël Deckers

Universiteit Antwerpen

Introduction & contents

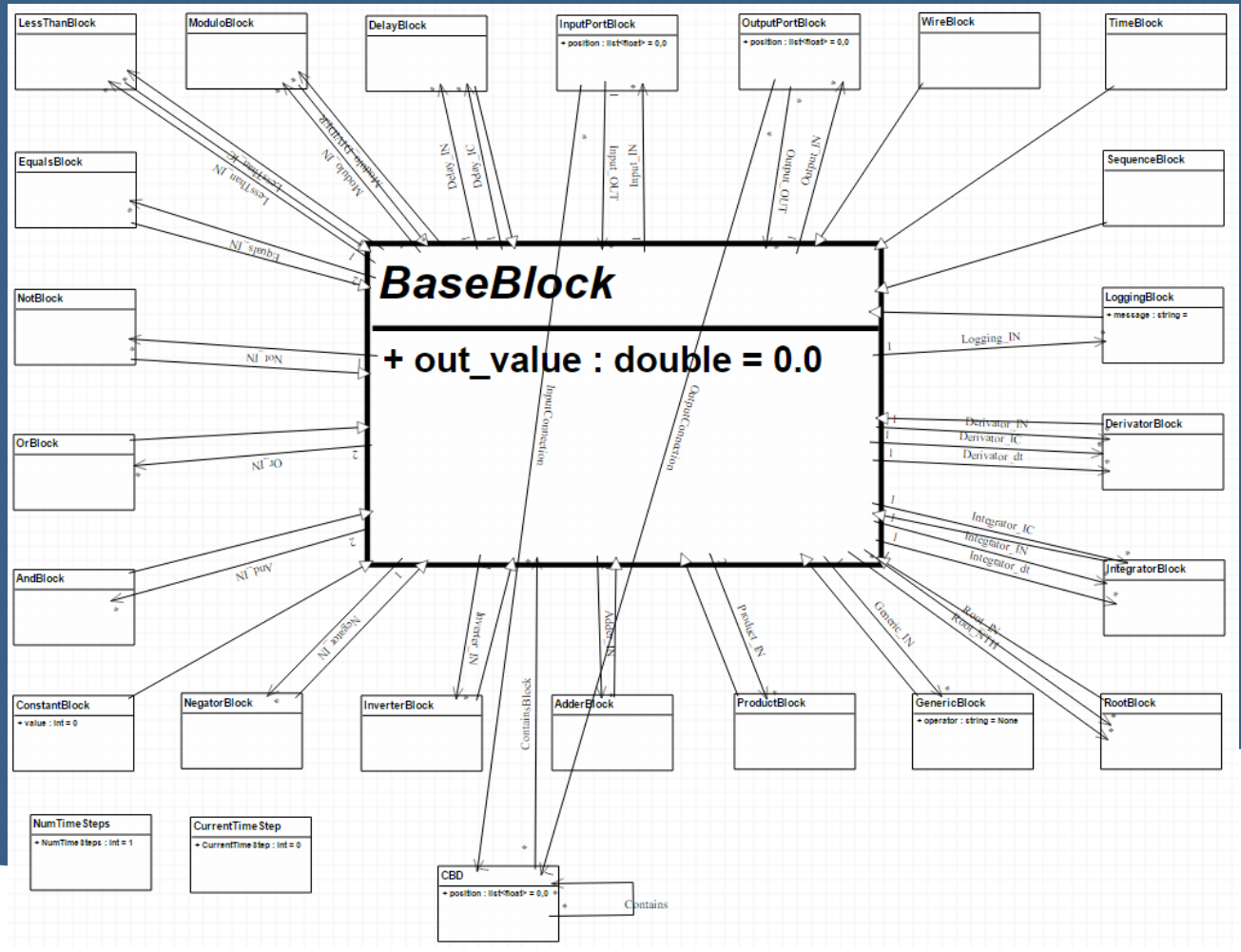
- Implementation (part 2 of project)
  - Designing CBD formalism for AToMPM
  - Export model to MetaDepth and compile to python
  - Generate simulation back-end
- Future work
- Conclusion
- Demonstration

Designing the CBD formalism

- Abstract syntax
  - Class for each block type
  - Blocks inherit from BaseBlock (class) to be easily interconnectible
  - CBD (class) can contains Blocks and other child CBD's
  - Extra classes for:
    - Total simulation steps
    - Current simulation step
  - Connections: choose type of input on connect

