Virtual Integration of Heterogeneous Data and Data Model Unification

Specification Calculus

Sergey Stupnikov

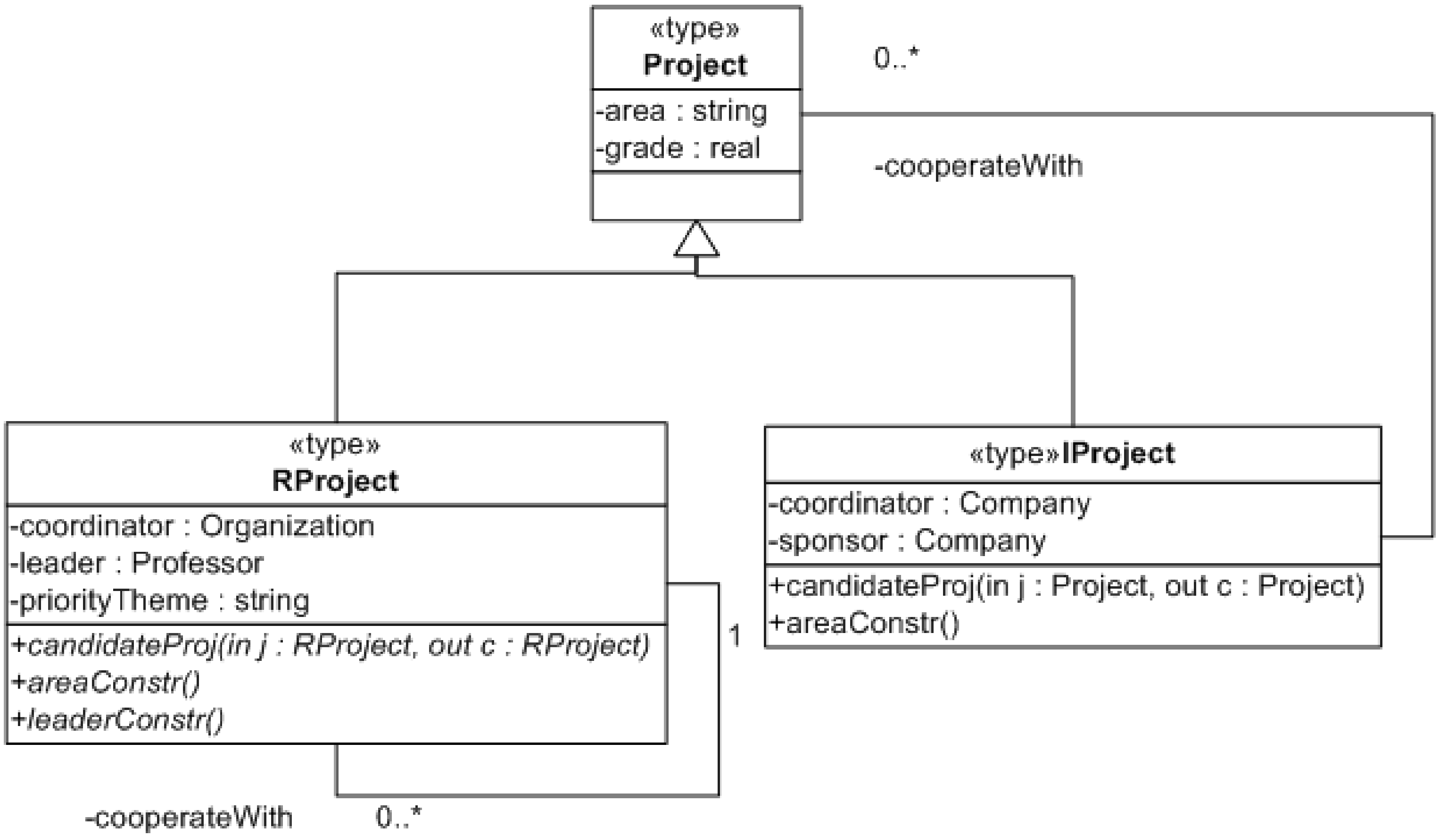
Institute of Informatics Problems, RAS

sstupnikov@ipiran.ru

Contents

Abstract Data Type

- ADT definition includes a specification defining a *behavior of the type values* by means of the *operation signatures* and of their *abstract descriptions*



Research Project Specification


```
{ RProject;     in: type;     supertype: Project;
 coordinator: Organization;
 leader: Professor;
 priority_theme: string;
 cooperate_with: {set; type_of_element: RProject; };

candidate_proj: {in: function;
    params: { +j/RProject, -c/Project};
    {{ this.area =  j.area  &  this.priority_theme = j.priority_theme & c' = j  }}};

 area_constr: {in: predicate, invariant;
    {{ all p/RProject (p.area = 'comp-sci' ->
          p.grade = 5 &
         (p.priority_theme = 'open systems' | p.priority_theme = 'interoperability'))   }}};

 leader_constr: {in: predicate, invariant;
    {{  all p/RProject (p.leader.degree = 'PhD')  }}}
 }
```

Industrial Project Specification

```
{IProject;
 in: type;
 supertype: Project;
 coordinator: Company;
 cooperate_with: {set; type_of_element: Project; };
 sponsor: Company;

 candidate_proj: {in: function;
     params: {+j/Project, -c/Project};
     {{ this.area =  j.area  & c' = j }}};

 area_constr: {in: predicate, invariant;
     {{ all p/IProject (p.area =  'comp-sci'  -> p.grade >= 3 )}}}
 }
```

