# Octopus:
# The Multisystem Framework

Micael Oliveira, Martin Lüders and the Octopus developers

Octopus Advanced Course 2023, MPSD Hamburg

# Motivation

At its core, Octopus [1] solves a set of differential equations describing the dynamics of a system of electrons and nuclei:

$$i\frac{\partial}{\partial t}\varphi_i(\boldsymbol{r},t) = \left\{ v_{\text{ext}}(\boldsymbol{r};\boldsymbol{R}) \ + v_{\text{Hxc}}[n](\boldsymbol{r},t) \right\} \varphi_i(\boldsymbol{r},t)$$

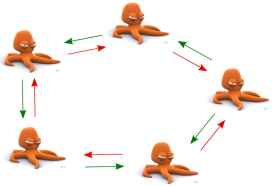
$$m_I\frac{\partial^2}{\partial t^2}\boldsymbol{R}_I(t) = \sum_{J\neq I}\boldsymbol{F}_{IJ}(\boldsymbol{R}_I,\boldsymbol{R}_J) + \ \boldsymbol{F}_{Ie}(\boldsymbol{R}_I;n)$$

- Electronic orbitals $\varphi_i(\boldsymbol{r},t)$ are discretized on a grid
- Nuclei are treated classically as point charges
- The equations are coupled by the nuclear coordinates $\boldsymbol{R}$ and by the electronic density $n(\boldsymbol{r},t) = \sum_i |\varphi_i(\boldsymbol{r},t)|^2$

# Motivation

Developers wanted to couple new types of systems with electrons and ions:

- Classical electromagnetic fields (Maxwell equations)
- Quantized EM fields (quantum electrodynamics)
- Solvent models
- ...



This can be challenging:

- Very different numerical methods are used to solve each subset of equations
- Time-scales can be very different (e.g., electrons move faster than nuclei)

# Motivation

The new framework should be able to:

- Handle arbitrary number of systems and system types
- Implement different types of interactions between systems
- Implement several different algorithms for each system type
- Decide what systems/interactions/algorithms to run from the input file
- Allow users to mix different types of algorithms (e.g., time-propagation and minimization)
- Give the user complete control over any possible approximations
- Allow for parallelization over systems

Very ambitious and not trivial to implement!

# How to design this

1. Clearly state the problem we are trying to solve
2. Write down all requirements
3. Choose a suitable programming paradigm (object-oriented, functional, etc)
4. Develop and test the code using a simple, well understood example application that covers most of the intended use-cases

# What problem are we trying to solve

- ~~Framework to simulate interacting physical systems~~
- ~~Framework to solve systems of coupled differential equations~~
- Framework to handle one or more iterative algorithms that need to exchange information at specific iterations

The framework needs to:

- Implement a mechanism for information exchange
- Implement conditions for information exchange
- Keep track of the internal state of systems, couplings, and algorithms

# What problem are we trying to solve

Some terminology:

- **System**: physical system characterized by some internal quantities (e.g, positions, densities, temperatures, etc) that are updated by some iterative algorithm
- **Coupling**: an internal quantity of a system that is required to execute the
- **Interaction**: a term required to execute the algorithm of a system that, in general, requires internal quantities from the system and some couplings to be evaluated (e.g., gravitational force)
- **Interaction partner**: some entity that contains couplings needed by other systems. All systems can be interaction partners, but not all interaction partners are systems (e.g., data models)

# Design requirements

**Main requirement**: framework plus all existing systems, interactions and algorithms should be easy to maintain and extend.

