Grav-Sim

Gravity Simulation on a Desktop Computer

Architecture of a Gravitational N-Body Simulator

Mark Ridler

Presentation To AMUSE Workshop

October 2009

Overview

1.Review of the “ACS” code
2.Porting the ACS code to C++
3.Architecture Diagrams
4.Plug-and-Play Integrators
5.Plug-and-Play Tree Codes
6.2-body Regularisation
7.Animations

1. Review of the “ACS” code

i.“The Art of Computational Science”

ii.Developed by Piet Hut and Jun Makino

iii.Written in the scripting language Ruby

iv.Intended as an introduction to N-Body systems

v.Includes individual time-steps

vi.Includes plug-and-play integrators, e.g:

- Forward Euler (1st order – not practical)
- Leap-Frog (2nd order – commonly used)
- Runge-Kutta (2nd to 4th order – used in the ‘70s)
- Hermite (4th to 8th order – amazing)

2. Porting the ACS code to C++

i.Mark Ridler was a C++ programmer for 7 years
ii.N-Body simulation needs high performance
iii.C++ is a standard compiled language
iv.Opens up the way for:

- Choice of Precision (float, double, dd_real etc)
- Multi-Core CPUs (OpenMP etc)
- General-Purpose GPUs (CUDA etc)
- Special-Purpose Hardware (GRAPE etc)

2 a. Porting the ACS code to C++ (continued ...)

i. Most of the ACS code world6.rb was ported as-is

ii. The “Multi-Step” integrator didn’t quite make it

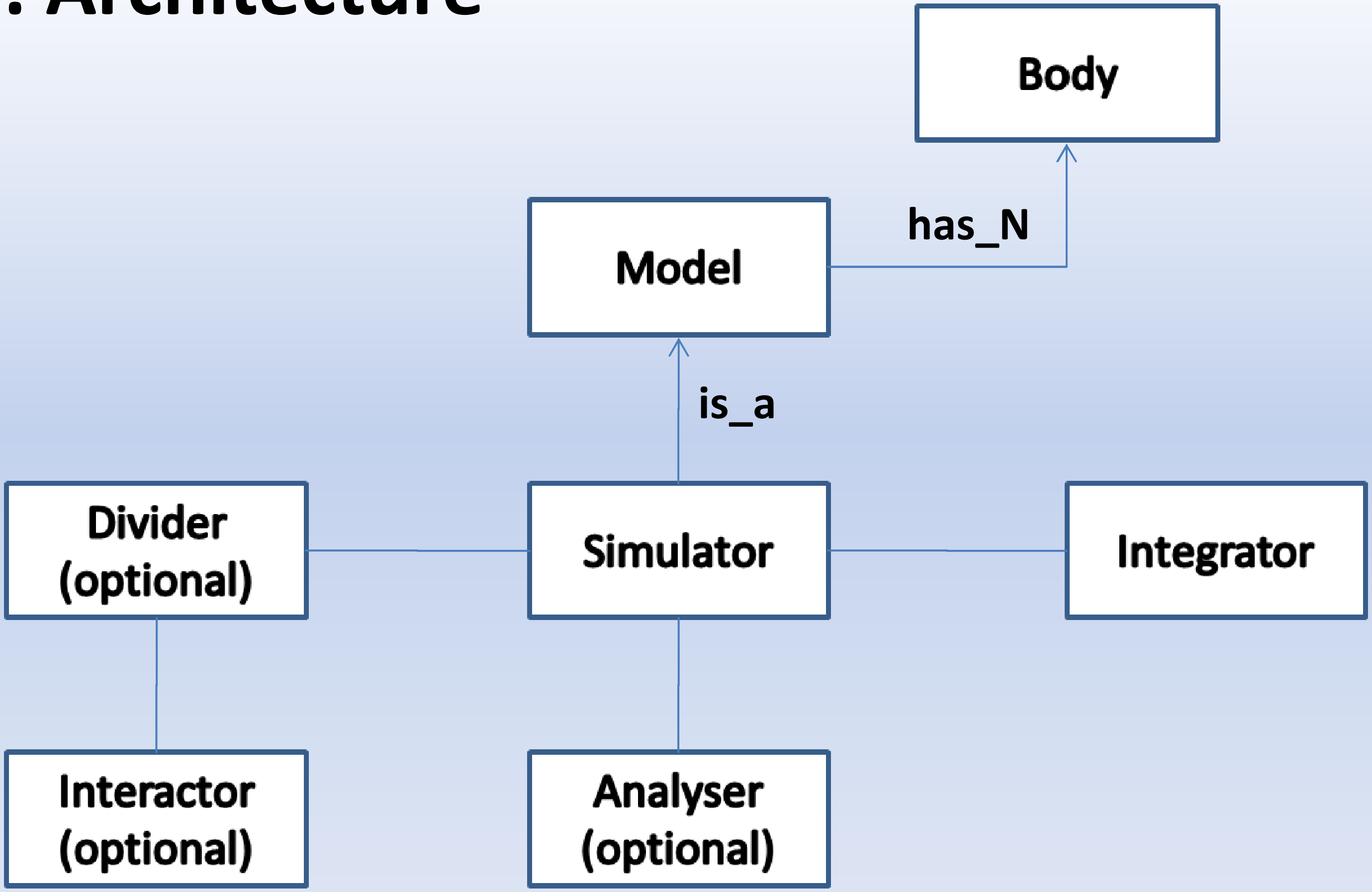
iii. Some parts of the code have been re-factored, e.g.:

- Calculates many-at-a-time rather than 1-at-a-time
- Transaction model commits changes together
- WorldLine limited to last 2 points only
- Mass moved from WorldPoint to Particle
- Body now has:
    a. Particle ... which represents the present
    b. WorldLine ... which represents the past

Grav-Sim

Gravity Simulation on a Desktop Computer

3. Architecture



Body

has_N

Model

is_a

Divider (optional)

Simulator

Integrator

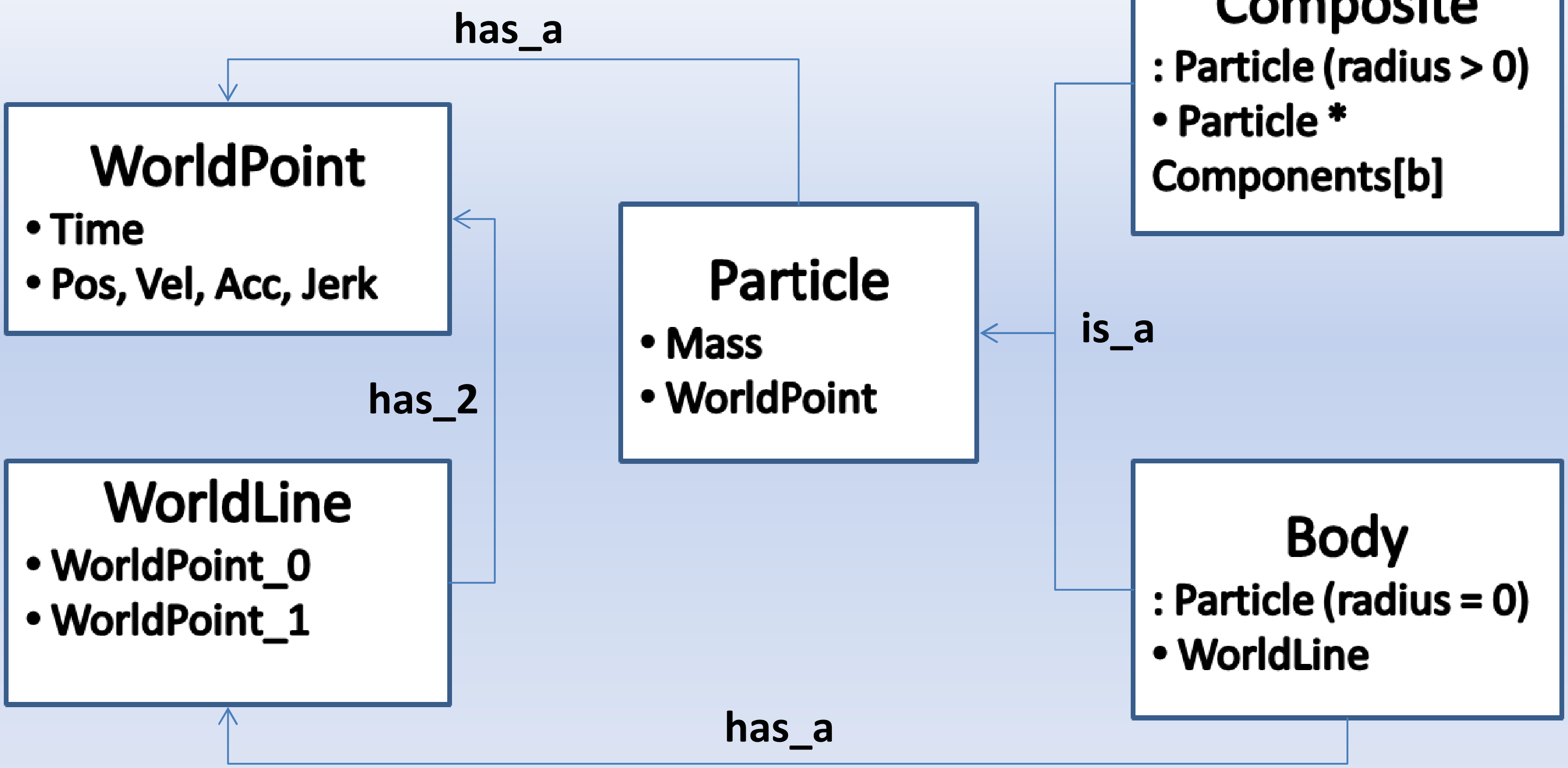
Interactor (optional)

Analyser (optional)

Grav-Sim

Gravity Simulation on a Desktop Computer

3 a. Particle



has_a

WorldPoint

- Time
- Pos, Vel, Acc, Jerk

has_2

WorldLine

- WorldPoint_0
- WorldPoint_1

Particle

- Mass
- WorldPoint

is_a

Composite

: Particle (radius > 0)

- Particle * Components[b]

