
Dominique.Colnet@loria.fr

http://SmartEiffel.loria.fr
Lisp
J. McCarthy

Smalltalk
A. Goldberg
72-80 Squeak

Eiffel
B. Meyer
86-89-ECMA

SmartEiffel

OCAML
96


Simula-67
O. J. Dahl K. Nygaard

Ada
R. Milner
78-84

Ada
83-9X

Self
D. Ungar
87

ML

CAML
87

OCAML
96

SmallEiffel
July 1995
-0.99 to -0.75

Lisaac
2003

“Les langages à objets”
G. Masini, A. Napoli, D. Colnet, D. Léonard, K. Tombre

http://SmartEiffel.loria.fr
Eiffel

Initial Author and Origins

Object-Oriented Software Construction

Bertrand Meyer

1988 Prentice Hall

http://SmartEiffel.loria.fr
SmartEiffel

Goals / Market

Quality Software Design
Re-Usability
Security
Efficiency

Large Team / Software

Keep those goals in mind.

http://shootout.alioth.debian.org

http://SmartEiffel.loria.fr
Efficiency: runtime requirements

integer: INTEGER
string1: STRING
string2: STRING

integer := 2
string1 := “foo”
string2 := Void

Expanded object
Eiffel Software Engineering

- Strong Static Typing (Genericity)
- Design by Contracts (Assertions)
- Truly Object-Oriented (Class Based)
- Multiple inherit / insert
- Powerful Exportation Rules
- Anchored Type Definition
- Infix / Prefix Operator Definition
- frozen Methods
- Abstract Class / Methods

http://SmartEiffel.loria.fr
Eiffel Software Engineering

- *once* Methods
- *obsolete* Class / Methods
- *agents* (Code Manipulation)
- low level interface with C
- ..., 
- SCOOP (concurrency)?

More than a Language!
All written in Eiffel (new technology)

Eiffel/S 1.3

SmallEiffel #1

SmallEiffel #2

Eiffel/S 1.3

SmallEiffel

SmallEiffel

BOOTSTRAP PROCESS

1997 data

9 minutes

56 seconds

35 seconds

56 / 35 = 1.6

http://SmartEiffel.loria.fr
The *Hello World* example

```eiffel
class HELLO
   -- My first class.
create
   world

feature

   world is
   do
      io.put_string("Hello World !\%N")
   end

end -- Of my HELLO class.
```

*Looks like Pascal / Ada.*
easy-to-read syntax

my_procedure (arg1: INTEGER; arg2: STRING) is
  -- The purpose of this procedure is just to show the syntax.
local
  i: INTEGER
do
  if arg1 > 0 then
    from
    i := arg1
    until
    i = 0
    loop
      print(arg2.to_string + "%N")
      i := i - 1
    end
  else
    print("Easy to read for beginners too.")
  end
end
Class declaration

deferred → class → Class_name → Formal_generic_list
expanded

obsolete → Manifest_string

inherit → Parent_list

insert → Parent_list

create → Creation_clause

feature → Feature_clause

invariant → Assertion

end → Class_name

Subtyping
multiple inheritance

Implementation
multiple inheritance

New
2004 - 2006
Inherit

Subtyping

Polymorphism

Insert

Implementation

No Polymorphism

\[ \text{ANY} : \text{the only one class with no ancestor (the common ancestor)}. \]

\[ \text{Inherit} + \text{Insert} = \text{Acyclic Directed Graph}. \]
Inherit relationship:
- B inherits from ANY
- D inherits from ANY
- D inherits from B
- D inherits from C
- E inherits from C
- F inherits from ANY

Insert relationship:
- C inserts ANY
- E inserts ANY
- F inserts B
- F inserts C
- F inserts D
Typing and Checking Policy

- What can be assigned into what?
- What type can be used when some method or attribute is overridden?
- What type can be used in case of a generic derivation?
- What about constrained genericity?
Typing and Checking Policy

• Rule 1 (assignment): *An expression of type* $A$ *can be assigned into a variable of type* $B$ *if and only if* $A$ *and* $B$ *are the same type or* $A$ *inherits from* $B$.

• Rule 2 (argument passing): *An expression of type* $A$ *can be passed as an argument of formal type* $B$ *if and only if* $A$ *and* $B$ *are the same type or* $A$ *inherits from* $B$. 
Typing and Checking Policy

• Rule 3 (redefinition under inheritance): When overriding an inherited method or attribute, the type of any argument and/or of its result can be replaced covariantly with a type that inherits from the replaced type (i.e. a subtype).

CAT calls are still there!
Typing and Checking Policy

• Rule 4 (redefinition under insertion): *When overriding an inserted method or attribute, the type of any argument and/or of its result can be replaced covariantly with a type that inherits from or inserts the replaced type.*

No CAT calls here!
-- Somewhere in a routine.

local
    point: POINT

do
    create point.with(2.5, 6.7)
    point.translate(1.1, 2.2)
    point.display_on(std_output)
...

Check for the traditional
POINT class in our tutorial.

class INTEGER
class STRING
class CHARACTER
class BOOLEAN
class REAL
class ...
class ...

-- Somewhere in a routine.

local
    string: STRING; integer: INTEGER

do
    integer := 2 + 2
    string := integer.to_string
...

http://SmartEiffel.loria.fr
Uniform access.

Function Call

\[
x := y.foo
\]

Attribute Read

\[
x := y.foo
\]

No direct write permission on attribute (Smalltalk way).
Uniform access.

**Function Call?**

\[ x := y.\text{foo} \]

Attribute Read?

*No direct write permission on attribute (Smalltalk way).*
Genericity

No Cast!

