Programming in the large

Bertrand Meyer
Lecture 12: Agents
Agenda for today

- Scope of this development
- Applications
- The mechanism
Agenda for today

- Scope of this development
  - Applications
  - The mechanism
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
Traditional input scheme

from
    open_file
until
    end_of_file
loop
    read_next
    process (last_item)
end
Event-driven programming

PUBLISHERS
trigger events

EVENTS

SUBSCRIBERS
handle events

ROUTINE

ROUTINE

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)

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

\[ my\_integrator.\text{integral} \text{ (agent my\_function, a, b)} \]
Integration example (2)

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

\(\text{my\_integrator\_integral(agent your\_function(?, u, v), a, b)}\)

- In the first example (one argument), the notation
  \(\text{agent my\_function}\)
- is a synonym for
  \(\text{agent my\_function(?)}\)
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
end
```
Error handling: with agents

\[
\text{execute}\ (\text{[agent } \text{action1}, \text{agent } \text{action2} (\ldots), \text{agent } \text{action3} (\ldots)\text{])}
\]

\[
\text{if glitch then}
\]
\[
\text{warning (glitch\_message)}
\]
\[
\text{end}
\]
Agenda for today

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

agent your_function (?, u, v)

- **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
Consider

\( my\_integer\_list: \text{LIST} [\text{INTEGER}] \)

in a class \( C \) that has the function

\( is\_positive (x: \text{INTEGER}): \text{BOOLEAN} \) is

\( \text{-- Is } x \text{ positive?} \)

\begin{verbatim}
   do
   \textbf{Result} := (x > 0)
   end
\end{verbatim}

To test that all integers in a list are positive:

\( all\_positive := \text{my\_integer\_list}.\text{for\_all} (\text{agent} \text{is\_positive}) \)
Consider

\[ my\_employee\_list: LIST \{EMPLOYEE\} \]

where class \textit{EMPLOYEE} has the feature

\[ is\_married: BOOLEAN \]

-- Does this object represent a married employee?

To test that all employees in a list are married:

\[ all\_married := my\_employee\_list.\text{for\_all} (\text{agent} \{EMPLOYEE\}.\text{is\_married}) \]
Compare the two examples (both in a class $C$):

$my_{\text{integer}_\text{list}}$: LIST [INTEGER]
$my_{\text{employee}_\text{list}}$: LIST [EMPLOYEE]

$\text{is\_positive}\ (x: \text{INTEGER}): \text{BOOLEAN}$ -- In class $C$

$\text{is\_married}: \text{BOOLEAN}$ -- In class EMPLOYEE

-- Abbreviated as
-- $my_{\text{integer}_\text{list}}.\text{for\_all}\ (\text{agent is\_positive})$:

$my_{\text{integer}_\text{list}}.\text{for\_all}\ (\text{agent is\_positive}\ (?))$

$my_{\text{employee}_\text{list}}.\text{for\_all}\ (\text{agent }\{\text{EMPLOYEE}\}.\text{is\_married})$
An EiffelBase contract (class HASH_TABLE)

extend (new: \textit{G}; key: \textit{H})
  -- Assuming there is no item of key \textit{key},
  -- insert \textit{new} with \textit{key}; set inserted.

require
  not_key_present: \textbf{not} has (key)

ensure
  insertion_done: item (key) = new
  key_present: has (key)
  inserted: inserted
  one_more: 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
  \[ a_0.f (a_1, a_2, a_3) \]

- Agent call (expression): preface it by keyword `agent`, yielding
  \[ \text{agent } a_0.f (a_1, a_2, a_3) \]

- For example:
  \[ u := \text{agent } a_0.f (a_1, a_2, a_3) \]

- 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