Eunomia inspired suggestions

TSVV-5 Code Camp 2023, EIRENE refactoring

DIFFER / Pieter Willem Groen

DIFFER

NWO

Introduction

- Eunomia Monte Carlo code for neutral transport [1]
  - (Rob Wieggers, Pieter Willem Groen; ~2012)
- From scratch

- Semi OOP (f90 compliant): modules as 'classes' [2]
- Design, using 'CRC' (Class Responsibility Collaborator; identifying 'nouns')

- Pruning and enrichment (algorithm)

[1] R.C. Wieggers et al. Contrib. Plasma Phys. 52, No. 5-6, 440 – 444 (2012)
[2] Decyk, Norton, and Szymanski "Introduction to Object-Oriented Concepts using Fortran90"

Status (as far as I know)

- Jorge Gonzalez worked on grouping variables (derived TYPE)
- also:
  - variable names should be clear
  - PROCEDURES with INTENT definitions (IN/OUT/INOUT)
  - removal of SAVE statements
  - Fortan free format (as of f90)
- Particular case MODCLF and MODCOL: MODCLF into REACDAT

```
TYPE REACTION_DATA
        TYPE (FIT_FORMS), POINTER :: POT, …, CRS, …, PHR
        LOGICAL :: LPOT, LCRS, … , LPHR
        …
END TYPE REACTION_DATA
```

Eunomia: semi OOP (structure)

- Class like functionality in modules (separation of concerns)
  - Derived type, subroutines with interface
- 'Main code' imports/uses these 'classes'
- Conventions:
  - (Variables, functions, classes, modules, ...)
  - Start with 'eu', then camelCase
  - Class module name ends with 'Class'
  - 'Ordinary' module name ends with 'Mod'
  - Derive type name ends with 'T'
  - An instance ends with 'I' (e.g. particleI)
  - An array of types ends with 'Array' (e.g. particleArray)
  - ...
  - To pass the type's instance use 'this' 'keyword'

Example (illustration! Now euParticleT, ...)

```
MODULE euParticleClass

  USE euTypesMod
  USE euConstantsMod

  USE euCoordinateClass
  USE euVelocityClass

  IMPLICIT NONE

  PRIVATE :: euParticleInit

  TYPE :: particleT
    INTEGER :: cellIndex
    INTEGER :: speciesIndex
    REAL (KIND = R8)       :: mass
    REAL (KIND = R8)       :: weight
    REAL (KIND = R8)       :: timeToCollision
    TYPE (coordinate3dT)   :: vel
    TYPE (coordinateCylT)  :: pos
  END TYPE particleT

  INTERFACE euInit
    MODULE PROCEDURE euParticleInit
  END INTERFACE

CONTAINS

  SUBROUTINE euParticleInit(this)
    IMPLICIT NONE

    TYPE (particleT), INTENT(OUT) :: this
    CALL euInit(this%pos)

  END SUBROUTINE euParticleInit

END MODULE euParticleClass
```

Semi OOP (structure)

- Extension of derived type
  - e.g. to distinguish coordinate system
- Interface takes care of corresponding type
- E.g.

```
TYPE (particle2DT) :: part
CALL euInit(part)
```