Designing the CBD formalism

- Abstract syntax

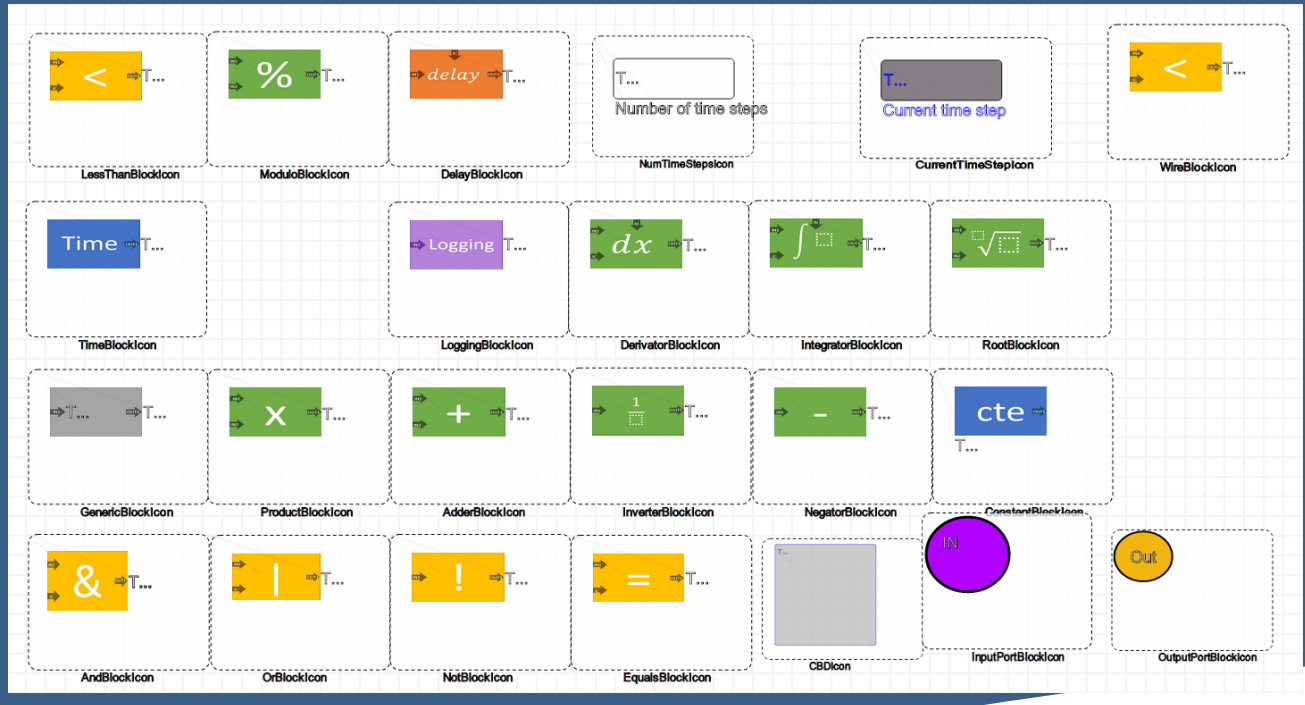


Designing the CBD formalism

- Concrete syntax
  - Each *block* has its own design
    - Shows input and output ports
    - Shows the operation it performs clearly
    - Color coded for type (e.g. green: mathematical, yellow: boolean)
    - Exceptions: purple circle: InputPortBlock, yellow circle: OutputPortBlock
  - Each type of *connection* has a certain color
    - Black: normal input
    - Blue: IC (initial component) or special input (divider or nth root)
    - Red: delta_t connection for derivator and integrator blocks

