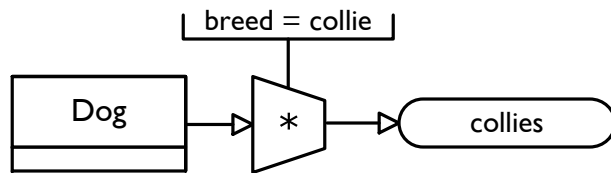


Scrall

Starr's Concise Relational Action Language

Leon Starr



August 5, 2003

Version 1.0
mint.scrall.tn.1



MODEL INTEGRATION, LLC.

CONFIDENTIAL

Introducing Scroll

Scroll is yet another action language at the same semantic level as other relational action languages such as SMALL, TALL, OAL and ASL. All of these languages are platform independent, object-oriented methods of specifying relational data access, computation, data processing and low level branching decisions. Most of Scroll is inspired by Steve Mellor's SMALL language. My language, however, is bigger.

What's different about Scroll?

Scroll is primarily a graphical language. Symbols are networked together with object, data and control flows.

Why graphical?

One of the problems with procedural style text languages is that potentially parallel actions are not clearly identified. In a data flow type diagram, parallel activity is obvious, so arbitrary sequencing can be completely avoided. The model translation process has the opportunity to take advantage of any parallelism to yield more efficient code.

Text languages are inherently 1D while networked graphical symbols are 2D. Data flow relationships in particular are easier to see in a graphical presentation.

Flows replace the need for creating, naming and scoping temporary variables.

A fresh approach to activity expression may prevent new developers from perpetuating inappropriate procedural and implementation habits.

But sometimes text is the best way to express an idea concisely and rapidly...

Scroll is a primarily, but not exclusively a graphical language.

Tray expressions make it possible to write comparison, selection, computation and navigation expressions in conventional text.

Blocks could be entirely written in a text based action language such as SMALL or OAL and integrated directly into a Scroll diagram.

So Scroll gives you the best of both an exclusively text and a hybrid graphic approach to expressing actions.

It will be possible to translate Scroll symbols into equivalent OAL or SMALLish text and vice versa.

Is Scroll UML compatible?

Scroll is fully compatible with UML 2.0 action semantics. It is slightly consistent with UML 2.0 activity and action symbols. Unfortunately, the current OMG standard is not sufficiently expressive in the area of relational data access. An attempt will be made to adjust some of the symbols and notation of Scroll so that it fits in as much as possible with standard UML notation. At the very least it should be possible to convert Scroll diagrams to fully compatible, though not as concise UML activity/action diagrams.

Scroll Concepts

Leon Starr
mint.scroll.tn.l



Contents

- Scroll Principles & Quick Syntax Note
- Storing Data
- Flowing Data
- Selecting Objects
- Finding Objects
- Creating and Deleting Objects
- Linking and Unlinking Objects
- Creating, Deleting and Reclassifying Specialized Objects
- Performing Set Actions on Objects
- Reading and Writing Attributes
- Connecting Data Flows
- Computing and Comparing Data
- Converting Data
- Branching on Conditions
- Explicit Sequencing
- Transmitting Signals
- Defining and Invoking Class Operations
- Defining and Invoking Services
- Functions
- Blocks
- Execution Rules
- UML Compliance
- Summary of Scroll Symbols

Scroll Concepts



Scroll Principles

I have attempted to adhere to the following principles in designing the Scroll symbology.

Symbol Design

Use symbols that can be drawn quickly and easily by hand. Any symbol that I found awkward to draw on unambiguously on a sketchpad or whiteboard was rejected.

Shape suggests function. The finder symbol is an arrow. The selector symbol looks a little like a funnel since it narrows the selection. This principle makes it easier for the developer to recall the proper shape while drawing on a whiteboard.

All sides of a symbol should be connectable. This makes it easier to find convenient anchor faces for connectors.

Use text where compact linear expressions are most descriptive. Use graphics where the focus should be on the flow of data and control. This keeps the diagrams neat, concise and helpful. A purely graphical action language would annoy the developer with endless flows and interconnections where a simple math equation would be easier to read and write. Not to mention taking up a lot less space.

Minimize the number of symbol types. The real power of any good language is in the ability to interconnect simple building blocks in many powerful ways.

Discarding Procedural Artifact

Discourage arbitrary sequencing of actions.

Discourage the use of temporary variables and object stores. In text only action language temporary variables and awkward scoping rules are necessary since the same data must be referenced at disparate locations throughout a one dimensional listing.

Discourage iteration. Most iteration in text action languages apply an action to a collection of objects. This is more concisely represented by a flow of multiple objects directly into an action.

Leverage the Relational Capabilities

At the model level data is organized relationally. The action language should maximize the power of this paradigm rather than suppress it in favor of object-oriented programming level semantics. Emphasis then is placed on powerful set manipulation features.

Execution

Keep execution rules as simple as possible. An action executes when all of its data and control inputs are available. That's all there is to it.

Scroll Concepts

Leon Starr
mint.scroll.tn.l



Quick Syntax Note

Syntax markers are light blue so that they won't be confused with the Scrall symbols.



Square braces indicate an optional item.



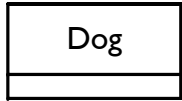
Squiggly brackets indicate multiple items are permitted.

Side comments, observations and unresolved issues are in purple text.



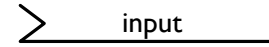
Storing Data

Class

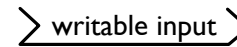


An abbreviated form of the symbol that represents a class on a UML class diagram is used on the AFD. Since only the name of the class is necessary, the other compartments are suggested by the lower horizontal segment. But this segment is not a compartment - just part of the symbol design.

Input Parameters



Passed in by transition event or operation parameter.



Operation parameter passed by reference - so the operation can change the input value.

Data from an input is accessed in a text expression as <in.input_name>. So an input labeled "speed" could be referenced as "in.speed".

Class operation return values are covered under "Connecting Data Flows" later in this document.

Object

Single object store

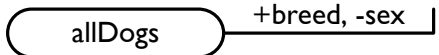


Object references are represented by single or multiple object stores. A rectangle suggests a single object whereas a puffed out rectangle suggests multiple objects.

Multiple object store



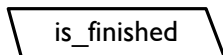
Ordered object store



+breed, -sex
+/- is ascending/
descending order

The contents of an object set may be ordered using one or more sorting keys. A half tray attached to an object set symbol holds the sorting keys. Each sorting key is an attribute of the object's class prefixed as ascending (+) or descending (-). Each key is separated by a comma. Ordering is done on each key proceeding left to right. An attribute may be used as a sorting key only if order is defined on the attribute's datatype.

Data



Intermediate values, of any type other than "object", are represented by a transient data store. The content of all data stores is undefined when the enclosing activity has completed.

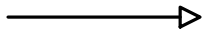
Scroll Concepts



Flowing Data

Data, objects and control flows along the following arrow symbols.

Object flow



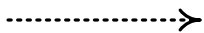
Objects selected from a class or subselected downstream or output from set operations travel as object references along an object flow.

Data flow



Explicit values, temporary variables and attribute values travel along a data flow.

Control flow

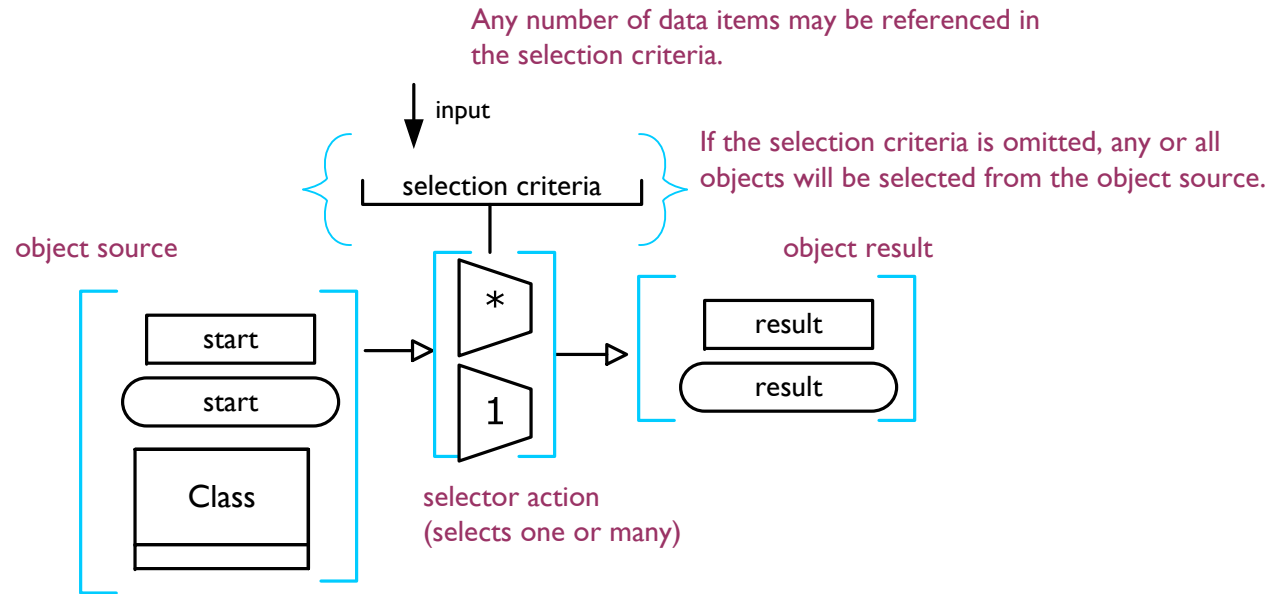


The control flow represents conditional branches and triggering of further activity.



Selecting Objects

A selector takes a class or an object store as input, applies selection and multiplicity criteria and then produces as output zero or more objects satisfying the criteria.



Selection Criteria Examples

`breed = collie`

The attr is an attribute of the source class. Relop is a relation operator such as =, <, > etc. defined for the attribute's datatype. The expr is an expression that yields a value of the same datatype as the attr.

`(breed = collie) and (age > 5)`

The selection criteria can be as complex as necessary.

Any number of data items may contribute to the selection criteria. An arrow is drawn for each input referenced within the expression.

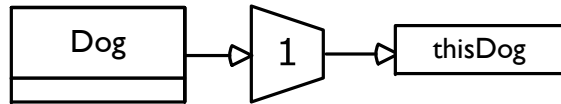
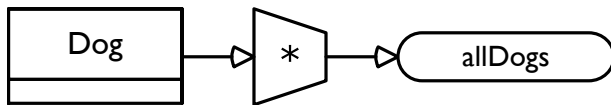


Scroll Concepts

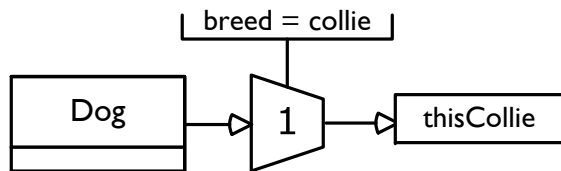
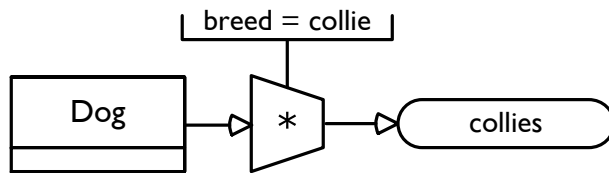


Object Selection Examples

Selection - no criteria

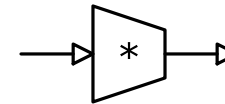


Selection - with criteria

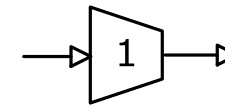


Free selectors

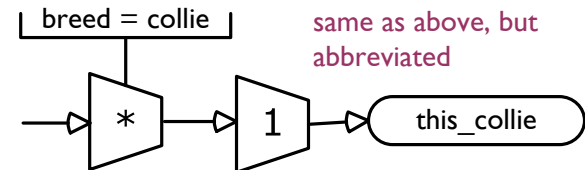
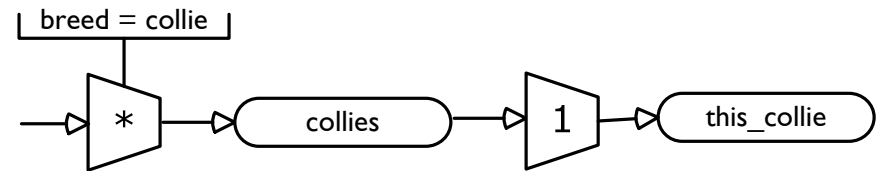
Select all



Select any



Subselection

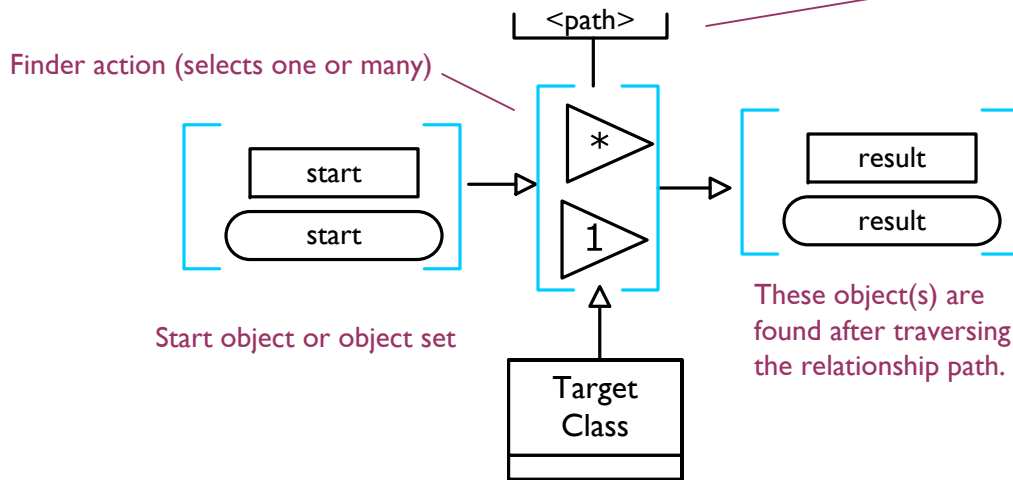


Scroll Concepts



Finding Objects

One or more objects may be found by tracing a route through a relationship path. The starting point is a single or multiple object set. Links along the relationship path are followed in sequence to produce zero, one or more related objects.



Results are found in the class

Relationship Path Syntax

<path>

```

path : starting traversal_list
traversal_list : traversal | traversal_list traversal
traversal : "->" relation_term
starting : relation_term
relation_term : rname | rname cname | rname phrase cname
rname : R[digit]+
cname : NAME
phrase : "/" STRING "/"
    
```

Relationship Path Examples

R22 Traverse from the start object(s) across R22 to the target class.

R7_Cabin->R10_Shaft->R21

Traverse from the start object(s) to Cabin via R7 to Shaft via R10 to the target class via R21.

R27/is executed before/

The phrase on either side of the association can be inserted for readability or to remove ambiguity on a reflexive access. The / symbol is easier to write by hand and looks better in squashed text (no whitespace) than quotation marks.

Note: The "->" symbol is chosen because it is easily drawn as an arrow by hand. That's also why it's a good idea to avoid the : symbol.

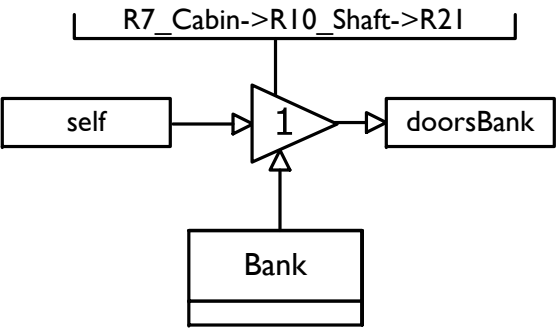
Scroll Concepts

Leon Starr
mint.scroll.tn.l

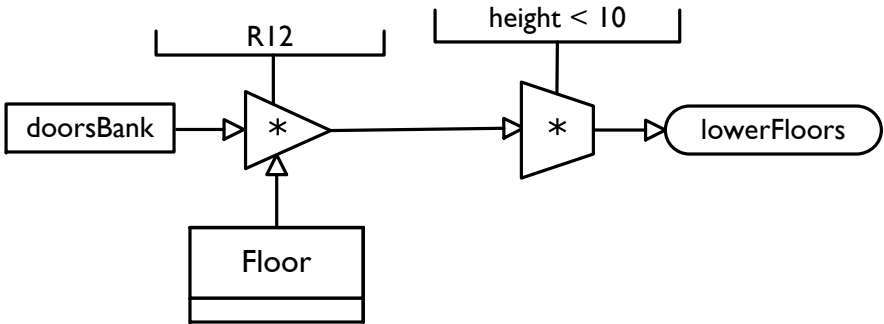


Finding Objects - Examples

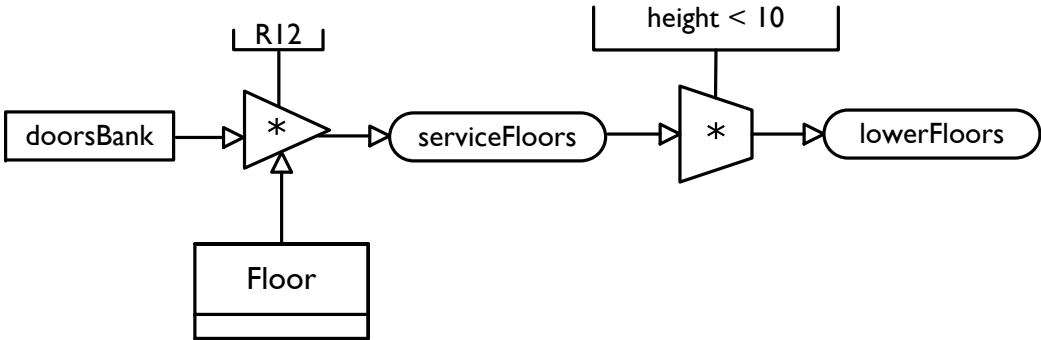
Navigating with no selection criteria



Selective navigation - the intermediate object set can be omitted since the data flow is implicitly typed as an object set.



Selective navigation (with criteria)



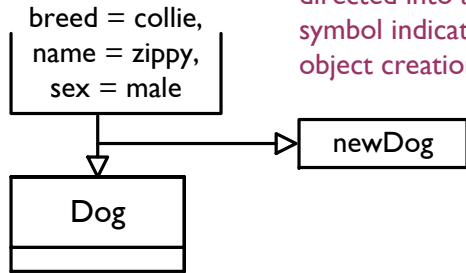
Scroll Concepts



Creating and Deleting Objects

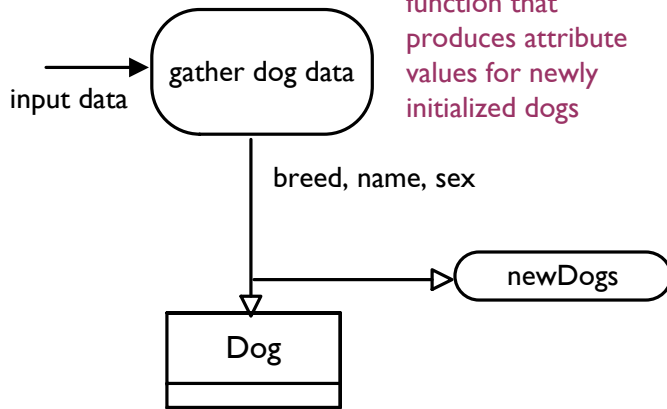
Creating

Creating an object



An object flow directed into a class symbol indicates object creation.

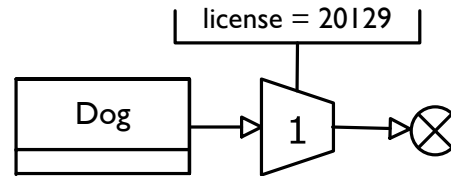
Creating many objects



function that produces attribute values for newly initialized dogs

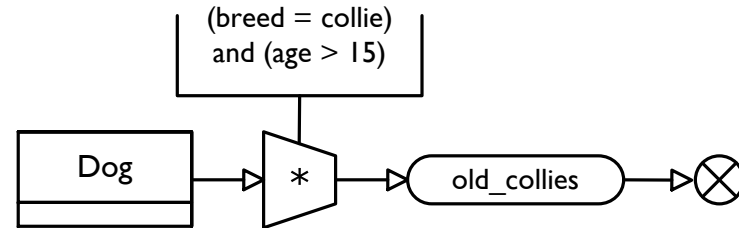
Deleting

Deleting a specific object

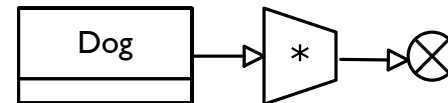
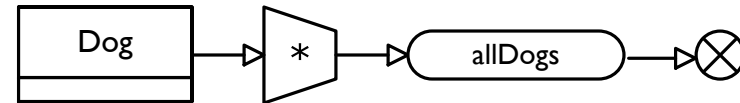


deletion action symbol

Delete selected objects



Deleting all objects



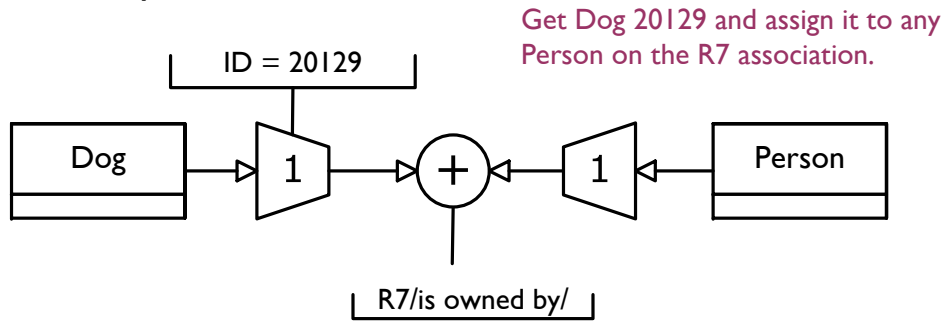
the intermediate object set is optional

Scroll Concepts

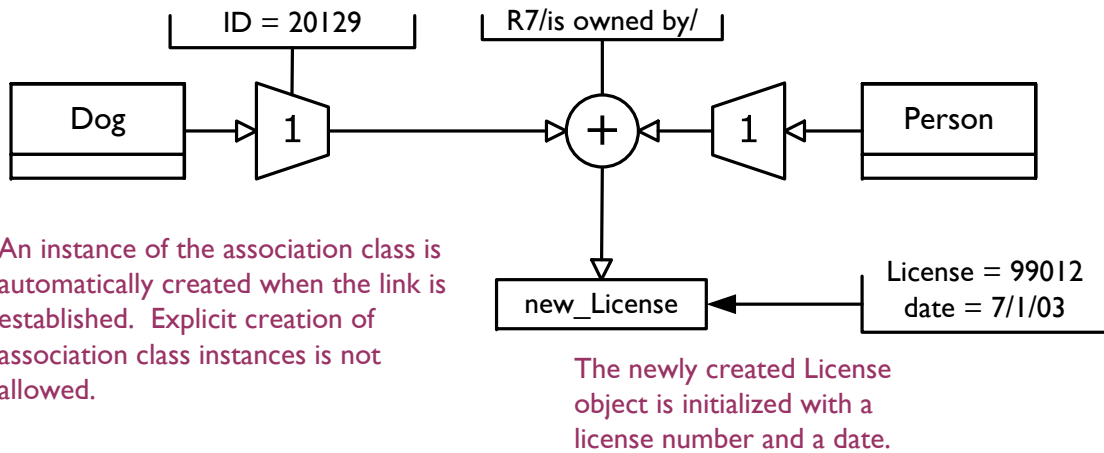


Linking Objects

Link - binary association



Link - association class

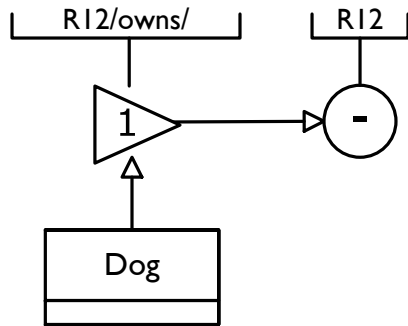


Scroll Concepts

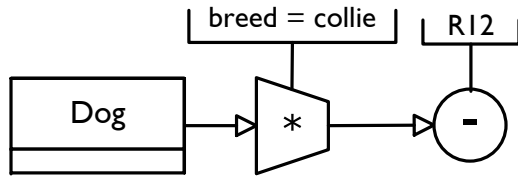


Unlinking Objects

Unlink - binary association



Unlink this Person from all his or her Dogs. If there are association class instances on the links, delete them.

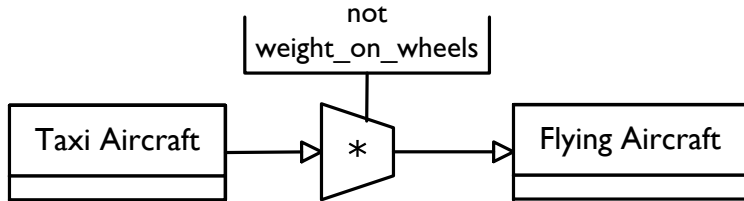


Unlinks all collies from their owners and deletes the associated License association class objects.



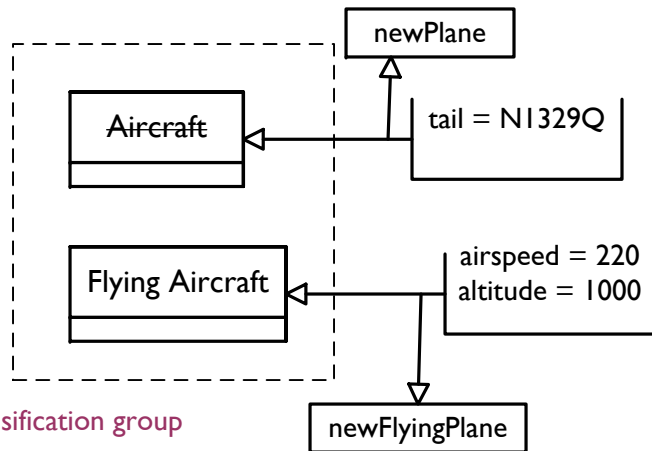
Creating, Deleting and Reclassifying Specialized Objects

Reclassification



All taxiing aircraft with no weight on the wheels are reclassified as flying aircraft.

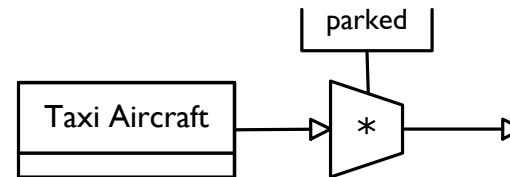
Creating



Classification group

All classes in the generalization requiring initialization values and/or producing object stores are shown surrounded by the dashed classification group symbol.

Deleting

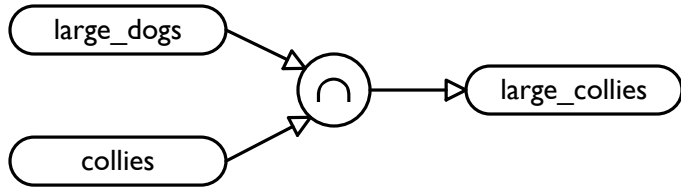


The Taxi Aircraft and Aircraft instances are deleted. Since no additional data is required to specify the deletion of the super/subclass instances, no explicit specification of these activities is required.

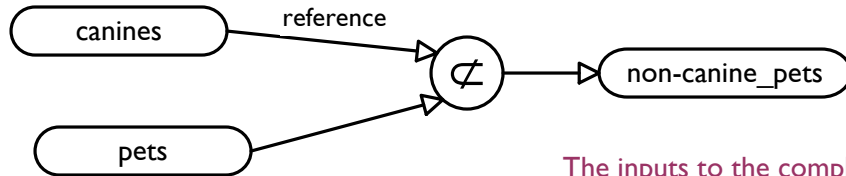
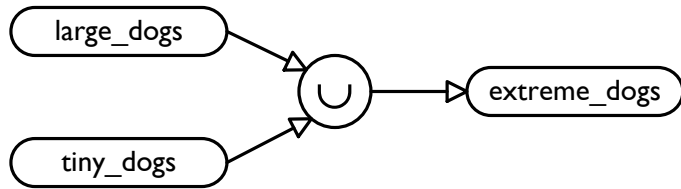
Scroll Concepts



Performing Set Actions on Objects



All object sets input to a set operation must belong to the same class.



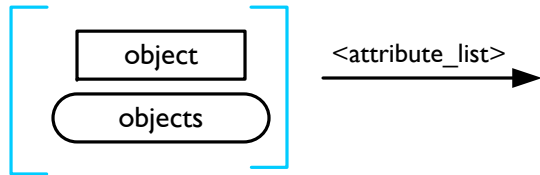
The inputs to the complement set action have specific roles. The reference set is marked by labeling the object flow. This operation produces as output that part of the input set that is not a member of the reference set.



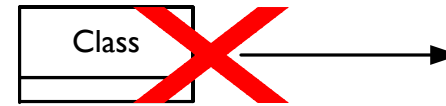
Reading and Writing Attributes

All data types, boolean, string, matrix, whatever, are connected via data flows.

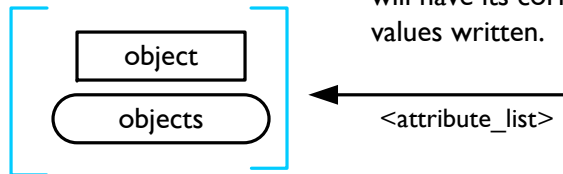
Attribute values may be read by directing an arrow out of an object store.



Since a class store represents the existence of a class and not any specific object, it doesn't make any sense to read data from a class. Only object data may be accessed.



The attributes to be read from an object store are named separated by commas or newline characters. If the source is a multiple object store, the flow represents the flow of attribute values per each object. If the object store is ordered, the attribute tuples will be ordered in the same sequence. Otherwise the attribute tuples will flow in arbitrary order.



A data flow pointing into an object store writes the specified attributes. If the object store is multiple, then each object will have its corresponding attribute values written.

```

<attribute_list>
attribute_list : attribute | attribute_list delim attribute
attribute : STRING
delim : “,” | NEWLINE

```

Scroll Concepts

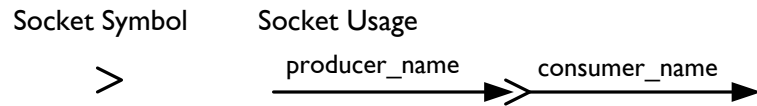
Leon Starr
mint.scroll.tn.l



Connecting Data Flows

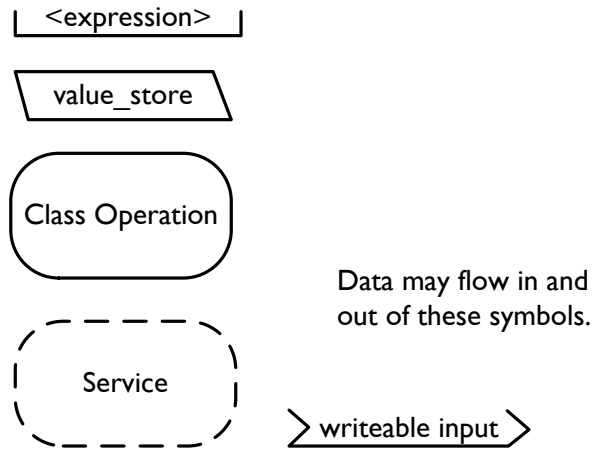
Flow Renaming

When the output of a producer element differs in meaning from the input to a consumer element it may be necessary to name the data content from each perspective. This is achieved with a socket.

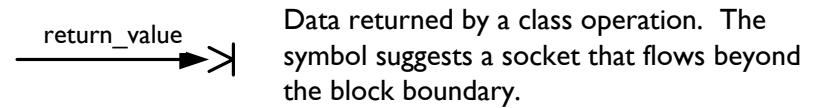


Sockets may be used with control and object flows in the same manner.

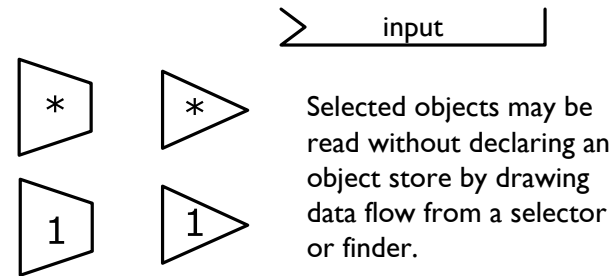
Sources and Destinations for Data Flows



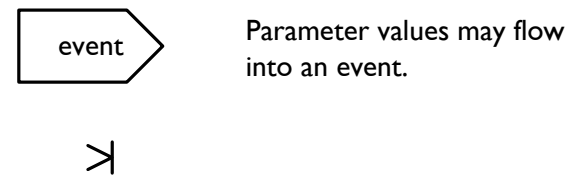
Returning a Value



Sources Only



Destination Only



Scroll Concepts



Computing and Comparing Data

Data is processed using text expressions. Text expressions are written in trays or text blocks.

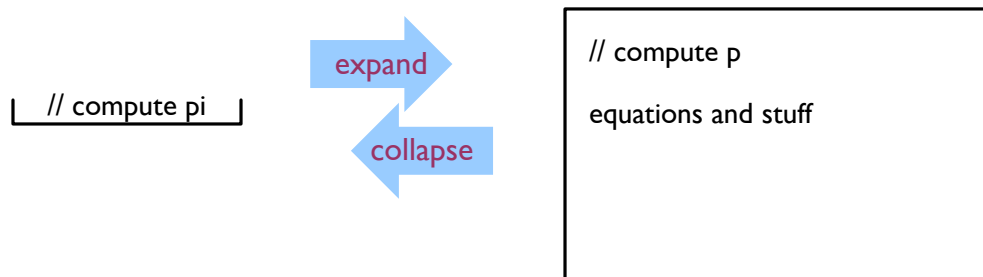
Trays



Trays are useful for writing short expressions. A tray may have any number of inputs or outputs. Longer expressions should be written in blocks so that the details do not obscure the complete activity.

Expanding and Collapsing Big Trays

It may be convenient to expand the tray to contain a particularly long expression. The user will be able to click on a tray and open up their favorite text editor and type away to their heart's content. On the whiteboard, the user can just write a detailed expression off to the side and then refer to it with an abbreviation in the tray. The abbreviation in the tray is expressed as a comment.



Scroll Concepts

Leon Starr
mint.scroll.tn.l



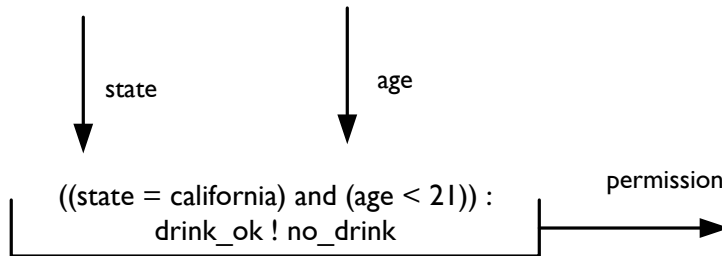
MODEL INTEGRATION, LLC. CONFIDENTIAL
All Rights Reserved.

Computing and Comparing Data

An expression operates on one or more input data items and produces a single output data item. The output data item may any desired structure. An expression consists of values and functions. Functions may be represented either infix (as operators such as +, *, =), unary (-) or as functions proper $f(x)$. The lhs of the expression will be missing as this is considered to be the single output of the function. So only the rhs is expressed.

Comparisons

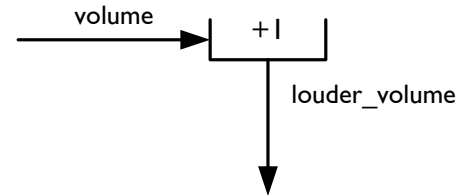
Boolean expression



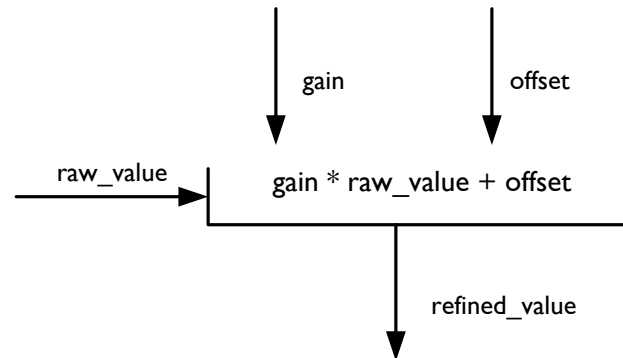
The user will be able to define operators and functions for non-native data types. The details will be spelled out in the datatype metamodel - not in the scope of the action language metamodel.

Computations

Increment the incoming data flow and pass the result.

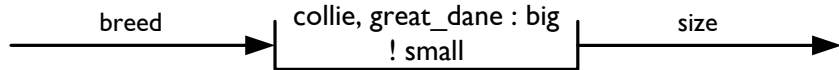
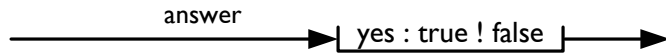


Apply offset and gain to an incoming raw value.

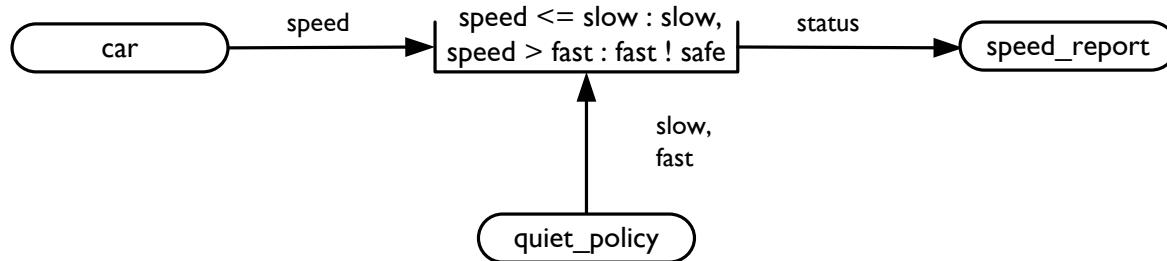
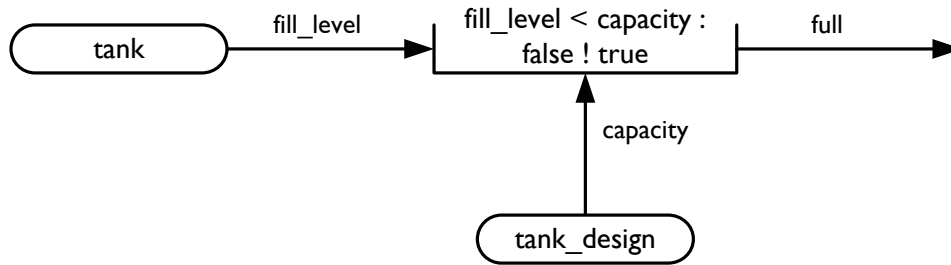


Converting Data

A value may be mapped from one data type to another using the “!” and “:” mapping operators.



The “!” operator guarantees that each input value will be mapped to an output value.



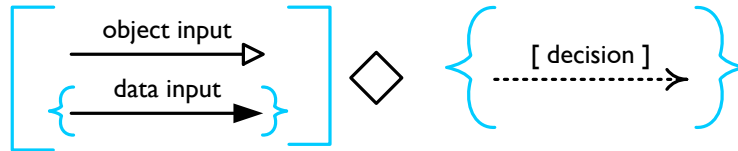
Scroll Concepts

Leon Starr
mint.scroll.tn.l



Branching on Conditions

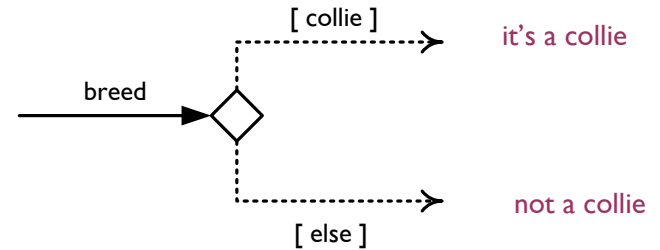
A decision is represented by a UML branch symbol.



One or more outputs represent decisions. A decision triggers downstream activity. Each output on a branch must be labeled with a decision expression enclosed within the [] guard symbol.

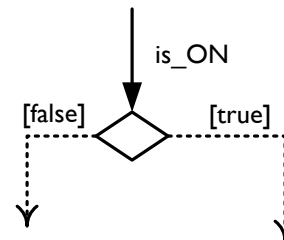
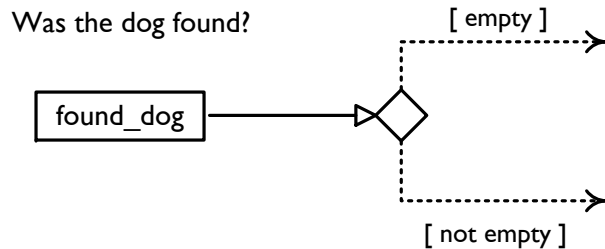
There is always at least one object or data input. An object input delivers the cardinality value [empty | not empty]. One or more data flow inputs deliver whatever data they carry.

A decision is an expression that matches one or more values that may arrive at the input. If there is a match, the associated control flow activates whatever it points to downstream.



An object flow is easily tested for content. Note that the decision cannot execute until after the upstream object store has been updated - see execution rules section.

The "else" keyword used by itself in a decision matches any values not covered by the other outputs on the same branch. This is usually more convenient than stating the complement of the other combined decisions.

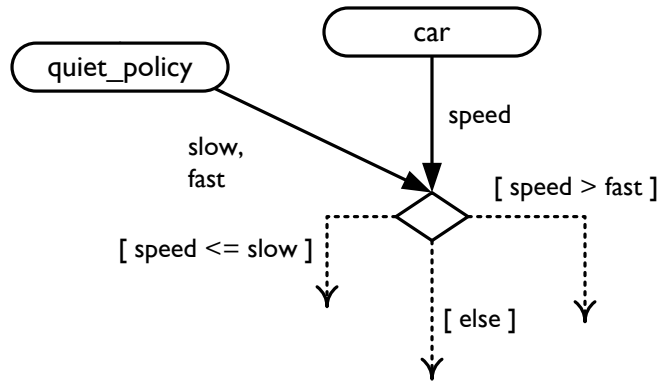


Scroll Concepts



Branching on Conditions

Decisions may be made by comparison to multiple data inputs.



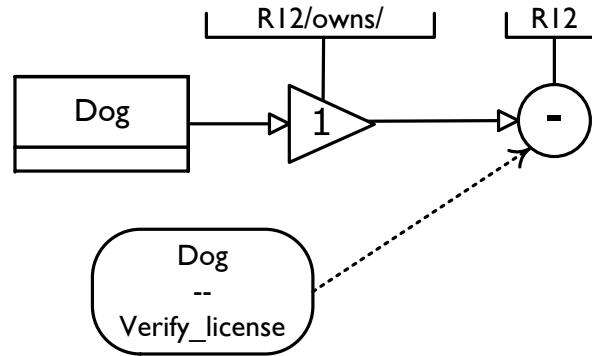
If there is more than one data input, each decision must reference the incoming data items by name.



Explicit Sequencing

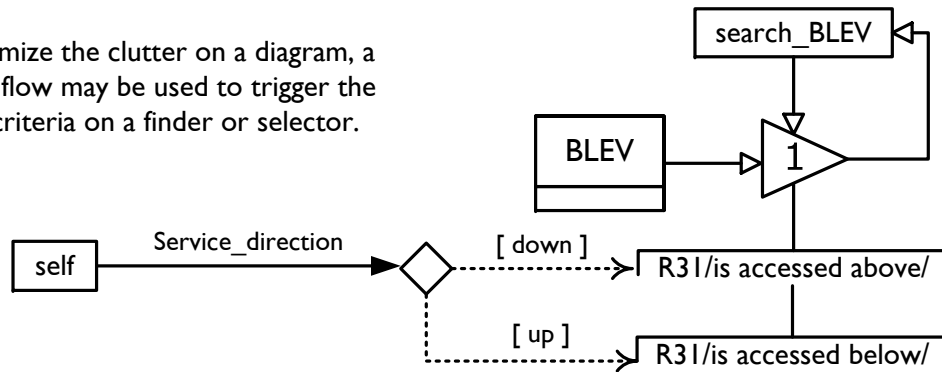
The purpose of a control flow is to explicitly sequence the execution of actions. This is done conditionally with labeled control flows or unconditionally with unlabeled control flows.

Any action may output an unlabeled (unconditional) control flow to sequence downstream activity.



The unlink operation proceeds only after both the finder action and Verify_license operation have completed.

To minimize the clutter on a diagram, a control flow may be used to trigger the search criteria on a finder or selector.



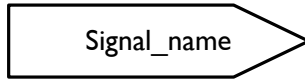
Once a process has its data and control inputs, it executes only once. (See execution rules.) So this is not an infinite loop!

Multiple trays may be stacked on a finder or selector action to allow conditional selection of criteria. This is only legal when the trays are attached to the mutually exclusive outputs from a single branch. Otherwise, tray selection could be ambiguous when multiple conditions for the same tray stack are active.

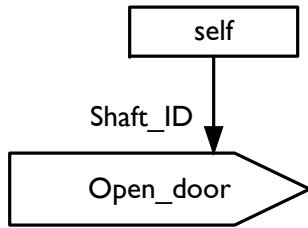
Scroll Concepts



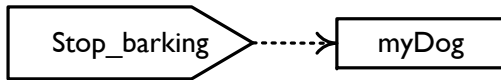
Transmitting Signals



The UML standard symbol is used for signal generation.



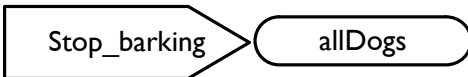
Data flows directed into a signal action



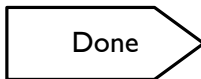
The destination of a signal is shown with an unlabeled control flow.



The control flow may be omitted if the the signal transmitter is pointed adjacent to the destination.



A signal may be directed at multiple instances. This results in the generation of a separate signal to each instance.



When not pointing to any specific instance, assume the event is generated to self.

Scroll Concepts

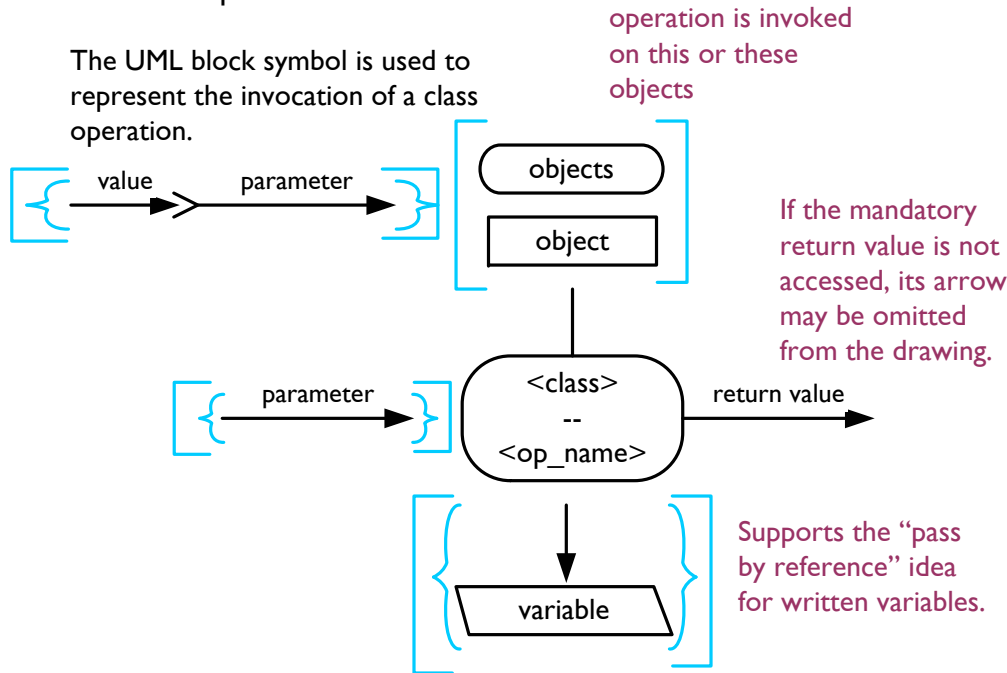


Defining and Invoking a Class Operation

An operation may be defined on a class that can be invoked separately by each object. (Also referred to as an object-based operation).

External Representation

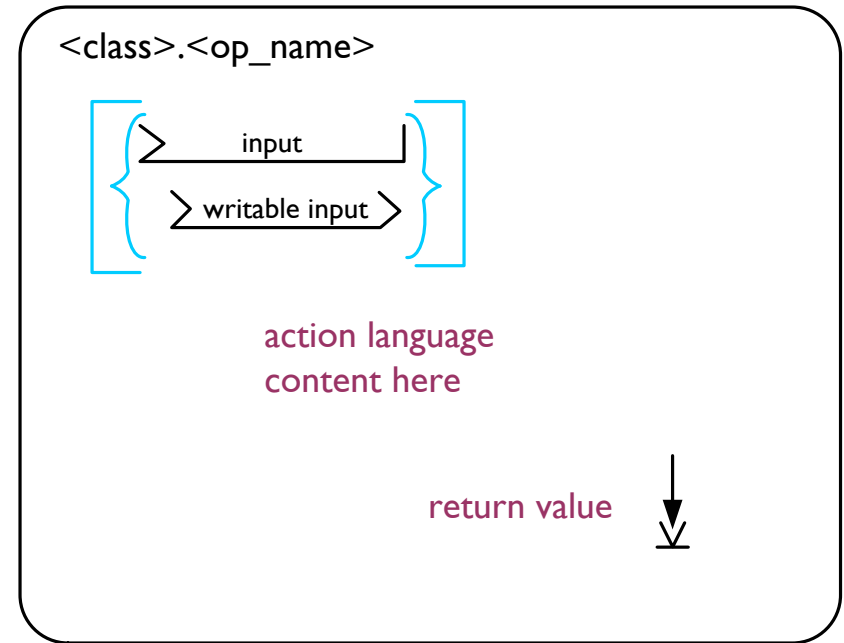
The UML block symbol is used to represent the invocation of a class operation.



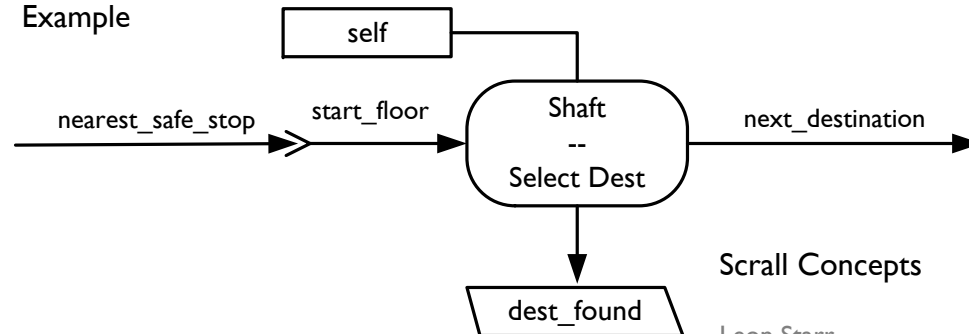
A class operation can accept any number of parameter inputs including zero. A single data value must be returned, but any number of external variables may be written.

Internal Representation

The content of a class operation is defined inside of an open block.



Example



Scroll Concepts

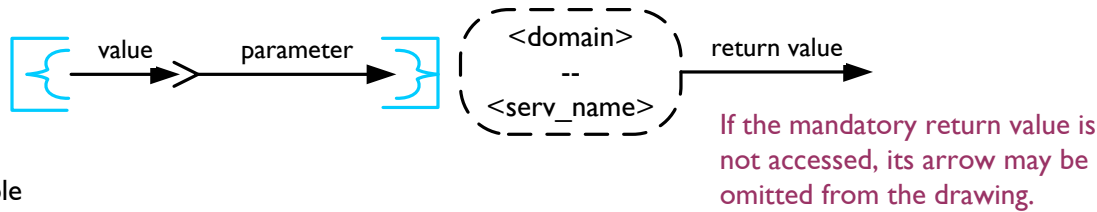
Leon Starr
mint.scroll.tn.l



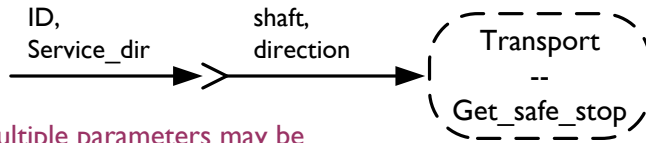
Defining and Invoking External Services

External Service

A service provided by (and implemented in) another domain is represented by a block with a dashed outline. An external service must return a value.



Example



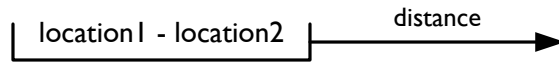
Multiple parameters may be shown with a single arrow as long as the names on the input and parameter sides are in the same order, comma separated.



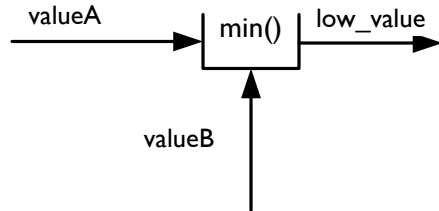
Functions and Operators

A domain specific data type such as geographic latitude, motion vector or quaternion may be defined using the, yet to be specified data type metamodel. Data type specific casts and functions will also be defined. Addition, for example, may have a special meaning when applied to latitude. Furthermore, advanced mathematical operations may be defined by the user for base data types such as boolean, integer and real.

In Scall there is no need for special graphics to represent functions operators and casts. These will be accommodated as text in expression trays. Here are some examples:



Both location1 and location2 are defined as (lat, long) pairs. Subtraction is defined for geographical coordinate processing.



Functions like min() and max() can be defined for various data types.

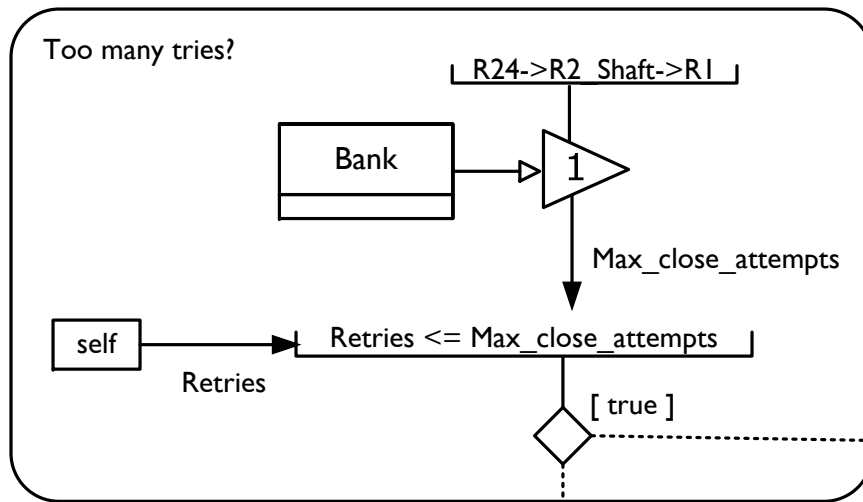
The data type model is not addressed in Scall. Scall supports any expression language that you might want to write in the trays. The point, here, is that there is no need for special global function or operator symbols. We don't want to clutter our diagrams with primitive data flows, best expressed as text.

Scall Concepts

Leon Starr
mint.scrall.tn.l

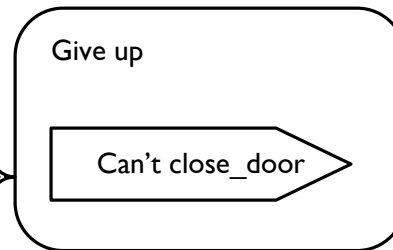


Name blocks may be drawn, much like xtUML subsystem boundaries, around clusters of actions. Any control, object or data flows may be intersected by the boundaries. A descriptive name may then be applied to each block.

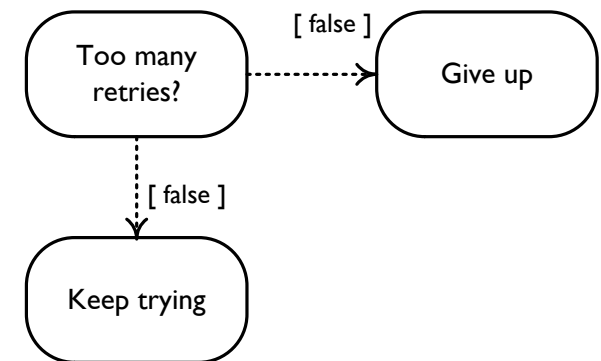
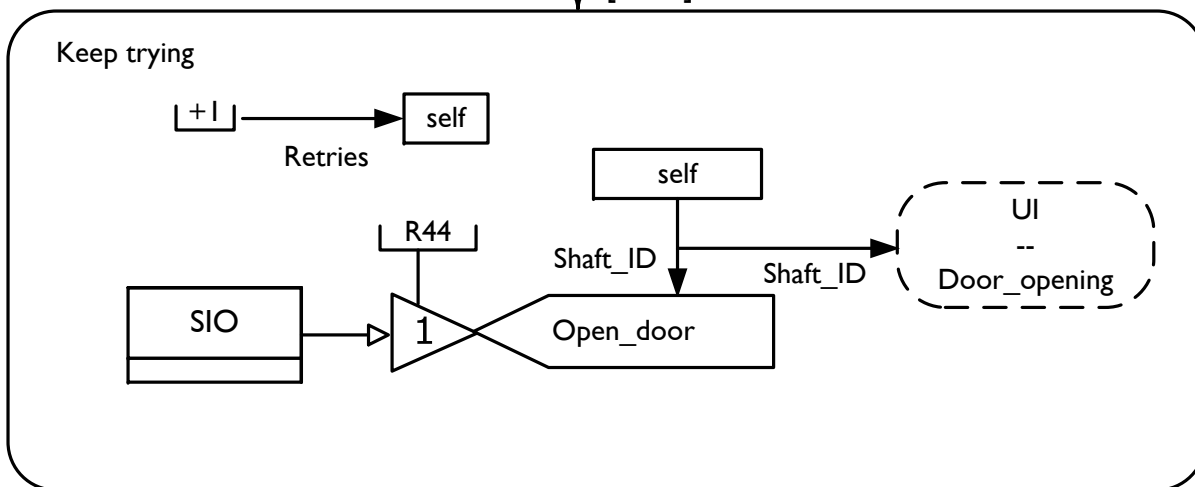


If the control flow touching the boundary of a block is not triggered, than nothing inside the block may execute.

Flows that pass through a block are accessed if and when the recipient actions execute.



When the block is collapsed, only the name and any input/output flows are shown.



Scroll Concepts



Execution Rules

Flow Dependency: An action executes once all of its data and control inputs are available.

An action executes once for each set of data items available.

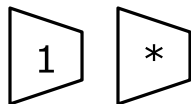
Unless explicitly sequenced, all actions may execute in parallel or in arbitrary order, the flow dependency rule notwithstanding.



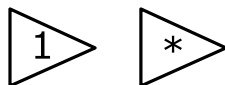
Summary of Scroll Symbols

Actions

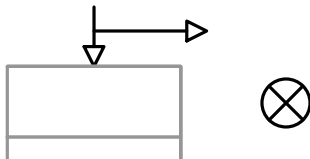
Selectors



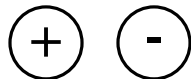
Finders



Creator and Deleter



Linker and Unlinker



Branch



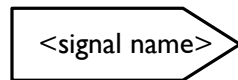
Set Operators



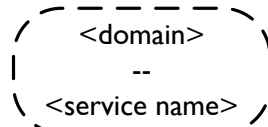
Tray Expression



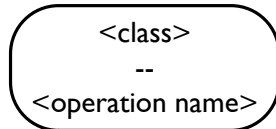
Signal Transmitter



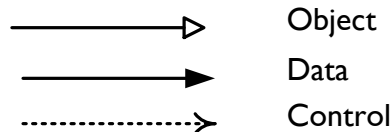
Service



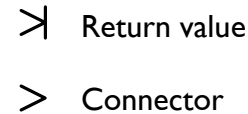
Class Operation



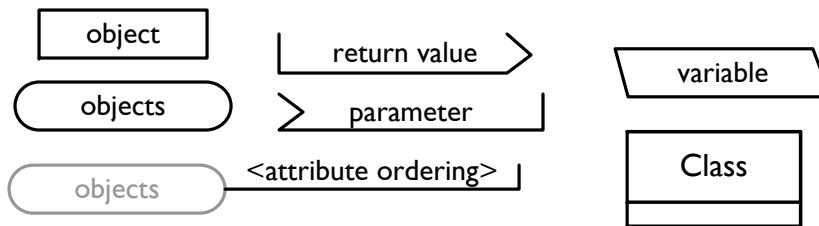
Flows



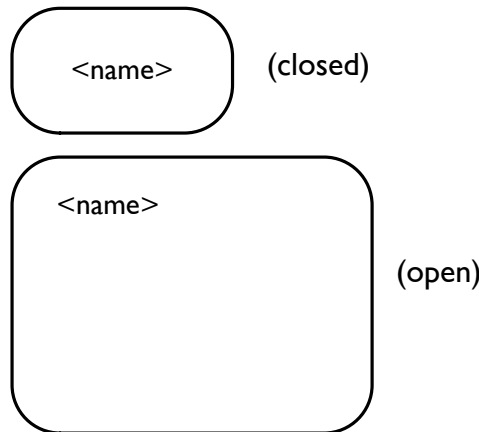
Sockets



Stores



Blocks



Comments

Comments may appear as a text block anywhere on a Scroll diagram. Ideally the text will be green or some offset color so that the comments are easily distinguished from the symbol labels.

Scroll Concepts



UML Compliance

The good news is that Scroll is 100% consistent with the UML 2.0 Action Semantics.

Unfortunately, the Scroll symbols are only about 10% consistent with standard UML graphics standards. With some effort, it may be possible to modify them to approach 15-20% compatibility. At most. Existing UML symbols just aren't up to the task Scroll was designed to address.

But there are many compelling advantages to adhering to industry accepted standards. What to do?

- ① Examine Scroll to see if some small changes can be made to reduce graphic conflicts with UML standards.
- ② Build translator that can convert the Scroll activity diagrams into a kind of improved OAL text script.
- ③ Be sure to save the graphical placement data in a separate file so that the graphics can be reconstituted from the OAL text + Placement data.
- ④ The OAL text + the class, state and other UML (non Scroll) diagram represents the 100% UML part of the models. The remaining Scroll data can be saved as a separate proprietary unit that tags along with the UML model set. The model set can be reviewed and tested without reference to the Scroll component. Yet the Scroll diagrams can be viewed and edited by developers who would prefer the Scroll notation.

I'm open to suggestions!

