Modular Quasiquotes for Scala

Denys Shabalin

École Polytechnique Fédérale de Lausanne

2 July 2013

Q: What are quasiquotes?

Q: What are quasiquotes?

A: A composable syntactical abstraction that vastly simplifies manipulation of ASTs.

Compactness

Syntax	`case class Foo(bar: Baz)`

Compactness

Syntax	`case class Foo(bar: Baz)`
AST	`ClassDef(Modifiers(...), TypeName("Foo"), List(), Template(List(Select(Ident(TermName("scala")), TypeName("Product")), Select(Ident(TermName("scala")), TypeName("Serializable"))), emptyValDef, List(ValDef(Modifiers(...), TermName("bar"), Ident(TypeName("Baz")), EmptyTree), DefDef(Modifiers(), nme.CONSTRUCTOR, List(), List(List(ValDef(Modifiers(...), TermName("bar"), Ident(TypeName("Baz")), EmptyTree))), TypeTree(), Block(List(pendingSuperCall), Literal(Constant(())))))))`

Compactness

Syntax	`case class Foo(bar: Baz)`

AST	`ClassDef(Modifiers(...), TypeName("Foo"),`
	`List(), Template(List(Select(Ident(`
	`TermName("scala")), TypeName("Product")),`
	`Select(Ident(TermName("scala")), TypeName(`
	`"Serializable"))), emptyValDef, List(`
	`ValDef(Modifiers(...), TermName("bar"),`
	`Ident(TypeName("Baz")), EmptyTree),`
	`DefDef(Modifiers(), nme.CONSTRUCTOR,`
	`List(), List(List(ValDef(Modifiers(...),`
	`TermName("bar"), Ident(TypeName("Baz")),`
	`EmptyTree))), TypeTree(), Block(List(`
	`pendingSuperCall), Literal(Constant(())))))))`

Quasiquote	`q"case class Foo(bar: Baz)"`

Composability

```
// it's easy to combine quasiquotes

val tree = q"simple tree"
val another = q"if ($tree) foo else bar"
```

Composability

```
// it's easy to combine quasiquotes

val tree = q"simple tree"
val another = q"if ($tree) foo else bar"

// and they can also be used to
// decompose trees in the same fashion

tree match {
  case q"$obj.$member" =>
    // obj & member are now available in this scope
}
```

Expressiveness

```
// you can splice lists of elements into a tree with
// the help of special cardinality annotation

val args = List(q"a", q"b")
q"f(..$args)"
```

Expressiveness

```
// you can splice lists of elements into a tree with
// the help of special cardinality annotation

val args = List(q"a", q"b")
q"f(..$args)"

// equivalent to

q"f(a, b)"
```

Expressiveness

```
// you can splice lists of elements into a tree with
// the help of special cardinality annotation

val args = List(q"a", q"b")
q"f(..$args)"

// equivalent to

q"f(a, b)"

// and non-tree data types

val i = 0
q"f($i)"
```

Modularity

```
@quasiquote object λ {






}
```

Modularity

```
@quasiquote object λ {
  sealed abstract class Tree
  case class Abs(v: Var, body: Tree) extends Tree
  case class App(f: Tree, arg: Tree) extends Tree
  case class Var(name: String) extends Tree









}
```

Modularity

```
@quasiquote object λ {
  sealed abstract class Tree
  case class Abs(v: Var, body: Tree) extends Tree
  case class App(f: Tree, arg: Tree) extends Tree
  case class Var(name: String) extends Tree

  object parse extends StdTokenParsers {






  }
}
```

Modularity

```
@quasiquote object λ {
  sealed abstract class Tree
  case class Abs(v: Var, body: Tree) extends Tree
  case class App(f: Tree, arg: Tree) extends Tree
  case class Var(name: String) extends Tree

  object parse extends StdTokenParsers {
    lexical.delimiters ++= List("(", ")", "\", ".")
    def main   = rep1(parens | varr | abs | hole)        ^^ App
    def abs    = ("\\" ~> (varr | hole) <~ ".") ~ main) ^^ Abs
    def varr   = ident                                   ^^ Var
    def parens = "(" ~> main <~ ")"
  }
}
```

Modularity

```
// now we can use our custom quasiquotes to construct

import λ._

val id = λ"\x. x"
val f = λ"\v. $id v"
```