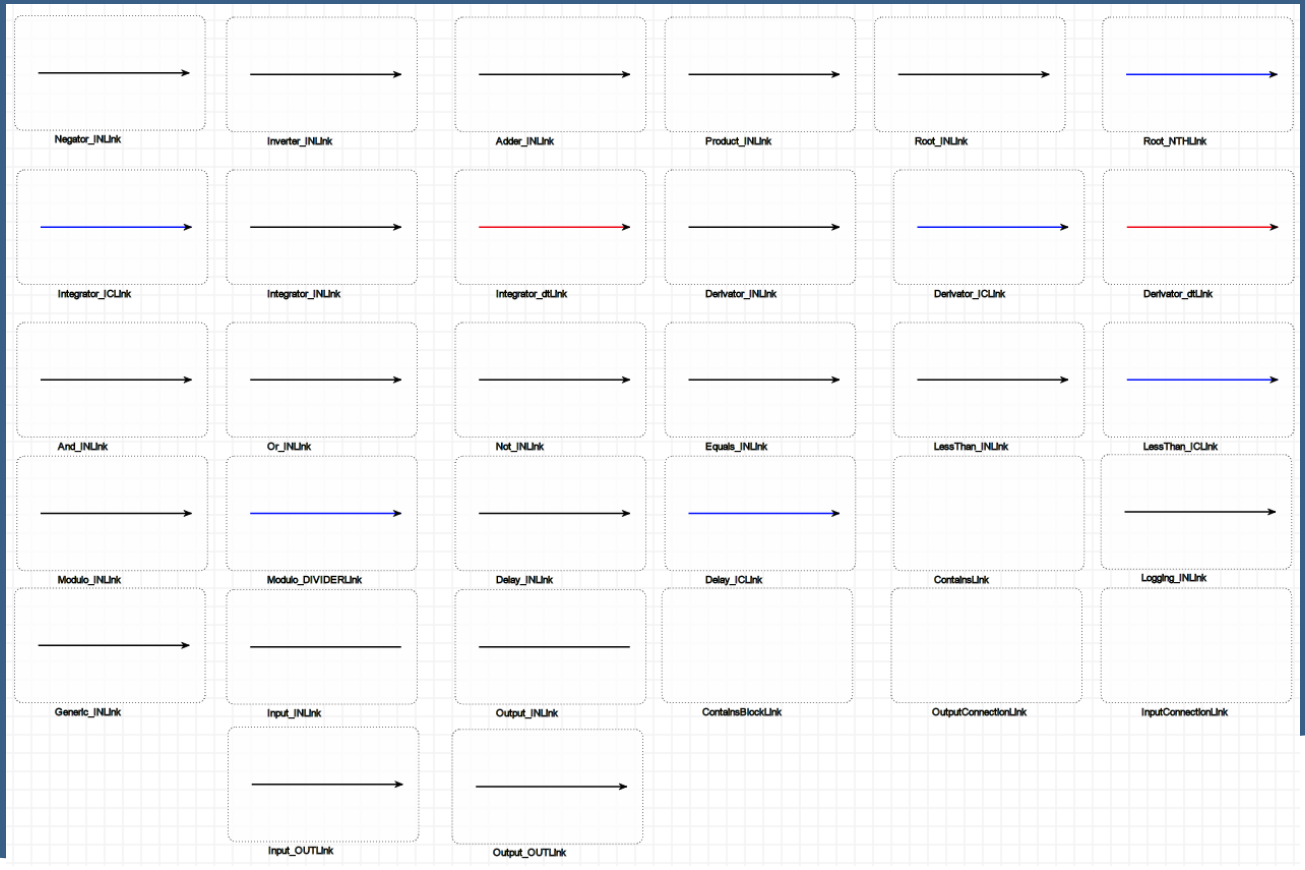
Designing the CBD formalism

- Concrete syntax



Designing the CBD formalism

- Concrete syntax



Designing the CBD formalism

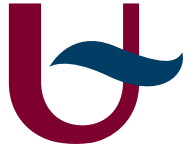
- Concrete syntax

Exporting to Python

- MetaDepth
  - Design or load model and metamodel
  - Manual compilation
    - Using the MetaDepth toolbar
    - On systems other than Windows
  - Automatic compilation
    - Using the CBD simulation toolbar (introduced later)
    - On Windows systems

Exporting to Python

- EGL
  - Export the MetaDepth models to be compatible with the Python generator (MoSIS)
  - Long process, the main parts are:
    - Adding child CBD's
    - Adding blocks and connections to child CBD's
    - Adding blocks and connections to main CBD
    - Retrieving results from the simulator and grouping them



Simulation

- Simulation toolbar

  - Export model to MetaDepth
  - Export metamodel to MetaDepth
  - Compile MetaDepth to Python
  - Run full (complete) simulation
  - Pause simulation
  - Perform one simulation step
  - Reset the simulation

