TRACK: Keeping Track of Highly Mobile Object

A Language-Level Proposal

Position Paper

Eric Jul

Professor II, IFI, University of Oslo

Bell Labs Ireland

What's my point?

Emerald introduced *object location* as a fundamental language concept and has constructs for *object mobility.*

This proposal is for a language construct that *tracks* object mobility in a manner similar to the *Observer* design pattern: applications can be notified when an object has moved.

What's the Problem?

In the Cloud world, thin clients, exemplified by smartphones, move along the edge of the cloud.

Applications running on the client may want to handle the change of location of the client, e.g., a gaming client may want to reconnect to the closest gaming server in the cloud, or a tablet may want to react to entering the office or its home.

Specific Problem

How are objects to be notified that they or another object has moved?

The Simple Brute Force Solution

Polling:

```
oldloc <- locate X
while oldloc == locate X do /* nothing */
act on the new location
```

Inefficient

May miss moves

Notification Based

Better solution:

Underlying system that performs the move *notifies* interested parties of each move.

How are these Notifications Passed to the Language Level?

There is no inherent way to do so!

Solutions:

- Library function
- Language construct

Library Function

Establish a thread to await a change:

```
thread
    newLoc = track.await(X)
    act on new location …
end thread
```

But this just pushes problem into the library – it is still *black magic* seen from the language viewpoint.

Language Construct

Will add a language construct to Emerald.

First, a short review of the Emerald language.

Emerald OO language

Emerald is an OO language:

- "Pure" OO like Smalltalk – all data represented as objects (no primitive types)
- Algol-family syntax (statements are NOT objects)
- Process concept (threads)
- Synchronization (Hoare monitors)
- Conformity based type system (worth several talks in itself)
- Like Java, but simpler

Distribution features

Concept of location: A *node* is merely a machine (within a *semi-closed* network)

- Mobility: `move X to Y`
- Attachment allows groups to be moved
- Location: `loc <- locate X`
- “Remote” object invocation
- Checkpoint: stable version to disk
- Node failure: failure handler, unavailability
- Immutable objects (instead of primitives)

Example: Kilroy

```
const Kilroy == object Kilroy
  process
    var  i:              Integer <- 0
    var  myNode:         Node <- locate self
    var  myList:         Nodelist
    var  remoteNode:     Node
    myList <- myNode.getActiveNodes
    for (i <- 0; i < myList.upperbound; i <- i+1)
       remoteNode <- myList(i)$theNode
       move Kilroy to remoteNode
    end loop
  end process
end Kilroy
```

The TRACK Construct

```
track X notifying Y
```

This asks that the underlying system invokes `Y` each time `X` has moved.

Analogous to the Observer design pattern.

Tracker Object Interface

```
objecttype notifiable
  op ObjectMoved[Any a, Node n, Time t]
  op ObjectAppearsUnavailable[Any a, Time t]
  op ObjectAppearsAvailable[Any a, Time t]
end notifiable
```

Quitting Tracking

```
detrack X from Y
```

Implementation

- Add a list of tracker objects to the tracked object
- After every move: notify each tracker

Fault Tolerance

- Maintain a list of tracked objects with each *tracker* object.

Garbage Collection Considerations

- Use weak reference from tracker object to tracked object – trackers should not keep tracked object alive.

Modelling Node Mobility

Whole nodes that migrate, e.g., smartphones moving can be modelled by tracking the special Node object associated with each Node.

Conclusion

A new language construct for tracking mobile objects.

Is model of *Node* mobility the right one?

URL

www.emeraldlanguage.org

www . emeraldlanguage . org