Mobile Ambients

Luca Cardelli

with Andrew Gordon, Cambridge Computer Lab

Introduction

Context

- Programming the Web.
- Lots of existing and forthcoming technology for mobile computation.

History

- Obliq, Telescript, (pre-RMI) Java: three different models of mobility.

Recent experiences

- Gone to several web meetings.
- Written a few position papers.
- Suddenly, ideas started precipitating.

Plan

- Devise and study mobility abstractions. (And use them within Java.)

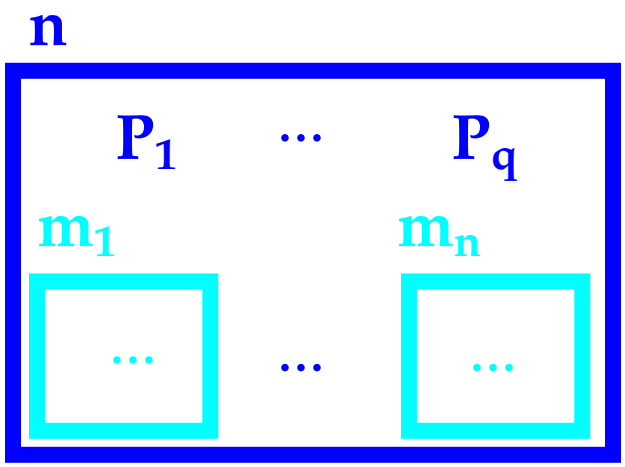
Ambients

- An ambient is:
  - A confined place where computation happens.
  - Also, something that can be nested within other ambients.
  - Also, something that can move as a whole.

- An ambient has:
  - A name. (Used to control access.)
  - A collection of local agents (threads).
  - A collection of sub-ambients.

- A name is:
  - Something that can be created, passed around, and used to name new ambients.
  - Something from which entry and exit capabilities can be extracted.

- Typical shape of an ambient:



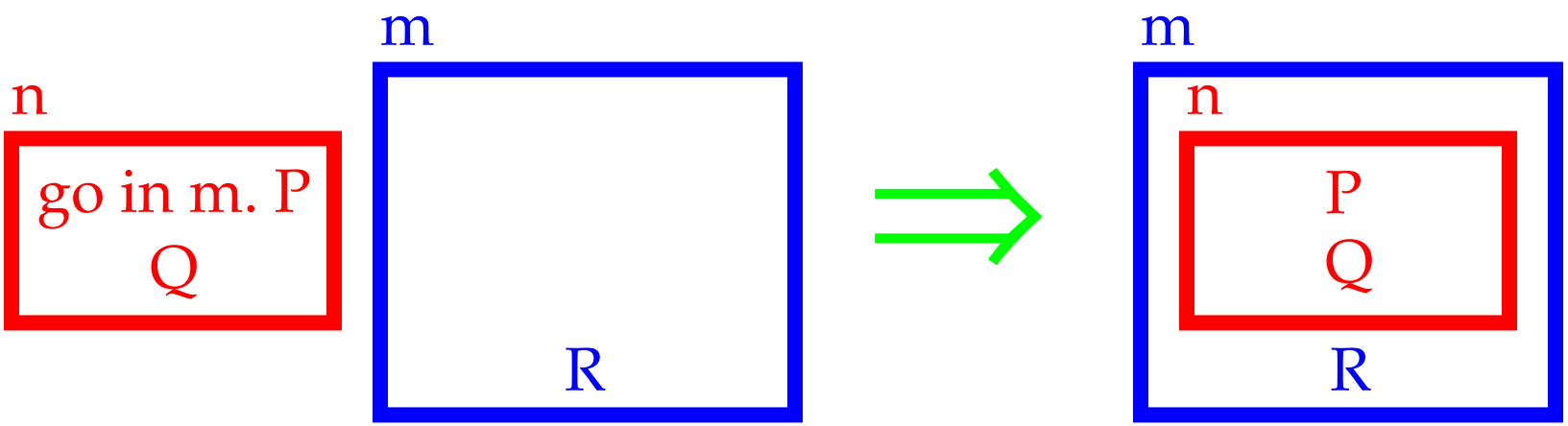
n [*name*
  P1 | ... | Pp | *agents*
  m1[...] | ... | mn[...] *sub-ambients*
]

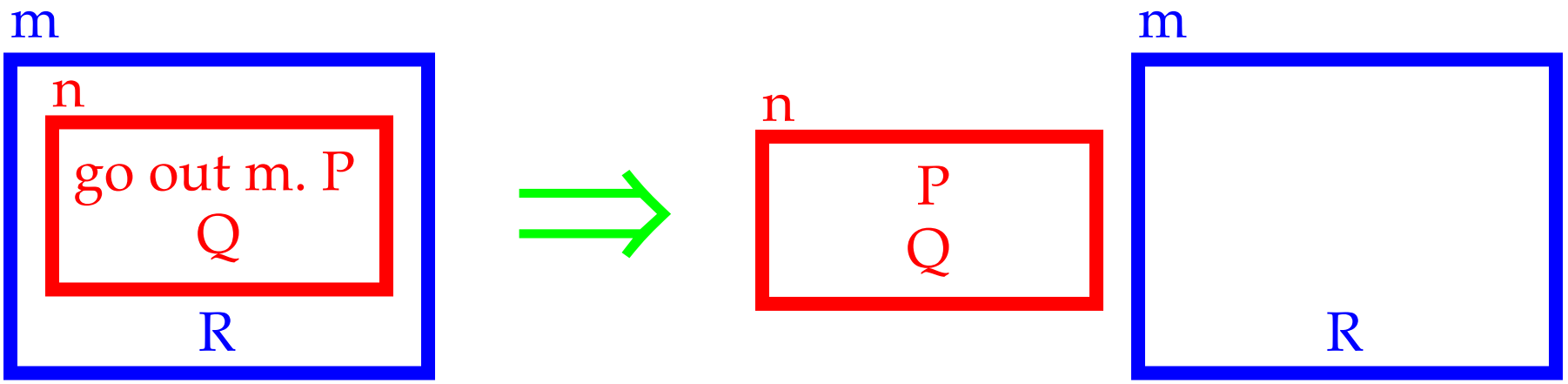
- Main operations on ambients:
  - ~ Enter. (Requires an entry capability.)
  - ~ Exit. (Requires an exit capability.)
  - ~ Be. (Change name.)

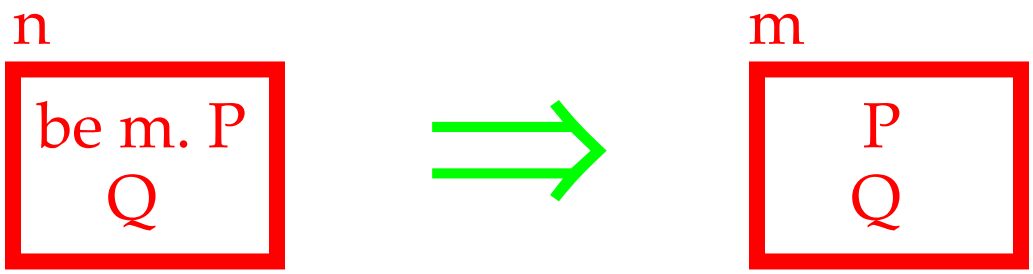
- Discussed today:

  Not computation, not communication. Just mobility.

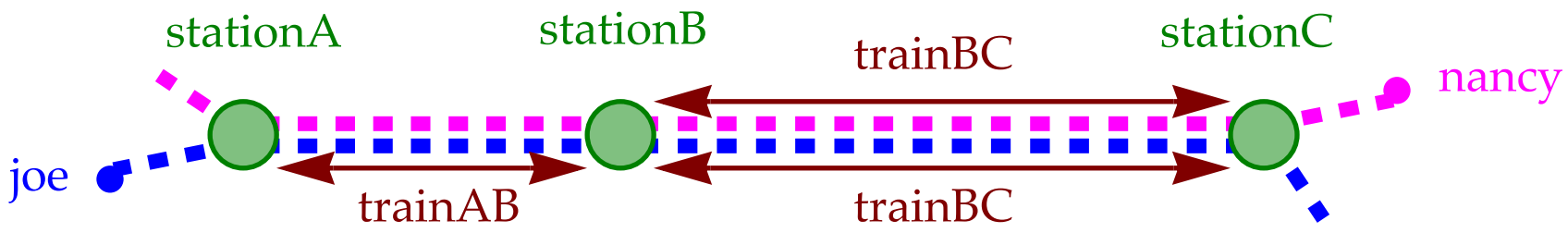
Ambient Dynamics







Example




```
let train(stationX stationY XYatX XYatY tripTime) =
  new moving.        // assumes the train originates inside stationX
    moving[rec T.
      be XYatX. wait 2.0.
      be moving. go out stationX. wait tripTime. go in stationY.
      be XYatY. wait 2.0.
      be moving. go out stationY. wait tripTime. go in stationX.
      T];

new stationA stationB stationC ABatA ABatB BCatB BCatC.
  stationA[ train(stationA stationB ABatA ABatB 10.0) ] |
  stationB[ train(stationB stationC BCatB BCatC 20.0) ] |
  stationC[ train(stationC stationB BCatC BCatB 30.0) ] |
```

```
new joe.
  joe[
    go in stationA.
    go in ABatA. go out ABatB.
    go in BCatB. go out BCatC.
    go out stationC] |

new nancy.
  nancy[
    go in stationC.
    go in BCatC. go out BCatB.
    go in ABatB. go out ABatA.
    go out stationA]
```

Execution trace

```
moving: Be ABatA
moving: Be BCatC
moving: Be BCatB
nancy: Moved in stationC
nancy: Moved in BCatC
joe: Moved in stationA
joe: Moved in ABatA
ABatA: Be moving
BCatC: Be moving
moving: Moved out stationC
BCatB: Be moving
moving: Moved out stationB
moving: Moved out stationA
moving: Moved in stationB
moving: Be ABatB
joe: Moved out ABatB
ABatB: Be moving
moving: Moved out stationB
moving: Moved in stationC
moving: Be BCatC
BCatC: Be moving
moving: Moved out stationC
moving: Moved in stationA
moving: Be ABatA
ABatA: Be moving
moving: Moved out stationA
moving: Moved in stationB
```

```
moving: Be BCatB
nancy: Moved out BCatB
joe: Moved in BCatB
BCatB: Be moving
moving: Moved out stationB
moving: Moved in stationB
moving: Be ABatB
nancy: Moved in ABatB
ABatB: Be moving
moving: Moved out stationB
moving: Moved in stationB
moving: Be BCatB
BCatB: Be moving
moving: Moved out stationB
moving: Moved in stationA
moving: Be ABatA
nancy: Moved out ABatA
nancy: Moved out stationA
ABatA: Be moving
moving: Moved out stationA
moving: Moved in stationB
moving: Be ABatB
moving: Moved in stationC
moving: Be BCatC
joe: Moved out BCatC
joe: Moved out stationC
moving: Moved in stationC
...
```

Basic Ambient Expressions

P ::=	an activity	
	n[P]	an ambient named n with contents P
	new n. P	create a new name for an ambient (then do P)
	go C. P	move the enclosing ambient (then do P)
	be n. P	rename the enclosing ambient (then do P)
	P | P	two activities in parallel
	–	inactivity
C ::=	a capability	
	in n	entry capability for name n
	out n	exit capability for name n
	C_1 & C_2	path

Java Interface

```
package Ambit;

public interface AnAmbient {
// Structure

  public Name getName();
  // The current name of this ambient.

  public Env getInitEnv();
  // Get initEnv, the environment at the time this ambient was created (never changes).

  public Ambient newOwnAmbient(Name name, Env env) throws AmbitException;
  // Creates an empty ambient with the given name. It becomes a child of the current ambient.
  // The env parameter becomes initEnv for the new ambient.

  public void startAgent(CodeProc code, Env env) throws AmbitException;
  // Start a new agent in this ambient. The agent runs code with initial environment env.
  // For a "fresh" agent, env should be set to initEnv.
  // For a "continuing" agent (e.g. one forked off by a par), env could be longer than initEnv.

// Movement

  public void moveOut(OutCap parentCap) throws AmbitException;
  // Move this ambient outside the parent (it becomes a sibling of the parent).
  // Requires an output capability to exit the parent.
  // Blocks until a parent's parent exists, and until a parent matches the capability.

  public void moveIn(InCap receiverCap) throws AmbitException;
  // Move this ambient inside a sibling ambient (it becomes a child of the sibling).
  // Requires an input capability to enter the sibling.
  // Blocks until a parent exists, and until a sibling matches the capability.

  public void become(Name newName) throws AmbitException;
  // Rename this ambient.
  // Blocks until a parent exists (to avoid races with other operations).

  public void implode() throws AmbitException;
  // The current ambient goes puff. (It is removed from its parent.)
  // Blocks until a parent exists.
```

```
// Communication

  public void give(Result result) throws AmbitException;
  // Offers to output a value into the current ambient's ether.
  // Blocks until it can match an input.

  public Result take() throws AmbitException;
  // Offers to input a value from the current ambient's ether.
  // Blocks until it can match an output.

  public void say(Result result) throws AmbitException;
  // Offers to output a value into the parent ambient's ether.
  // Blocks until a parent exists in which it can match an input.

  public Result ask() throws AmbitException;
  // Offers to input a value from the parent ambient's ether.
  // Blocks until a parent exists in which it can match an output.

// Utility

  public void scream(String screamMsg);
  // Scream a message from this ambient to a global console.

  public String toString();
  // Display the current state of the ambient.
  // If the ambient is changing, it may display an inconsistent configuration.

}
```