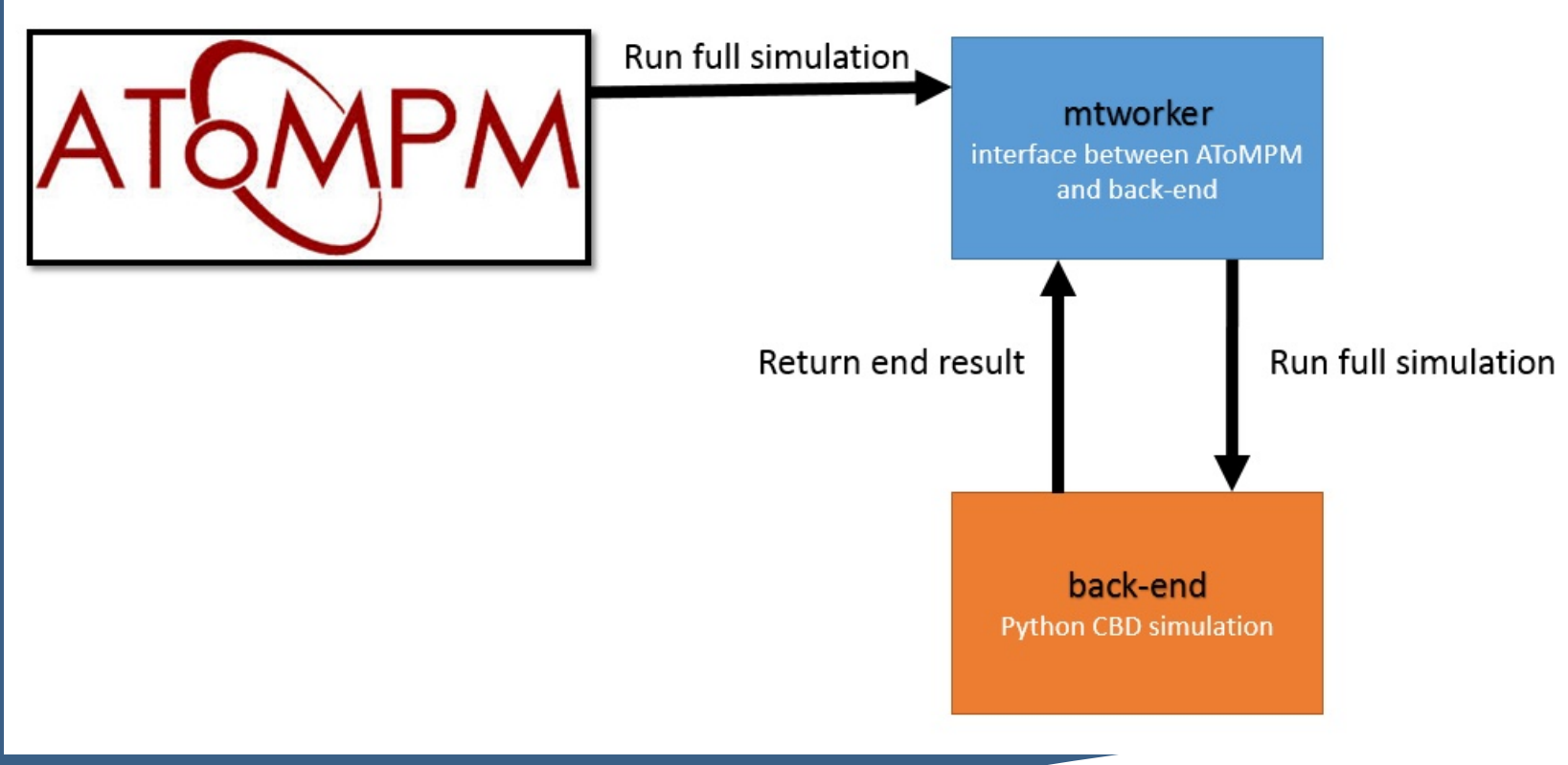
Simulation

- Simulation was developed in multiple iterations
  1. Running the simulation
  2. Updating the AToMPM model
  3. Using Statecharts for simulation
  4. De/reconstruction of the simulator to/from Statechart
  5. Eliminating full simulation
  6. Reset
  7. Pausing the simulation

Simulation - Running the simulation

- Connection layer converted from ParallelDevs model
- Do simulation call (to existing python CBD simulator) from this connection layer
- Main challenges:
  - Finding out which parts are necessary
  - Adapting this back-end to work with (much simpler) CBD models

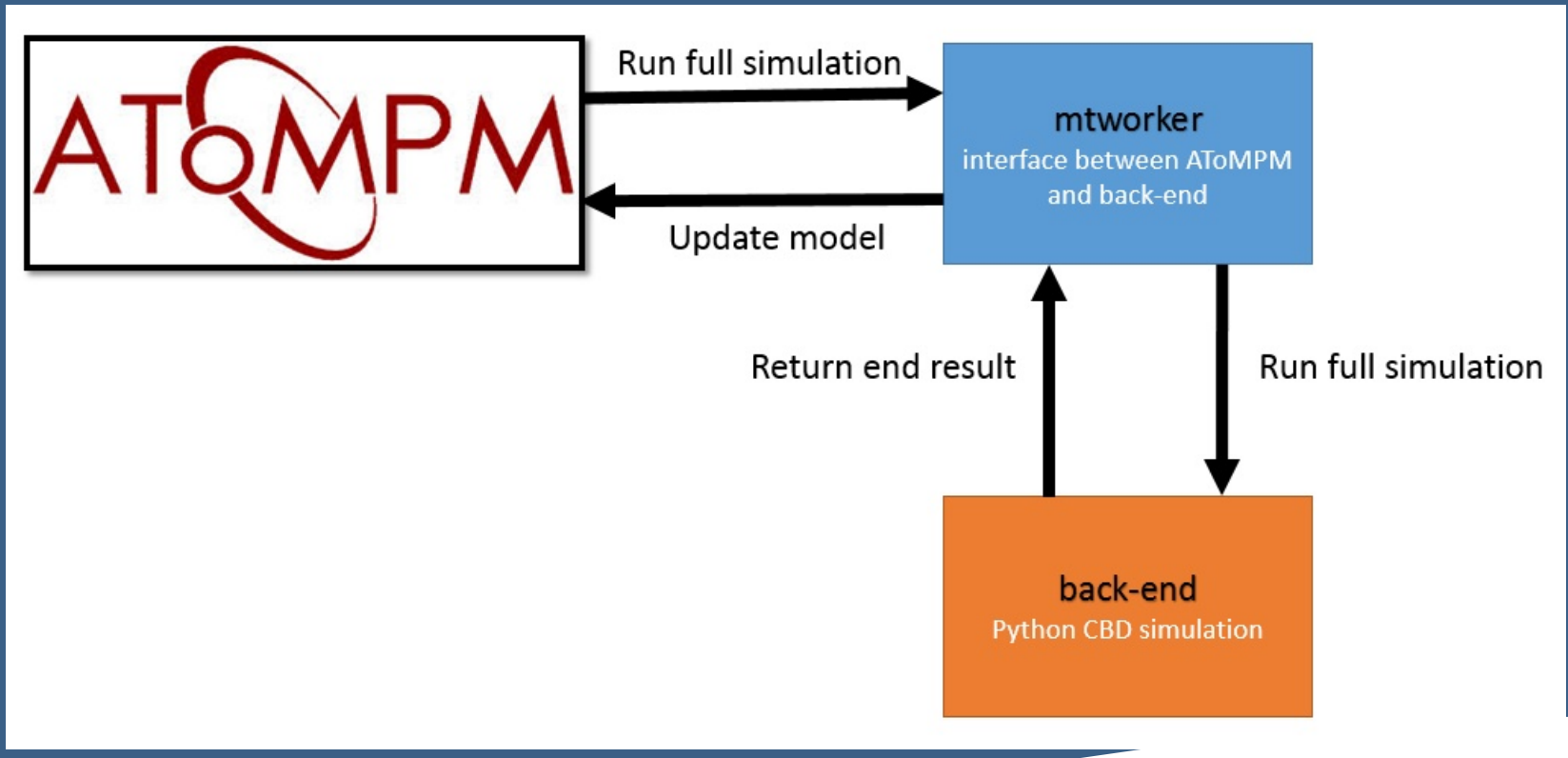
Simulation - Running the simulation



Simulation - Updating the AToMPM model

- Results from simulation have been received in connection layer in the form of a list of tuples
  - (blockname, blockvalue)
- For each tuple, update the value of the block in AToMPM with the correct value
- Main challenge:
  - Figuring out how and where to make the right calls

