

Early Experience with Language Interoperability

Porting the BETA language to Java and .NET

Peter Andersen

Ole Lehrmann Madsen

Center for Pervasive Computing
Aarhus University

Content

- Background & goals
- Virtual machines
 - ◆ The Java- and .NET platforms
- BETA
- Mapping BETA to Java and .NET
- Experience with the platforms
- Language interoperability
- Demo
- Conclusion

Background

- Modern language implementations:
 - ◆ Virtual machines
 - ◆ Bytecode
 - ◆ Just-in-time compilers
 - ◆ Run-time type information
 - ◆ Verification of code before execution
- Main stream platforms
 - ◆ Java virtual machines
 - ◆ Microsoft .NET

Main stream OO languages

- Java
 - ◆ The one and only language for the Java-platform
- C# and VB.Net
 - ◆ The dominant languages for .NET
- A family of languages
 - ◆ Java/C#-like languages
 - ◆ And many more

Goals I

- Implement BETA for
 - ◆ .NET
 - ◆ Java-VM
- Investigate the suitability of Java and .NET as platforms for general language implementation

Goals II

- Investigate language interoperability
- Traditional language interoperability for procedural languages
 - ◆ Procedure libraries in different languages
 - ◆ Common calling sequence for procedures
- Language interoperability for OO languages
 - ◆ Uses of classes from different languages
 - ◆ Inheritance from classes in different languages
 - ◆ An explicit goal for .NET
 - ◆ No expectations for Java

Goals III

- Provide a common language for .NET and Java-VM
- (Component architecture)
 - ◆ Web services – SOAP
- (BETA for PDA's)
 - ◆ .NET compact framework

Challenges I

Mapping of BETA to .NET & Java-VM

- .NET & Java-VM are typed virtual machines modelled from Java and C#
- BETA is much more general
- One issue: just finding a mapping

Challenges II

Language interoperability

- BETA should be able to use classes from other languages
- Other languages should be able to use BETA classes (patterns)
- BETA should be able to inherit classes from other languages
- Other languages should be able to inherit from BETA
- The BETA mapping should be 'nice' when seen from other languages

Java/C# language model

- A program is a collection of classes
- A class defines
 - ◆ Data-items/attributes
 - ☞ an attribute has a type
 - ◆ Methods
 - ☞ a method has arguments and a possible result value
- Classes may be nested
 - ◆ Java – real nesting
 - ◆ C# - only a scope mechanism
- No nesting of methods
- Dynamic exceptions
- Concurrency in the form of threads

BETA

- Class and method unified into the pattern mechanism
- General nesting of patterns
- INNER instead of super
- Genericity in the form of virtual patterns
- Multiple return values
- Active objects
 - ◆ Coroutines and concurrency
 - ◆ Basis for writing schedulers
- No constructors
- No dynamic exceptions (yet)

Challenges with BETA mapping

- Pattern mapping to class and methods
- Nested patterns
- Nested procedures
- Enter-do –exit-semantics
- Inner
- Virtual patterns – class and procedure
- Multiple return values
- Leave/restart out of nested method activations
- Coroutines and concurrency
- Pattern variables
- Basic values – signed/unsigned

The mapping

- Generating code for Java- and .NET corresponds to making a BETA source mapping into Java- and/or C#-source code
- In the following the implementation of BETA is shown as a mapping into a Java/C#-like language

BETA example

Calc:

```
(# R: @integer;
```

```
  set:
```

```
    (# V: @integer enter V do V → R #);
```

```
  add:
```

```
    (# V: @integer enter V do R+V → R exit R #);
```

```
  #);
```

```
C: @Calc; X: @integer;
```

```
12 → C.set;
```

```
5 → C.add → X
```

Naive mapping into Java/C#

```
Class Calc extends Object  
{ int R;  
  void set(int V) { R = V; };  
  int add(int V)  
    { R = R + V; return R; }  
}
```

```
Calc C = new Calc(); int X;  
C.set(12);  
X = C.add(5);
```

Instances of add

- More complex mapping needed
- Possible to create instances of pattern add


```
C: @Calc; X: @integer;
```

```
A: ^C.add;
```

```
&C.add[] → A[];
```

```
6 → A → X
```

Creation of an instance of C.add



Class add

```
class add extends Object
{ Calc origin ;
  int V;
  void add(Calc org) {origin = org; }
  void enter(int a) { V = a; }
  void do() { origin.R = origin.R + V };
  int exit() { return origin.R; }
}
```

```
C: @Calc;
X: @integer;
12 → C.set;
5 → C.add → X
```

```
add A; int X;
A = new add(C);
A.enter(5);
A.do();
X = A.exit();
```

