ModCR - refactoring reaction datastructure

Meeting TSVV-K 19-06-26

P.W. Groen, M. Gordon, D. Borodin

Motivation/background + overview

- Addition of spectroscopy task (calculate line intensities) to ModCR
- Code-wise resemblance between reactions and (emission) transitions
  - species
  - states
  - rates - PECs (ADF data)
- This suggests a superclass for Reaction and Transition (new class): "Transformation"
- The current Reaction class is abstract, with subclasses Reaction1d, Reaction2d
- "1d" and "2d" refers to the data (rates) defined as a function of Te, Ne (2D) or Te (1D) and the corresponding linear interpolation function
- The Transition class needs these interpolation functions as well (PECs)

Current situation

- Abstract type / class cRMReaction_t
- Two extensions / subclasses: cRMReaction1d_t, cRMReaction2d_t
- Dimensions 1d and 2d refer to the data (rate coefficients) to be interpolated (only)
  - 1d: Te (or Ne) dependent
  - 2d: Ne and Te dependent
- The types cRMReaction1d_t and cRMReaction2d_t only differ in the way
  - the "coordinates" the data (rates) are defined on, and
  - how they calculate (interpolate) the rate

- To be able to place the subclasses in a list (cRMReactionList_t), ...
- ... this list is a list of "containers" (cRMReactionContainer_t) containing objects of subclasses of cRMReaction_t (the only way to do this in Fortran)
- Call sequence goes via this list:
  - type :: cRMReactionList_t has linterp => getLinearInterp1d, getLinearInterp2d
  - that call "linterp" on either cRMReaction1d_t or cRMReaction2d_t
  - linterp=>linearInterp1d, linterp=>linearInterp2d, resp.

Diagram (current)

MCRm_Reaction

cRMReactionList_t
- Integer: nReac
- Integer: maxReac
- init()
- add()
- Real«dp» linterp(Integer: Index, Real«dp»: x)
- Real«dp» linterp(Integer: Index, Real«dp»: x, Real«dp»: y)

1 — reactionContainer — 1..*

cRMReactionContainer_t

1 — reaction — 1

cRMReaction_t
- String«len=prStateLen»: name
- Integer: type
- Integer: nIn
- Integer: nOut
- init()
- delete()

1 — states — 1

cRMReactionStates_t

1 — adf11Ind_t — 1

adf11Ind_t

cRMReaction1d_t
- Integer: nDim1
- Real«dp»: dim1[1..*]
- Real«dp»: rate[1..*]
- Real«dp»linterp(Real«dp»: x)

cRMReaction2d_t
- Integer: nDim1
- Integer: nDim2
- Real«dp»: dim1[1..*]
- Real«dp»: dim2[1..*]
- Real«dp»: rate[1..*][1..*]
- Real«dp» linterp(Real «dp»: x, Real «dp»: y)

[Summarised]

Spectroscopy task

- Addition of spectroscopic data, to be able to calculate emission line intensities from population densities and photon emissivity coefficients (PECs)
- The photon emissions are related to transitions of states
- The PECs are read from a database and are also given for Ne, Te

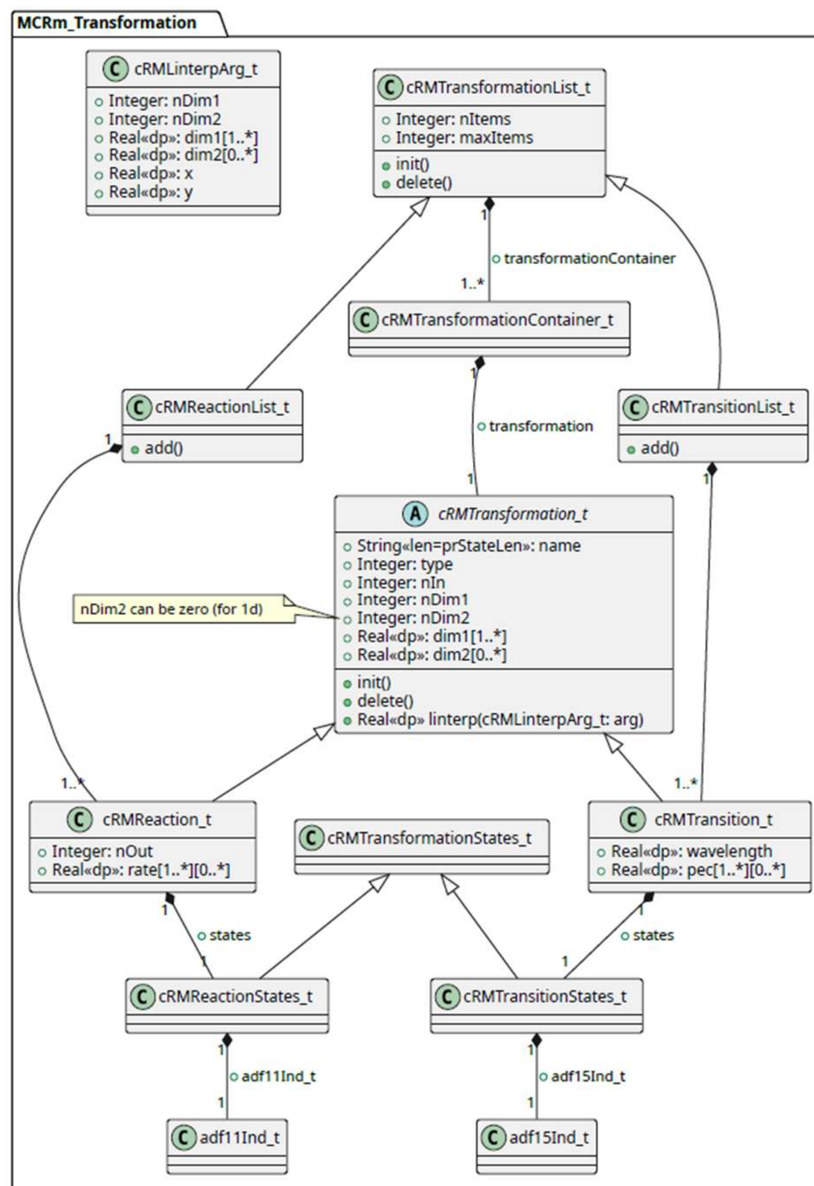
- Analogy (in code) between reactions (with rates) and transitions (with PECs)
- "Copying" the reaction structure to the transition structure will cause too much overhead