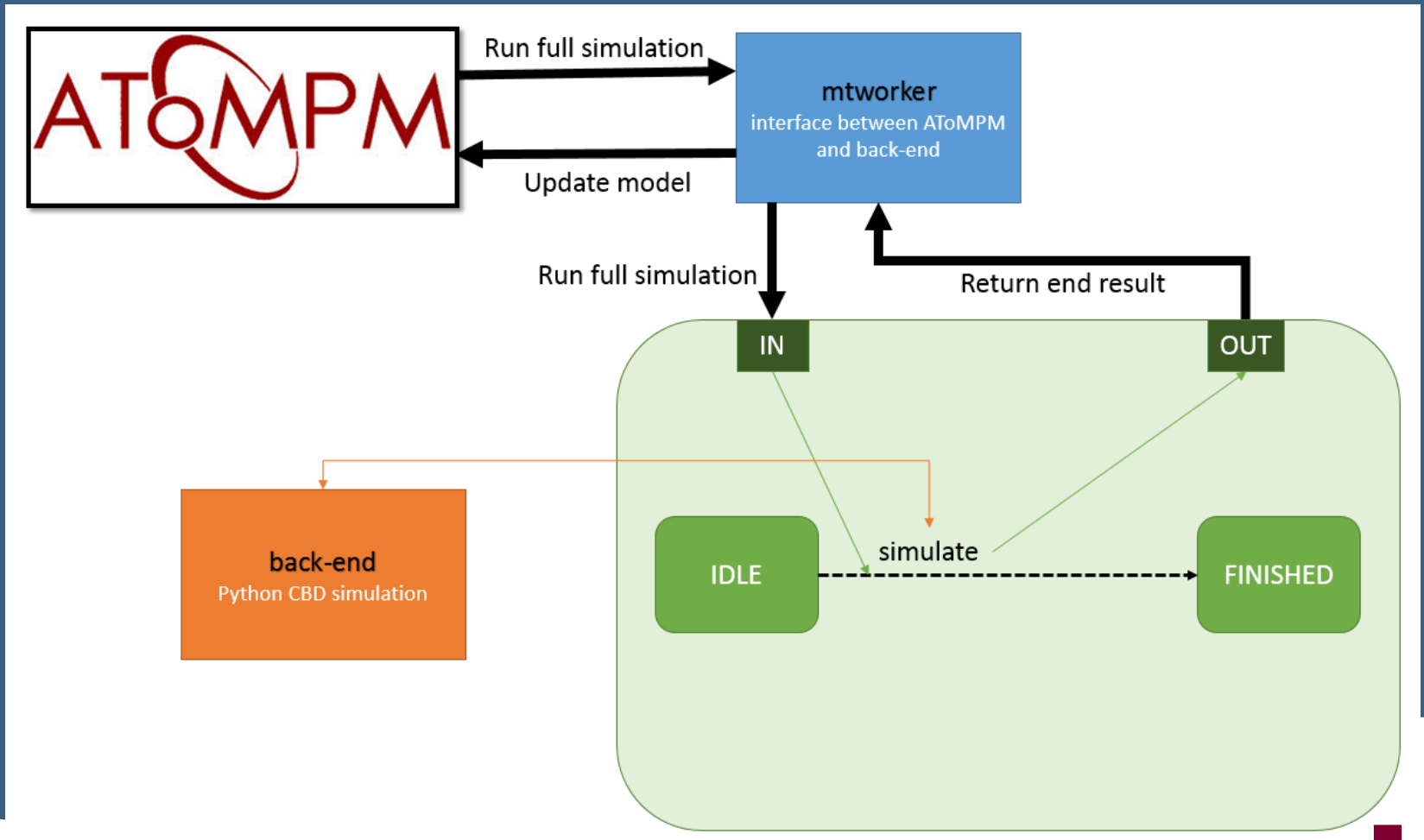
Simulation - Updating the AToMPM model



Simulation - Using Statecharts for simulation

- Previously: call the simulation from the connection layer
- Now: the simulation is called by a Statechart transition, which interacts with the python simulator
- Statechart currently has 2 states and 1 transition
  - Idle (simulator is doing nothing)
  - Finished (simulator is done)
- Main challenge:
  - Figuring out how to use the Statecharts as an extra layer