- Framework should be completely independent of existing systems, interactions and algorithms
- Adding new systems should not require modifying existing systems, interactions or algorithms
- Adding new interactions should only require to modify systems that want to use those interactions
- Adding new algorithms should only require to modify systems that want to use those algorithms
- Modifications to the framework should only be required when adding new **features**, not when adding new systems/interactions/algorithms

# Programming paradigm and design choices

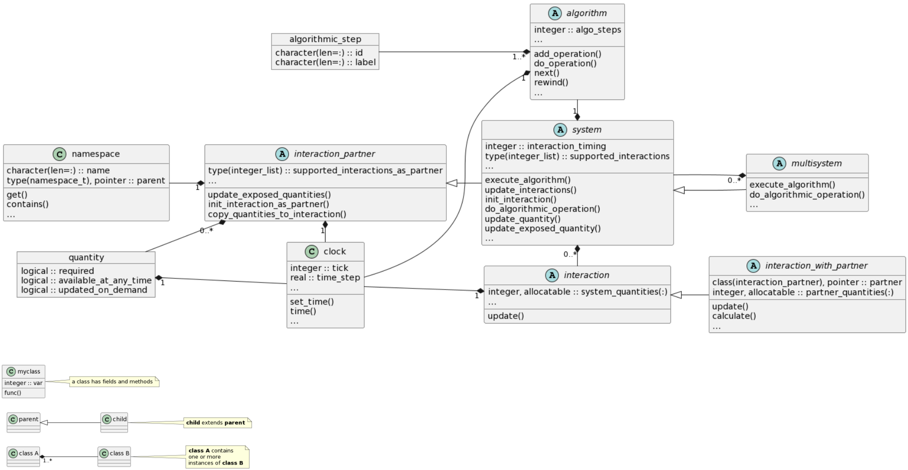
The way NOT to do it:

```
if (system_A%is_electrons) then
  ...
  select case (system_A%propagator)
  case (AETRS)
    ...
    if (system_A%has_interaction_X_with_system_B) then
      ...
    end if
  end select
  ...
else if (system_A%is_ions) then
  if ((system_A%has_interaction_Y_with_system_B) then
    ...
  end if
  ...
end if
```

# Programming paradigm and design choices

- Object Oriented Programming
- Framework defines several abstract classes for systems, interactions and algorithms
  - Actual systems, interactions and algorithms provide implementations for the required deferred methods
  - Clean separation between the framework and the math/physics
- Systems do not know about each other directly, instead they know interactions
- An interaction connects a system with an interaction partner
- Interactions are uni-directional
- Algorithms are implemented as a set of state machine atomic operations (algorithmic operations)
- Systems do not inherit from the algorithms, instead, they implement algorithmic operations (Fortran does not allow multiple inheritance!)

# UML Class diagram of the framework



# The general algorithm

**repeat**
  **for all** systems **do**
    $algo\_op \leftarrow$ next algorithmic operation
    $break \leftarrow false$
    **while** not $break$ **do**
      **if** $algo\_op \neq$ update interactions **then**
        execute algorithmic operation
        $algo\_op \leftarrow$ next algorithmic operation
      **else**
        try updating interactions
        **if** interactions updated **then**
          $algo\_op \leftarrow$ next algorithmic operation
        **end if**
        $break \leftarrow true$
      **end if**
    **end while**
  **end for**
**until** all algorithms finished

# Conditions for interaction update

When a system request an interaction to be updated, the following conditions must be met for a successful update:

- The necessary system quantities must be at the exact requested time
- The partner's algorithm clock must be at the requested time or is going to reach the requested time in the next time-step
- The necessary partner quantities must be either:
  - at the exact requested time (if user requested the interaction timing to be exact)
  - at the closest possible time in the past (if the user allowed for retarded interactions)
  - at the closest possible time in the future (if the user allowed for time interpolation)

# Updating clocks

- The algorithm's clock is tentatively advanced when interactions are being updated and rewound if failed
- The algorithm's clock is advanced when interactions are successfully updated
- The system's clock is advanced when a time-step/iteration is finished
- A quantity's clock is updated whenever the quantity is updated

# Design in practice (continued)

- Three general types of algorithmic operations:
  - System and algorithmic generic: implemented in the framework
  - Algorithm specific and system generic: implemented in the algorithm classes
  - System specific: implemented in the system classes
- The framework keeps track of time (iterations) with clocks (counters)
  - Systems, algorithms and quantities all have clocks attached
  - The algorithm's clock is advanced when interactions are being updated
  - The system's clock is advanced when a time-step/iteration is finished
  - A quantity's clock is updated whenever the quantity is updated