Modularity

```
// now we can use our custom quasiquotes to construct

import λ._

val id = λ"\x. x"
val f = λ"\v. $id v"

// and deconstruct our lambda-calculus trees

f match {
  case λ"\$arg. $body" =>
}
```

Summary

- Quasiquotes are an extremely powerful abstraction over ASTs
- Primary usage is to simplify manipulation of Scala trees
- However they can be generalized to arbitrary languages
- Our framework derives implementations from declarative definitions

Model Manipulation Using Embedded DSLs in Scala

Filip Křikava
I3S laboratory, Université Nice Sophia-Antipolis

Context
Model Manipulation

- Essential in *Model-Driven Engineering* (MDE)
- Automating operations such as
  - *model consistency checking*
  - *model-to-model transformation* (M2M)
  - *model-to-text transformation* (M2T)

ECLIPSE MODELING FRAMEWORK

	external
GPL	**DSL**

Approaches

Java

OCL EVL
QVT ETL ATL
MOFM2T EGL Xpand
Kermeta ...

http://eclipse.org/modeling/

emf

ECLIPSE MODELING FRAMEWORK

	GPL	*external* **DSL**
Approaches	Java	**OCL EVL QVT ETL ATL MOFM2T EGL Xpand Kermeta ...**
Issues	✓ Tool support and performance ✓ Versatility and integration ✗ Low level of abstraction ✗ Limited domain-specific error checking and optimizations	✓ High level of abstraction ✓ Domain-specific error checking and optimizations ✗ Limited tool support and performance ✗ Limited versatility and interoperability

http://eclipse.org/modeling/

= giving raise to accidental complexities, albeit of a different nature.

Towards Embedded DSLs

- External model manipulation DSLs
  - embed general-purpose programming constructs into a specific model-manipulation DSL

Towards Embedded DSLs

- External model manipulation DSLs
  - embed general-purpose programming constructs into a specific model-manipulation DSL

- *We explore* Internal / embedded model manipulation DSLs, that
  - embed domain-specific model manipulation constructs into a GPL
  - aiming at
    - similar features and expressiveness
    - increased versatility
    - improved tool support
    - *with significantly reduced engineering effort*

Quick Example
Model Consistency Checking

ISBN 978-3-16-148410-0

9 783161 484100

Checking Library books' ISBN codes.

OCL

```
context Book:

invariant UniqueISBN:

  self.library.books->forAll(book |
    book <> self implies book.isbn <> self.isbn);
```

Quick Example
Model Consistency Checking

ISBN 978-3-16-148410-0

Checking Library books' ISBN codes.

OCL

```
context Book:

invariant UniqueISBN:

  self.library.books->forAll(book |
    book <> self implies book.isbn <> self.isbn);
```

Scala

```
class BookContext extends ValidationContext
  with BookPackageScalaSupport {

  type Self = Class

  def invUniqueISBN =
    self.library.books forall { book =>
      book != self implies book.isbn != self.isbn
    }
}
```

- ✓ Tool support (rich editor, debugger)
- ✓ Unit testing
- ✓ Invariant inheritance
- ✓ Integration (build tools, workflows)

Model Consistency Checking

```
def invHasCapitalizedName =
  // guards
  guardedBy {
    // invariant dependency
    self satisfies invHasNonEmptyName
  } check {
    if (self.name.split(" ") forall (_(0).isUpper)) {
      Passed
    } else {
      // detailed error messages
      Error(s"Book ${self.name} does not have capitalized name")
        // with quick fixes
        .quickFix("Capitalize book name") {
           self.name = self.name.split(" ") map (_.capitalize) mkString (" ")
        }
    }
  }
```

- ✓ Context and invariant guards
- ✓ User feedback including quick fixes
- ✓ Invariant inheritance, modularity
- ✓ Different levels of severity

M2M Transformation

Simple object-oriented model into relational model

```
class OO2DB extends M2M with OOPackageScalaSupport with DBPackageScalaSupport {

  def ruleClass2Table(cls: Class, tab: Table, pkey: Column) {
    tab.name = cls.name;
    tab.columns += pkey
    tab.columns ++= ~cls.properties

    pkey.name = "Id"
    pkey.dataType = "Int"
  }

  @Lazy
  def ruleProperty2Column(prop: Property, col: Column) = guardedBy {
    !prop.multi
  } transform {
    col.name = prop.name
    col.dataType = prop.type_.name
  }
}
```

