A declarative mapping language

- Motivations for a declarative style
  - Abstract from underlying representations
  - Abstract from implementation language
  - Capture of intent of a mapping
  - Able to generate mapping code
- VML (View Mapping Language)
  - Bi-directional mapping specification

Structure of VML

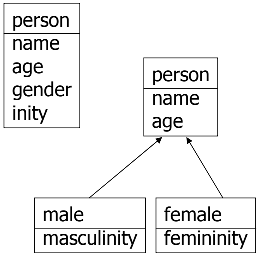
- inter_view
  - Describes the 2 schemas being mapped between
    - Versions being mapped between
    - Type of information transfer required (read-only, read_write, integrated)
    - Whether this is a complete or partial mapping
- inter_class
  - Describes sets of classes that need to combine for a mapping
  - Three parts to each inter_class description
    - Invariants: what must hold true for this mapping to proceed
    - Equivalences: the mappings to perform
    - Initialisers: values to be set when a new object is created

inter_class example

```
inter_view(idm, integrated, view1, read_write, complete).

inter_class([person],[male],
        invariants(     gender = 'male'),
        equivalences(   name = name,
                        age = age,
                        inity = masculinity)
).

inter_class([person],[female],
        invariants(     gender = 'female'),
        equivalences(   name = name,
                        age = age,
                        inity = femininity)
).
```



inter_class classes

- Can specify one or more classes from each schema
  - If one class then inter_class is applied to every object of that class (as long as the invariants are satisfied)
  - If more than one class then the cross product of objects is used for the mapping
  - For example:
    - Class a has objects o1 and o2
    - Class b has objects o3, o4, and o5
    - inter_class([a, b], [c], …) evaluates the mapping for:
      - [o1, o3], [o1, o4], [o1, o5], [o2, o3], [o2, o4], [o2, o5]
    - group() function allows all objects of a class to be grouped
    - E.g., inter_class([a, group(b)], [c], …) evaluates the mapping for:
      - [o1, [o3, o4, o5]], [o2, [o3, o4, o5]]

invariants

- Define the conditions under which an inter_class is applicable (e.g., gender = ‘male’)
  - Reduce the set of objects which are evaluated
- Each individual invariant may only reference attributes and objects from one of the schemas.
- A constraining condition applied in one direction is a default value in the opposite direction.
  - E.g., when creating a ‘person’ object from one of type ‘male’ in the previous example then the ‘gender’ attribute of the ‘person’ object is set to ‘male’.

initialisers

- Assignment statements for attributes
- Only applicable to newly created objects
  - Can call methods of new objects

```
initialisers(
  idm_space_face.face_property = 'idm_space_face',
  idm_material_face.face_property = 'idm_material_face',
  idm_material_face.material=>type_of_material = 'idm_window_material',
  idm_material_face.material=>type_of_window = 'idm_single',
  idm_material_face.material=>window_subtype = 'clear',
  fe_opening@create(idm_space_face.plane, idm_space_face.plane, 'space', 0, 0,
            idm_space_face.min=>x, 0 - idm_space_face.min=>y,
            idm_space_face.max=>x, 0 - idm_space_face.max=>y,
            idm_material_face.material=>window_subtype)
)
```

equivalences

- Equations, functions, and procedures to perform a mapping
- Ordering of specification is unimportant
- Types of equivalence equations include:
  - Initialisers (e.g., gloss_factor = 90.0)
  - Equality (e.g., name = planeName)
  - Pointer equality (e.g., plane = fe_face_window)
  - Simple equations (e.g., r*sin(theta) = y_coord)
  - Pointer references (e.g., apex1=>x = apex2=>x
  - Functions (e.g., exists(end_point=>z)
  - Aggregate functions (e.g., sum(windows=>(height*width))) = area

equivalences

- Types of equivalence equations include:
  - List and array references (e.g., axes[2] = v_ref)
  - List and array iteration (e.g., classified_by[] = material[].name)
  - Conditional list and array iteration, for example,
    bijection(spaces[]@class(‘idm_space’), spaces=>list[])
    bijection(spaces[]@class(‘idm_roof’), roofs=>list[])
  - Functions (e.g., list_splitter(vals, splitvals))
  - Procedures (e.g., map_to_from(procA(), procB()))
  - Method invocation (e.g., plane@view_plane = fe@create_view(name))
  - Type conversion – implicit evaluation or cast explicitly
  - Unit conversion – explicit modelling
  - Temporary/intermediate attributes (e.g., _temp)