Use known *structures* to construct models.
Create a well-informed instance of the model.
Reformulate the model+data based on existing transformation rules.
Pass the selected instance to a solving mechanism that exploit the structure information.
I-DARE(lib) - structure library

- defines the available structures to build models,
- how structures will interact.
- Structures = classes in a hierarchy.
- This hierarchy can be dynamically extended.

- Different linking variables must contain disjoint variable patterns.
- The unification mechanism was extended for that purpose.

```csharp
// Component
[abstract,
 // Methods
 => wellformed,
 vars => _list,
 freeinds => _list,
 pureinds => _list,
 allinds => _list
].
```

```csharp
// LeafProblem
:: d_Component_C
[d_leafProblem_C
[abstract,
 args *-> _list,
 [local]].
```

```csharp
// Block
:: d_Component_C
[d_Block_C
[abstract
subsC *-> _list,
link *-> _list
].
```
I-DARE(IM) - The internal Model

- Defines the way the model is stored internally.
- Permits the representation of *dimensions*, *constants*, *variables*, *vectors*, *leaf-problems*, *blocks*
- Formally defines the concept of well-formed formulation.

```
!-DARE(lib) - structure library
!-DARE(im) - The internal Model
!-DARE(ei) - Enhanced Instance
!-DARE(r) - reformulation package
!-DARE(solve) - Solution framework
```

- Formally defines the concept of well-formed formulation.

```
dimension(plant).
d_index(i, plant).

capacity : d_property.
capacity : d_constant [dims -> [plant]].

$(capacity(i), [i], i>1)

?id:?class [args -> ?args, [condition -> ?cond]].

?id:?class [subs -> ?subs, subVP -> ?subVP, [freet -> ?freet]].
```

```
d_Formulation [dimensions => _list, properties => _list, indexes => _list, root => d_Component].
```
Having the model of a problem is not enough,
To solve the problem we need the data that will instantiate the model pieces.
I-DARE(ei) uses the structural information expressed in I-DARE(im) to construct the instance.
The data will be divided in two types, *global data* and *local data*.

```
I-DARE(ei) - Enhanced Instance
```

```
 d_InstanceWrapper
 [ // Properties
    formulation  => d_Formulation,
    global       => d_DataHandler,
    local(prob)  => _list,

    // Methods
    dimSize(Dim_name)  => _integer,
    dimList(Dim_name)  => _list,
    constVal(Cons_name, ?dims) => _double,
    constValList(Cons_name) => _list
 ]
```

```
Formulation
subscription
Global Data
```

```
Local Data
```

```
Meta DataFile & File Structure
```

```
d_Component_C
```

```
d_DataHandler
```

```
d_InstanceWrapper
```

```
metadata file generation
```

---

**Having the model of a problem is not enough,**
**To solve the problem we need the data that will instantiate the model pieces.**
**I-DARE(ei) uses the structural information expressed in I-DARE(im) to construct the instance.**
**The data will be divided in two types, *global data* and *local data*.**

```
I-DARE(ei) - Enhanced Instance
```

```
 d_InstanceWrapper
 [ // Properties
    formulation  => d_Formulation,
    global       => d_DataHandler,
    local(prob)  => _list,

    // Methods
    dimSize(Dim_name)  => _integer,
    dimList(Dim_name)  => _list,
    constVal(Cons_name, ?dims) => _double,
    constValList(Cons_name) => _list
 ]
```

```
Formulation
subscription
Global Data
```

```
Local Data
```

```
Meta DataFile & File Structure
```

```
d_Component_C
```

```
d_DataHandler
```

```
d_InstanceWrapper
```

```
metadata file generation
```

---

**Having the model of a problem is not enough,**
**To solve the problem we need the data that will instantiate the model pieces.**
**I-DARE(ei) uses the structural information expressed in I-DARE(im) to construct the instance.**
**The data will be divided in two types, *global data* and *local data*.**

```
I-DARE(ei) - Enhanced Instance
```

```
 d_InstanceWrapper
 [ // Properties
    formulation  => d_Formulation,
    global       => d_DataHandler,
    local(prob)  => _list,

    // Methods
    dimSize(Dim_name)  => _integer,
    dimList(Dim_name)  => _list,
    constVal(Cons_name, ?dims) => _double,
    constValList(Cons_name) => _list
 ]
```

```
Formulation
subscription
Global Data
```

```
Local Data
```

```
Meta DataFile & File Structure
```

```
d_Component_C
```

```
d_DataHandler
```

```
d_InstanceWrapper
```

```
metadata file generation
```

---
A structure $S_1$ can be transformed in $S_2$ if we have

- An argument mapping of the $S_1$ arguments into $S_2$ arguments ($argMap$) and
- A back mapping of the $S_2$ results into $S_1$ results ($backMap$) or

A reformulation algorithm can be designed, starting from well-formed instances to obtain equivalent well-formed instances.
Common solver interface.
- Extensible way of reporting solutions.
- Extensible solver configuration.
- The input is a *solving tree* + the enhanced instance.
- Each solver is registered to one structure in I-DARE(lib).
- It is the responsibility of each solver to call the proper sub-solvers (already set in the solving tree)