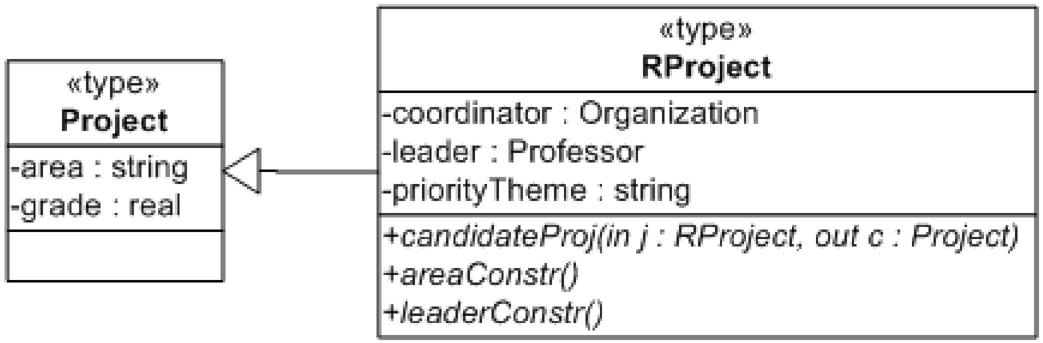
Subtype relation informally

- a value of a subtype can be used in all cases where a supertype value is expected
- correspondence of type operations
  - supertype's invariant should be *implied* by subtype's invariant
  - supertype's operations should be *refined* by subtype's operations
- multiple subtyping is allowed (for a subtype a set of supertypes can be defined)
- operations of a subtype to which operations of a supertype correspond can be renamed in case of multiple subtyping

Type Specification

- *Type specification* is a triplet $<V_T, O_T, I_T>$
  - V_T – extension the type (carrier of the type) - set of admissible instances of the type
  - O_T – operation symbols, indicating operation arguments and result types
  - I_T – invariant symbols
- Conjunction of all invariants in I_T constitutes the type invariant Inv_T
- Every instance must satisfy the invariant Inv_T

Type Specification - Example



- $V_{RProject}$ = {
  <self→uid1, area→'comp-sci', grade→5, coordinator→uid2, leader→uid3, proiorityTheme→'interoperability'> ,

  <self→uid4, area→'biology', grade→3, coordinator→uid5, leader→uid6, proiorityTheme→'gene-analysis'> ,

  … }
- $O_{RProject}$ = { candidateProj(+j/Rproject, -c/Project) }
- $I_{RProject}$ = { areaConstr, leaderConstr }
- $Inv_{RProject}$ =

  all p/RProject (p.area = 'comp-sci' -> p.grade = 5 &

  (p.priority_theme = 'open systems' | p.priority_theme = 'interoperability')) &

  all p/RProject (p.leader.degree = 'PhD')

Subtype relation formally

- *Invariant rule:* $\forall v: V_{sub}\ (I_{sub}(v) \Rightarrow I_{sup}(Abs(v)))$
- *Precondition rule:* subtype operation should terminate whenever a supertype operation is guaranteed to terminate

  $\forall v_s: V_{sub}, x?: X\ (preO_{sup}(Abs(v_s), x?) \Rightarrow preO_{sub}(v_s, x?))$
- *Postcondition rule:* the state after the subtype operation (marked by ‘) represents one of those abstract states in which an operation of a supertype could terminate

  $\forall v_s: V_{sub}, v'_s: V_{sub}, x?: X, y?: Y\ ($

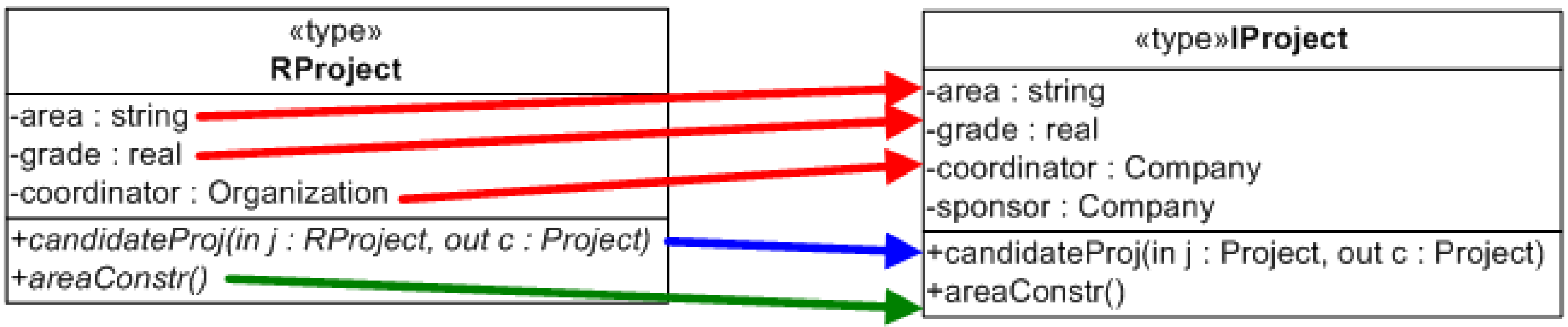
  $preO_{sup}(Abs(v_s), x?) \wedge postO_{sub}(v_s, v'_s, x?, y?) \Rightarrow$

  $postO_{sup}(Abs(v_s), Abs(v'_s), x?, y?)\ )$
- Type specification (ex. T_{sup}) is *correct* if
  - it has a model $\exists v: V_{sup}\ (I_{sup}(v))$
  - type operations preserve type invariants

    $\forall v: V_{sup}, v': V_{sup}, x?: X, y?: Y\ ($

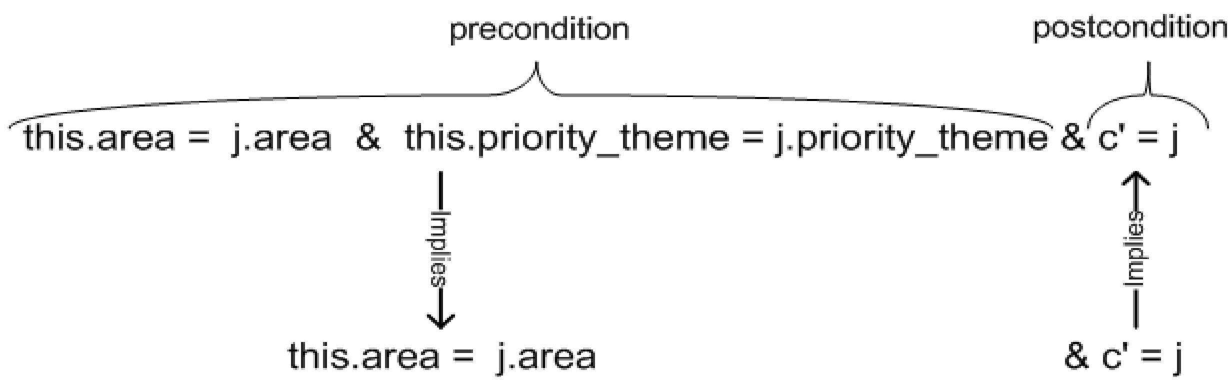
    $I_{sup}(v) \wedge preO_{sup}(v, x?) \wedge postO_{sup}(v, v', x?, y?) \Rightarrow I_{sup}(v')\ )$

Subtype relation example (I)

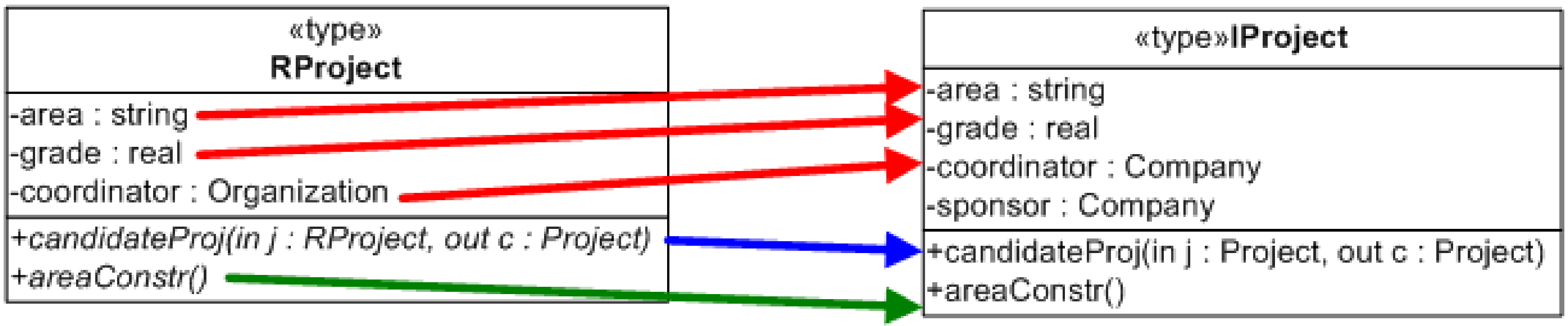


- *Organization* is a supertype of *Company*

- *RProject.candidateProj* is refined by *IProject.candidateProj*



Subtype relation example (II)



- *Iproject.areaConstr*

  is implied by

  Rproject.areaConstr

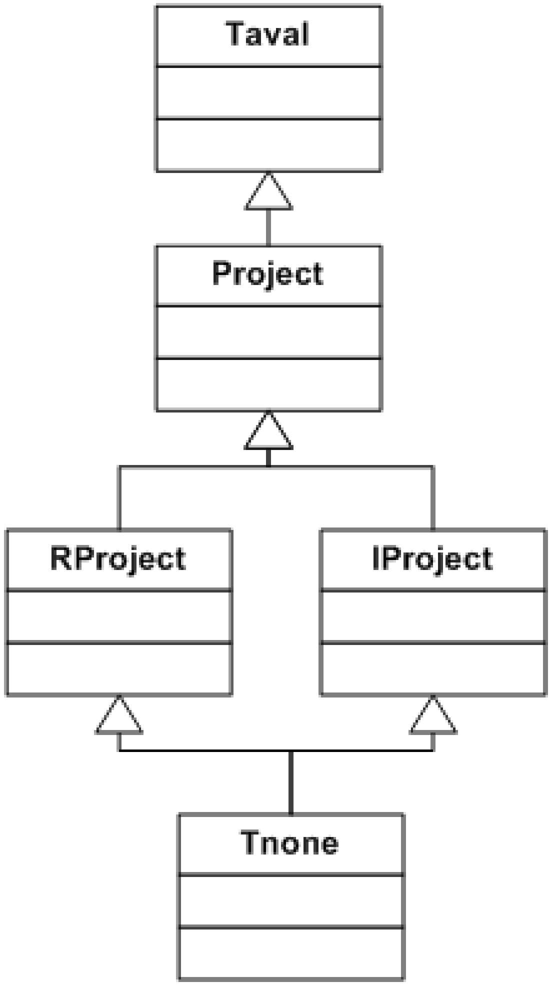
p.area = 'comp-sci' => p.grade >= 3

↑ Implies

p.area = 'comp-sci' => p.grade = 5 &
(p.priority_theme = 'open systems' | p.priority_theme = 'interoperability')

