Static Population of Preexisting Instances

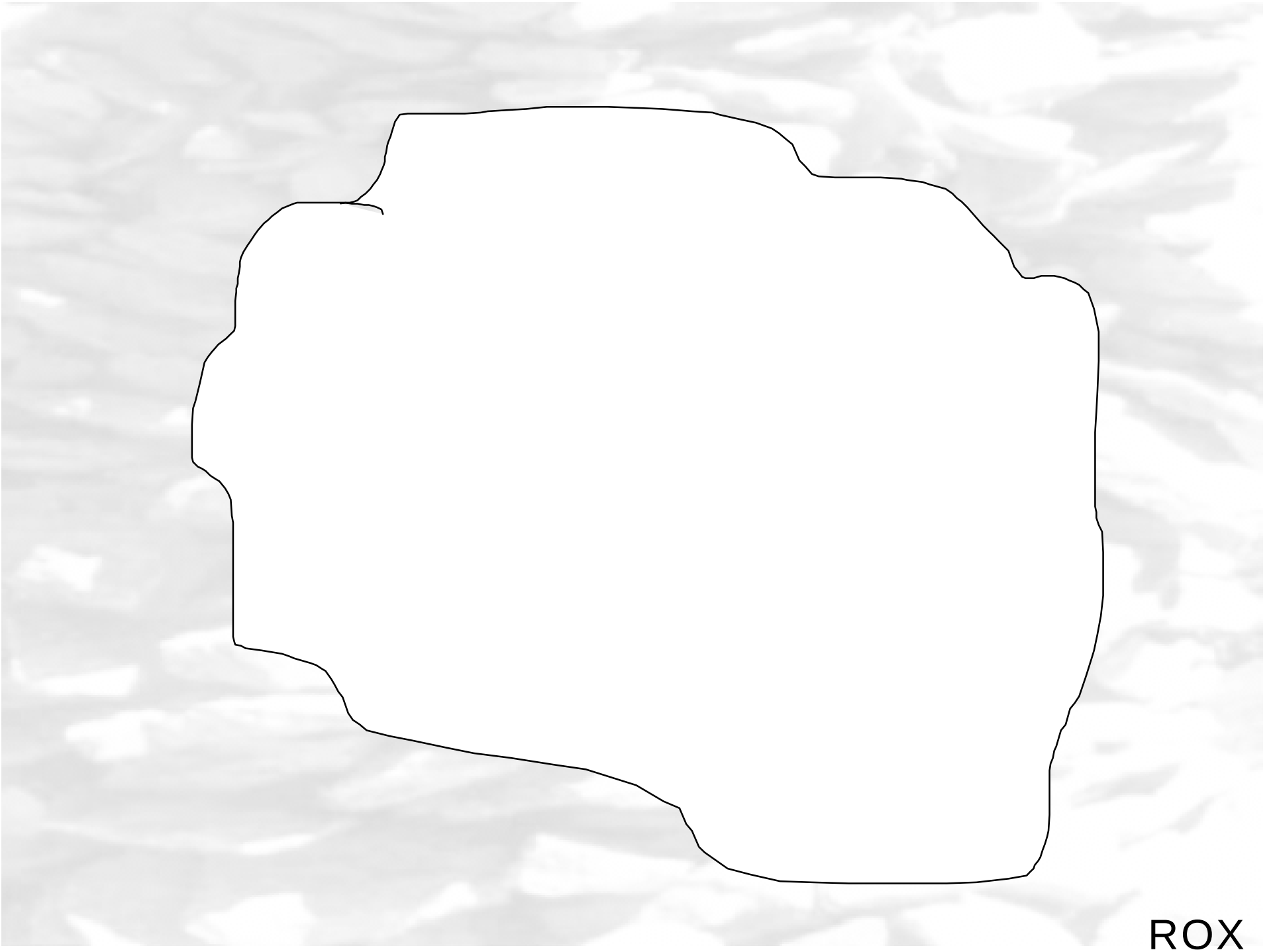
Cortland Starrett
ROX Software, Inc.

