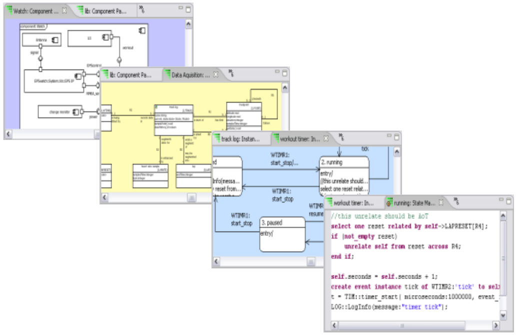
The xtUML method – Specifying Activities

- **Analysis** – questioning, thinking, sketching...
  - Descriptive UML diagrams
    - use case, sequence, ...
- **Executable Modeling** – formalizing the analysis:
  - Component Diagrams (partitioning/interfaces)
  - Class Diagrams (data)
  - State Machines (control)
  - **Activities (processing)**
- **Verification**
  - Interpretive Model Execution
- **Code generation**
  - Template and Rule-Based Translation



```
//this unrelate should be AoT
select one reset related by self->LAPRESET[R4];
if (not_empty reset)
    unrelate self from reset across R4;
end if;

self.seconds = self.seconds + 1;
create event instance tick of WTIMR2:'tick' to sel
t = TIM::timer_start( microseconds:1000000, event_
LOG::LogInfo(message:"timer tick");
```

Activities

- An activity specifies processing within the model
- An action can be associated with the following modeled elements:
  - states
  - bridge operations
  - functions
  - class and instance-based operations
  - mathematically-derived attributes
  - interface reference operations and signals
- The Object Action Language (OAL) is used to define the semantics for the processing that occurs in an action.

Object Action Language [OAL]

- Since 2001, the UML standard has incorporated a defined action semantics... but has not yet defined a syntax for specifying actions.

- Object Action Language is a concrete syntax which implements the UML standard

- OAL is complete enough to be executable, but abstract enough that it does not prescribe implementation specifics.

```
create object instance request of REQ;

select one channel related by device->CHAN[R100];

assign device.priority = lastpriority + 1;

generate CHAN11:'host relinquish' to channel;
```

What OAL can do:

- Create and delete instances.
- Link and unlink associations between instances.
- Select instances across association links.
- Select instances based on attribute values.
- Read and write attribute values.
- Compute new values.
- Control statements.
- Generate events.
- Invoke interface operations.

Data Types

- **Implicit Typing**
  - All data items are implicitly typed by the value assigned to them on their first use within an action.
- **Simple Data Types**
  - Integer
  - Real
  - String
  - Boolean
- **System Data Types**
  - Date
  - Timestamp
  - Unique ID
- **Reference Types**
  - Timer Handle
  - Instance Handle
  - Instance Handle Set
  - Event Instance
  - Component Handle

Operators

- **Arithmetic**
  - + - * / %
  - Unary -
- **Boolean**
  - and or
  - Unary not
- **Logical**
  - == !=
  - < <= > >=
- **Assignment**
  - assign x = 1;
  - Assign keyword optional
- **Instance Handles**
  - == !=
  - empty not_empty
  - cardinality

e.g.

```
expired = (account.balance == 0.00) and
((TIM::get_current_time() - last_pay_time)  >=max_wait) ;
```

Expressions

```
a = 3 ;                    /* integer typed local variable */
assign x = 3.14 ;          /* floating point value (real) */
y = 11.0 ;                 /* another real */
done = false;              // boolean typed local variable
z = x + y * x;             /* Operator Precedence */
b = a % 2;                 /* remainder operator */
s1 = “Hello”;              /* String Variable – dynamic size */
s2 = “World!”;             //  C++ Comments also allowed
s3 = s1 + “   “ + s2;      // String Concatenation
```