Proposed setup

- create a common superclass for reactions and (to be added) transitions: 'transformation',
- and a transformation list (with subclasses reaction list and transition list)
- 'transformation' has an interface to a linear interpolation function that takes a derived type as an argument,
- the subclasses implement their own linear interpolation function (with the same argument)
- the "rate" array (reactions) and "pec" array (transitions) are of rank 2, but one dimension can be zero
- let cRMReaction_t and cRMTransition_t (subclasses of cRMTransformation_t) be able to take care of *both* 1d and 2d interpolation (using a derived type as an argument)

Diagram (proposed)

- The method "linterp" returns a real value (rate or PEC)
- Takes a derived type as an argument that contains the 1d/2d information needed
- This enables to have an interface that can treat "1d" and "2d" type instances in the same way



[Summarised]

Test (proof of concept)

- Create reaction and transition lists with instances of Reaction, Transition resp.
- Allocate the "rate" array per instance, one 2 dimensional, one 1 dimensional
- Loop over the lists calling an "interpolation" function on each instance

- First also: create a hybrid list of reactions and transitions: worked, but not used

DIFFER

Code snippets

- Create reaction and transition lists with instances of Reaction, Transition resp.
- Allocate the "rate" array per instance, one 2 dimensional, one 1 dimensional
- Loop over the lists calling an "interpolation" function on each instance

```fortran
type, extends(transformation_t) :: reaction_t
    character(len=16) :: type
    real, allocatable, dimension(:,:) :: rate
contains
    procedure, pass :: init => init_reaction
    procedure, pass :: del => del_reaction
    procedure, pass :: linterp => linterp_reaction
end type reaction_t

real function linterp_reaction(this, arg) result (rate)
    class(reaction_t), intent(inout) :: this
    class(linterpArg_t), intent(in) :: arg

    rate = arg%x
end function linterp_reaction

type, extends(transformationList_t) :: reactionList_t
contains
    procedure, pass :: add => addReaction
!    procedure, pass :: del => deallocateReactions ! Not necessary
end type reactionList_t
```

```fortran
type(reactionList_t) :: reactionList
TYPE(transitionList_t) :: transitionList
type(linterpArg_t) :: interpArg

integer :: nrReac, nrTransit, i
real :: interpVal

nrReac   = 2
nrTransit = 2

interpArg%x = 2.0
interpArg%y = 3.0

! Create a reaction list (and circumvent a types list)
call reactionList%init(nrReac)
! Add a reaction with index 1, first dimension 2, second dimension 2
! The first dimension corresponds to the number of Ne points, the second to the number of Te points
call reactionList%add(1, 2, 2)
! Add a reaction with index 2, first dimension 2, second dimension 0
call reactionList%add(2, 2, 0)
do i = 1, nrReac
    write (*, *) i, reactionList%trc(i)%tr%name
    write (*, *) i, reactionList%trc(i)%tr%index
    write (*, *) i, reactionList%trc(i)%tr%nDim1
    write (*, *) i, reactionList%trc(i)%tr%nDim2
    interpVal = reactionList%trc(i)%tr%linterp(interpArg)
    write (*, *) i, reactionList%trc(i)%tr%name, interpVal
end do
```

test/general_fortran_tests/test_multiple_type_list.f90

Code snippets

- Mixed list of reactions and transitions (deprecated)

```
type(transition_t) :: transition
type(transformationList_t) :: trList

integer :: myIndex, nrTransf, i

nrTransf = 2

call trList%init(nrTransf)
call trList%initTypesList(nrTransf)
call trList%addTransformations()
do i = 1, nrTransf
    write (*, *) i, trList%trc(i)%tr%name
end do
call trList%del()
```