Static Population of Preexisting Instances

- Overview
- Requirements
- PIs at Analysis
- PIs at Design
- Components
- Code
- Other Stuff

Overview

- “Vacuous World” Initialization
- Procedural Initialization
- Considering Real World Systems


ROX

Procedural Initialization

(example from Elevator) Thank you, Leon.

```
// Create the Building

create object instance theBuilding of
    BLDNG;
theBuilding.Name = "Gengar";

totalFloors = 7;

// Create the Bank in the Building

create object instance theBank of BANK;
theBank.Pass_load_time = 10;
theBank.Block_clear_time = 5;
theBank.Top_floor = totalFloors;
theBank.Bottom_floor = 1;
theBank.Max_close_attempts = 10;
relate theBank to theBuilding across
    R24;
```

```
// Create the Shaft

create object instance theShaft of
    SHAFT;
theShaft.Service_direction = "up";
theShaft.In_service = TRUE;
relate theShaft to theBank across R1;

// Create the Cabin

create object instance theCabin of CAB;
theCabin.Current_floor = 1;
relate theCabin to theShaft across R2;
```

ROX

Procedural Initialization (cont)

```
// Create the Stop Window

create object instance theSwin of
    STOP_WIN;
theSwin.Nearest_upper_floor =
    totalFloors;
theSwin.Nearest_lower_floor = 1;
relate theSwin to theCabin across R22;

// Create the Door

create object instance theDoor of DOOR;
theDoor.Retries = 0;
theDoor.Open_wait_time = 0;
theDoor.Lock_enabled = FALSE;
relate theDoor to theCabin across R4;
```

```
// Create the Floors, Bank Levels and Shaft
    Levels
fn = 1;
while (fn <= totalFloors)
  // Add a Floor to the Building
  create object instance thisFloor of FLOOR;
  thisFloor.Number = fn;
  thisFloor.Name = "noname";
  relate thisFloor to theBuilding across R23;

  // Add a Shaft Level for this Floor
  create object instance thisSlev of SLEV;
  thisSlev.Floor_number = fn;
  thisSlev.Stop_requested = FALSE;
  relate thisSlev to theShaft across R28;

  // Add a Bank Level for this Floor
  create object instance thisBlev of BLEV;
  thisBlev.Floor_number = fn;
  relate thisBlev to theBank across R25;
  fn = fn + 1;
end while;
// ...
```