Lab 1: Exercise 5

- **Run the model in the xtUML Debugging Perspective**

IF Statement

- No semicolon after the IF statement
- As many ELIF clauses as desired
- Nested IF statements allowed, END IF; terminates statement.

```
if (<Boolean or Logical equation>)
    // do something
elif (<Boolean or Logical equation>)
    // do something
else
    // or something
end if;
```

```
if (empty firstPoint)
  // this is the first trackPoint in the log
  relate self to trackPoint across R1.'has
first';
  relate self to trackPoint across R3.'has last';
else
  unrelate self from lastPoint across R3.'has
last';
  relate self to trackPoint across R3.'has last';
  relate lastPoint to trackPoint across
R2.'follows';
end if;
```

Loops

- WHILE and FOR EACH. Use WHILE to implement a FOR loop.
- Can be nested.
- Defines a local scope.

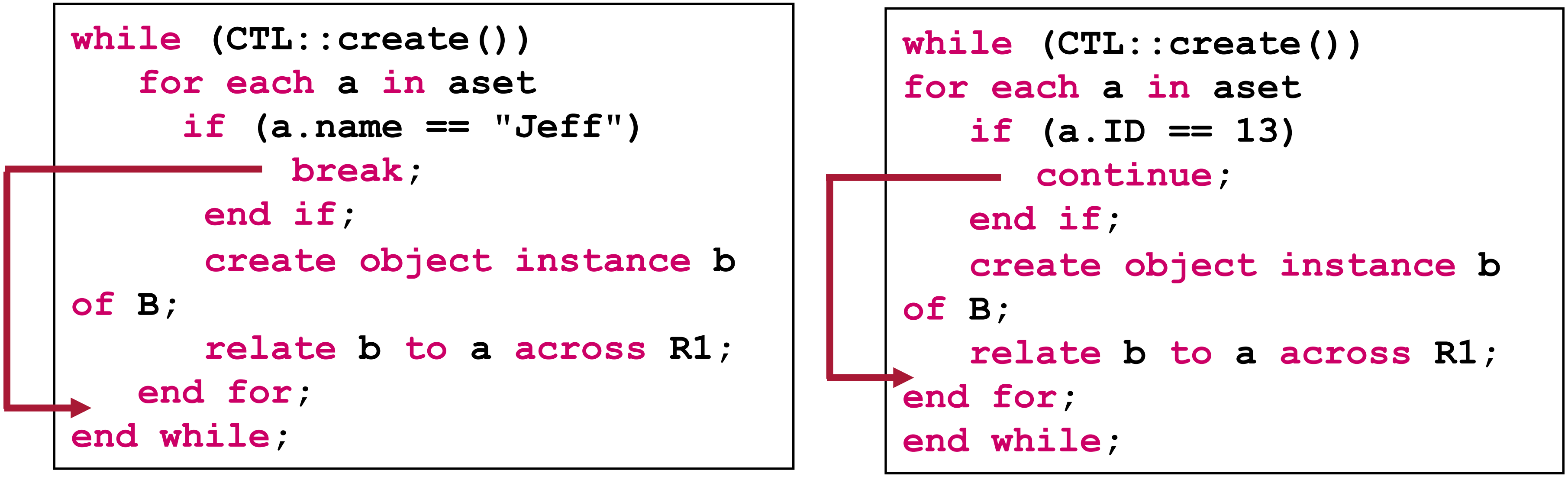
```
for each mobile in mobiles
       // do something
end for;

i = 0;
while (i < 4)
       // do something
       i = i + 1;
end while;
```

Nesting

```
for each this_Cabin in bank_Cabins
   select one its_Shaft related by this_Cabin->Shaft[R2];
   if (its_Shaft.In_service)
      cab_delay =
this_Cabin.Estimate_travel_delay(Floor:my_Floor.Name,
        Calling_dir:param.Dir);
      if ((cab_delay < shortest_delay) or (first_cabin))
         shortest_delay = cab_delay;
         param.OUT_Shaft = its_Shaft.ID;
      end if;
   end if; // in service
   first_cabin = false;
end for;
```