- ✓ Hybrid M2M transformation
- ✓ Rule inheritance, modularity
- ✓ Matched rules, lazy rules, partial rules
- ✓ Unit testing

M2M Transformation

Simple object-oriented model into relational model

```
class OO2DB extends M2M with OOPackageScalaSupport with DBPackageScalaSupport {

  def ruleClass2Table(cls: Class, tab: Table, pkey: Column) {
    tab.name = cls.name;
    tab.columns += pkey
    tab.columns ++= ~cls.properties

    pkey.name = "Id"
    pkey.dataType = "Int"
  }

  @Lazy
  def ruleProperty2Column(prop: Property, col: Column) = guardedBy {
    !prop.multi
  } transform {
    col.name = prop.name
    col.dataType = prop.type_.name
  }
}
```

- ✓ Hybrid M2M transformation
- ✓ Rule inheritance, modularity
- ✓ Matched rules, lazy rules, partial rules
- ✓ Unit testing

M2M Transformation

Simple object-oriented model into relational model

```
class OO2DB extends M2M with OOPackageScalaSupport with DBPackageScalaSupport {

  def ruleClass2Table(cls: Class, tab: Table, pkey: Column) {
    tab.name = cls.name;
    tab.columns += pkey
    tab.columns ++= ~cls.properties

    pkey.name = "Id"
    pkey.dataType = "Int"
  }

  @Lazy
  def ruleProperty2Column(prop: Property, col: Column) = guardedBy {
    !prop.multi
  } transform {
    col.name = prop.name
    col.dataType = prop.type_.name
  }
}
```

Executes

- ✓ Hybrid M2M transformation
- ✓ Rule inheritance, modularity
- ✓ Matched rules, lazy rules, partial rules
- ✓ Unit testing

M2T Transformation

Simple object-oriented model into Java code

```
class OO2Java extends M2T with OOPackageScalaSupport {
  type Root = Class // input type for this transformation

  def main =
    !s"public class ${root.name}" curlyIndent {
      !endl // extra new line

      for (o <- root.operations) {
        genOperation(o)
        !endl // extra new line
      }
    }

  def genOperation(o: Operation) =
    !s"public ${o.returnType.name} ${o.name}()" curlyIndent {
      !s"""
        // TODO: should be implemented
        throw new UnsupportedOperationException("${o.name}");
        """
    }
}
```

- ✓ Template based, code-centric M2T
- ✓ Template inheritance, modularity
- ✓ Smart white space handling
- ✓ Unit testing

Further Work: Exploring Deep Embedding

- Translating invariant expression into first-order logic

```
library.books exists { book =>
  book.pages > 300 }
```

$$\exists(x)(\texttt{Book}(x) \land (\texttt{pages}(x) > 300))$$

Further Work: Exploring Deep Embedding

- Translating invariant expression into first-order logic

```
library.books exists { book =>
  book.pages > 300 }
```

$$\exists(x)(\mathtt{Book}(x) \land (\mathtt{pages}(x) > 300))$$

- Resolving M2M transformation rules at compile time

```
def ruleClass2Table(cls: Class, tab: Table, pkey: Column) {
  tab.name = cls.name;
  tab.columns += pkey
  tab.columns ++= ~cls.properties
```

No conversion rule between oo.Property and B.

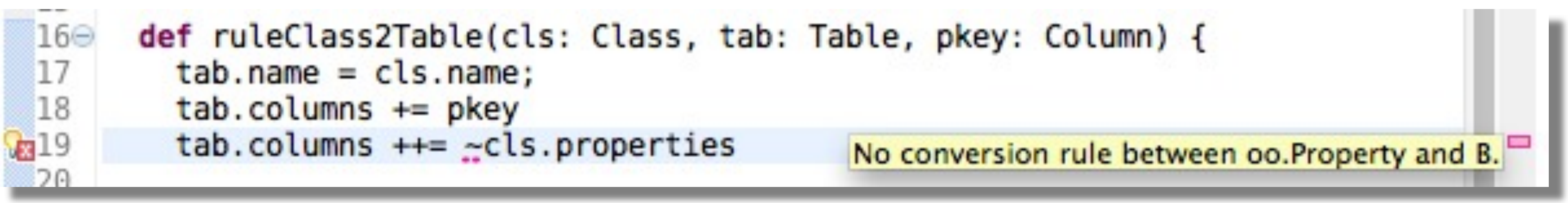
Further Work: Exploring Deep Embedding

- Translating invariant expression into first-order logic

```
library.books exists { book =>
  book.pages > 300 }
```