- Disadvantage: particle coordinate type determined in preprocessing

Example

```
TYPE :: particleT
   INTEGER :: cellIndex
   INTEGER :: speciesIndex
   REAL (KIND = R8)      :: mass
   REAL (KIND = R8)      :: weight
   REAL (KIND = R8)      :: timeToCollision
   TYPE (coordinate3dT)  :: vel
END TYPE particleT

TYPE, EXTENDS(particleT) :: particleCylT
   TYPE (coordinateCylT) :: pos
END TYPE particleCylT

TYPE, EXTENDS(particleT) :: particle2DT
   TYPE (coordinate2DT)  :: pos
END TYPE particle2DT

 INTERFACE euInit
   MODULE PROCEDURE euParticleCylInit
   MODULE PROCEDURE euParticle2DInit
   MODULE PROCEDURE euParticleParentInit
 END INTERFACE

CONTAINS

 SUBROUTINE euParticleCylInit(this)
   IMPLICIT NONE
   TYPE (particleCylT), INTENT(OUT) :: this

   CALL euInit(this%particleT)   ! This is the way to pass the parent as an argument
   CALL euInit(this%pos)         ! This will go to the interface in the coordinate class,
                                 ! redirecting it for cylindrical coordinate type arguments
 END SUBROUTINE euParticleCylInit

 SUBROUTINE euParticle2DInit(this)
   IMPLICIT NONE
   TYPE (particle2DT), INTENT(OUT) :: this

   CALL euInit(this%particleT)
   CALL euInit(this%pos)         ! This will be redirected to the procedure
                                 ! for arguments of type coordinate2DT
 END SUBROUTINE euParticle2DInit

 SUBROUTINE euParticleParentInit(parent)
   IMPLICIT NONE
   TYPE (particleT), INTENT(OUT) :: parent

   ! Initialise common attributes here
   parent%mass            = 0.0
   parent%weight          = 0.0
   parent%timeToCollision = 0.0
   CALL euInit(parent%vel)       ! This will go to the interface in the velocity class

 END SUBROUTINE euParticleParentInit
```

Particle - states

- Particle
  - speciesIndex
  - …

- Species
  - type
  - state
  - …

euParticleClass.f90

```
  ! Basic particle type, that contains geometry independent part
  TYPE :: euParticleT
    INTEGER :: cellIndex !< cell index of test-particle
    INTEGER :: speciesIndex !< species index of test-particle
    REAL (KIND = R8) :: mass !< mass in amu of particle (rcw: or is this only stored in the species typ
    REAL (KIND = R8) :: weight !< relative weight of test-particle
    REAL (KIND = R8) :: rosenbluthWeight !< the Rosenbluth weight. Used to improve statistics on averag
    REAL (KIND = R8) :: time !< time in seconds left to be simulated. It counts down during simulation
    INTEGER :: steps !< number of steps the test particle is simulated
    REAL (KIND = R8) :: timeToCollision !< time in seconds to next collision
    REAL (KIND = R8) :: speed !< speed of particle
    TYPE (euCoordinate3dT) :: vel !< 3d cartesian velocity vector
#ifdef EU_GEOMETRY_CYL_2D
    TYPE (euCoordinateCylT) :: pos !< position of particle in 2d cylindrical symmetric geometry
    TYPE (euCoordinateCylT) :: pos0 !< starting position of particle in 2d cylindrical symmetric geomet
#endif
#ifdef EU_GEOMETRY_CAR_2D
    TYPE (euCoordinate2dT) :: pos !< position of particle in 2d cartesian geometry
    TYPE (euCoordinate2dT) :: pos0 !< starting position of particle in 2d cartesian geometry
#endif
#ifdef EU_GEOMETRY_CYL_3D
    TYPE (euCoordinate3dT) :: pos !< position of particle in 3d cylindrical geometry
    TYPE (euCoordinate3dT) :: pos0 !< starting position of particle in 3d cylindrical geometry
#endif
#ifdef EU_GEOMETRY_CAR_3D
    TYPE (euCoordinate3dT) :: pos !< position of particle in 3d cartesian geometry
    TYPE (euCoordinate3dT) :: pos0 !< starting position of particle in 3d cartesian geometry
#endif
  END TYPE euParticleT
```

euSpeciesClass.f90

```
TYPE :: euSpeciesT
  INTEGER :: type !< 0: atom, 1: molecule, 2: atomic ion (electrons including), 3: molecular ion, 4: photon
  INTEGER :: state !< ground state, or a certain vibrational state
  INTEGER :: parentSpecies !< for vibrational states, this index indicates the parent species, to which a global density, temp
  LOGICAL :: isParent
  REAL (KIND = R8) :: mass !< mass expressed in AMU
  CHARACTER (LEN = clen) :: chemicalFormula
  LOGICAL :: simulated !< true in case the species is simulated
  LOGICAL :: doubleDist !< true if we assume 2 populations, a drifting and non-drifting (default: .FALSE.)
  INTEGER :: charge !< charge of species in elementary charge units
  REAL (KIND = R8) :: ionisationPotential
  INTEGER :: lengthElementList
  INTEGER, ALLOCATABLE, DIMENSION(:, :) :: elementList
  INTEGER :: lengthCollisionList
  INTEGER, ALLOCATABLE, DIMENSION(:) :: collisionGroup
  INTEGER, ALLOCATABLE, DIMENSION(:) :: collisionList
  INTEGER, ALLOCATABLE, DIMENSION(:) :: collisionPartnerList
  ! Collision list
END TYPE euSpeciesT
```