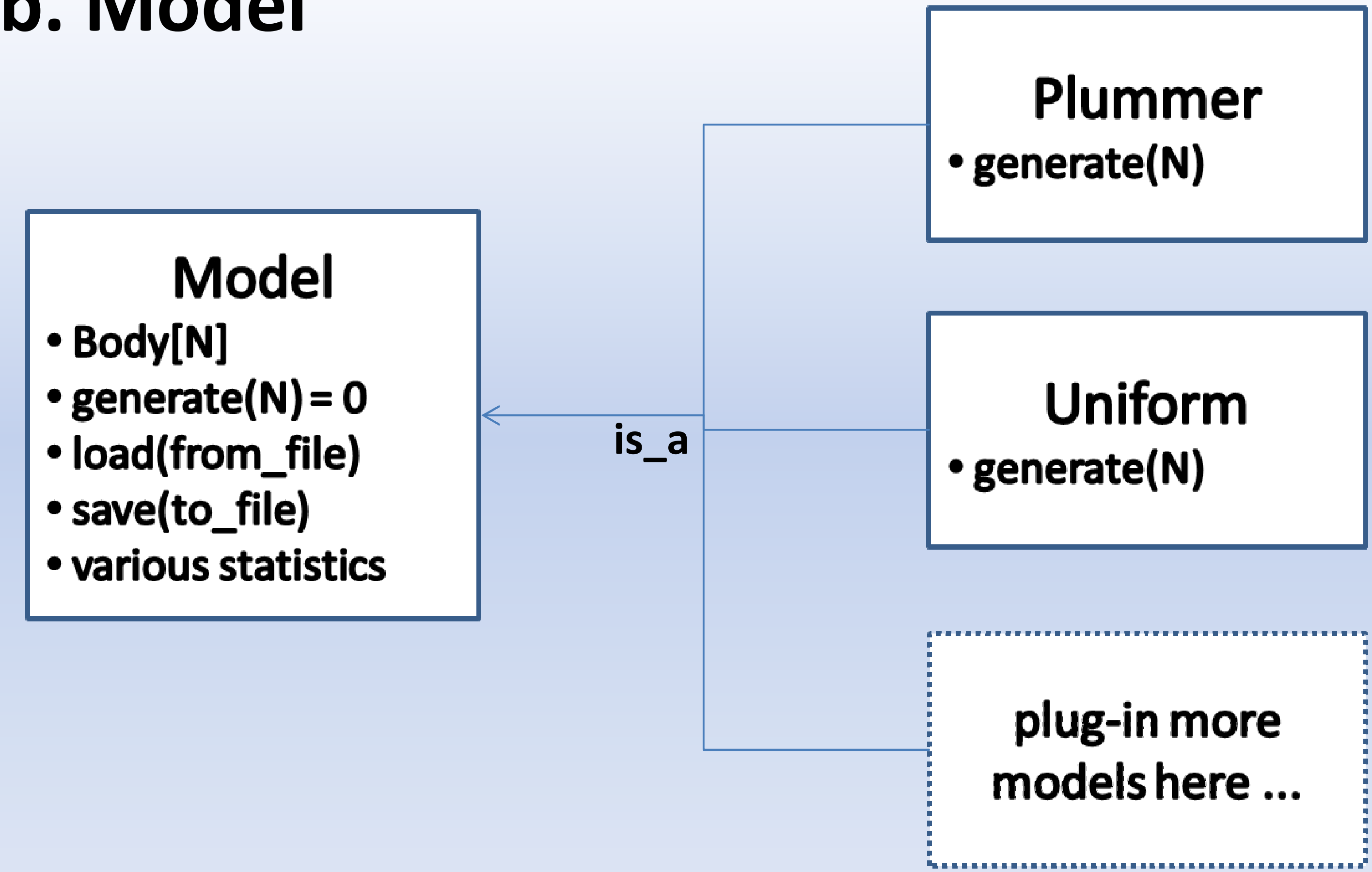
Body

: Particle (radius = 0)

- WorldLine

has_a

3 b. Model



Model

- Body[N]
- generate(N) = 0
- load(from_file)
- save(to_file)
- various statistics

