Pleiad

Programming Languages and Environments for Intelligent, Adaptable and Distributed systems

dcc UNIVERSIDAD DE CHILE

fcfm FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS UNIVERSIDAD DE CHILE

Domain-Specific Aspect Languages (DSALs)

Johan Fabry
jfabry@dcc.uchile.cl

Overview

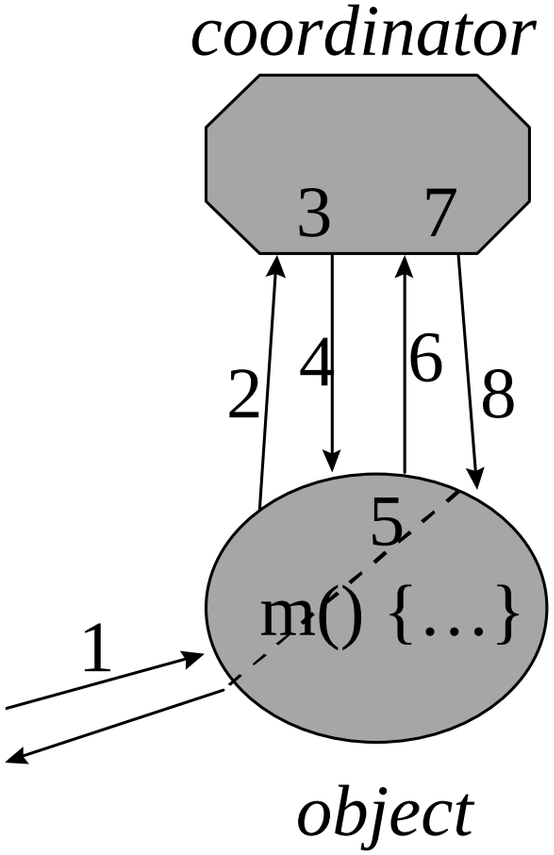
- Reminder: Fundamental concepts of AOP
- Example: COOL
- What is a DSAL?
- Example: KALA
- Domain-Specific what?
- Example: AspectLISA
- Making a DSAL

AOP Fundamentals

- Separation of concerns
- Joinpoint = point of composition of concerns
- Joinpoint model = kind, use, access of joinpoints
- Aspect
  - pointcut = joinpoint predicate = **where**
  - advice = functionality = **what**
- Most popular aspect language = AspectJ
  - joinpoint model: methods, fields, exceptions
  - pointcut: type/method/... patterns
  - advice: +- standard Java

COOL: another way to do aspects

"**mutual exclusion** of threads, **synchronization** state, **guarded suspension** and **notification**" [Lopes 97]



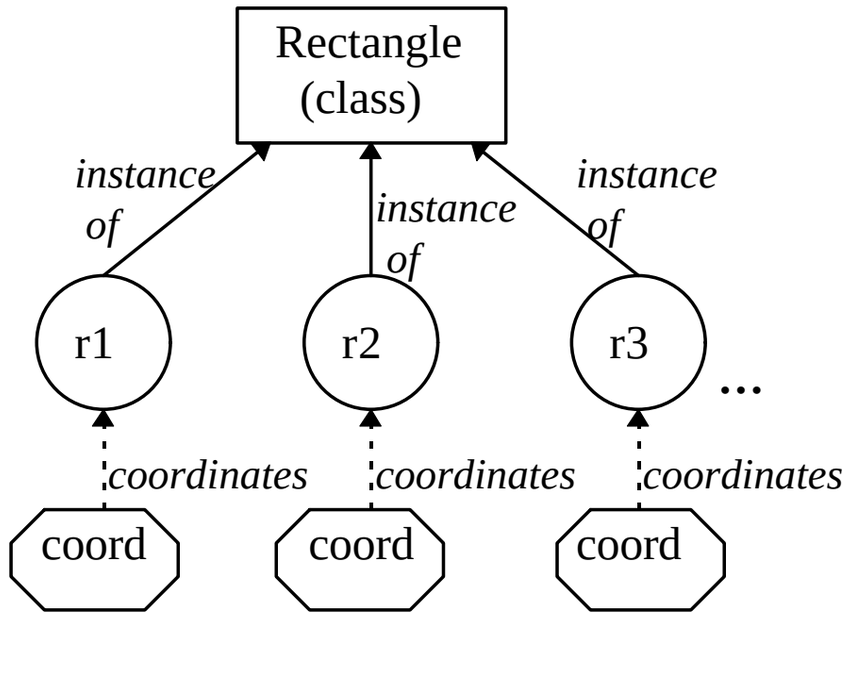
1: Within thread T, there is a method invocation to method m of the object, say obj.m()
2: The request is first presented to the object's coordinator
3: The coordinator checks exclusion constraints and pre-conditions for method m. If any of those constraints is not met, T is suspended. When all constraints are met, T has the right to execute method m. Just before it does so, the coordinator executes its on_entry statements for method m.
4: The request proceeds to the object.
5: Thread T executes method m in the object.
6: As the method invocation returns, the return is presented to the coordinator.
7: The coordinator executes its on_exit statements for method m.
8: The method invocation finally returns.