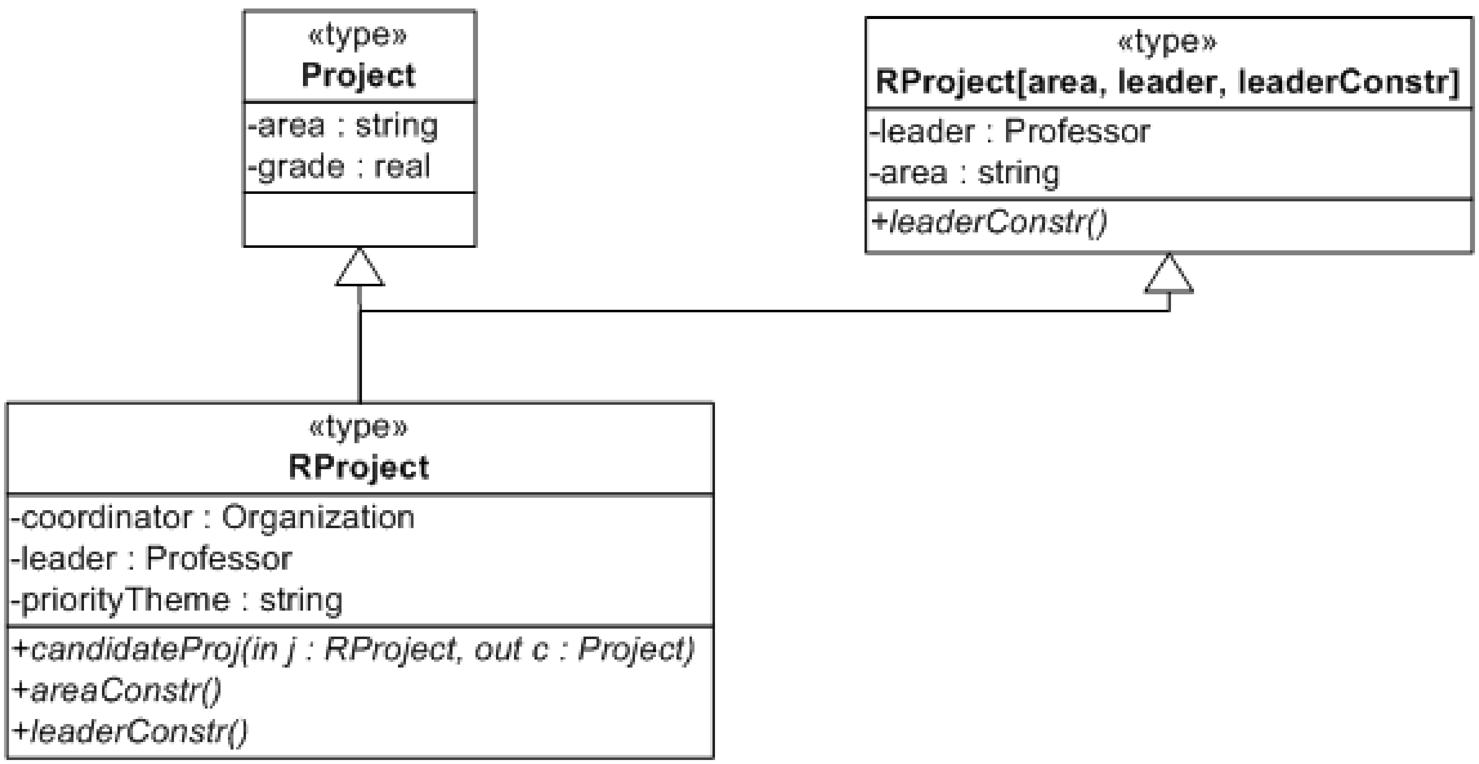
Overdefined and Least Informative Types

- **Taval** is *least informative type*, any type is a subtype of Taval
- **Tnone** is *overdefined* type, any type is a supertype of Tnone
  - predefined *none* value is of type Tnone and may be returned by a function as an empty result of any type

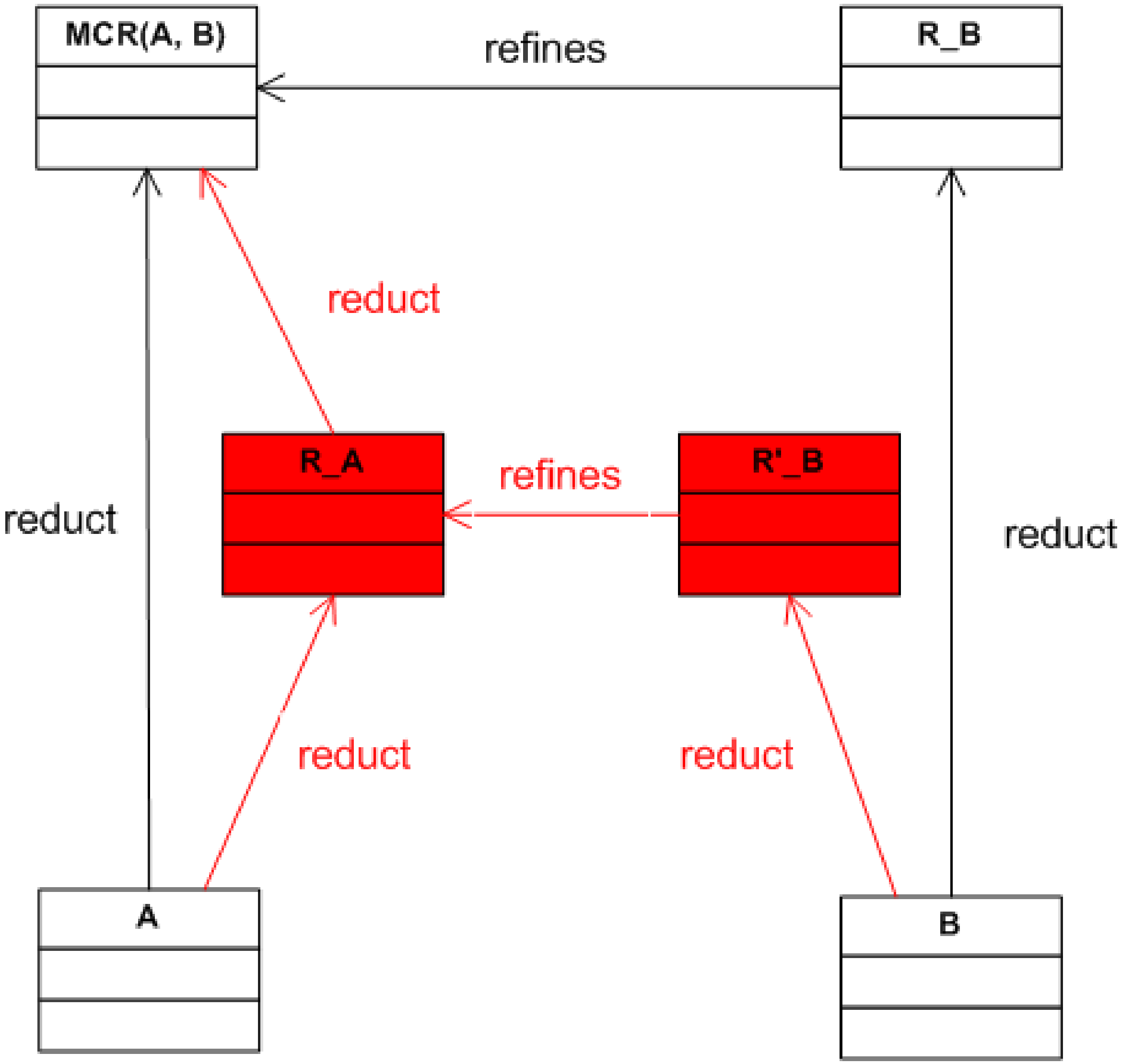


Type Reduct

- *Reduct* $R_T = \langle V_R, O_R, I_R \rangle$ of type $T = \langle V_T, O_T, I_T \rangle$ is a subspecification of type *T:*
  - $V_R = V_T$
  - $O_R \subseteq O_T$
  - $I_R \subseteq I_T$