$$\exists(x)(\texttt{Book}(x) \wedge (\texttt{pages}(x) > 300))$$

- Resolving M2M transformation rules at compile time



- Fast M2M transformation by translating declarative rules

```
def ruleClass2Table(cls: Class, tab: Table, pkey: Column)
def ruleProperty2Column(prop: Property, col: Column)
```

```
source.eAllContents collect {
  case x: Class ⇒
    ruleClass2Table(x, create[Table], create[Column])
  case x: Property ⇒
    ruleProperty2Column4(x, create[Column])
  case x =>
    logger warn (s"No rule to transform ${x}.")
}
```

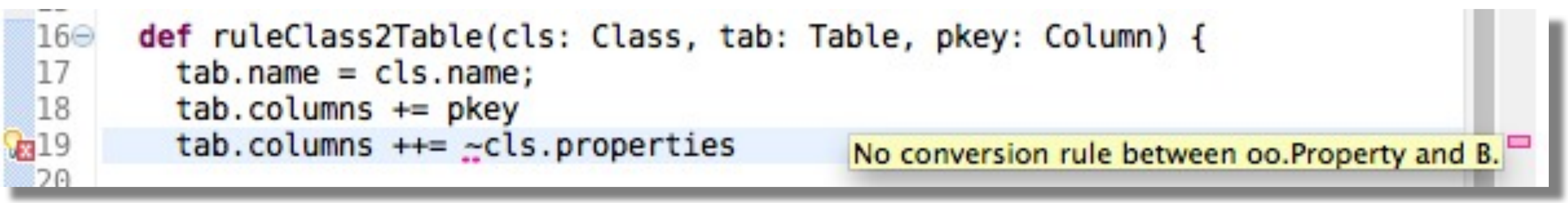
Further Work: Exploring Deep Embedding

- Translating invariant expression into first-order logic

```
library.books exists { book =>
  book.pages > 300 }
```

$$\exists(x)(\mathtt{Book}(x) \wedge (\mathtt{pages}(x) > 300))$$

- Resolving M2M transformation rules at compile time



- Fast M2M transformation by translating declarative rules

```
def ruleClass2Table(cls: Class, tab: Table, pkey: Column)
def ruleProperty2Column(prop: Property, col: Column)
```

```
val it = source.eAllContents
while (it.hasNext) {
  val x = it.next
  if (x.isInstanceOf[Class])
    ruleClass2Table(x.asInstanceOf[Class], create[Table], create[Column])
  else if (x.isInstanceOf[Property])
    ruleProperty2Column4(x.asInstanceOf[Property], create[Column])
  else
    logger warn (s"No rule to transfrom ${x}.")
}
```

Thank You

https://fikovnik.github.io/Sigma

https://salty.unice.fr

Filip Křikava

filip.krikava@i3s.unice.fr

The work is partly funded by the ANR SALTY project under contract ANR-09-SEGI-012.

Common Infrastructure Support

- Generate a Scala support trait for each EMF package

```
with LibraryPackageScalaSupport
```

- Navigation

```
self.getLibrary().getIsbn()
```

```
self.library.isbn
```

```
self.library.books collect {
  case Book(title, Author(author),_,_,_,_) => (title, author)
}
```

- Improving null handling
  - Multiplicity 0..1 is wrapped into Option[T]
- Modification

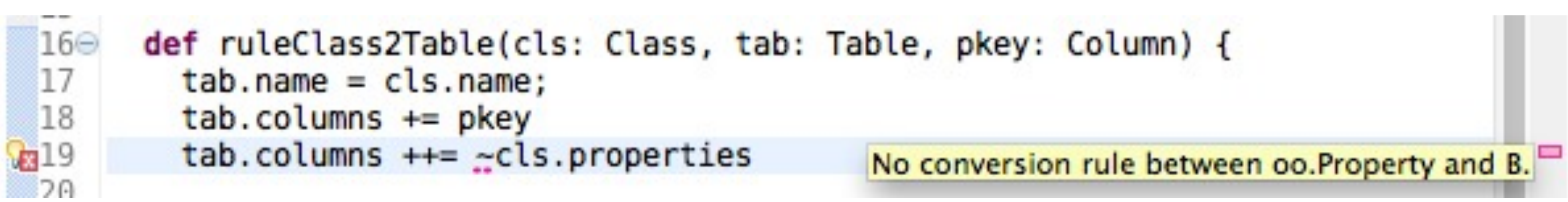
```
val sicp = Book(name = "SICP", copies = 2)
sicp.author = library.authors find (_.name == "H. Abelson")
```