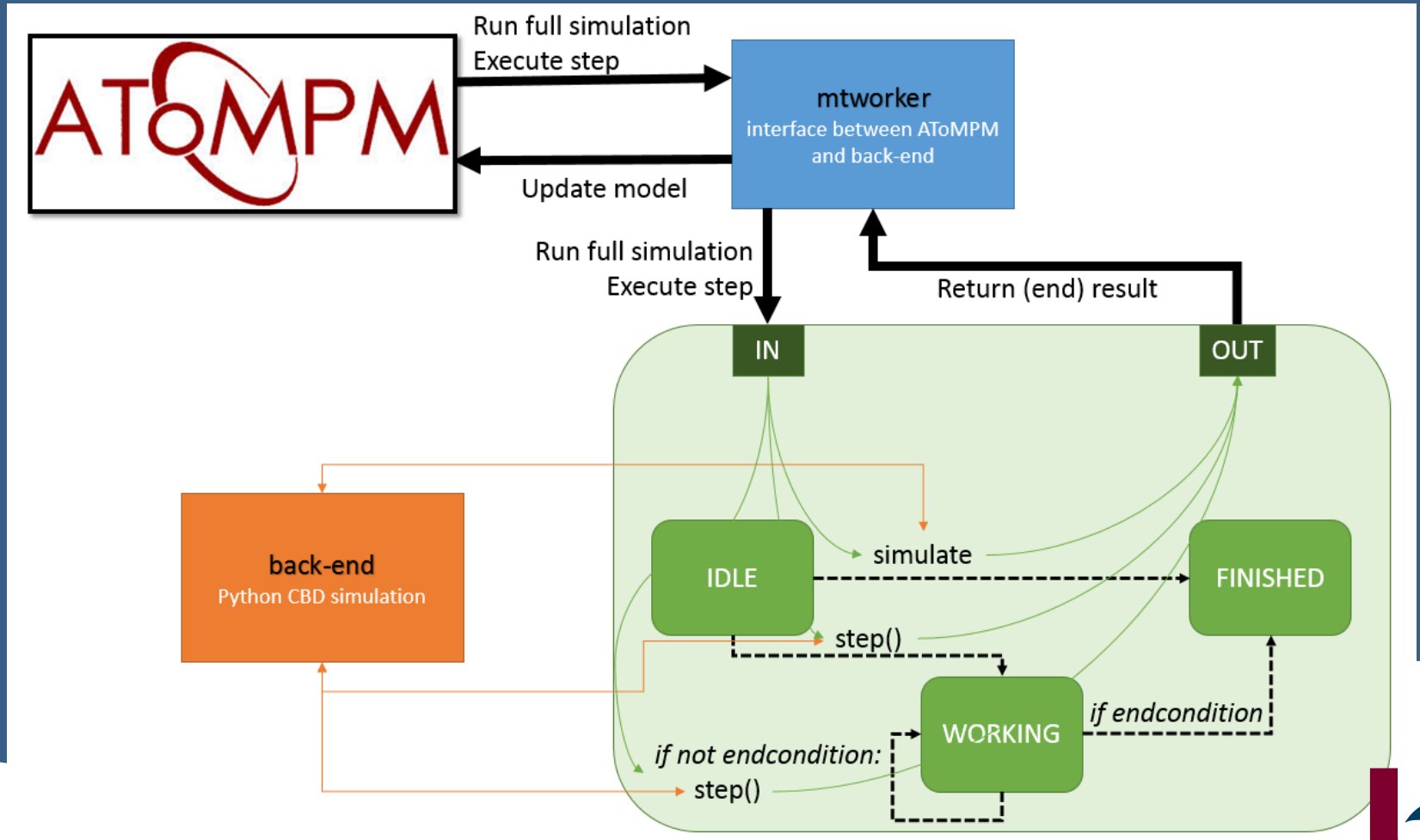
Simulation - Using Statecharts for simulation



Simulation - de/reconstruction of the simulator to/from Statechart

- Previously: the Statechart would make a call that runs the entire simulation and only returns the end result
- Now: it is possible to step through the simulation
- Statechart currently has 3 states
  - Idle (simulator is doing nothing)
  - Finished (simulator is done)
  - Working (individual steps are being simulated)
- Modify the (existing) Python CBD simulator and the EGL exporter
- Main challenge:
  - Modifying all required files

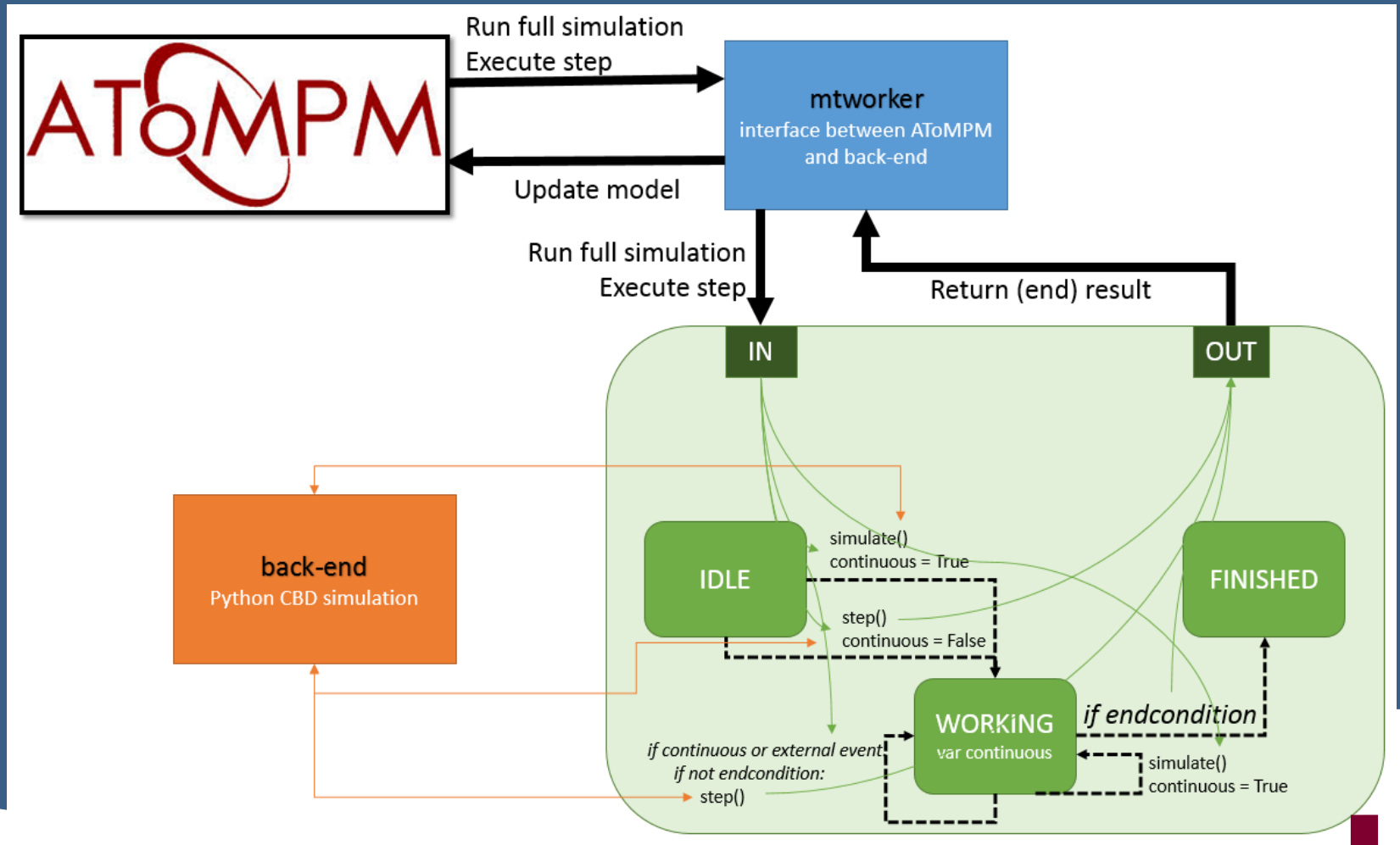
Simulation - de/reconstruction of the simulator to/from Statechart



