Agenda for today

- Scope of this development
- Applications
- The mechanism
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
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.

![Diagram]

- 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)

\[
\int_{a}^{b} \text{my_function}\ (x) \, dx
\]

\[
\text{my_integrator.integral (agent my_function, a, b)}
\]

### 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
  \[
  \text{agent my_function (?)}
  \]

### Metrics example

```plaintext
create source_line_metric.make
("Source_lines",
  [
    [feature_scope, agent feature_line_counter],
    [class_scope, agent class_line_counter]
  ]
)
```
Error handling example: without agents

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

Error handling: with agents

```execute ([agent action1,
          agent action2 (...),
          agent action3 (...)])
if glitch then
  warning (glitch_message)
end```
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)

Iteration

- Consider
  
  my_integer_list: LIST [INTEGER]

- in a class C that has the function
  
  is_positive (x: INTEGER): BOOLEAN is
  
  -- 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
  
  \[ my\_employee\_list: \text{LIST} \{\text{EMPLOYEE}\} \]

- where class \text{EMPLOYEE} has the feature
  
  \[ \text{is\_married}: \text{BOOLEAN} \]
  
  -- Does this object represent a married employee?

- To test that all employees in a list are married:

  \[
  \text{all\_married} := \text{my\_employee\_list}\text{.for\_all (agent \{\text{EMPLOYEE}\}.is\_married)}
  \]

Target or argument open

- Compare the two examples (both in a class \text{C}):

  \[
  \text{my\_integer\_list}: \text{LIST} \{\text{INTEGER}\} \\
  \text{my\_employee\_list}: \text{LIST} \{\text{EMPLOYEE}\} \\
  \text{is\_positive}(x: \text{INTEGER}): \text{BOOLEAN} \quad \text{-- In class C.} \\
  \text{is\_married}: \text{BOOLEAN} \quad \text{-- In class EMPLOYEE} \\
  \]

  -- Abbreviated as
  
  \[
  \text{my\_integer\_list}\text{.for\_all (agent \text{is\_positive})} \\
  \text{my\_employee\_list}\text{.for\_all (agent \{\text{EMPLOYEE}\}.is\_married)}
  \]

An EiffelBase contract (class HASH\_TABLE)

\[
\text{extend} \ (\text{new: G; key: H})
\]

\[
\text{-- Assuming there is no item of key key, insert new with key; set inserted.}
\]

\[
\text{require}
\]

\[
\text{not\_key\_present: not has (key)}
\]

\[
\text{ensure}
\]

\[
\text{insertion\_done: item (key) = new} \\
\text{key\_present: has (key)} \\
\text{inserted: inserted} \\
\text{one\_more: count = old count + 1}
\]

Agents’ potential for contracts

- Express general properties such as "none of the elements from positions 1 to count = 1 have been changed".

Normal call vs. agent

- Normal call
  \[ a0.f(a1, a2, a3) \]
- Agent call (expression): preface it by keyword 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:
  \[ u.call([\]) \]
- Recall original name of agents: "delayed calls".

End of lecture 12