Break and Continue

- Break completely exits the inner-most loop
- Continue exits the current iteration of the inner-most loop



```
while (CTL::create())
   for each a in aset
     if (a.name == "Jeff")
          break;
      end if;
      create object instance b
of B;
      relate b to a across R1;
   end for;
end while;
```

```
while (CTL::create())
for each a in aset
   if (a.ID == 13)
      continue;
   end if;
   create object instance b
of B;
   relate b to a across R1;
end for;
end while;
```

Functions

- ## Function Invocation

```
::fnName(ParamName1:ParamValue1, …);
::start();
probe = ::getProbe(probeId: p);
```

- ## Return value

```
return <expression>;  // <expression> is optional
return “down”;
```

- ## Accessing Parameters
  - *param* is a pre-pended keyword to access function arguments

```
select any probe from instances of SP where
      selected.probe_ID == param.probe_id;
trackPoint.latitude  = param.location.latitude;
```

Attributes

- **Writing Attributes**
  - ***[assign] <instance handle>.<attribute> = <expression>;***
  - ***assign*** **keyword is optional**

```
create object instance my_account of ACCT;
my_account.branch = rcvd_evt.this_branch;
```

- **Reading Attributes**

```
myx = myrobot.x_position;
```

- **Writing Mathematically Derived Attributes**
  - **In Model Explorer, set as derived attribute**
  - **Then select and Open With > Activity Editor**

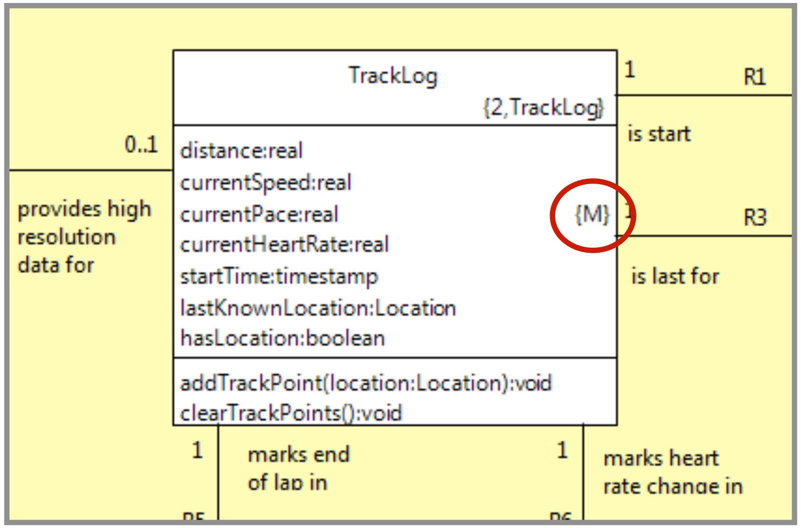
```
self.volume =
self.length*self.width*self.height;
```

Mathematically Derived Attributes

- **Writing Mathematically Derived Attributes**
  - **In Model Explorer, set as derived attribute**
  - **Then select and Open With > Activity Editor**

```
self.volume = self.length * self.width * self.height;
```

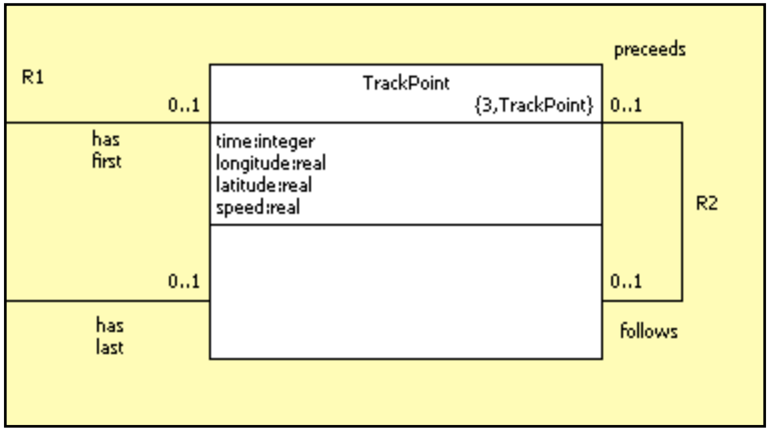
  - **No return statement required**
  - **Access attribute via self**
  - **Mathematically derived attributes are read-only in all other places**