Most Common Reduct



$$MCR(A, B) \not\equiv MCR(B, A)$$

Type Refinement

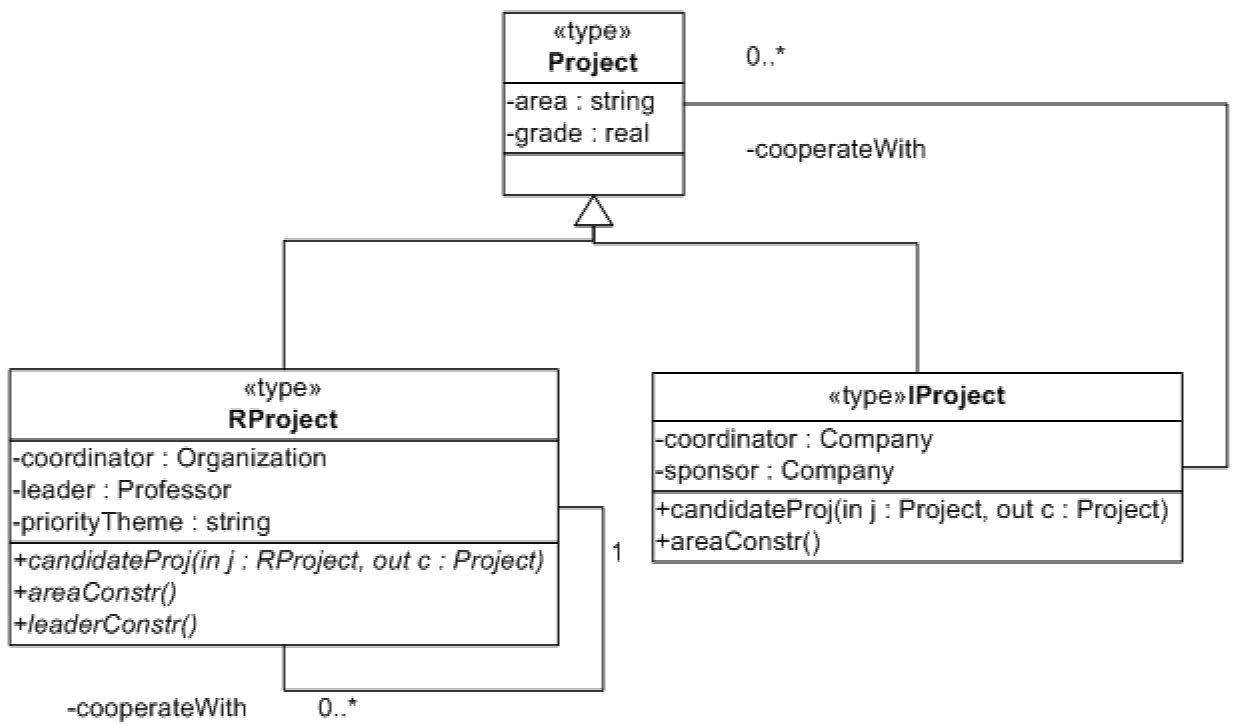
- Type *U* is a *refinement* of type *T* iff
  - there exists an injective mapping *Ops*: $O_T \rightarrow O_U$;
  - there exists an abstraction function *Abs:* $V_U \rightarrow V_T$ that maps each admissible state of *U* into the respective state of *T*
  - $\forall x \in V_T, y \in V_U (Abs(x, y) \Rightarrow I_U(y) \wedge I_T (x))$
  - for every operation $o \in O_T$ the operation $Ops(o) = o' \in O_U$ is a refinement of *o*
    - $pre(o) \Rightarrow pre(o')$
    - $post(o') \Rightarrow post(o)$.

MCR(RProject, IProject)

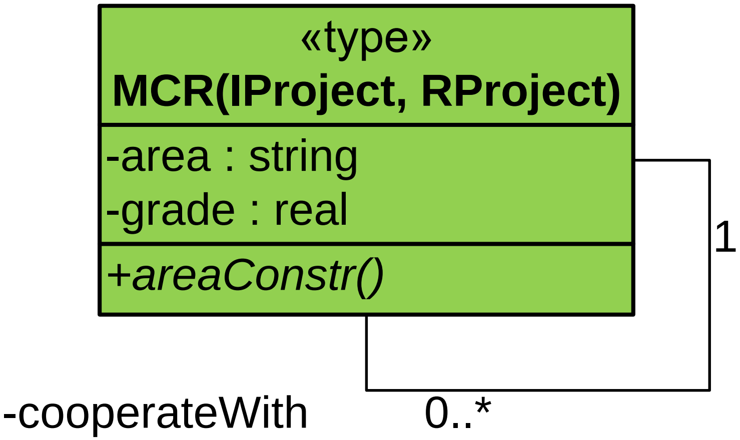
«type»
MCR(RProject, IProject)

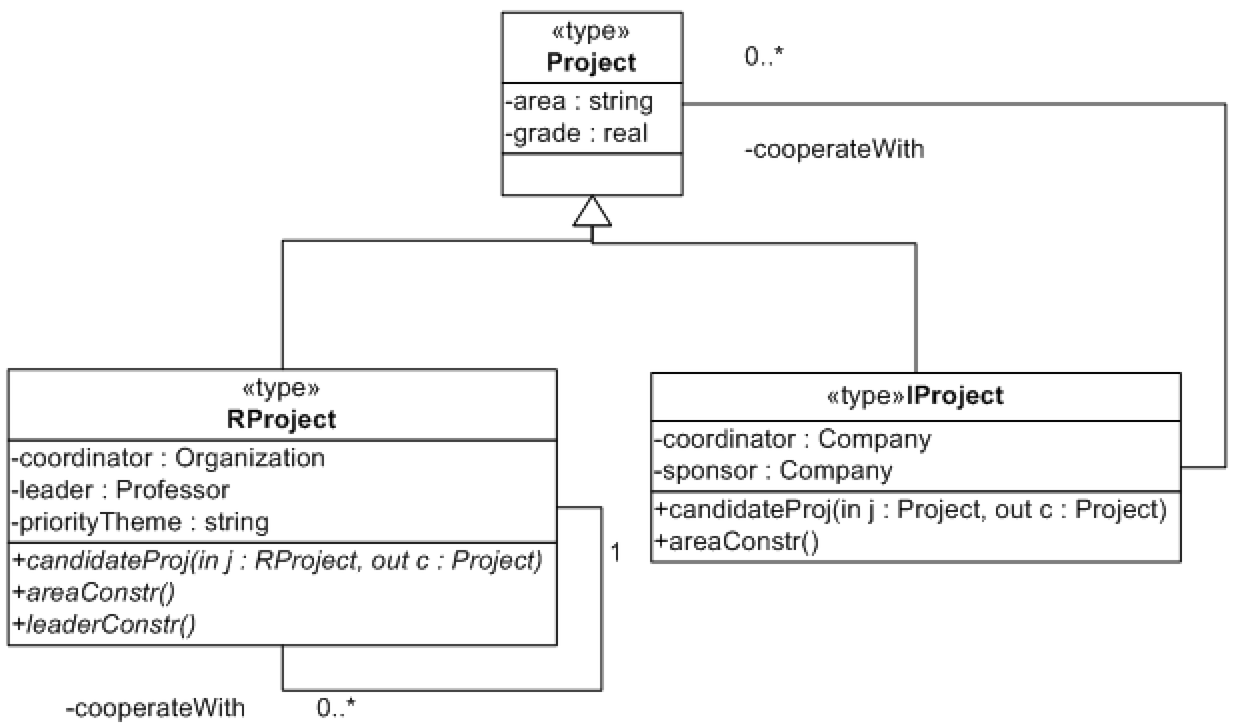
-area : string
-grade : real
-coordinator : Organization