Calc with call-add operation

```
Class Calc extends Object
{ int R;
  void set(int V) { R = V; };
  int add(int V)
  { add A;
    A = new add(this);
    A.enter(V);
    A.do()
    return A.exit();
  }
}
```

We hope the JIT compilers inlines these calls!

```
Calc C = new Calc(); int X;
C.set(12);
X = C.add(5);
```

Calc with call-add & new-add

Class **Calc** extends **Object**

```
{ int R;
```

```
void set(int V) { R = V; };
```

Call method
for add

```
int add(int V) { ...; return A.exit(); }
```

New method
for add

```
→ add add () { return new add(this); }  
}
```

```
Calc C = new Calc(); int X; add C.A;
```

```
C.set(12);
```

```
X = C.add(5);
```

```
A = C.add();
```

General scheme

```
myClass:
```

```
(# ...;  
  f1: (# ... #);  
  f2: (# ... #);  
  f3: (# ... #)  
#)
```

```
class myClass extends Object
```

```
{ ...;  
  T1 f1(...) { ... } // call f1  
  f1 f1() { ... } // new f1  
  T2 f2(...) { ... } // call f2  
  f2 f2() { ... } // new f2  
  T3 f3(...) { ... } // call f3  
  f3 f3() { ... } // new f3  
}
```

With language restrictions

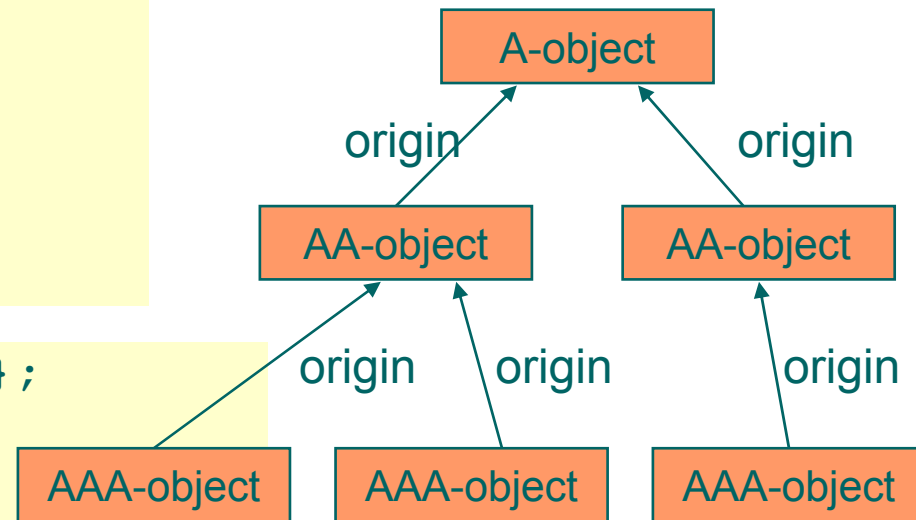
```
myClass: class
  (# ...;
    f1: proc (# ... #);
    f2: proc (# ... #);
    T: class (# ... #)
  #)
```

```
class myClass extends Object
{ ...;
  T1 f1(...) { ... } // call f1
  T2 f2(...) { ... } // call f2
  T T() { ... }      // new T
}
```

Nested/inner classes in general

```
A: (# ...  
    AA: (# ...  
        AAA: (# ... #)  
        #)  
    #)
```

```
class A extends Object { ... };  
class AA extends Object  
{ A origin,  
  void AA(A org) { origin = org; }  
}  
class AAA extends Object  
{ AA origin;  
  void AAA(AA org) { origin = org; }  
}
```



Method combination with inner

Calc:

```
(# ...  
  add:  
    (# V: @integer  
      enter V  
      do R+V → R; inner  
      exit R  
    #);  
  ...  
#)
```