Real World System Consideration

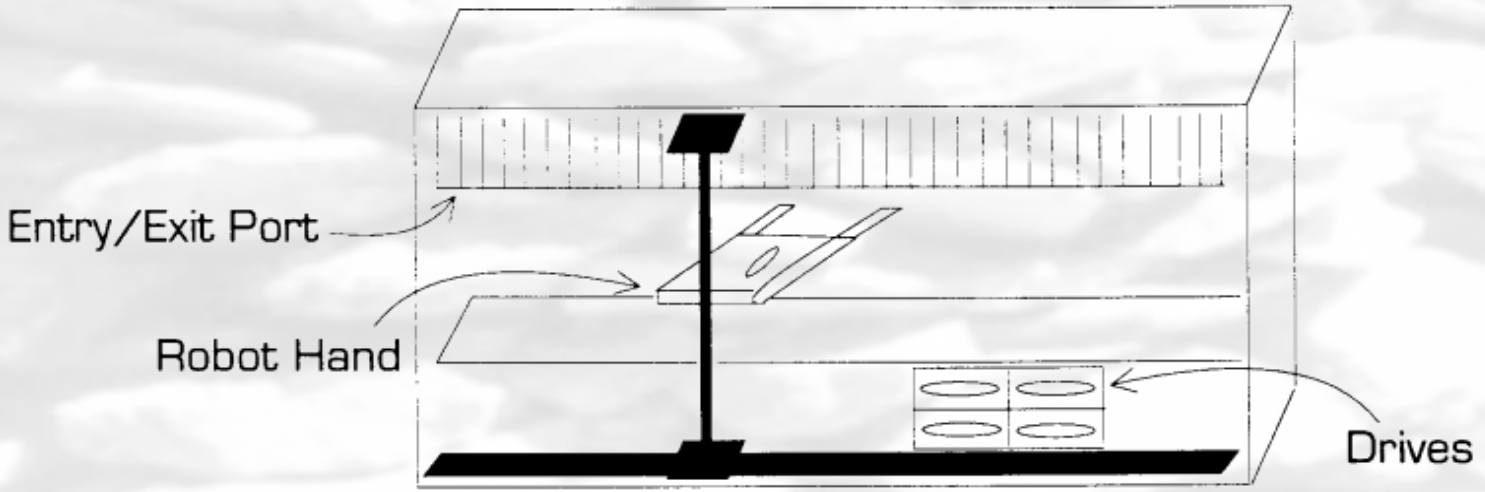


Figure 2: The Optical Disk Cabinet

Requirements:

Means of...

- ...specifying preexisting class instances.
- ...specifying the attribute values of specified preexisting class instances.
- ...specifying preexisting relationships between specified preexisting instances.
- ...statically establishing preexisting instances in a target system with little or no dynamic run-time code execution.

Extended Requirements:

- Means of specifying preexisting events!
- Means of specifying class operations!
- etc...

Preexisting Instances in Analysis

- Object Testing
- Simulation (including penny simulation)
- responsibility of System Designer/Architect

Preexisting Instances at Design

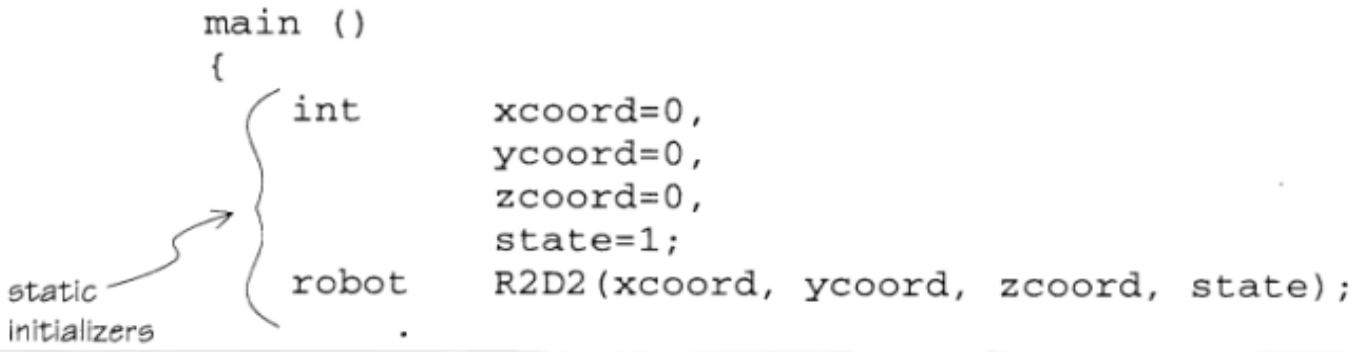
- Recursive Design Notes
- OOA of Instance Subsystem
- MC-3020 Population Process

Pre-existing Instance

A pre-existing instance is an instance that is known to exist before the system begins to execute.

If there are only a few pre-existing instances, initialize them by declaration.

Make sure the real world agrees with your declaration.



ROX

Many Pre-existing Instances

If there are many pre-existing instances,

- create a database with a schema to match the information model of this domain
- populate the database to match the initial conditions of this domain
- . . .

The main program then

- reads the pre-existing instances from the database
- invokes the appropriate `Create` operation.

Application Classes

Create instances of application classes by:

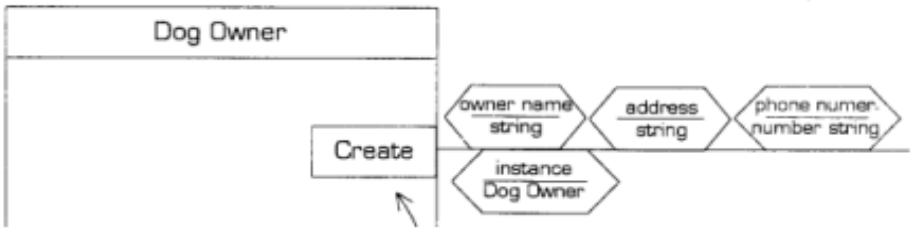
- reading in an external file database that specifies the instances to create,