Bounded Buffer in COOL

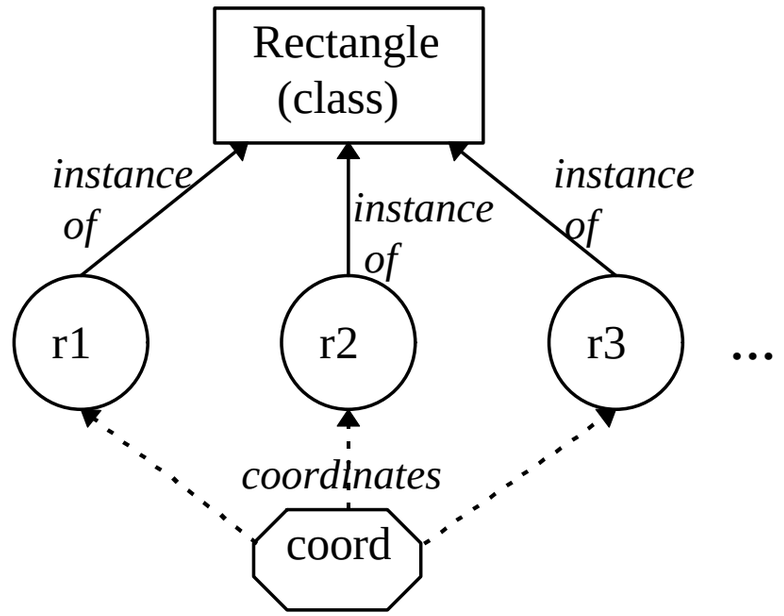
```
coordinator BoundedBuffer {
  selfex put, take;
  mutex {put, take};
  condition empty = true, full = false;

  put: requires !full;
    on_exit {
      if(empty) empty = false;
      if(usedSlots == capacity) full = true;}
  take: requires !empty;
    on_exit {
       if (full) full = false;
       if (usedSlots == 0) empty = true;}
}
```