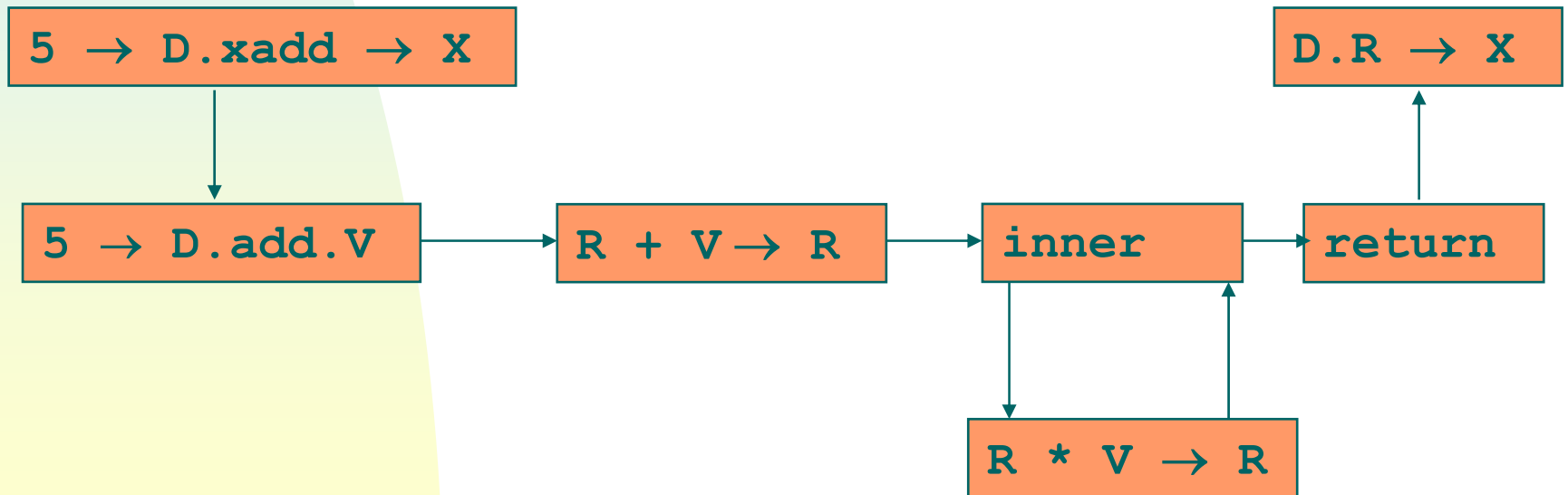
Xcalc: Calc

```
(#  
  xadd: add (# do R * V → R #)  
#)
```

Semantics of inner

```
D: @Xcalc; X: @integer  
12 → D.set;  
5 → D.xadd → X
```

```
add: (# V: @integer  
      enter V  
      do R+V → R; inner  
      exit R  
      #);  
xadd: add (# do R * V → R #)
```



Class add with inner

```
class add extends Object
{
  ...
  void do()
  {
    origin.R = origin.R + V;
    do_1(); // inner
  };
  void do_1();
}

```

```
class xadd extends add
{
  ...
  void do_1()
  {
    origin.R = origin.R * V;
  };
  ...
}

```

Inner at several levels

```
A: (# do X1; inner; Y1 #);  
AA: A (# do X2; inner; Y2 #);  
AAA: AA(# do X3; inner; Y3 #)
```

```
class A: extends Object  
  { void do() { X1; do_1(); Y1; };  
    void do_1()  
  }  
class AA: extends A  
  { void do_1() { X2; do_2(); Y2; };  
    void do_2()  
  }  
class AAA: b AA  
  { void do_2() { X3; do_3(); Y3 };  
    void do_3()  
  }
```

Virtual patterns

- BETA has virtual patterns
- Virtual patterns used as virtual procedures
 - ◆ Like virtual methods in Java/C#
- Virtual patterns used as virtual classes
 - ◆ A mechanism for defining a class parameterized by another class
- No counter parts in Java/C#
 - ◆ Proposal on its way for Java

Pattern List

```
List:  
  (# element:< Object;  
    insert:  
      (# e: ^element  
        enter e[] do ... #);  
    ...  
  #)  
StudentList: List  
  (# element::< Student;  
  #)
```

```
L: @List;  
aPerson[] → L.insert;  
aStudent[] → L.insert;
```

```
S: @StudentList;  
aPerson[] → S.insert;  
aStudent[] → S.insert;
```

Illegal

Detected by the
BETA compiler



Virtual class in Java/C#

```
class List extends Object
{ ...
  void insert(Object e) { ... }
}
```

```
class StudentList extends List
{ ...
  void insert(Object e)
  { Student e1 = (Student) e;
    ...
  }
}
```

Multiple return values

```
Calc:
```

```
(# R,Rx: @integer;  
...;  
getReg: (# exit (R,Rx) #) ;  
#) ;
```

```
C: @Calc; x,y: @integer;
```

```
C.getReg → (x,y) ;
```

- A field is added to the class for each return value
- At the calling site, the exit fields are fetched

Leave/restart of nested method calls

```
foo:
  (#
do L: (# bar: (# do leave L #);
      go: (# do restart L #)
do (if b then
    bar
    else
      go
    if)
  #)
#);
```

To be implemented using dynamic exceptions

Remaining issues

- Active objects
 - ◆ Coroutines and concurrency
- Pattern variables
 - ◆ Classes and methods as first-class values
- Basic values – signed/unsigned
- And lots of details

Compiler organization

- Frontend
 - ◆ Parser, abstract syntax trees, semantic checking, semantic analysis, abstract code generation
 - ◆ Platform independent
- Backend
 - ◆ Code generation for a specific target
 - ◆ Platform dependent
 - ☞ Sun Solaris
 - ☞ Linux
 - ☞ Windows
 - ☞ Macintosh

Compiler reorganization

- Frontend
 - ◆ Generate (typed) stack code
- Backend
 - ◆ Abstract stack code
 - ◆ Specific targets
 - ☞ .NET
 - ☞ Java-VM

Platform issues I

- No static link on stack frames
 - ◆ Nested procedures/methods cannot be implemented using the stack
- Very rigid typing of class-fields
 - ◆ And method-locals on .NET
- Constructor initialization
 - ◆ Impossible to make setups before calling super constructor
- General exit out of nested method calls
- No support for covariant arguments
- No support for covariant return types
- Active objects

Platform issues II

- .NET class references must be fully qualified, including location of binding
 - ◆ Problems with separate compilation
- Large number of classes generated due to the generality of BETA patterns
 - ◆ Java class files can only contain one (non-static) class per file
 - ◆ Lots of class files are generated
- .NET assemblies can contain any number of classes

Platform issues III (minor)

- No swap and dup_x1/x2 on .NET
- Requirements for fully typed local variables in .NET.
 - ◆ Means that swap cannot be implemented in backend using local variables unless type of two top-of-stack objects are known.
- Type of field: super/this class

Platform advantages

- Well-defined run-time format
 - ◆ More than just a calling convention for procedures
- Memory management
 - ◆ Storage allocation
 - ◆ Object format & method activation format
 - ◆ Garbage collection
- Efficient code generation
- We will have BETA for all Java- and .NET platforms
- Can this be utilized efficiently?
 - ◆ If simple and direct mapping – yes
 - ◆ If complex mapping – no?
 - ◆ Inlining may help
 - ☞ Can rely on generating methods that the JIT inlines

Language interoperability I

- Java/C# class inherited from BETA
 - ◆ Straight forward
 - ◆ Since name & type-based,
 - ☞ just specify the methods you will use
 - ◆ Simpler than for COM
 - ☞ COM uses offsets
 - ☞ All preceding methods must be declared
- BETA pattern inherited from Java/C# class
 - ◆ Also straight forward
 - ◆ Issues with default constructor
 - ☞ BETA object requires surrounding object (origin)

Language interoperability II

■ Issues

- ◆ Libraries and frameworks in Java/C#
- ◆ Java-string, C#-String, BETA-text
 - ☞ Automatic coercion implemented
- ◆ Overloading
- ◆ Constructors

■ External class interface

- ◆ Interface syntax needed – currently clumsy
- ◆ External name & location
- ◆ Automatic include of interface files
 - ☞ .NET assemblies
 - ☞ Java class-files

Demo

- Inheritance in BETA of external class
 - ◆ Java Applet
- Inheritance of BETA pattern in Java and C#
- Implementation of program using Google web-service
 - ◆ In C# and Java
- Debugging of Google search in Visual Studio.Net

Conclusion

- Not trivial to map all parts of BETA
- We still need to measure efficiency
- Generation of bytecode: straightforward
 - ◆ Java issue: no standard assembler
- Generation of type information: a lot of work
- No real problems in supporting both platforms
- Use of Visual Studio to edit and debug BETA programs
 - ◆ Impressive
 - ◆ Source level debugging, breakpoints, object-inspection, ...

Language interoperability

- .NET/C#: seems to work as promised
 - ◆ Class instance
 - ◆ Inheritance
 - ◆ Visual Studio
- Java: works well too – to our surprise
 - ◆ Class instance
 - ◆ Inheritance
 - ◆ Has not found any (free) SDE's that work as well as Visual Studio (which is expensive, but stand alone graphical debugger part of free .NET framework)
- It seems realistic to mix libraries and frameworks from different languages
- Easier to introduce new languages

BETA Language changes

- Constructors
- Restricting a pattern as class or method
- Add properties as in C#
- Issue with super/inner call sequencing
- Overloading?

Status

- Large subset of BETA has been implemented
- To be done
 - ◆ General leave/restart
 - ◆ Active objects
 - ◆ Pattern variables
 - ◆ Lots of details
 - ◆ Porting BETA libraries and frameworks to Java and .NET

Acknowledgements

- Project in
 - ◆ Center for Pervasive Computing
 - ◆ www.pervasive.dk
- Supported by
 - ◆ Microsoft Denmark
 - ◆ Sun Microsystems Denmark
- Ideas borrowed from
 - ◆ Kresten Krab Thorup
 - ◆ Henry Michael Lassen
 - ◆ Kim Falk Jørgensen
- Peter Andersen - datpete@daimi.au.dk
- Ole Lehrmann Madsen - olm@daimi.au.dk