Create / delete statement

Syntax:

create object instance <instance handle> of <keyletter>;
create object instance of <keyletter>;
delete object instance <instance handle>;



```
create object instance trackPoint of TrackPoint;

delete object instance trackPoint;
```

Relate / unrelate statement

- OAL is used to manage relationships between specific instances of classes.

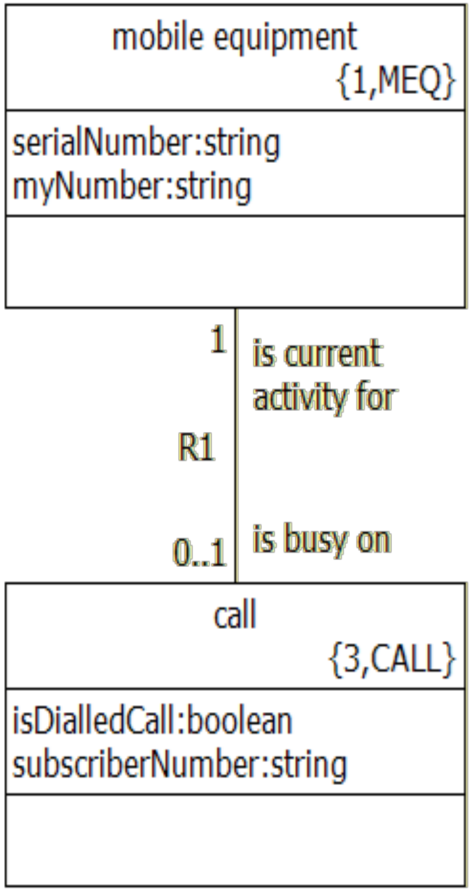
```
relate mobile to call across R1;
```

Local instance reference variable

Association label

```
unrelate mobile from call across R1;
```

Local instance reference variable



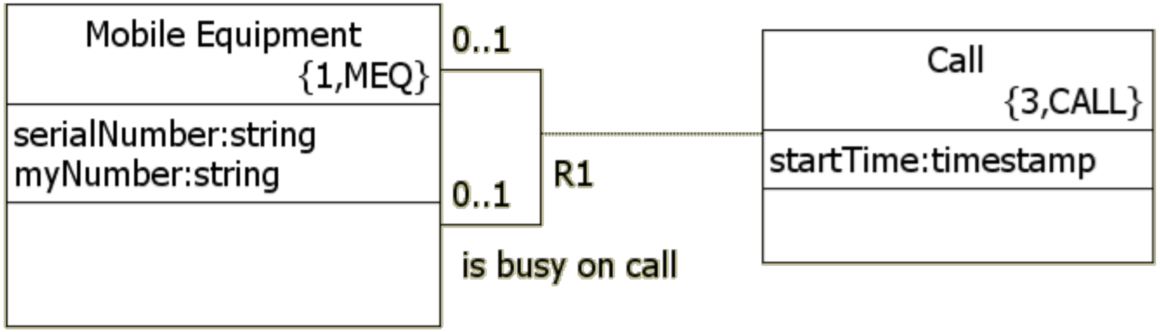
Relate / unrelate “using“ statement

- Connecting two classes that have an associative class stemming from their relationship.

```
relate mobile1 to mobile2
        across R1.’is busy on call’ using call;
```

Local instance reference variable

```
unrelate mobile1 from mobile2
        across R1.’is busy on call’ using call;
```



Select any / many

- **Selecting instances of a class**

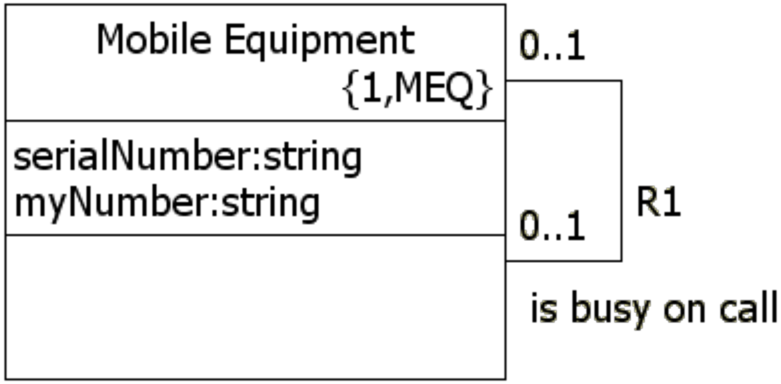
```
select any mobile from instances of MEQ;
```

Local instance reference variable

Key letters

```
select many mobiles from instances of MEQ
      where selected.serialNumber > 10000;
```

Where clause



Select one / many … related by …

- **Select one requires the use of the related by clause**
- **‘Self’ is the instance of the class that originates an action**

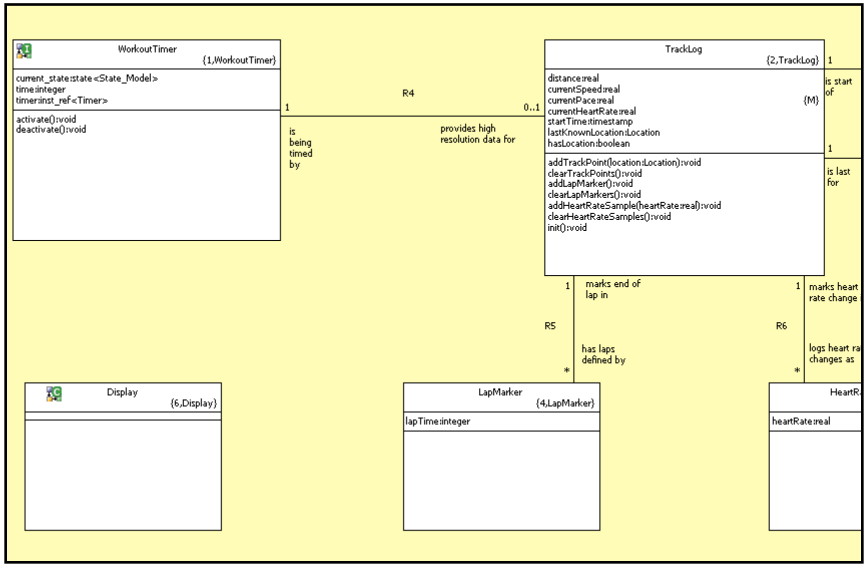
```
select one timer related by self->WorkoutTimer[R4];
```