+candidateProj(in j : RProject, out c : Project)



MCR(IProject, RProject)

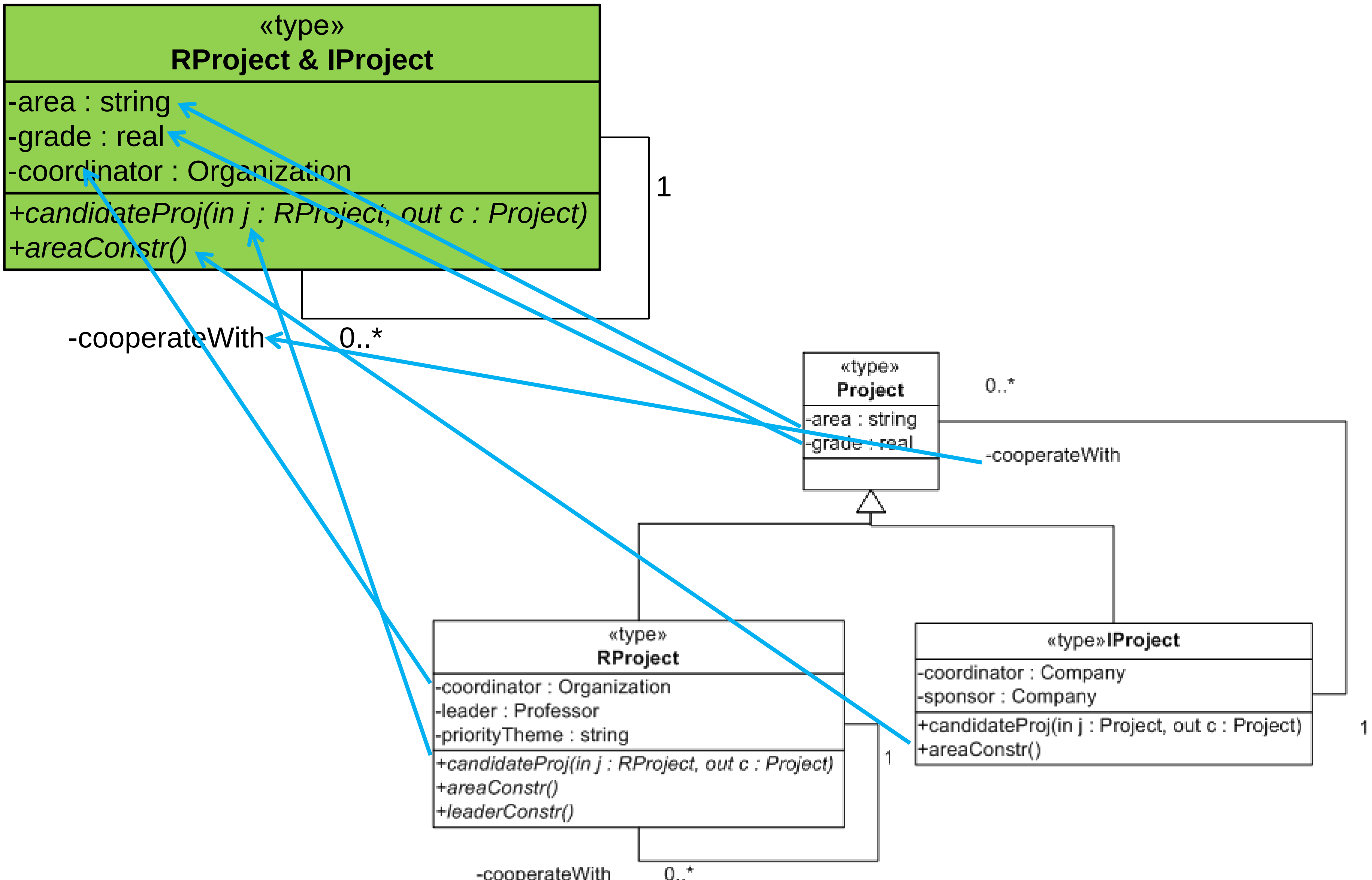




Type MEET

- The *meet* operation T_1 & T_2 of produces a type *T* as an "intersection" of specifications of the operand types
- Common elements of the types are defined by most common reducts $MCR(T_1, T_2)$ and $MCR(T_2, T_1)$
- $O_{T1 \& T2} = O_{MCR(T1,T2)} \cup O_{MCR(T2,T1)}$
- Type invariant of *T* is defined as a disjunction of operand types invariants $Inv_{MCR(T1,T2)} \mid Inv_{MCR(T2,T1)}$
- T_1 & T_2 is a supertype of both T_1 and T_2