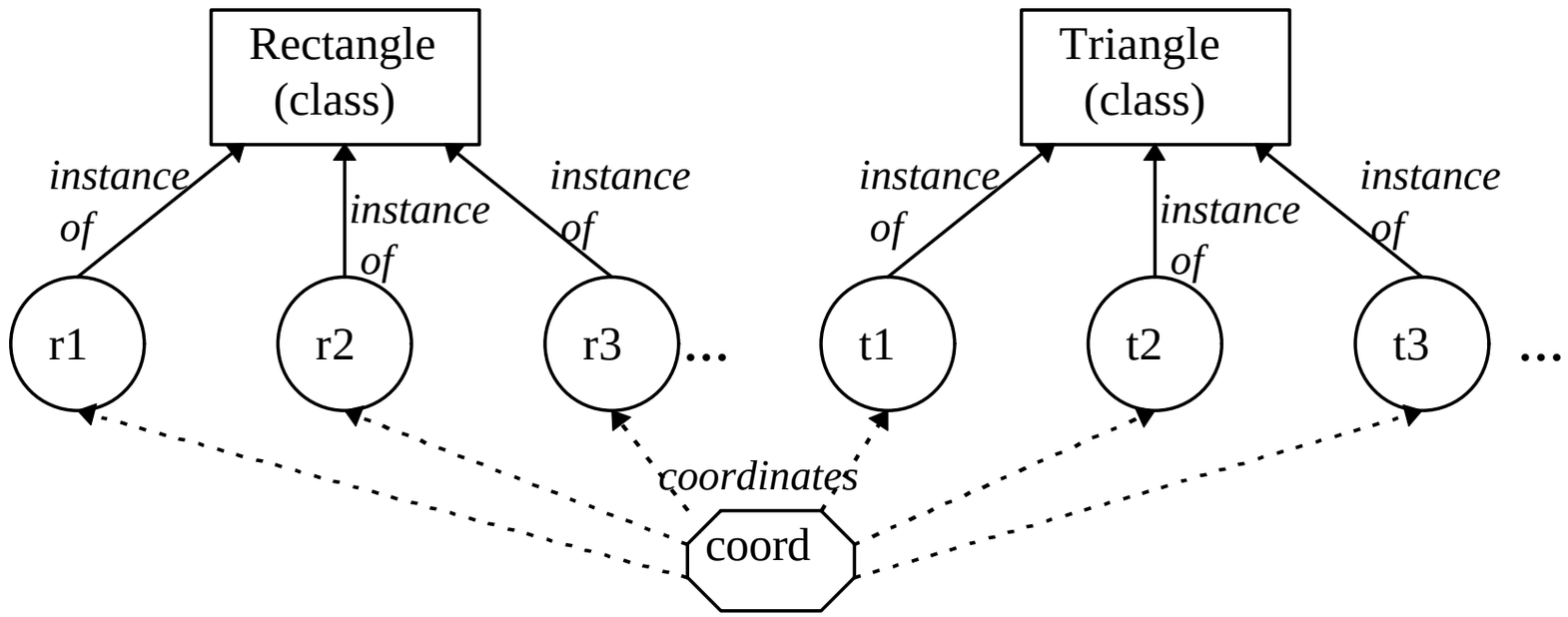
Per object / Per class coordinators



a) per_object coordination



b) per_class coordination -- single class



c) per_class coordination – 2 classes

Dining Philosophers in COOL

```
per_class coordinator Philosopher {
  condition OKToEat[]={true, true, true, true, true};
  boolean eating[]={false, false, false, false, false};
  eat: requires OKToEat[mynumber];
       on_entry {
         OKToEat[(mynumber+1) % max] = false;
         OKToEat[(mynumber-1) % max] = false;
         eating[mynumber] = true;   }
       on_exit {
         if (eating[(mynumber+2) % max] == false)
           OKToEat[(mynumber+1) % max] = true;
         if (eating[(mynumber-2) % max] == false)
           OKToEat[(mynumber-1) % max] = true;
         eating[mynumber] = false;  }
}
```

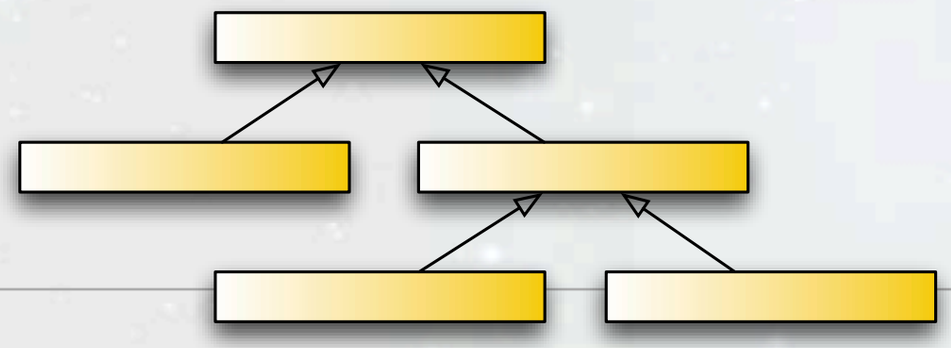
What is a DSAL?