or

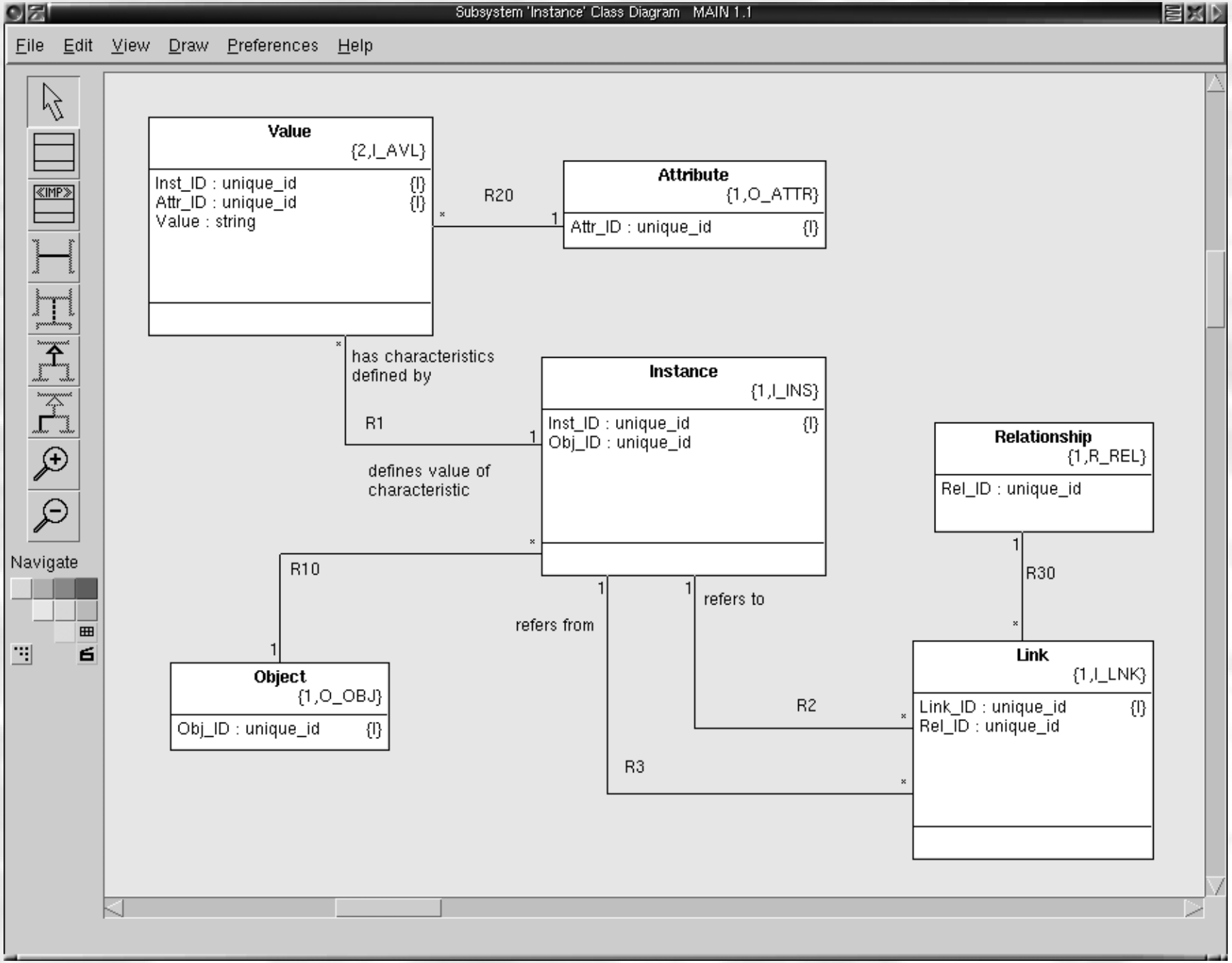
- calling a `create` operation in the initialization code,

or

- compile-time creation of instances using static initializers.



ROX


Subsystem 'Instance' Class Diagram MAIN 1.1
File Edit View Draw Preferences Help
Navigate
Value
{2,I_AVL}
Inst_ID : unique_id {I}
Attr_ID : unique_id {I}
Value : string
R20
Attribute
{1,O_ATTR}
Attr_ID : unique_id {I}
has characteristics defined by
R1
defines value of characteristic
Instance
{1,I_INS}
Inst_ID : unique_id {I}
Obj_ID : unique_id
Relationship
{1,R_REL}
Rel_ID : unique_id
R10
R30
refers from
refers to
Object
{1,O_OBJ}
Obj_ID : unique_id {I}
Link
{1,I_LNK}
Link_ID : unique_id {I}
Rel_ID : unique_id
R2
R3

ROX

SIP Work Flow….

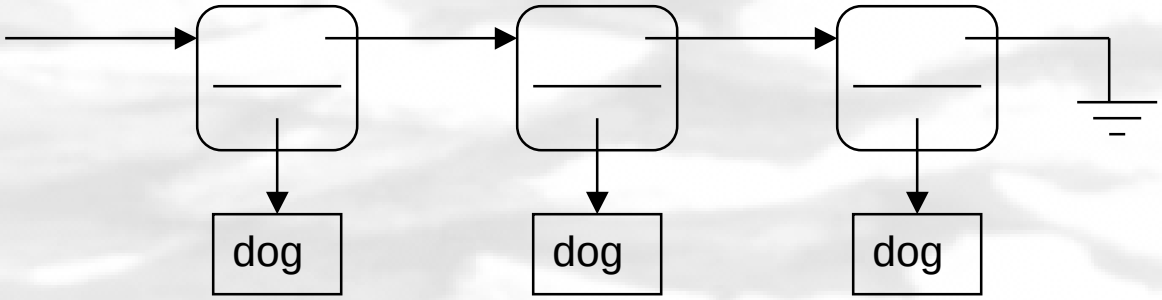
Components

- Containers
- Instances
- Attribute Values
- Links

Containers



Container Object (“containoid”)



Instances

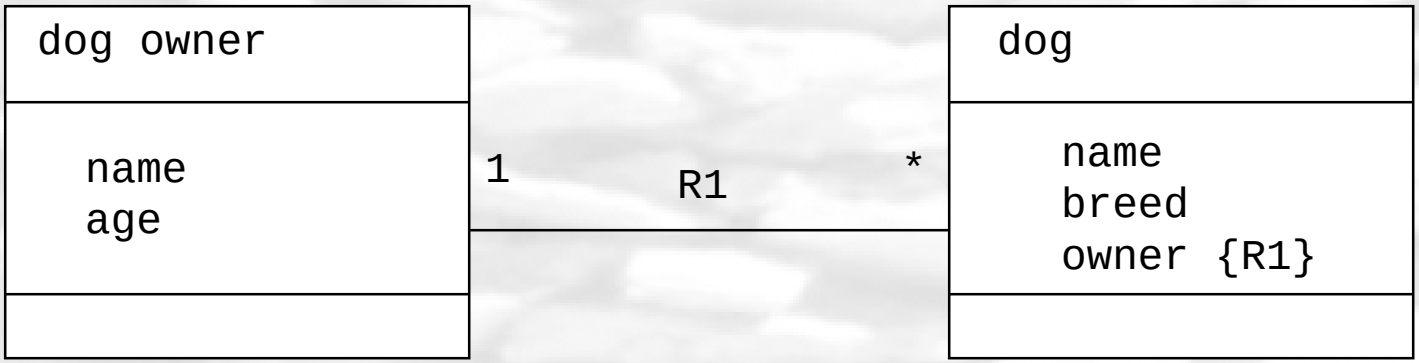



```
DOG_s DOG_ObjectPool_s[ DOG_MAX_EXTENT_SIZE ];

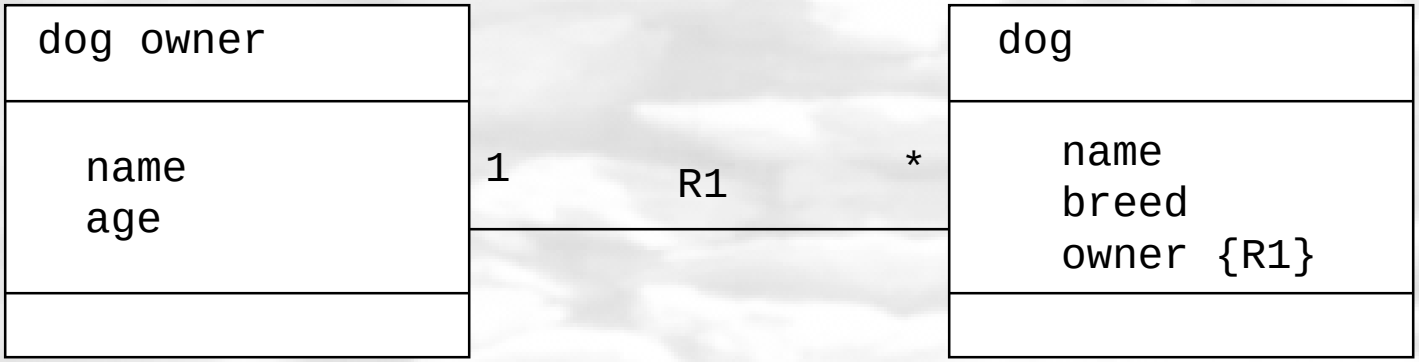
Escher_ObjectSet_s * p_DOG_extent;

static Escher_ObjectSet_s * p_DOG_inanimate;
```

Statically populated instances occupy the front of the array. The end of the array represents the “free list”.

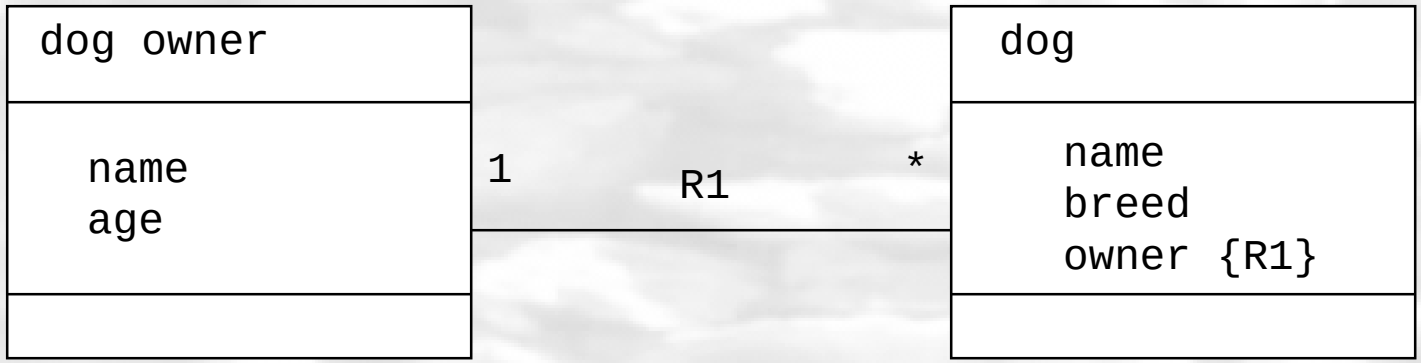


ROX



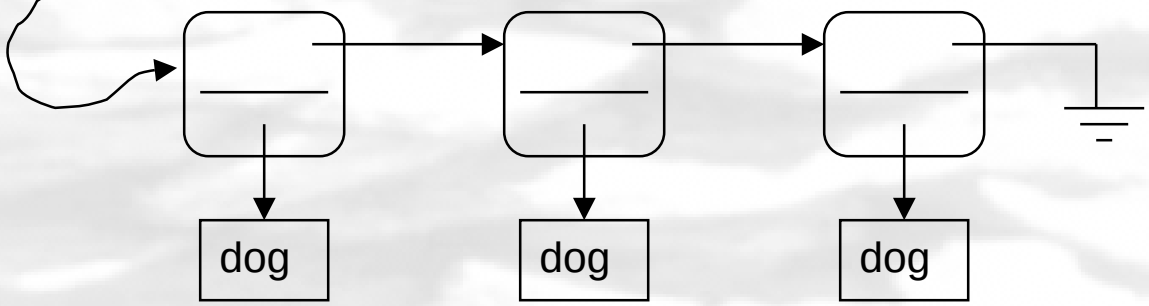

```
struct dog_owner_s {
  char name[ MAX_LEN ];
  int age;
  set_t * dog;
} dog_owner_t;
```

```
struct dog_s {
  char name[ MAX_LEN ];
  breed_t breed;
  /* owner OPTIMIZED OUT */
  owner_t * owner;
}
```



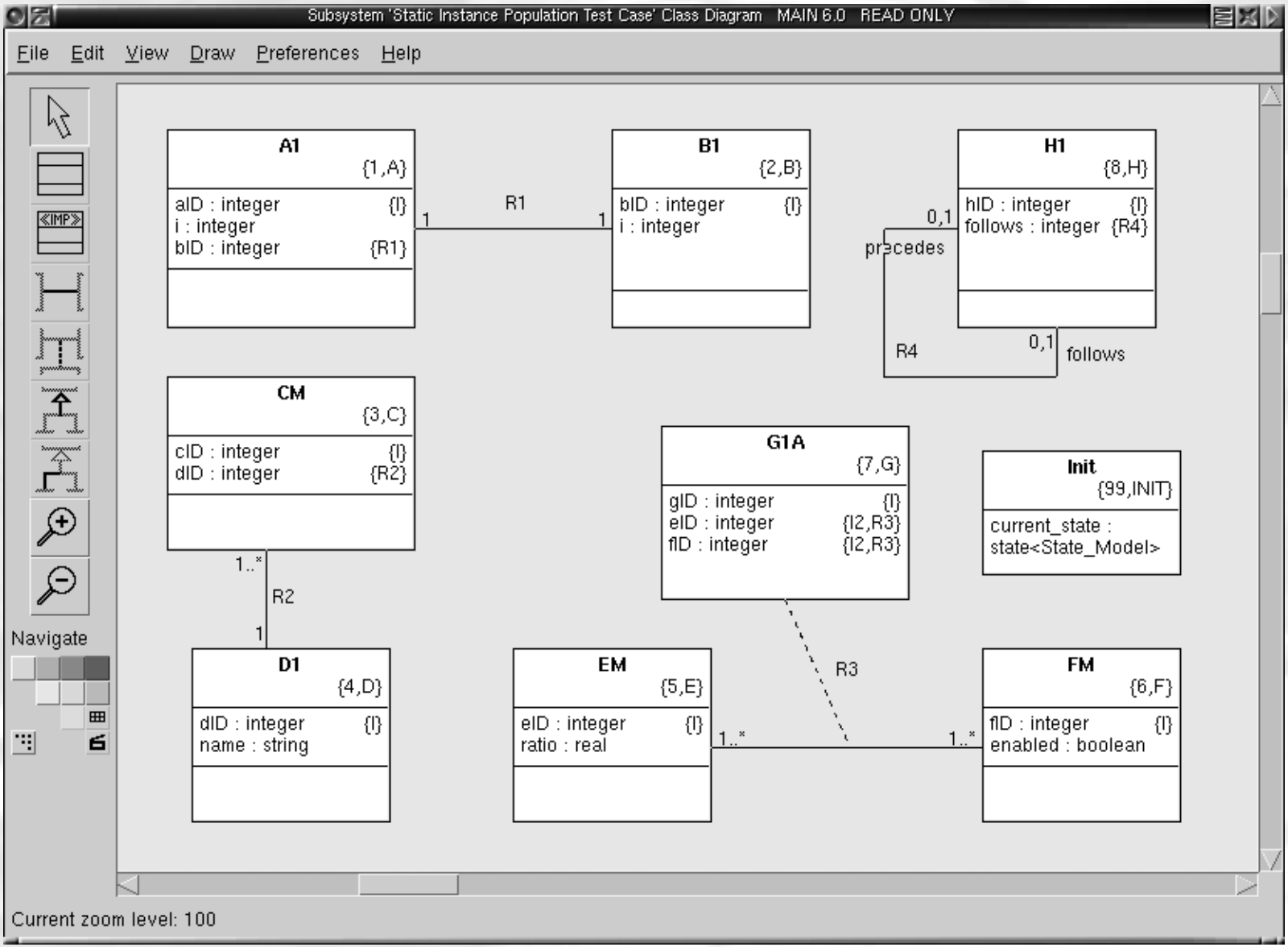

```
struct dog_owner_s {
  char name[ MAX_LEN ];
  int age;
  set_t * dog;
} dog_owner_t;
```

```
struct dog_s {
  char name[ MAX_LEN ];
  breed_t breed;
  /* owner OPTIMIZED OUT */
  owner_t * owner;
}
```



Code

- 1:1
- 1:1 reflexive
- 1:M


Subsystem 'Static Instance Population Test Case' Class Diagram MAIN 6.0 READ ONLY
File Edit View Draw Preferences Help
Navigate
A1
{1,A}
aID : integer {I}
i : integer
bID : integer {R1}
1
R1
1
B1
{2,B}
bID : integer {I}
i : integer
H1
{8,H}
hID : integer {I}
follows : integer {R4}
0,1
precedes
R4
0,1
follows
CM
{3,C}
cID : integer {I}
dID : integer {R2}
1..*
R2
1
D1
{4,D}
dID : integer {I}
name : string
G1A
{7,G}
gID : integer {I}
eID : integer {I2,R3}
fID : integer {I2,R3}
R3
Init
{99,INIT}
current_state :
state<State_Model>
EM
{5,E}
eID : integer {I}
ratio : real
1..*
1..*
FM
{6,F}
fID : integer {I}
enabled : boolean
Current zoom level: 100

ROX

A 1:1 B

```
A_t A[ MAX_LEN ] =
{
  { 1, 37, 0, &B[3] }
};
```

```
B_t B[ MAX_LEN ] =
{
  { 0, 8, &A[0] }
};
```

1:1 reflexive

```
H_t H[ MAX_LEN ] =
{
  { 0, 0, 0, &H[1] },
  { 1, 0, &H[0], &H[2] },
  { 2, 1, &H[1], 0 }
};
```

D 1:M C

```
D_t D[ MAX_LEN ] =
{
  { 7, “Fred”, &Cset[0] }
};

set_t Cset[] =
{
  { &Cset[1], &C[0] },
  { &Cset[2], &C[1] },
  { &Cset[3], &C[2] },
  { 0, &C[3] }
};
```

```
C_t C[ MAX_LEN ] =
{
  { 111, 7, &D[0] },
  { 222, 7, &D[0] },
  { 333, 7, &D[0] },
  { 444, 7, &D[0] }
};
```

Other Stuff

- XML
- RD Ideas
- other approaches

XML

```
<?xml version="1.0" ?>
<system> <content>
<domain dom_id=’329755456’ name=’sip1’ full_der=’0’ config_id=’268435457’ >
  <ids obj_id1=’329755456’ />
  <descrip name=’Descrip’></descrip>
  <data-type dt_id=’268959758’ dom_id=’329755456’ name=’date’ >
    <ids obj_id1=’268959758’ obj_id2=’268959758.329755456’ />
    <rels rel14=’329755456’ />
    <descrip name=’Descrip’></descrip>
    <user-data-type dt_id=’268959758’ cdt_dt=’268959756’ gen_type=’1’ >
      <ids obj_id1=’268959758’ />
      <rels rel17=’268959758’ rel18=’268959756’ />
    </user-data-type>
  </data-type>
  <data-type dt_id=’268959759’ dom_id=’329755456’ name=’timestamp’ >
    <ids obj_id1=’268959759’ obj_id2=’268959759.329755456’ />
    <rels rel14=’329755456’ />
    <descrip name=’Descrip’></descrip>
    <user-data-type dt_id=’268959759’ cdt_dt=’268959756’ gen_type=’2’ >
      <ids obj_id1=’268959759’ />
      <rels rel17=’268959759’ rel18=’268959756’ />
    </user-data-type>
  </data-type>
  ...
```