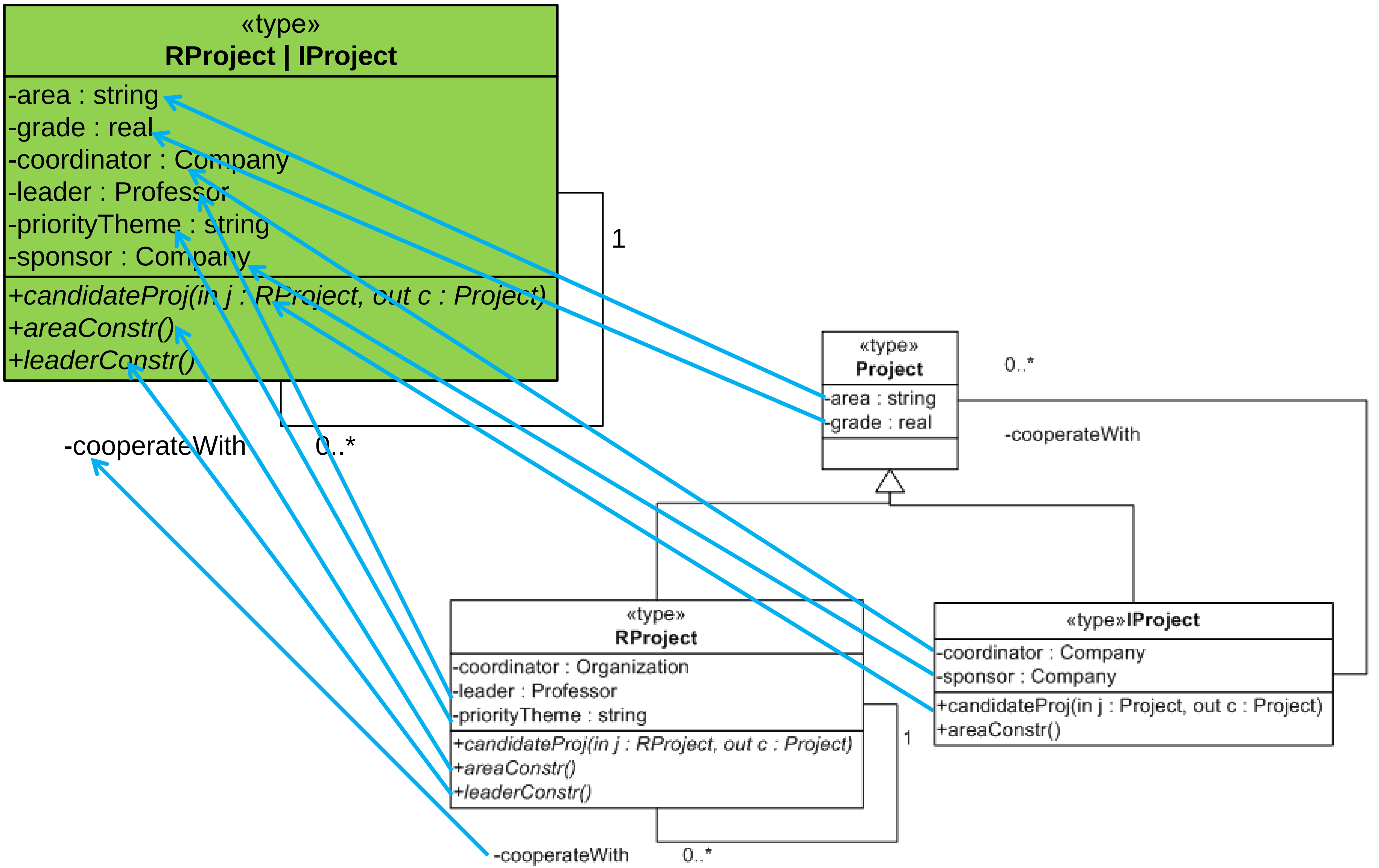
Type Meet Example



Type JOIN

- The *join* operation $T_1 \mid T_2$ produces a type *T* as a "join" of specifications of the operand types, common elements are included only once
- Common elements of the types are defined by most common reducts $MCR(T_1, T_2)$ and $MCR(T_2, T_1)$
- $O_{T1 \mid T2} = (O_{T1} \backslash O_{MCR(T1,T2)}) \cup (O_{T2} \backslash O_{MCR(T2,T1)}) \cup (O_{MCR(T1,T2)} \cap O_{MCR(T2,T1)})$
- Type invariant of *T* is defined as a conjunction of operand types invariants Inv_{T1} & Inv_{T2}
- $T_1 \mid T_2$ is a subtype of both T_1 and T_2

Type Join Example



Type Lattice

- Set $\mathcal{V}$ of types is a *lattice* over *meet* and *join* operations
  - commutativity
    - $T_1 \& T_2 = T_2 \& T_1$
    - $T_1 \mid T_2 = T_2 \mid T_1$
  - associativity
    - $T_1 \mid (T_2 \mid T_3) = (T_1 \mid T_2) \mid T_3$
    - $T_1 \& (T_2 \& T_3) = (T_1 \& T_2) \& T_3$
  - idempotence
    - $T \& T = T$
    - $T \mid T = T$
  - absorption
    - $T_1 \& (T_1 \mid T_2) = T_1$
    - $T_1 \mid (T_1 \& T_2) = T_1$