Simulation - eliminating full simulation

- Previously: when running full simulation, the result of the entire simulation would be requested from the Python simulator
- Now: full simulation is modelled by repeating single steps
- Main challenge:
  - Figuring out how to distinguish between a single step or repeated, automatic steps

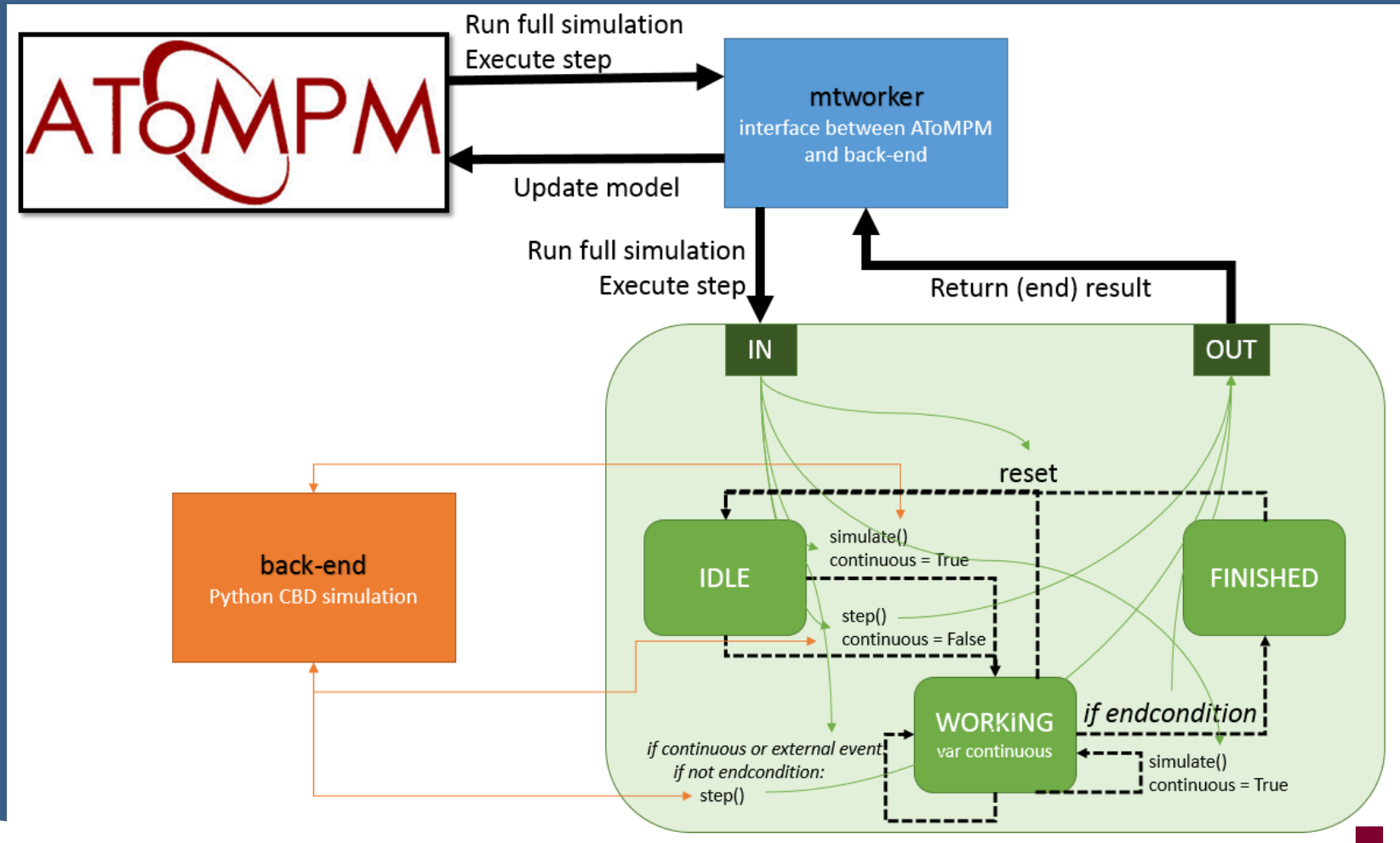
Simulation - eliminating full simulation



