Programming in the large

Bertrand Meyer

Lecture 12: Agents

Agenda for today

- Scope of this development
- Applications
- The mechanism

Scope

- Starting from an object-oriented basis, add a new kind of objects representing potential computations.
- Such objects are called "agents".
- Earlier names:
  - Delayed calls
  - Routine objects
- Similar to:
  - "Closures"
  - Delegates (.NET: C#, Visual Basic .NET...)
  - Blocks (Smalltalk)
  - Lambda expressions

O-O structure

Action

Object

Processor
Traditional input scheme

```
from open_file until end_of_file loop
  read_next
  process (last_item)
end
```

Event-driven programming

```
PUBLISHERS
  trigger events

SUBSCRIBERS
  handle events

EVENTS

ROUTINE
```

Compare to...

- "Functional" style of programming, e.g. Haskell
- Conjecture: Haskell should be an Eiffel library (Eifskell?)

The starting idea of object-technology

- Organize software architecture around data types.
- Agents: Can an object represent an action?

Agenda for today

- Scope of this development
- Applications
- The mechanism

Applications of agents

- Iteration
- High-level contracts
- Numerical programming
- Introspection
- High-level functionals, type-safe
Integration example (1)

\[
\int_{a}^{b} my\_function(x) \, dx
\]

\[
my\_integrator\_integral(agent\ my\_function, a, b)
\]

Integration example (2)

\[
\int_{a}^{b} your\_function(x, u, v) \, dx
\]

\[
my\_integrator\_integral(agent\ your\_function(?, u, v), a, b)
\]

- In the first example (one argument), the notation `agent my_function` is a synonym for `agent my_function(?)`.

Metrics example

```python
create source_line_metric.make
("Source_lines",

[ [feature_scope, agent feature_line_counter],
  [class_scope, agent class_line_counter]
  ]
)
```

Error handling example: without agents

```plaintext
action1
if ok1 then
  action2
  if ok2 then
    action3
    ... More processing, more nesting ...
  end
end
```

Error handling: with agents

```plaintext
execute ([agent action1,
  agent action2 (...),
  agent action3 (...)])

if glitch then
  warning (glitch_message)
end
```

Agenda for today

- Scope of this development
- Applications
- The mechanism
### Open and closed arguments

- **agent** `your_function (?, u, v)`
  - **Open**
  - **Closed**: set at the time of the agent’s definition
  - **Open**: set at the time of any call to the agent

### Using a routine from another class

- **agent** `some_object.some_routine (?, u, v)`
  - **Target**

### Iteration

- Consider
  - `my_integer_list: LIST [INTEGER]`
  - in a class `C` that has the function
    - `is_positive (x: INTEGER): BOOLEAN`
      - **Open**: is `x` positive?
    - **do**
      - `Result := (x > 0)`
    - **end**
  - To test that all integers in a list are positive:
    - `all_positive := my_integer_list.for_all (agent is_positive)`

### Iteration

- Consider
  - `my_employee_list: LIST [EMPLOYEE]`
  - where class `EMPLOYEE` has the feature
    - `is_married: BOOLEAN`
      - **Open**: Does this object represent a married employee?
  - To test that all employees in a list are married:
    - `all_married := my_employee_list.for_all (agent {EMPLOYEE}.is_married)`

### Target or argument open

- Compare the two examples (both in a class `C`):
  - `my_integer_list: LIST [INTEGER]`
  - `my_employee_list: LIST [EMPLOYEE]`
    - In class `C`
    - In class `EMPLOYEE`
  - Abbreviated as
    - `my_integer_list.for_all (agent is_positive)`
    - `my_employee_list.for_all (agent {EMPLOYEE}.is_married)`

### An EiffelBase contract (class HASH_TABLE)

- **extend** `(new: G; key: H)`
  - **Open**: Assuming there is no item of key `key`,
    - **require**
      - `not_key_present: not has (key)`
    - **ensure**
      - `insertion_done: item (key) = new`
      - `key_present: has (key)`
      - `inserted: inserted`
      - `count = old count + 1`
Agents’ potential for contracts

- Express general properties such as "none of the elements from positions 1 to \( \text{count} - 1 \) have been changed".

Normal call vs. agent

- Normal call
  \[ a0.f(a1, a2, a3) \]
- Agent call (expression): preface it by keyword \texttt{agent}, yielding
  \[ \text{agent } a0.f(a1, a2, a3) \]
- For example:
  \[ u := \text{agent } a0.f(a1, a2, a3) \]
- This represents the routine, ready to be called. To call it:
  \[ u.call() \]
  -- For type of \( u \), see next
- Recall original name of agents: "delayed calls".

End of lecture 12