is_a

Plummer

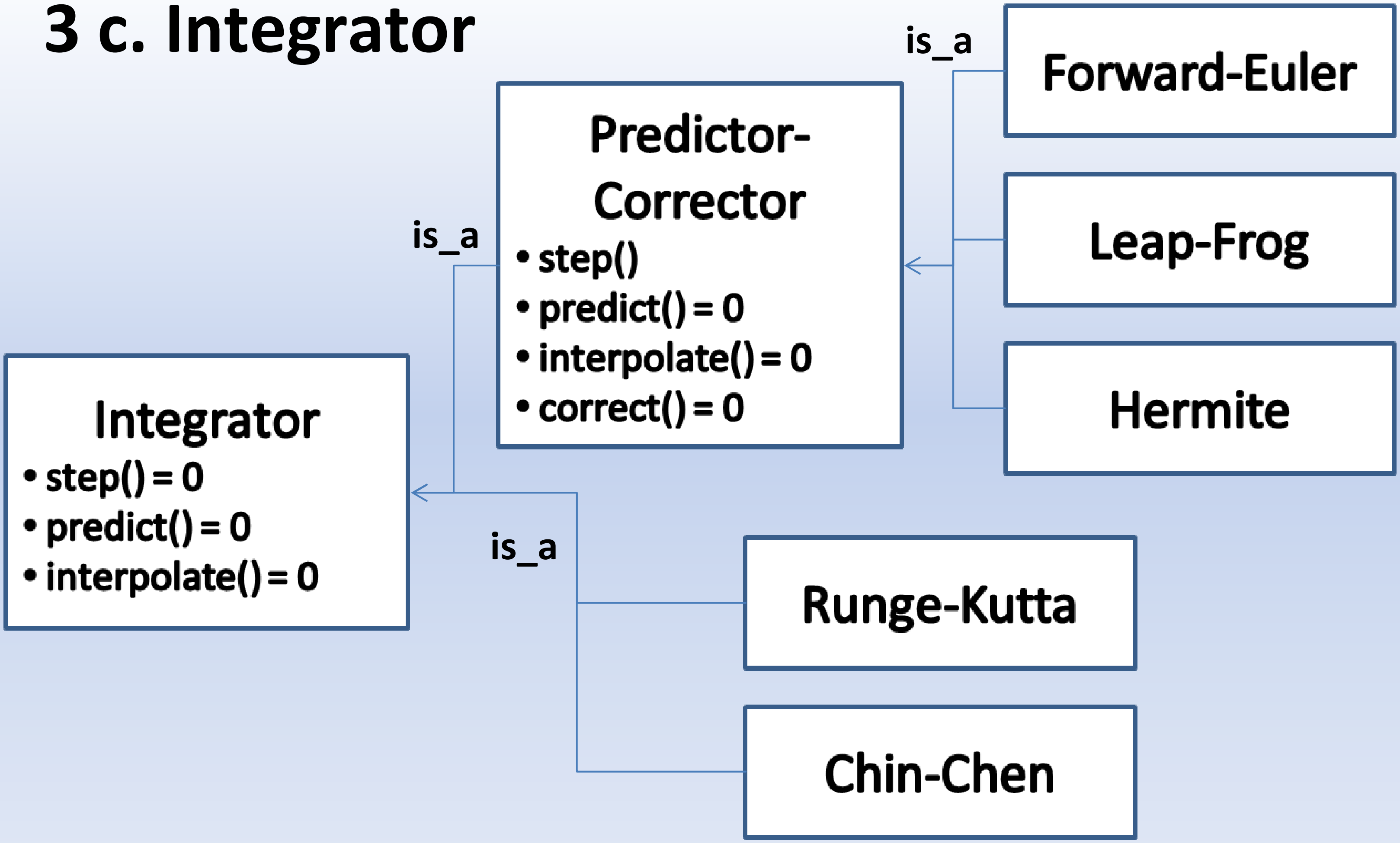
- generate(N)

Uniform

- generate(N)