M2M Transformation
partially type-safe

- Instead of reflection, explicitly register each rule

```
implicit val _ruleClass2Table = rule(ruleClass2Table _)
```

- Using ~ without corresponding rule yields compile time error



- Realized using

```
@implicitNotFound("No conversion rule between ${Source} and ${Target}.")
trait Rule[S <: EObject, T <: EObject]
```

```
implicit class EListM2MSupport[A <: EObject: ClassTag](that: EList[A]) {
  def unary_~[B <: EObject](implicit rule: Rule[A, B]) = // ...
}
```

Further Work
Formal Reasoning

- Translating invariant expression into first-order logic
- To check invariant unsatisfiability[1]

```
library.books exists { book => book.pages > 300 }
```

$$\exists(x)(\texttt{Book}(x) \wedge (\texttt{pages}(x) > 300))$$

```
library.books forall { book => book.pages < 300 }
```

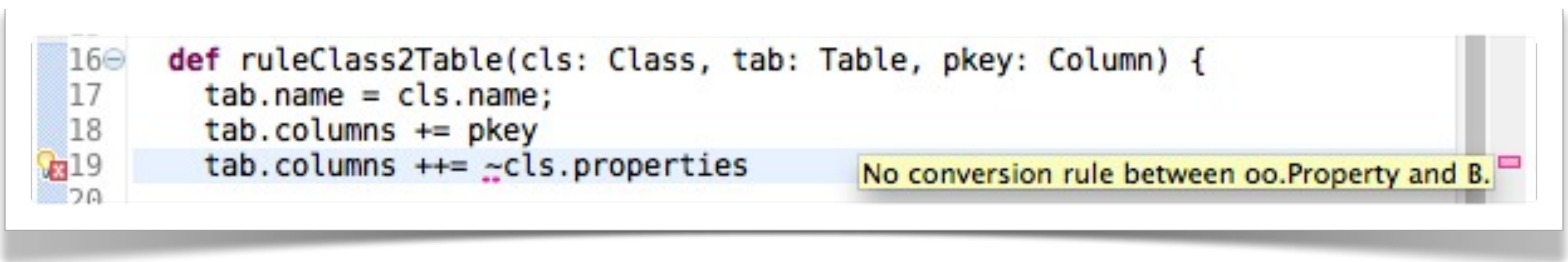
$$\forall(x)(\texttt{Book}(x) \implies (\texttt{pages}(x) < 300))$$

- Automatic verification using SMT solvers

[1] Clavel, M., Egea, M., Garcia de Dios, M.A. (2009) *Checking unsatisfiability for OCL constraints*. OCL Workshop, MODELS 2009

Further Work
Domain-Specific error checking

- Using ~ without corresponding rule yields compile time error




```
@implicitNotFound("No conversion rule between ${Source} and ${Target}.")
trait Rule[S <: EObject, T <: EObject]
```

```
implicit class EListM2MSupport[A <: EObject: ClassTag](that: EList[A]) {
  def unary_~[B <: EObject](implicit rule: Rule[A, B]) = // ...
}
```

```
implicit val _ruleClass2Table = rule(ruleClass2Table _)
```

Further Work
Faster M2M Transformations

Translate declarative rules into imperative code

```
class OO2DB extends M2M with OOPackageScalaSupport with DBPackageScalaSupport {

  def ruleClass2Table(cls: Class, tab: Table, pkey: Column)

  def ruleProperty2Column(prop: Property, col: Column)
}
```

```
source.eAllContents collect {
  case x: Class ⇒
    ruleClass2Table(x, create[Table], create[Column])
  case x: Property ⇒
    ruleProperty2Column4(x, create[Column])
  case x =>
    logger warn (s"No rule to transform ${x}.")
}
```