- **Definition:** “A DSAL is a DSL for expressing cross-cutting concerns separately”
- “More formally: a DSL whose programs are **not** functionally composed with other programs”

- DSAL vs DSL: Composition is essential in DSAL
- DSAL vs GPAL = DSL vs GPL
- Separation of concerns is the goal

Advanced Transactions

- Transaction: Concurrency & failure management in distributed systems
- ACID = Atomic, Consistent, Isolated, Durable
- Objects & Transactions:
  - Method = transaction
  - Data access within the scope of the transaction
- Advanced Transactions: go beyond the classical model



KALA: DSAL for Advanced Transactions

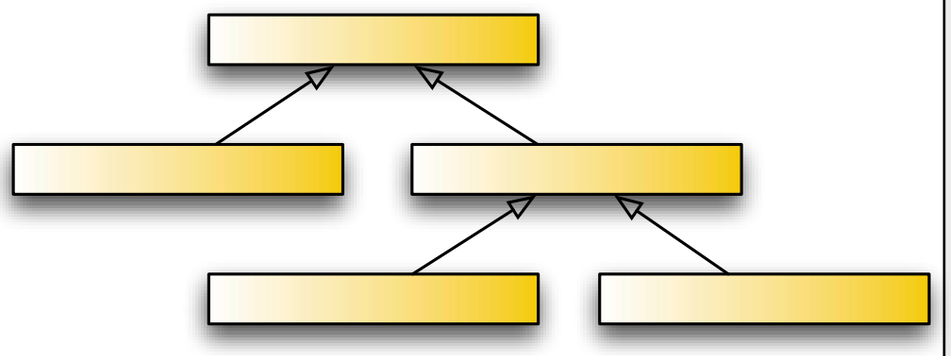
- Transactions = classical AOP example
  - Multiple implementations using AspectJ

- KALA: DSAL for Advanced Transactions [Fabry 05]
  - Base: ACTA formal model [Chrysanthis 91]

- (Not all DSALS are for Concurrency!)

KALA: DSAL for Advanced Transactions

```
util.strategy.Hierarchical.*(){
 alias(root Thread.currentThread());
 name(self Thread.currentThread());
 begin{
   dep(self wd root, root cd self);
   view(self root)}
commit{ del(self root);
   name(root Thread.currentThread());
   terminate(self)}
abort{
   name(root ...);
   terminate(self)}}
```



KALA vs General-Purpose

```
Cashier.transfer
    (BankAccount from, BankAccount to , int amount) {
  alias (Saga Thread.currentThread());
  groupAdd(self ”StepOf”+Saga);}
  autostart (transfer(BankAccount, BankAccount, int)
    <dest, source, amount> {
     name(self "CompOf"+Saga);
     groupAdd(self "CompOf"+Saga);});
begin {
  alias (Comp "CompOf"+Saga);
  dep(Saga ad self, self wd Saga, Comp bcd self); }
commit {
  alias (Comp "CompOf"+Saga>);
  dep(Comp cmd Saga, Comp bad Saga); }}
```

KALA vs general-purpose

```
private void transfer
    (BankAccount from_orig, BankAccount to_orig, int amount)
    throws TxException
{
    TransactionManager txmgr = TransactionManager.getCurrent();
    Integer self = txmgr.newID();
    txmgr.addTransaction(self);
    Integer RCS = txmgr.lookup(Thread.currentThread());
    txmgr.addToGroup("RCS"+ RCS + "Step",self);

    final Integer comp_id = txmgr.newID(); //for compensation
    txmgr.addTransaction(comp_id);
    txmgr.addToGroup("RCS"+ comp_id+ "Comp",comp_id);
    txmgr.bind("RCS"+ comp_id+ "Comp",comp_id);

    final BankAccount compfrom = from_orig; //for inner class
    final BankAccount compto = to_orig; //for inner class
    final int compamount = amount; //for inner class

    Runnable compensator = new Runnable()
{
    public void run(){
        undoTransfer(compfrom, compto, compamount, comp_id);
    }
};
    txmgr.addDependency(RCS, "ad", self);
    txmgr.addDependency(self, "wd" ,RCS);
    txmgr.addDependency(comp_id, "bcd" ,self);

    new Thread(compensator).run();
```

```
Forcing bf = txmgr.mayBegin(self);
if (bf == null){
    Object preView = txmgr.lookupGroupBinding("RCS"+ RCS + "View");
    txmgr.begin(self);
    txmgr.removeViewGroup(RCS, preView);
    txmgr.delegate(RCS, self);
}
else {
    txmgr.rollback(self);
    return;
}
try {
    BankAccountWrap from = new BankAccountWrap(from_orig);
    BankAccountWrap to = new BankAccountWrap(to_orig);
```