plug-in more models here …

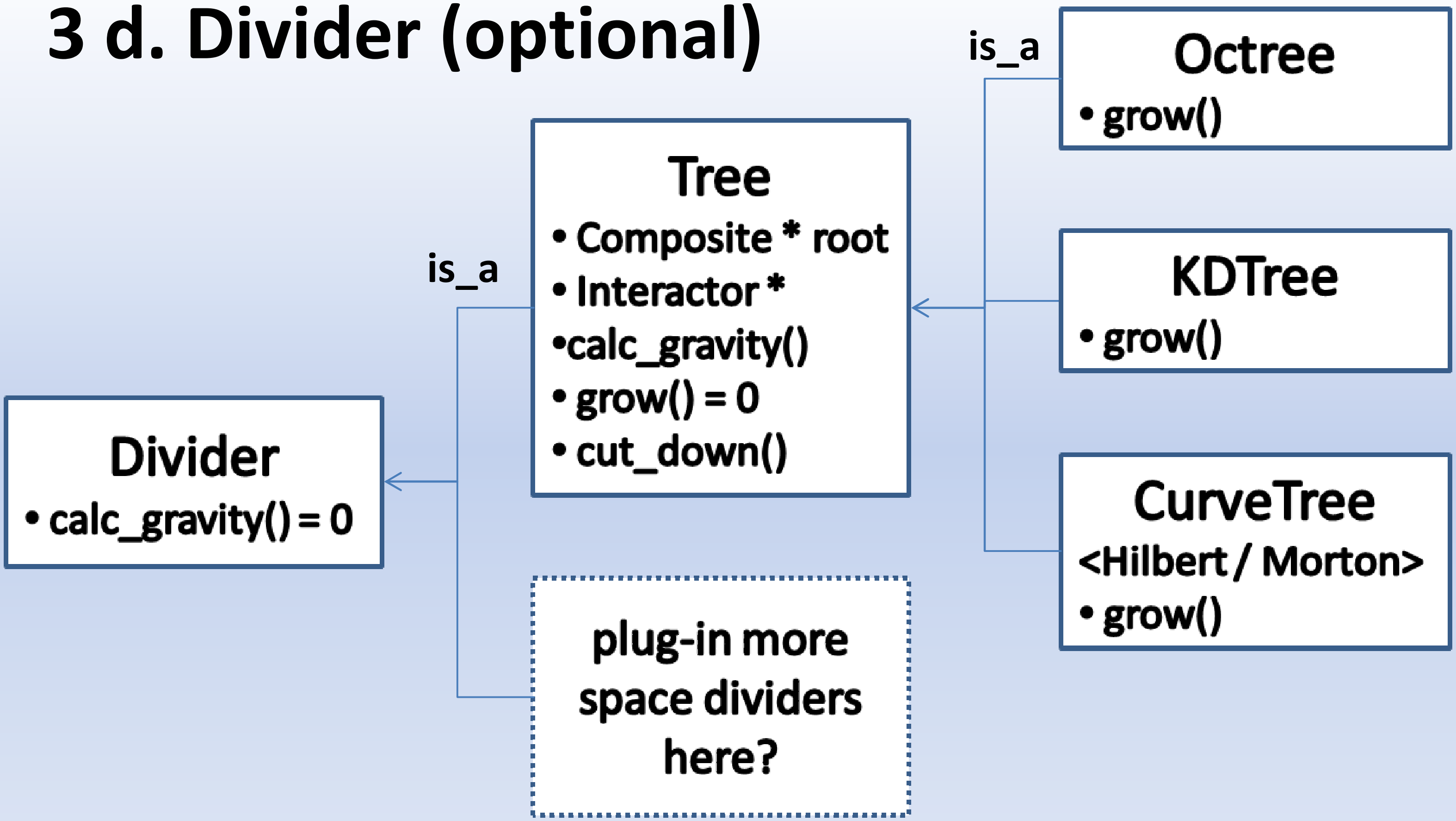
3 c. Integrator



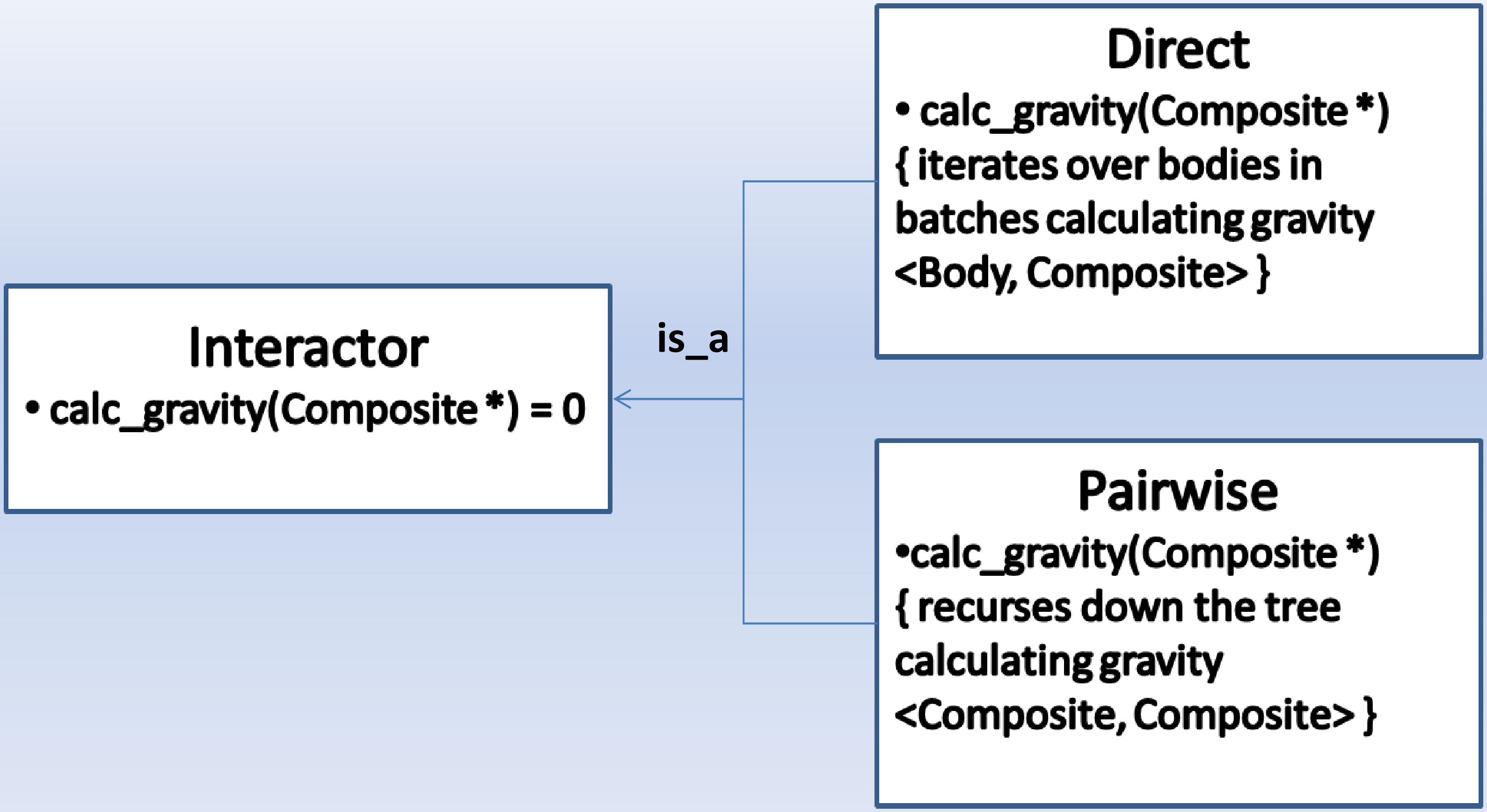
Integrator
- step() = 0
- predict() = 0
- interpolate() = 0

Predictor-Corrector
- step()
- predict() = 0
- interpolate() = 0
- correct() = 0

3 d. Divider (optional)



3 e. Interactor (optional)



Interactor

- calc_gravity(Composite *) = 0

is_a

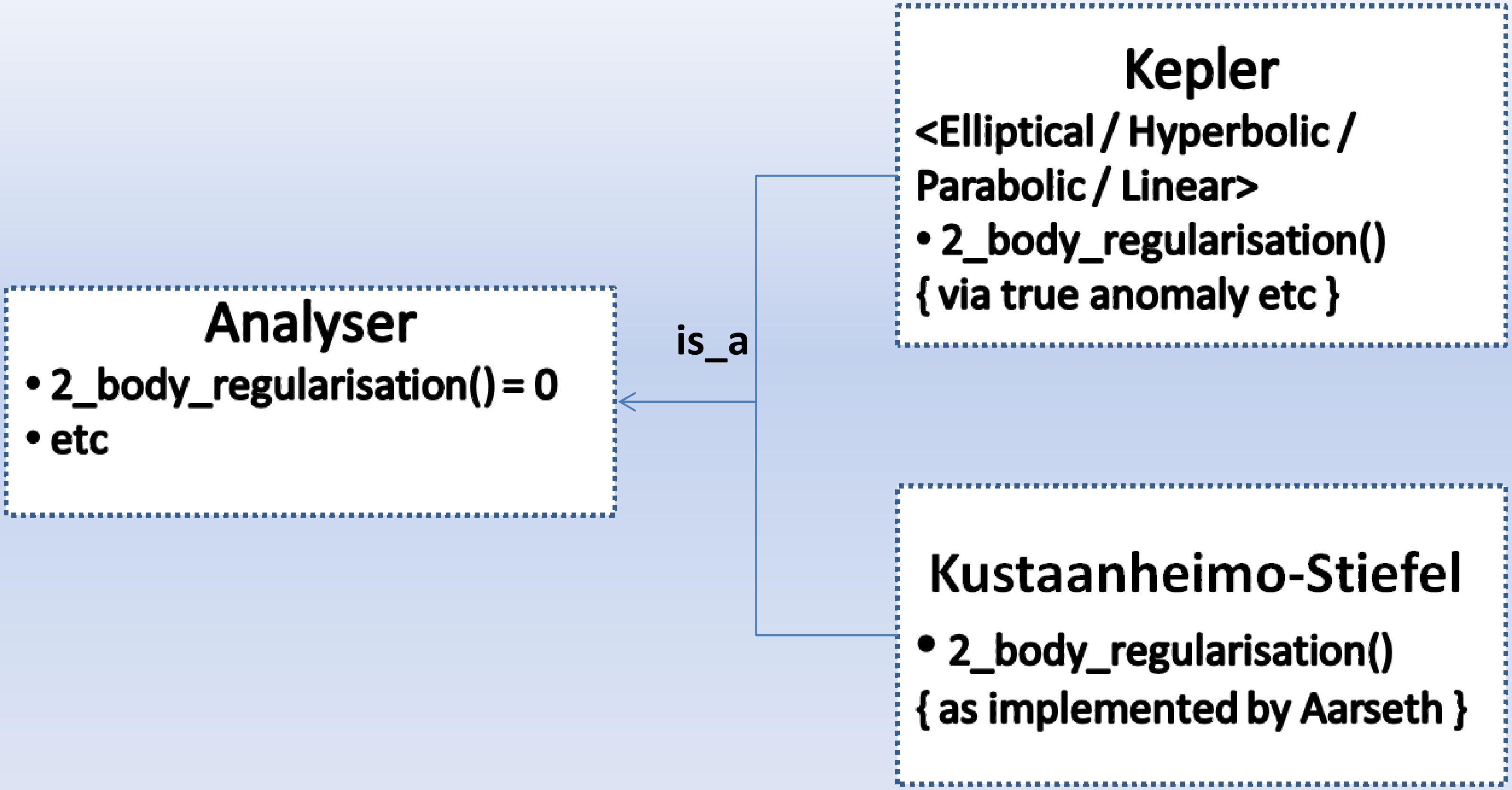
Direct

- calc_gravity(Composite *) { iterates over bodies in batches calculating gravity <Body, Composite> }

Pairwise

•calc_gravity(Composite *) { recurses down the tree calculating gravity <Composite, Composite> }

3 f. Analyser (optional)



Analyser
- 2_body_regularisation() = 0
- etc

is_a

Kepler
<Elliptical / Hyperbolic / Parabolic / Linear>
- 2_body_regularisation()
{ via true anomaly etc }

Kustaanheimo-Stiefel
- 2_body_regularisation()
{ as implemented by Aarseth }