Further Work
Very Faster M2M Transformations

Translate declarative rules into imperative code

```
class OO2DB extends M2M with OOPackageScalaSupport with DBPackageScalaSupport {

  def ruleClass2Table(cls: Class, tab: Table, pkey: Column)

  def ruleProperty2Column(prop: Property, col: Column)
}
```

```
    val it = source.eAllContents
    while (it.hasNext) {
      val x = it.next
      if (x.isInstanceOf[Class])
        ruleClass2Table(x.asInstanceOf[Class], create[Table], create[Column])
      else if (x.isInstanceOf[Property])
        ruleProperty2Column4(x.asInstanceOf[Property], create[Column])
      else
        logger warn (s"No rule to transfrom ${x}.")
    }
```

Further Work
Beyond Shallow Embedding

- Using Scala facilities for *deep embedding*
  - scala-virtualized https://github.com/TiarkRompf/scala-virtualized
  - lightweight modular staging http://scala-lms.github.io/
- For
  - formal analysis
  - domain-specific error checking
  - domain-specific optimization

Approaches in

ECLIPSE MODELING FRAMEWORK

GPL	emf	*external* **DSL**
Java	Model consistency checking	OCL EVL Kermeta
Java	M2M transformation	QVT ETL Kermeta ATL
Java	M2T transformation	MOFM2T EGL Kermeta Xpand

http://eclipse.org/modeling/

Issues

Java

Kermeta ATL EGL
Xpand ETL OCL
QVT EVL MOFM2T

- ✓ Versatility
- ✓ Excellent tool support
- ✓ Performance
- ✓ Integration

- ✗ Low level of abstraction
- ✗ Limited expressiveness (lack of functional aspects in the language)
- ✗ Limited domain-specific error checking and optimizations

- ✓ High level of abstraction
- ✓ Expressiveness and ease of use
- ✓ Domain-specific error checking and optimizations

- ✗ Limited tool support
- ✗ Limited performance
- ✗ Limited versatility (fall back to Java)
- ✗ Limited interoperability

Issues

Java

Kermeta ATL EGL
Xpand ETL OCL
QVT EVL MOFM2T

✓ Versatility
✓ Excellent tool support
✓ Performance
✓ Integration

✗ Low level of abstraction
✗ Limited expressiveness (lack of functional aspects in the language)
✗ Limited domain-specific error checking and optimizations

✓ High level of abstraction
✓ Expressiveness and ease of use
✓ Domain-specific error checking and optimizations

✗ Limited tool support
✗ Limited performance
✗ Limited versatility (fall back to Java)
✗ Limited interoperability

= giving raise to accidental complexities, albeit of a different nature.

Towards Embedded DSLs

- External model manipulation DSL
  - based on some common infrastructure for model navigation and modification (usually a subset of OCL)
  - embed general-purpose programming constructs into a specific model-manipulation DSL
- *We explore* Internal / embedded model manipulation DSL
  - based on some host GPL language
  - embed domain-specific model manipulation constructs into the GPL
  - gain similar expressiveness, versatility and tool-support