Proceed()

```
    Forcing cf = txmgr.mayCommit(self);
    if (cf != null)
        throw new TxAbortedException();

    txmgr.addDependency(comp_id, "cmd" ,RCS);
    txmgr.addDependency(comp_id, "bad" ,RCS);

    txmgr.bindGroup("transferGroup","RCS"+ RCS + "View")
    Object newView = txmgr.lookupGroupBinding("RCS"+ RCS + "View");
    txmgr.addViewGroup(RCS, newView);
    txmgr.delegate(self, RCS);

    txmgr.commit(self);
}
catch (TxException ex){
    txmgr.mayAbort(self);//will always succeed
    txmgr.rollback(self);
    throw ex;
}}
```

KALA advantages

- Separation of Concerns
  - App logic: Java, Transactions: KALA
- High level of abstraction
  - State transactional properties
- Conciseness
  - Sagas: 267 lines pure Java = 37 Java + 52 KALA
  - i.e. 3x code reduction

Domain-Specific what?

- Joinpoint model? Pointcuts? Advice?

Domain-specific Joinpoint Model

Domain-specific Pointcut

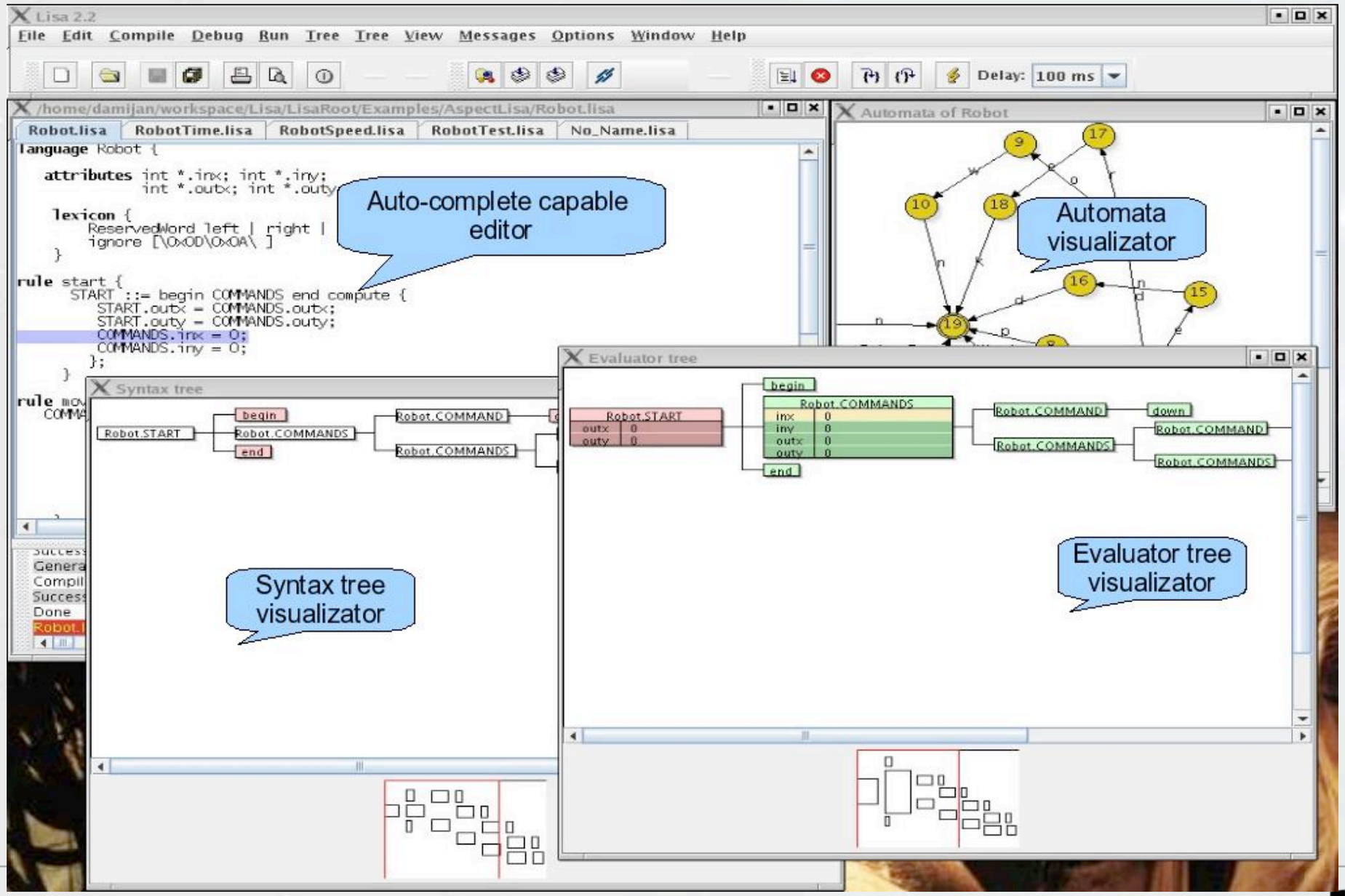
Domain-specific Advice

KALA

COOL

DSL as a base language : Aspect LISA

- LISA compiler compiler [Rebernak 06]
  - Regular expressions, BNF, Attribute Grammars



LISA: Example Robot language

```
language Robot {
lexicon {
   Commands left | right | up | down
   ReservedWord begin | end
   ignore [\0x0D\0x0A\ ] // skip whitespaces
}
rule start {
   START ::= begin COMMANDS end compute {
      START.outp = COMMANDS.outp; // robot position in the beginning
      COMMANDS.inp = new Point(0, 0); };
}
rule move { // each command changes one coordinate
   COMMAND ::= left compute {
      COMMAND.outp = new Point((COMMAND.inp).x-1,(COMMAND.inp).y); };
   COMMAND ::= right compute {
      COMMAND.outp = new Point((COMMAND.inp).x+1,(COMMAND.inp).y); };
   COMMAND ::= up compute {
      COMMAND.outp = new Point((COMMAND.inp).x,(COMMAND.inp).y+1); };
   COMMAND ::= down compute {
      COMMAND.outp = new Point((COMMAND.inp).x,(COMMAND.inp).y-1); };
} [...] }
```

AspectLISA

- Joinpoint model
  - Static
  - Syntactic Production Rules | Generalized LISA Rules

- Pointcuts
  - match terminal or non-terminal symbols
  - ‘..’ = 0 or more symbols
  - ‘*’ = (parts of) literals representing a symbol

AspectLISA Example Pointcuts

`pointcut *.* : * ::= .. ;`

any production in any rule in all languages across the current language hierarchy

`pointcut Robot.m* : * ::= .. ;`

any production in all rules which start with **`m`** in the **`Robot`** language

`pointcut Robot.move : COMMAND ::= left ;`

matches only a production **`COMMAND ::= left`** in the rule **`move`** of the Robot language

`pointcut Time<COMMAND> *.move : COMMAND ::= *;`

all productions in **`move`** with **`COMMAND`** as the left-hand non-terminal

AspectLISA Advice

- Parameterized semantic rules
- Written as native Java assignment statements
- Define additional semantics, not impacting structure/ syntax
- Adding `COMMAND.time=1` to all productions within move:

```
pointcut Time<COMMAND>*.move:COMMAND::= *;
advice TimeSemantics<C>
    on Time { C.time=1; }
```

Domain-Specific what?

- Joinpoint model? Pointcuts? Advice?

