Expressive and Safe Static Reflection with MorphJ

Yannis Smaragdakis
UMass, Amherst

Shan Shan Huang
Georgia Tech

Motivation

- A synchronization proxy: (from Java Collections)

```
class SynchronizedSet implements Set {
  final Set l;
  final Object mutex;

  public int size () {
    synchronized(mutex) { return l.size();    }
  }
  public boolean remove (int i) {
    synchronized(mutex) { return l.remove(i); }
  }
  ...
  // repeat for all methods in Set
}
```

Wouldn't it be great if I can say...

```
“Synchronization” for any type T {
  for each method of T
    declare a method with the same signature {
      synchronize on a mutex
      delegate the call to the real object of T
    }
}
```

- We call this: ***Structural Abstraction***

MorphJ Offers Structural Abstraction

```
class SynchronizeMe<interface T> implements T {
  final T me;
  final Object mutex;

  <R,A*>[m] for ( R m (A) : T.methods )
  public R m (A args) {
    synchronized(mutex) { return me.m(args); }
  }




}
```

Morphing Your Code

```
class SynchronizeMe<interface T>
  final T me;
  final Object mutex;

  <R,A*>[m] for ( R m (A) : T.methods )
  public R m (A args) {
    synchronized(mutex) { return me.m(args); }
  }
}
```

```
interface List {
  int size();
  Object get (int i);
  ...
}
```

```
class SynchronizeMe<List> implements List {
  final List me;
  final Object mutex;

                                        me.size();

    synchronized(mutex) { return me.get(i);
  }
  ...
}
```

MorphJ: 18 LOC

Hardcoded Proxies: 315 LOC

What Do People Do In Practice?

- Meta-programming techniques
  - Reflection, combined with bytecode engineering, templates
  - Aspect-Oriented Programming
  - etc.
- No ***separate type checking!***

MorphJ Offers Separate Type Checking!

```
class Foo {
  public float bar (int a, int b) {...
  public int   bar (int a)        {...
}
```

```
class CallWithMax<class T> extends T {

  <R,A>[m] for ( public R m (A) : T.methods )
  public R m ( A arg1, A arg2 ) {
    if ( arg1.compareTo(arg2) > 0 )
      return super.m(arg1);
    return super.m(arg2);
  }
}
```

```
class CallWithMax<Foo> extends Foo {
  public int bar (int arg1, int arg2)
    if ( arg1.compareTo(arg2) > 0 )
      return super.bar(arg1);
    return super.bar(arg2);
  }
}
```

Type Checking Idea

Challenges in Type Checking

```
class CallWithMax<T> extends T {

  <R,A>[m] for ( public R m (A) : T.methods )
  public R m ( A arg1, A arg2 ) {
    if ( arg1.compareTo(arg2) > 0 )
      return super.m(arg1);
    return super.m(arg2);
  }

}
```

- Unknown number of methods/fields
- Unknown method/field names
- Unknown type signatures of methods/fields
- Unknown supertypes

- Can we even do this?

YES,WE CAN !

Main Idea

- Morphed members represented as abstract set
  - defined by: (superset, pattern)
- We can still answer subset/disjointness on abstract sets based on patterns
  - declaration uniqueness = disjointness of declaration-set with all others
    - two-way unification of patterns
  - reference validity = use-set subset of declaration-set
    - one-way unification of patterns

Easy-to-Show Validity

-
  ```
  class EasyReflection<X> {
    X x; ... // code to set x field

    [n] for(void n (int): X.methods )
    void n (int i) { x.n(i); }
  }
  ```

Validity in Full Glory

-
  ```
  class Declaration<Y> {
     <R,B*>[m] for(R m (B): Y.methods )
     void m (B b) { ... }
  }
  ```

-
  ```
  class Reference<X> {
     Declaration<X> dx; ... //code to

     <A*>[n] for(String n (A): X.metho
     void n (A a) { dx.n(a); }
  }
  ```
  - decl-set subsumes use-set if patterns unify (*one way*)

```
Y <: X
R ↦ String
B ↦ A
m ↦ n
```

Nested Patterns

Adding more expressiveness

Making Patterns More Expressive

- Field getters?

```
class AddGetter<class X> extends X {
  <T>[f] for ( T f : X.fields )
  public T get#f () {
    return f;
  }
}
```

```
class Foo {
  Meal lunch;
  ...
  boolean getlunch() {
    return isNoon() ? true : false;
  }
}
```

```
class AddGetter<Foo> extends Foo{
  public Meal getlunch() {
    return lunch;
  }
}
```

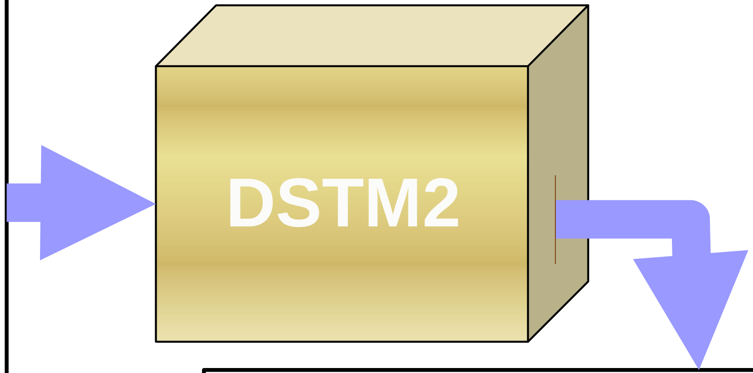
Negative Nested Pattern

```
class AddGetter<class X> extends X {
  <T>[f] for ( T f : X.fields )
  no get#f () : X.methods )
  public T get#f () {
    return f;
  }
}
```

Case Study: DSTM2

- Software transactional memory framework, by *Herlihy* et al.

```
interface INode {
  int    getValue();
  void   setValue(int);

  INode getNext();
  void   setNext(INode);
}
```

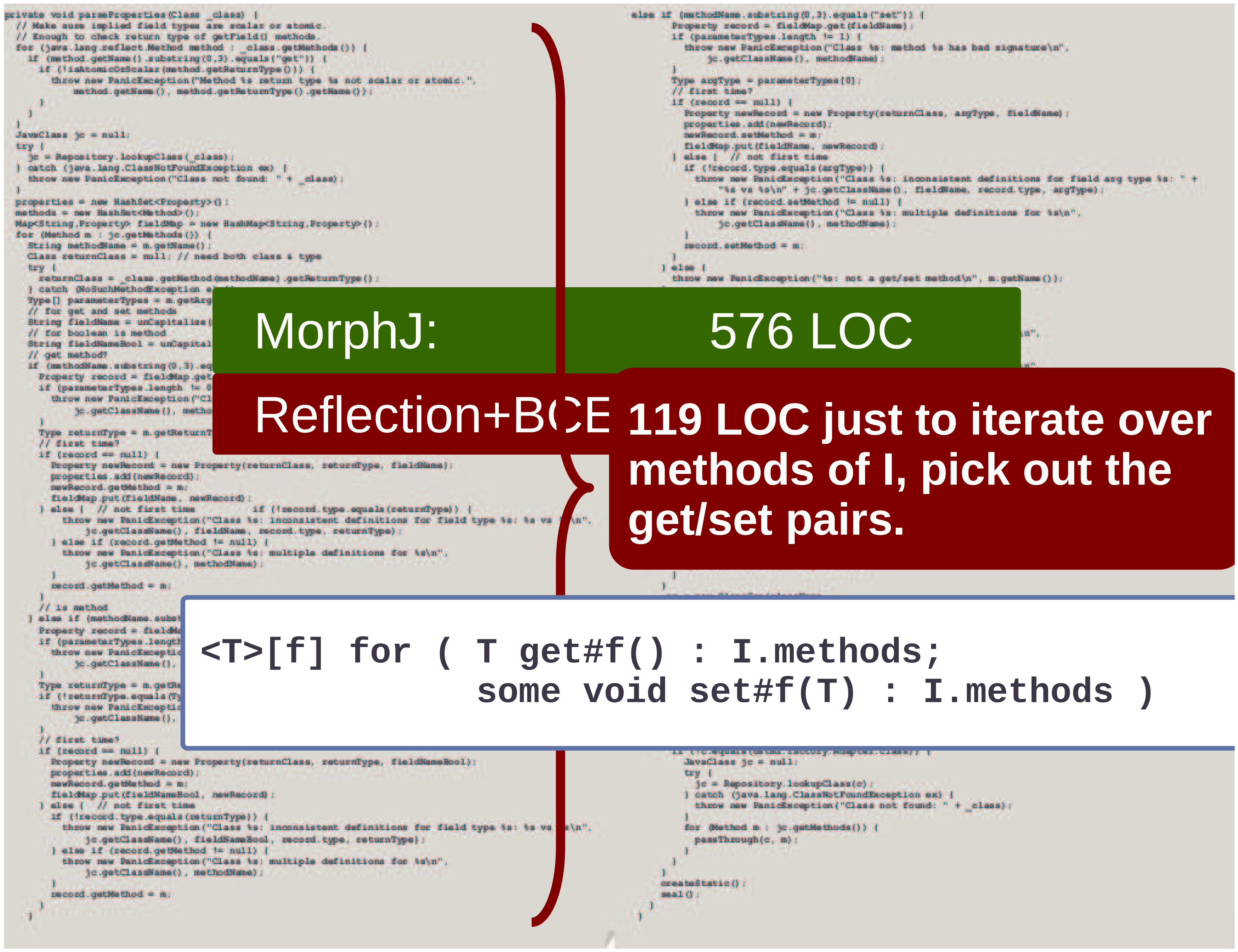


```
class AtomicNode
            implements INode {
  int    value;
  int    getValue() { ... }
  void   setValue(int) { ... }

  INode next;
  INode getNext() { ... }
  void   setNext(INode) { ... }
}
```

- Implemented using reflection, BCEL

DSTM2 with MorphJ

```
public class Atomic<interface I> implements I {

  <T>[f] for ( T get#f() : I.methods;
               some void set#f(T) : I.methods ) {|
  T f;
  public void set#f(T value) {
    ... // open transaction.
    f = value;
    ... // resolve conflict.
  }
  public T get#f() { ... }
  |}
  ...
}
```



```
private void parseProperties(Class _class) {
  // Make sure implied field types are scalar or atomic.
  // Enough to check return type of getField() methods.
  for (java.lang.reflect.Method method : _class.getMethods()) {
    if (method.getName().substring(0,3).equals("get")) {
      if (!isAtomicOrScalar(method.getReturnType())) {
        throw new PanicException("Method %s return type %s not scalar or atomic.",
          method.getName(), method.getReturnType().getName());
      }
    }
  }
  JavaClass jc = null;
  try {
    jc = Repository.lookupClass(_class);
  } catch (java.lang.ClassNotFoundException ex) {
    throw new PanicException("Class not found: " + _class);
  }
  properties = new HashSet<Property>();
  methods = new HashSet<Method>();
  Map<String,Property> fieldMap = new HashMap<String,Property>();
  for (Method m : jc.getMethods()) {
    String methodName = m.getName();
    Class returnClass = null; // need both class & type
    try {
      returnClass = _class.getMethod(methodName).getReturnType();
    } catch (NoSuchMethodException e
    Type[] parameterTypes = m.getArg
    // for get and set methods
    String fieldName = unCapitalize(
    // for boolean is method
    String fieldNameBool = unCapital
    // get method?
    if (methodName.substring(0,3).eq
      Property record = fieldMap.get
      if (parameterTypes.length != 0
        throw new PanicException("Cl
          jc.getClassName(), metho
      }
      Type returnType = m.getReturnT
      // first time?
      if (record == null) {
        Property newRecord = new Property(returnClass, returnType, fieldName);
        properties.add(newRecord);
        newRecord.getMethod = m;
        fieldMap.put(fieldName, newRecord);
      } else {  // not first time          if (!record.type.equals(returnType)) {
          throw new PanicException("Class %s: inconsistent definitions for field type %s: %s vs  \n",
            jc.getClassName(), fieldName, record.type, returnType);
        } else if (record.getMethod != null) {
          throw new PanicException("Class %s: multiple definitions for %s\n",
            jc.getClassName(), methodName);
        }
        record.getMethod = m;
      }
      // is method
    } else if (methodName.subst
      Property record = fieldMa
      if (parameterTypes.lengt
        throw new PanicExcepti
          jc.getClassName(),
      }
      Type returnType = m.getRe
      if (!returnType.equals(Ty
        throw new PanicExcepti
          jc.getClassName(),
      }
      // first time?
      if (record == null) {
        Property newRecord = new Property(returnClass, returnType, fieldNameBool);
        properties.add(newRecord);
        newRecord.getMethod = m;
        fieldMap.put(fieldNameBool, newRecord);
      } else {  // not first time
        if (!record.type.equals(returnType)) {
          throw new PanicException("Class %s: inconsistent definitions for field type %s: %s vs  \n",
            jc.getClassName(), fieldNameBool, record.type, returnType);
        } else if (record.getMethod != null) {
          throw new PanicException("Class %s: multiple definitions for %s\n",
            jc.getClassName(), methodName);
        }
        record.getMethod = m;
      }
  }
```

```
    else if (methodName.substring(0,3).equals("set")) {
      Property record = fieldMap.get(fieldName);
      if (parameterTypes.length != 1) {
        throw new PanicException("Class %s: method %s has bad signature\n",
          jc.getClassName(), methodName);
      }
      Type argType = parameterTypes[0];
      // first time?
      if (record == null) {
        Property newRecord = new Property(returnClass, argType, fieldName);
        properties.add(newRecord);
        newRecord.setMethod = m;
        fieldMap.put(fieldName, newRecord);
      } else {  // not first time
        if (!record.type.equals(argType)) {
          throw new PanicException("Class %s: inconsistent definitions for field arg type %s: " +
            "%s vs %s\n" + jc.getClassName(), fieldName, record.type, argType);
        } else if (record.setMethod != null) {
          throw new PanicException("Class %s: multiple definitions for %s\n",
            jc.getClassName(), methodName);
        }
        record.setMethod = m;
      }
    } else {
      throw new PanicException("%s: not a get/set method\n", m.getName());
```

```
        JavaClass jc = null;
        try {
          jc = Repository.lookupClass(c);
        } catch (java.lang.ClassNotFoundException ex) {
          throw new PanicException("Class not found: " + _class);
        }
        for (Method m : jc.getMethods()) {
          passThrough(c, m);
        }
      }
    }
    createStatic();
    seal();
  }
}
```

MorphJ: 576 LOC

Reflection+BCE

119 LOC just to iterate over methods of I, pick out the get/set pairs.

```
<T>[f] for ( T get#f() : I.methods;
             some void set#f(T) : I.methods )
```

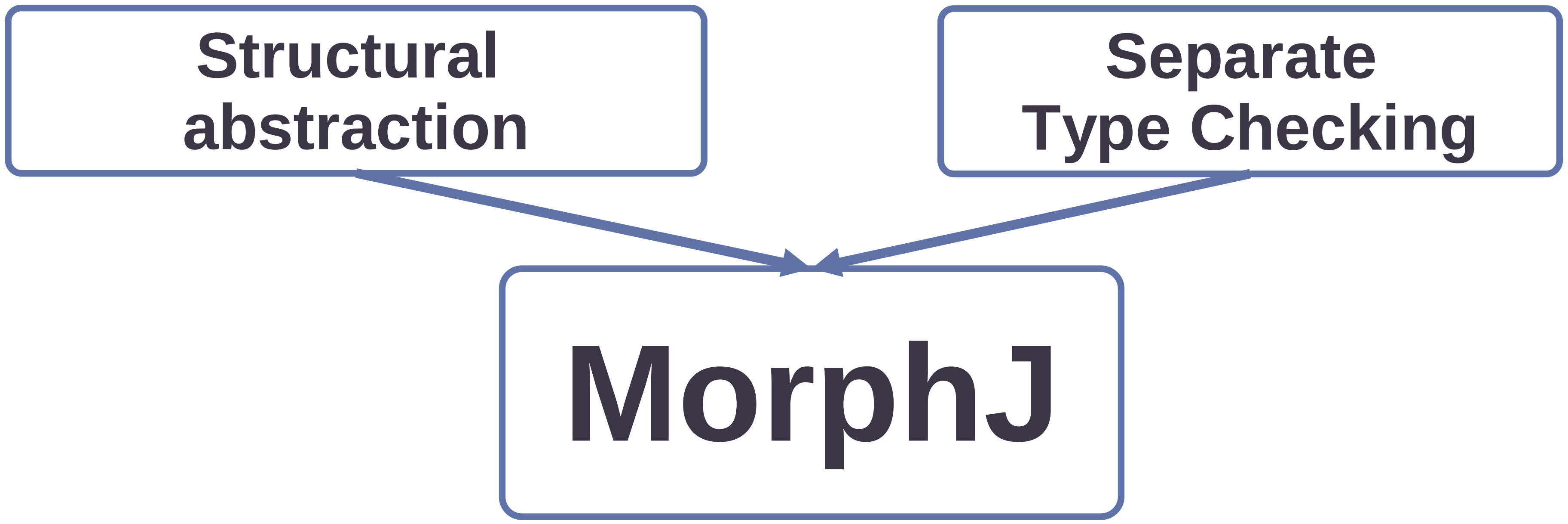
More In The Paper

- “**`if`**” patterns (in addition to “**`for`**”)
- “**`error`**” patterns (encode assumptions: like type casts)
- More examples
- Details of type system.

Related Work

- Compile-Time Reflection (*Fähndrich, Carbin, and Larus*)
- SafeGen (*Huang, Zook, Smaragdakis*)
- Genoupe (*Draheim, Lutteroth, Weber*)
- Staging languages: MetaML, MetaOCaml (*Calcagno, Taha, Sheard, et al.*)

In Summary



- MorphJ implementation available:
  - ***http://code.google.com/p/morphing/***