euParticleClass.f90

```
  ! Basic particle type, that contains geometry independent part
  TYPE :: euParticleT
    INTEGER :: cellIndex !< cell index of test-particle
    INTEGER :: speciesIndex !< species index of test-particle
    REAL (KIND = R8) :: mass !< mass in amu of particle (rcw: or is this only stored in the species typ
    REAL (KIND = R8) :: weight !< relative weight of test-particle
    REAL (KIND = R8) :: rosenbluthWeight !< the Rosenbluth weight. Used to improve statistics on averag
    REAL (KIND = R8) :: time !< time in seconds left to be simulated. It counts down during simulation
    INTEGER :: steps !< number of steps the test particle is simulated
    REAL (KIND = R8) :: timeToCollision !< time in seconds to next collision
    REAL (KIND = R8) :: speed !< speed of particle
    TYPE (euCoordinate3dT) :: vel !< 3d cartesian velocity vector
#ifdef EU_GEOMETRY_CYL_2D
    TYPE (euCoordinateCylT) :: pos !< position of particle in 2d cylindrical symmetric geometry
    TYPE (euCoordinateCylT) :: pos0 !< starting position of particle in 2d cylindrical symmetric geomet
#endif
#ifdef EU_GEOMETRY_CAR_2D
    TYPE (euCoordinate2dT) :: pos !< position of particle in 2d cartesian geometry
    TYPE (euCoordinate2dT) :: pos0 !< starting position of particle in 2d cartesian geometry
#endif
#ifdef EU_GEOMETRY_CYL_3D
    TYPE (euCoordinate3dT) :: pos !< position of particle in 3d cylindrical geometry
    TYPE (euCoordinate3dT) :: pos0 !< starting position of particle in 3d cylindrical geometry
#endif
#ifdef EU_GEOMETRY_CAR_3D
    TYPE (euCoordinate3dT) :: pos !< position of particle in 3d cartesian geometry
    TYPE (euCoordinate3dT) :: pos0 !< starting position of particle in 3d cartesian geometry
#endif
  END TYPE euParticleT
```