Domain-specific Joinpoint Model

Domain-specific Pointcut

Domain-specific Advice

KALA

COOL

Name/att filtering

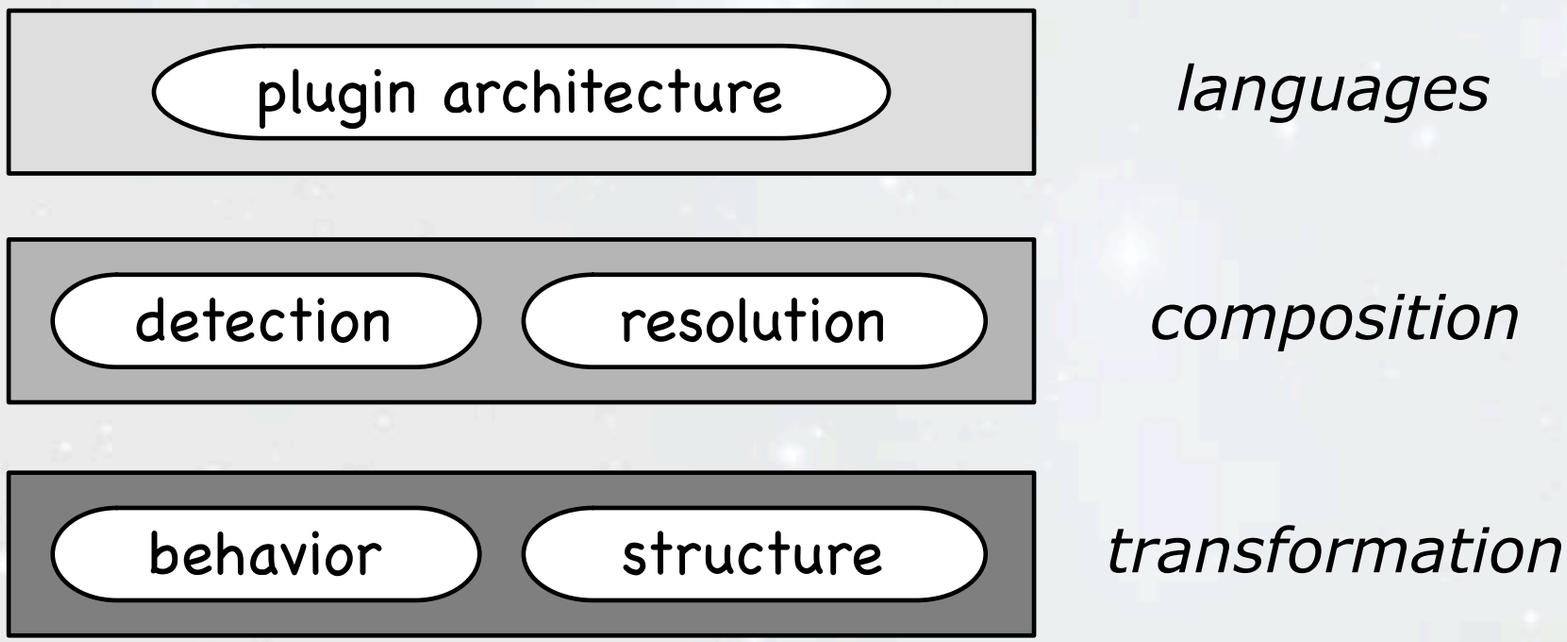
Where vs what

Making a DSAL -- Implementation

- By hand: Parser + Weaver
  - - Lots of work & hard
  - + Total freedom
- Transform to AspectJ
  - - Limited by AspectJ features
  - + Communication, robustness
- DSAL weaving infrastructure, e.g. Reflex [Tanter 05]
  - Best of both worlds?

The Reflex Angle

- Combine the advantages of framework-based approach with those of language-based approaches
- Extensible core: Reflex
  - AOP kernel for Java based on reflective model
- Extensible syntax/assimilation:
  - MetaBorg (SDF+Stratego) [Visser & Bravenboer]



Making a DSAL -- Design

- Advice Specification ?
- Domain-Specific joinpoint model?
  - What are the joinpoints?
  - Static / dynamic?
  - Granularity?
- Pointcut Specification?
- Context exposure ?
- Pointcut / Advice separation ?
- Aspect reuse ?

Making a DSAL -- Challenges

- Analysis of the domain
  - Methodology? (Ad-hoc)
  - DSAL = DSL + crosscut specification
- Making the weaver
  - Infrastructure, e.g. Reflex
  - Reuse of language/weaver definitions
- Dependencies and Interactions of Aspects
  - Various aspects in 1 application
  - DSALs provide domain information

Questions?

Pleiad

Programming Languages and Environments for Intelligent, Adaptable and Distributed systems