Octopus: Implementation of the multisystem framework

Martin Lüders and the Octopus developers

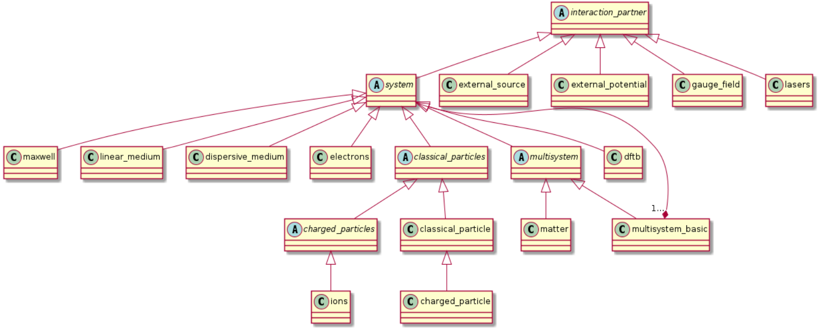
Octopus Advanced Course 2023, MPSD Hamburg

System classes

- Examples of systems:
  - maxwell
  - classical particles
  - charged particles
  - ions
  - electrons
  - tight binding model
  - etc.
- re-use as much code as possible between different systems
- use object oriented approach!
- represent systems as classes and use inheritance

System classes

Currently implemented system classes:



System classes

The abstract class `interaction_partner_t`:

- abstract class: cannot be instantiated
- defines basic variables and interface for all classes which can be partner in an interaction
  - namespace
  - clock
  - list of 'supported interactions as partner'
  - defines list of exposed quantities
  - interface for routine to update exposed quanities

System classes

The externally driven partners: e.g. `lasers_t`

- no proper propagation
- not affected by other partners
- use 'static propagator'

System classes

The abstract class `system_t`:

- abstract class: cannot be instantiated
- inherits all from `interaction_partner_t`
- defines basic variables and methods for all systems
- implements methods which are common to all systems
- defines deferred methods which are common to all systems, but depend on specifics

System classes

Child classes add more features to the parent class.

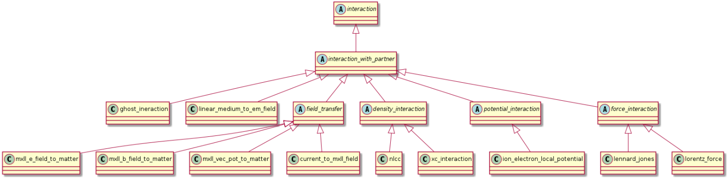
- deferred functions need to be implemented
- functions of parent can be overridden

Performing algorithmic steps (until barrier): `execute_algorithm()`

- perform general tasks
- call `do_algorithmic_operation()` of child class.

Interaction classes

Currently implemented interaction classes:



Interaction classes

The abstract class `interaction_t`:

Basic attributes:

- *label*: name for debug output
- *clock*: keep track of the time when last updated
- *system quantities*: which quantities are needed from the system?
- *intra interaction*: Is the an interaction of the system with itself?
- *energy*: energy associated with that interaction

Interaction classes

The abstract class `interaction_t`:

Deferred interfaces:

- `update()`:
  attempt to update the interaction, if not not the right time.
- `calculate()`:
  calculate the fields, or potentials used by the system owning the interaction
- `calculate_energy()`:
  calculate the energy associated with the interaction

Interaction classes

The abstract class `interaction_with_partner_t`:

Added attributes:

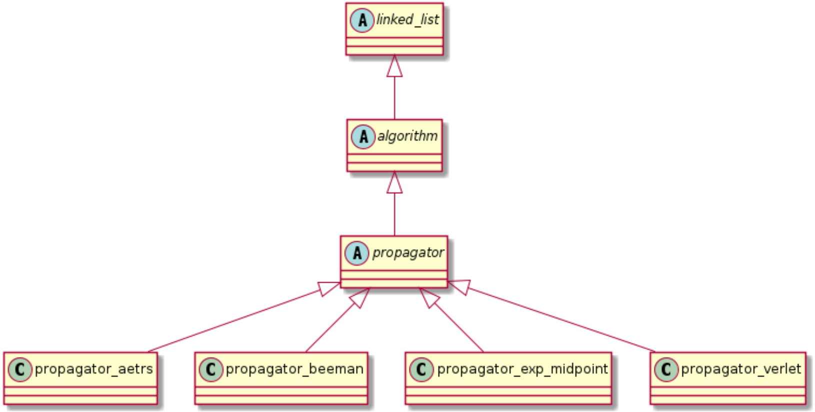
- pointer to partner
- list of the partner's exposed quantities

Implement:

- `update()`

System classes

Some of the implemented system classes:



Algorithms

The abstract class `algorithm_t`:

- abstract class: cannot be instantiated
- extends a linked list
- adds algorithm specifics
    - iterator
    - clock
    - time step
    - number of algorithmic steps

Propagators

The abstract class `propagator_t`

- extends `algorithm_t`
- adds pointer to system
- adds implementation of system-independent algorothmic operations, e.g. start/end scf loop

Specific propagators extend `propagator_t` and add the algorithm in the constructor.

Time-dependent multisystem run

```
! Initialize all propagators
call systems%init_algorithm(propagator_factory_t(systems%namespace))

call systems%init_clocks()
call systems%initial_conditions()

call systems%propagation_start()

! The full TD loop
do while (.not. systems%algorithm_finished())

  ! Execute algorithm until next barrier
  call systems%execute_algorithm()

  ...
end do

call systems%propagation_finish()
```

Executing the algorithm

`system_execute_algorithm()` performs loop until barrier:

- get current operation
- try to execute system-specific operation (`system_do_algorithmic_operation()`)
- update quantities
- if required try to update interactions
- if required perform algorithm specific or generic operation

Implementing algorithmic operations

`system_do_algorithmic_operation()`:

- implements all algoritmic operations for a system
- this combines all Algorithms
- implementation in big `select case` construct