REACTIVE-SIM

REACTIVE FRAMEWORK FOR COMPLEX COMPUTATIONS

http://github.com/ellis/reactive-sim

by Ellis Whitehead

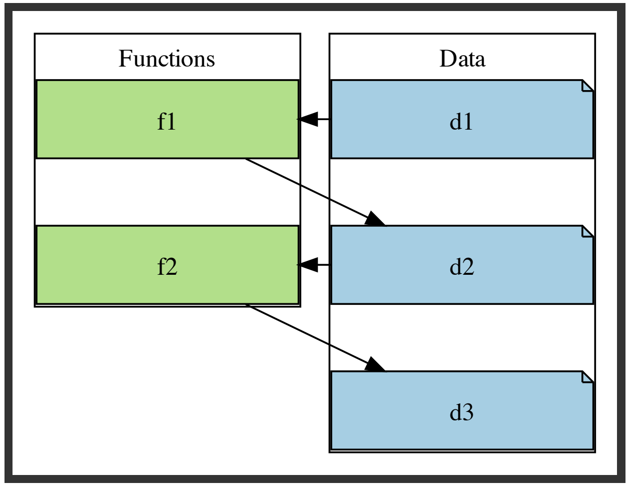
REACTIVE FRAMEWORKS

Outputs are automatically updated when inputs change

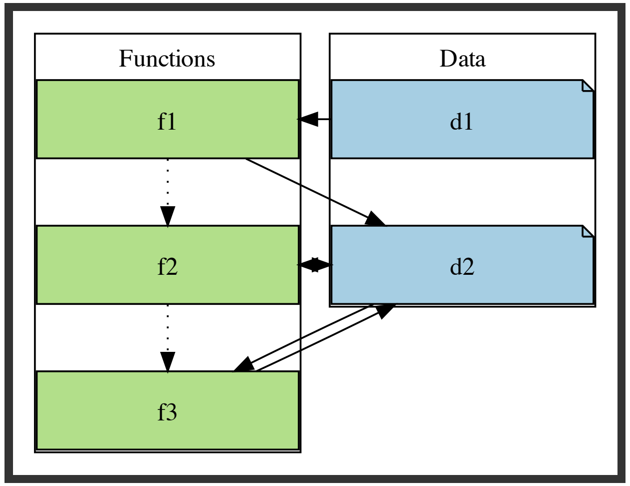
EXAMPLES

- AngularJS
- Interactive visualization (Bret Vector)
- Functional Reactive Programming
- JavaFX/ScalaFX

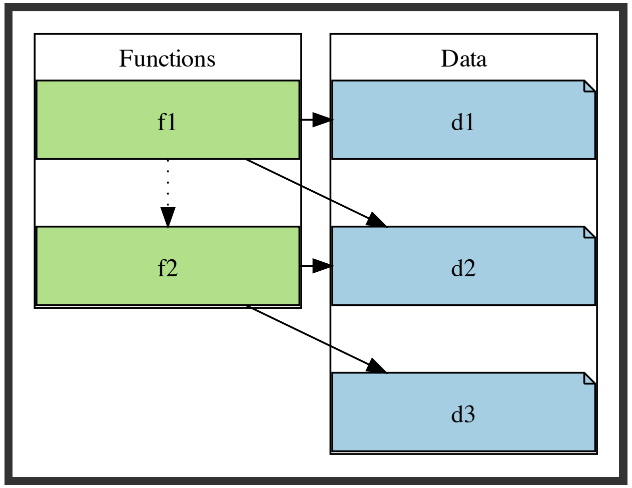
COMPUTATION GRAPHS



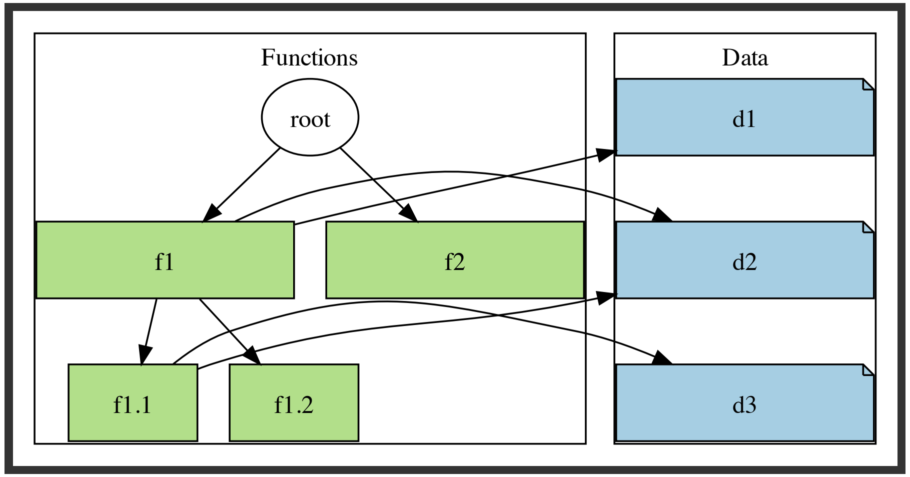
Declarative



Sequence and State



Output Pushing



Function Tree

REACTIVE-SIM LIBRARY

GENERAL USE-CASE

- Step through, trouble-shoot, or visualize a computation
- Explore impact of parameter changes on computation

OUR APPLICATION

- Robot simulation
- Troubleshooting
- Optimization
- Robot control with sensor feedback

REACTIVE-SIM FEATURES

- Scala classes for reactive simulation framework
- DSL to ease construction of computation graph
- Errors and warnings
- Selectors for inputs

Being ported: * Commands, events, dynamically calculated entities * Automatic parallelization * Control

CONCLUSION

- Step through, trouble-shoot, or visualize a computation
- Explore impact of parameter changes on computation

http://github.com/ellis/reactive-sim

Thanks to: