A Framework for Mobile Java Applications

Andrew Hughes
http://www.dcs.shef.ac.uk/~andrew

Department of Computer Science
University of Sheffield

Ja4Mo07 - 05/09/2007

Outline

1. Introduction
2. Example 1: Broadcasting
3. Example 2: Extending to Mobility

Introduction

- The Dynamic Theory Execution (DynamiTE) framework simplifies creating concurrent object-oriented applications.
- Allows a formal specification to be translated directly into code.
- Grounded in a process calculus with a formal semantics.
- Facilitates inter-process signalling and process movement.

Example 1: Broadcasting

- **Scenario**: We want to send a message from one process to an arbitrary number of other processes.
- Number of recipients = number which choose to listen.
- Same broadcaster can handle any number.
- Can be represented in the calculus.
- Thus most of the work is done by the framework and not the developer.

Step 1: Specify the System

- How do we want the broadcaster to behave?

1. Create the data to broadcast.
2. Repeatedly send the data until all recipients have it.

- This is specified as:

Example

$$\tau.\mu X.\lceil \overline{o}.X \rceil \sigma(\mathbf{0})$$

Breaking It Down

Example

$$\tau.\mu X.\lceil \overline{o}.X \rceil \sigma(\mathbf{0})$$

- τ – some internal behaviour.
- μX – $X = \lceil \overline{o}.X \rceil \sigma(\mathbf{0})$
- $\overline{o}$ – send a message on the channel o
- $\lceil \overline{o}.X \rceil \sigma(\mathbf{0})$ – do $\overline{o}.X$ if the clock σ doesn't tick, $\mathbf{0}$ if it does.
- $\mathbf{0}$ – no explicit behaviour.

Step 2: Turn This Into Code

- Each construct becomes an instance of `Process`.
- `Tau` is an *abstract class*.
- The method `execute()` provides internal behaviour.

Example

```
public class CreateData extends Tau {
      public void execute() { }
     }
```

Translating It

Example

$$\tau.\mu X.\lceil \overline{o}.X \rceil \sigma(\mathbf{0})$$

- $\overline{o}$ – `out = new OutputChannel("o")`
- $\mathbf{0}$ – `Nil.NIL`
- $\lceil \overline{o}.X \rceil \sigma(\mathbf{0})$ – `to = new StableTimeout(new Prefix(out,new Var("X")),Clock.get("σ"),Nil.NIL)`
- μX – `rec = new Mu(to)`
- Whole thing – `new Prefix(new CreateData(), rec)`

What About The Receiver?

- Specified as simply *o.P*
- *o* – `in = new InputChannel("o")`
- *P* – any instance of *Process* which carries on
- Communication requires matching channel names
- Sender and receiver are connected by `new Par(sender, receiver)`
- Matching channels produce a special `Tau` instance, `Synchronisation` when *in parallel*.

Handling Data

- No formal representation of data
- DynamiTE has *local* and *global* contexts
- *Local context* works up to parallelism – maps to `ThreadLocal` where `Par` uses threads
- *Global context* is simply the *environ tree*
- *Environs* exist between the two

Implementation Details

- DynamiTE ensures:
  - A top-level *environ* so clocks can operate.
  - A top-level `Par` within each environ so local contexts work
- Channels and parallelism work by a plugin framework
- Other minor issues such as internal identifiers

Extending to Mobility

- Instead of sending data, we can send processes.

Example

$$\mu X.\lceil \mathit{on\ move} \ovee \mathit{host}.X \rceil \sigma(\mathbf{0})$$

- *on move* $\ovee$ *host* – Move process available on *move* to sibling environ *host*
- Recipient looks the same (*move.P*) but enters an *environ* called *host* to do *P*
- Channel type (input/output) is irrelevant

Translating It

Example

$$\mu X.\lceil \textit{on move} \varovee \textit{host}.X\rceil\sigma(\mathbf{0})$$

- *on move* $\varovee$ *host* – `new ProcIn(new Channel(“move”),Environ.get(“host”))`
- Rest stays the same.
- Connected using `new Par(sender, receiver, Environ.get(“host”))`

Implementation Details

- Environs can be on either the same or a different host.
- Migrating process is ‘frozen’; no execution yet performed.
- Migration is thus well defined: just the code for *P* and its local context need be transferred.
- Migration is always local; to a sibling or to outside the parent.

Future Work and Conclusions

- Still in heavy development; some of this may change.
- A lot yet to be realised and lots of room for further exploration
- Can provide a simpler way of implementing concurrent and mobile concepts in programs
- Also useful as an interesting way to present theory to students
- Leveraging of object-oriented techniques makes it easier to alter/extend this calculus (TNT) and implement others

The End

Thanks for listening.
Any questions?