Simulation - reset

- Previously: when the simulation was done, a reload was required
- Now: the simulation can be reset and restarted
- Reset all the values of blocks in AToMPM to their initial values (0)

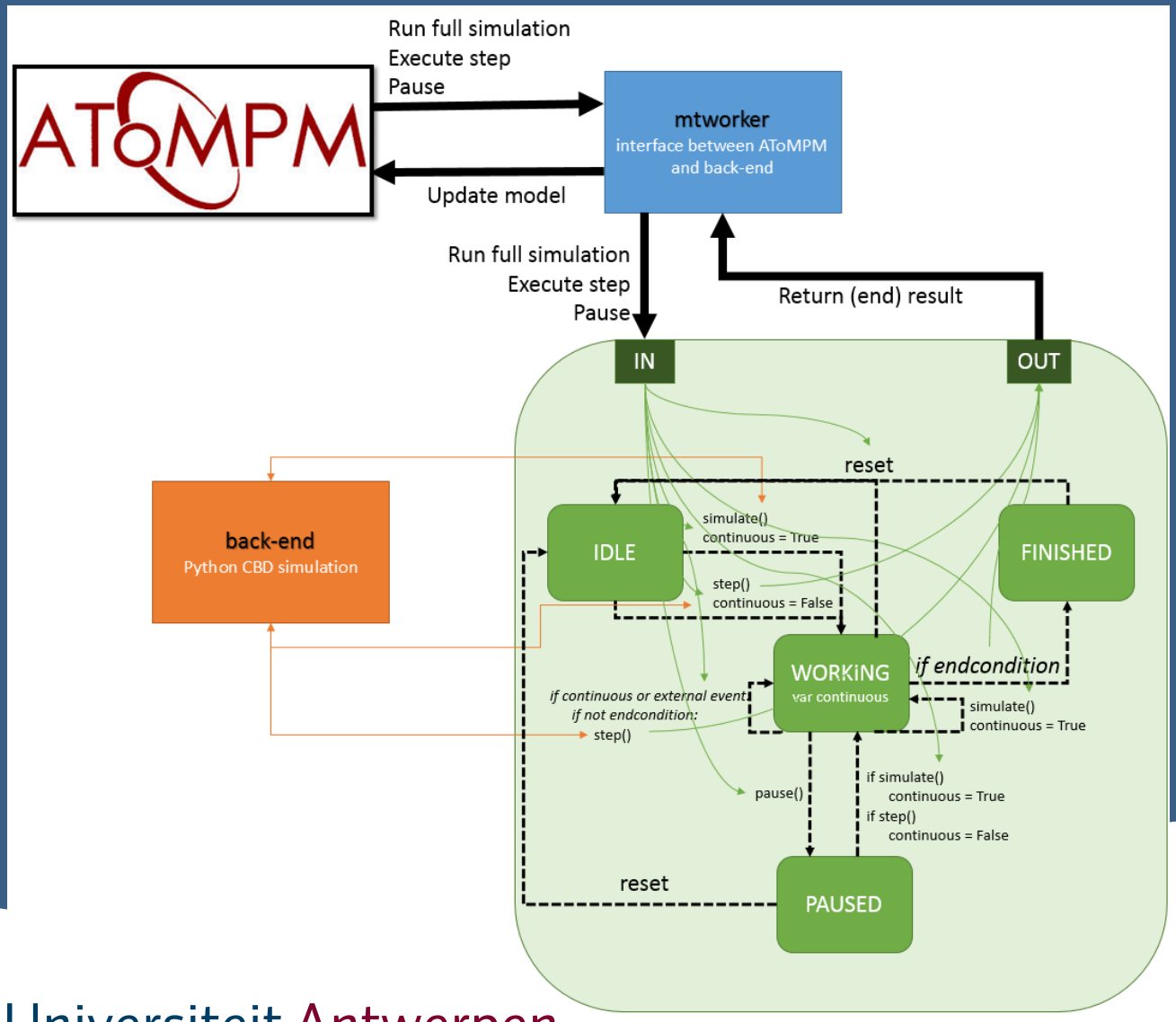
Simulation - reset



Simulation - pausing the simulation

- Previously: once the automatic simulation was started , it cannot be stopped
- Now: the simulation can be paused and resumed
- Main challenge:
  - The Statechart engine was not adapted to do what I needed
  - Trying to find some solution for this problem

Simulation - pausing the simulation



Future work

- AToMPM syntax
  - Constraints
    - Enforce the user to generate correct models
  - Visual improvements
    - Input/OutputPortBLocks should snap to their CBD
    - Improve visual appearance
- Simulation
  - Following simulation/debugging options can be added
    - Small steps (one block at a time)
    - Backwards stepping (Big and small steps)
    - Breakpoints, these were introduced in the reading assignment but not implemented

Conclusion

- MoSIS course missed a visual environment for an important part of the course: CBD's
- A lot of subjects/assignments from the MDE course were used in this project
- Creating the simulator was very frustrating
  - Starting from an existing project and modifying it
  - Choosing between the perfect solution and time limitations
- Decent functionality and usability for the time I was able to invest