Local instance reference variable

Originating class instance

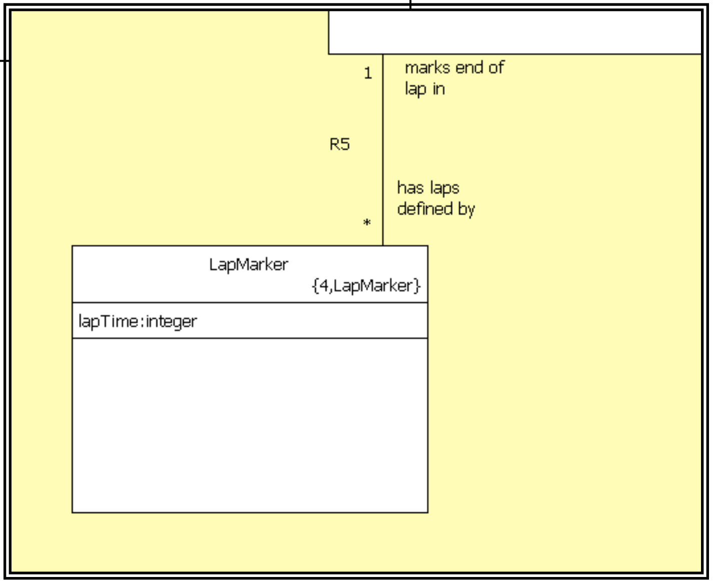
Key letters

Association label



Example: Lap Time

```
select one timer related by self->WorkoutTimer[R4];
create object instance lapMarker of LapMarker;
lapMarker.lapTime = timer.time;
relate self to lapMarker across R5;
```



```
select many lapMarkers related by self->LapMarker[R5];
for each lapMarker in lapMarkers
  unrelate self from lapMarker across R5;
  delete object instance lapMarker;
end for;
```

Lab 2: Exercise 1

- **Create a class diagram for the Tracking subsystem in the GPS Watch**

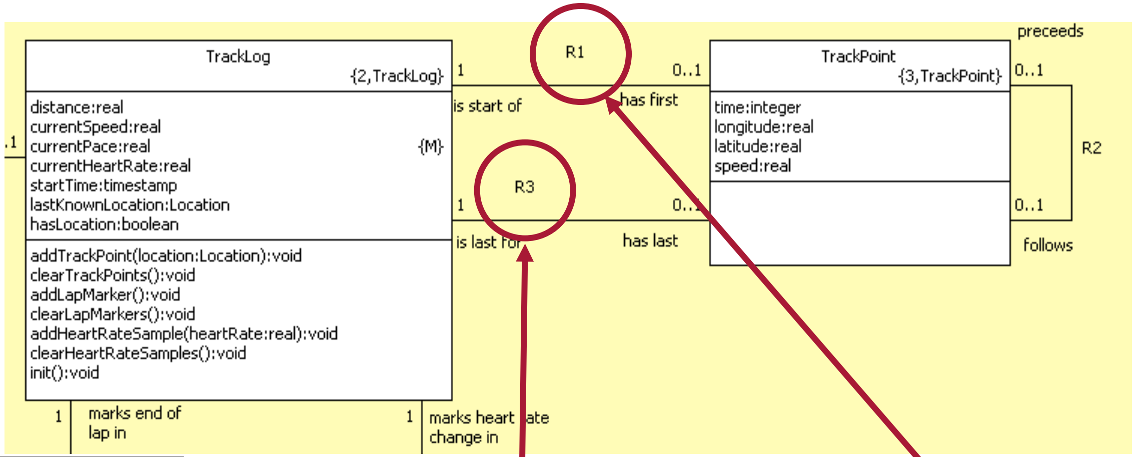
Control Structures

- **Example:**

```
// Send a 'time for bed' event to all children 5 and under.
select many children from instances of C;
for each child in children
  if (child.age <= 5)
      while (child.awake)
          generate C1:'time for bed' () to child;
          if (not lights.out)
              generate C2:'turn off lights' () to child;
          end if;
        end while;
    end if;
end for;
```

Example: Creating an Ordered List

- At each TrackPoint update, this operation is run on an instance of TrackLog



```
create object instance trackPoint of TrackPoint;
trackPoint.time        = workoutTimer.time;
trackPoint.longitude = param.location.longitude;
trackPoint.latitude    = param.location.latitude;

select one firstPoint related by self->TrackPoint[R1];
select one lastPoint related by self->TrackPoint[R3];

if (empty firstPoint)
  // this is the first trackPoint in the log
  relate self to trackPoint across R1.'has first';
  relate self to trackPoint across R3.'has last';
else
  unrelate self from lastPoint across R3.'has last';
  relate self to trackPoint across R3.'has last';
  relate lastPoint to trackPoint across R2.'follows';
end if;
```

Select head of list

Select tail of list

Handle first point

Update new last point
Relations

Lab 3: Exercise 1

- **Relate and unrelate class instances using OAL**