Works on all types.

Compile-Time checks!

local
   integer: INTEGER
   list: LINKED_LIST[INTEGER]
do
   create list.make
   list.add_last(1)
   integer := list.last
...

local
   string: STRING
   list: LINKED_LIST[STRING]
do
   create list.make
   list.add_last("foo")
   string := list.last
...

local
   string: STRING
   array: ARRAY[STRING]
   list: LINKED_LIST[ARRAY[STRING]]
do
   create array.make(1, 1)
   array.put("foo")
   create list.make
   list.add_last(array)
   string := list.last.item(1)
...
Design-by-Contract

Automatic Documentation Extraction.

class STRING

put (c: CHARACTER; i: INTEGER) is
  -- Put `c' at index `i'.
  require
    valid_index(i)
do
  storage.put(c, i - 1)
ensure
  item(i) = c
end

remove_last (n: INTEGER) is
  -- Remove `n' last characters.
  -- If `n' >= `count', remove all.
  require
    n >= 0
do
  if (n > count) then
    count := 0
  else
    count := count - n
  end
ensure
  count = (0).max(old count - n)
end
class STRING
-- Resizable STRINGs of CHARACTERs indexed from `1` to `count`.
...
...
...

count: INTEGER
    -- Actual length.

capacity: INTEGER
    -- Capacity of the `storage` area.

invariant
    count >= 0
    count <= capacity

end -- of class STRING
Now looking at the interface of class STRING using command short ... and then, the SmartEiffel web site.
deferred class COLLECTION[E_]

ARRAY[E_]
RING_ARRAY[E_]
FAST_ARRAY[E_]
LINKED_LIST[E_]
TWO_WAY_LINKED_LIST[E_]

Design-by-Contract + inherit

Assertions are written only once!
Now looking at the interface of class COLLECTION[E].
Relationship between modules (i.e. who can do what.)
Anchored Types

collection: ARRAY[STRING]

is_equal (other: like Current): BOOLEAN is
...

Just change the anchor ... others will follow automatically.

storage: LINKED_LIST[INTEGER];

foo: like storage;

bar: like foo;

clone: like Current is
...

Smart Typing !

http://SmartEiffel.loria.fr
Now looking at the source code of class COLLECTION[E].
class DICTIONARY [V, K -> HASHABLE]
--
-- Associative memory. Values of type `V' are stored using Keys of type `K'.
--
feature
  ...
  ...
end -- DICTIONARY

Documentation + Validation + Testing + Performances!
**class** DICTIONARY [V, K -> HASHABLE]

at (key: K): V is
  require
    has(key)
  do not_yet_implemented
end

put (value: V; key: K) is
  require
    key /= Void
  do not_yet_implemented
ensure
  value = at(key)
end

has (key: K): BOOLEAN is
  require
    key /= Void
  do not_yet_implemented
end

add (value: V; key: K) is
  require
    not has(key)
  do not_yet_implemented
ensure
  count = 1 + old count
  value = at(key)
end
class DICTIONARY [V, K -> HASHABLE]

at (key: K): V is
  require
    has(key)
  do not_yet_implemented
end

put (value: V; key: K) is
  require
    key /= Void
  do not_yet_implemented
ensure
  value = at(key)
end

has (key: K): BOOLEAN is
  require
    key /= Void
  do not_yet_implemented
end

add (value: V; key: K) is
  require
    not has(key)
  do not_yet_implemented
ensure
  count = 1 + old count
  value = at(key)
end

http://SmartEiffel.loria.fr
How can Eiffel handles code as data ?

Ordinary valid code example used for next slides

```eiffel
object.method(arg1, arg2)
```
code := agent object.method(?, arg2)

step-1  Capture of given operands in the agent object.

step-2  Passing missing operands and fires the method.

code.call([arg1])

Agents at Work

http://SmartEiffel.loria.fr
code := agent object.method(arg1, ?)

step-1 Capture of given operands in the agent object.

step-2 Passing missing operands and fires the method.

code.call([arg2])

Agents at work
code := agent object.method(?, ?)

step-1 Capture of given operands in the agent object.

code.call([arg1, arg2])

step-2 Passing missing operands and fires the method.

Agents at work
code := agent {TOBJECT}.method(?, ?)

step-1 Capture of given operands in the agent object.

code.call([object, arg1, arg2])

step-2 Passing missing operands and fires the method.

Agents at work
\[
\text{code} := \textbf{agent} \ \text{object.method}(\text{arg1, arg2})
\]

\textbf{step-1} Capture of given operands in the agent object.

\[
\text{code.call([])}
\]

\textbf{step-2} Passing missing operands and fires the method.

Agents at Work
http://SmartEiffel.loria.fr

Thank you for your attention.

Contact: Dominique.Colnet@loria.fr

Ideal for CS teaching too.
Give it a try!

Questions?
CAT call example (1/3)

class POINT
  x, y: REAL

  binary ( other: POINT ) is
  do
    ... other.x ...
  end

  ...

end – class POINT
class PIXEL

inherit POINT redefine binary

status: BOOLEAN

binary ( other: PIXEL ) is
  do
    ... other.status ...
  end

...
CAT call example (3/3)

```plaintext
local
    point1, point2: POINT;

do
    point1 := create {POINT};
    point2 := create {PIXEL};
    point2.binary(point1)

... Bang ! ...
```
CAT call (generics Java version)

```java
public class JavaCATcall {
    public static void main(String[] args) {
        Integer[] integerArray = new Integer[10];
        Object[] objectArray = null;

        objectArray = integerArray;
        objectArray[1] = "BANG!";
    }
}
```