euSpeciesClass.f90

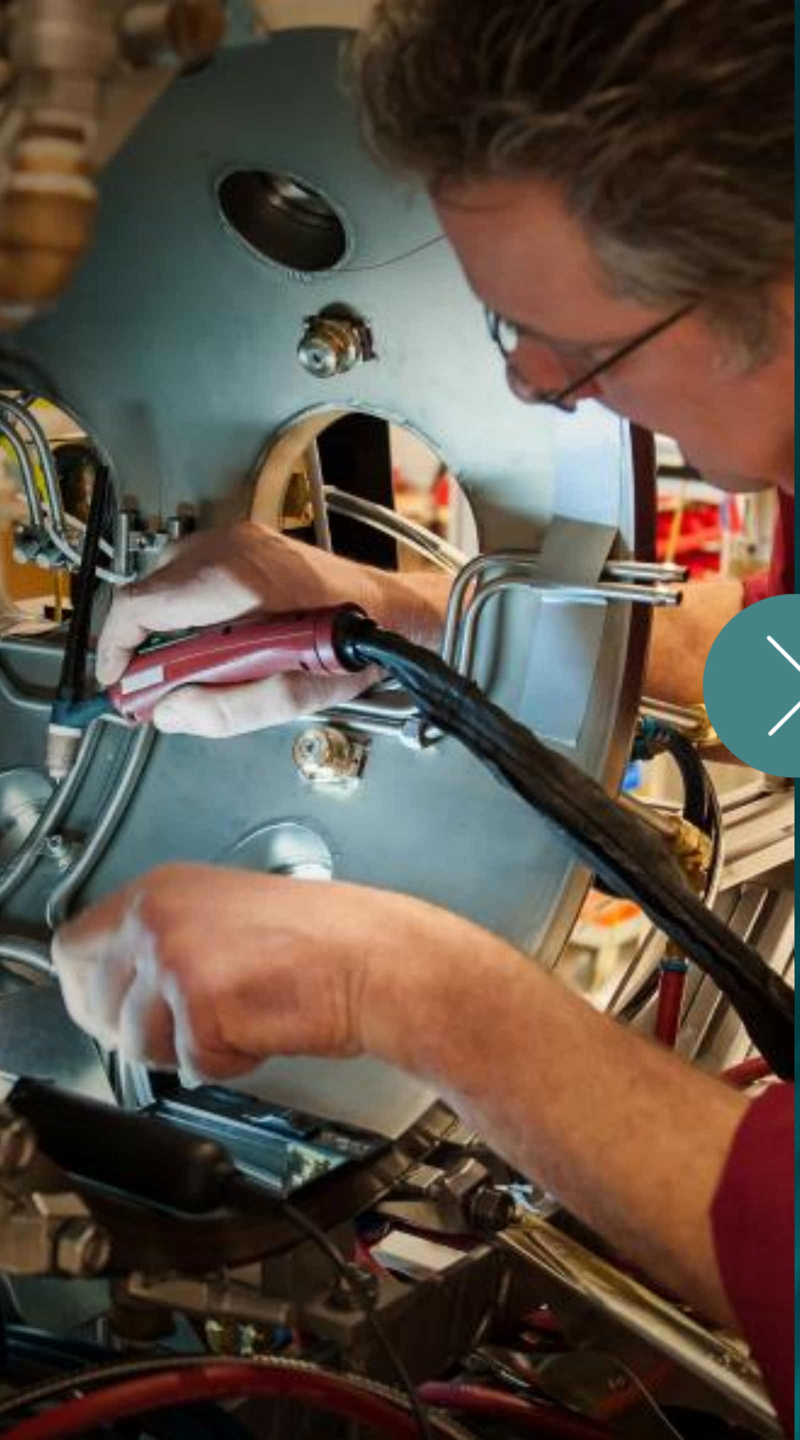
```fortran
TYPE :: euSpeciesT
  INTEGER :: type !< 0: atom, 1: molecule, 2: atomic ion (electrons including), 3: molecular ion, 4: photon
  INTEGER :: state !< ground state, or a certain vibrational state
  INTEGER :: parentSpecies !< for vibrational states, this index indicates the parent species, to which a global density, temp
  LOGICAL :: isParent
  REAL (KIND = R8) :: mass !< mass expressed in AMU
  CHARACTER (LEN = clen) :: chemicalFormula
  LOGICAL :: simulated !< true in case the species is simulated
  LOGICAL :: doubleDist !< true if we assume 2 populations, a drifting and non-drifting (default: .FALSE.)
  INTEGER :: charge !< charge of species in elementary charge units
  REAL (KIND = R8) :: ionisationPotential
  INTEGER :: lengthElementList
  INTEGER, ALLOCATABLE, DIMENSION(:, :) :: elementList
  INTEGER :: lengthCollisionList
  INTEGER, ALLOCATABLE, DIMENSION(:) :: collisionGroup
  INTEGER, ALLOCATABLE, DIMENSION(:) :: collisionList
  INTEGER, ALLOCATABLE, DIMENSION(:) :: collisionPartnerList
  ! Collision list
END TYPE euSpeciesT
```

Pruning and enrichment

- Rob Wieggers, PhD thesis
- Gain more statistics where needed
- Inspired by axially symmetric problem (small cells near axis)

Roughly:

- In cells where more particles are needed: enrichment
  - split up test-particle into multiple test-particles
  - divide the weight
- In cells where less particles are needed: pruning
  - terminate test-particle
  - or continue with increased weight



DIFFER