

Datalog as a Pointcut Language in Aspect-Oriented Programming

abc team, University of Oxford, United Kingdom

Aspect-Oriented Programming:

improved modularity through:

- Event interception (Advice)
- Adding members (Open Classes)

Pointcuts are the key to find locations of interest in the code

Wish list:

- Robust pointcuts
- Expressive semantic pointcuts
- Easy to learn, write and read

Current problems:

• Lack of expressiveness

Just syntactic pattern matching

Nearly all methods have to be explicitly listed

What if there are hundreds of methods?

• Not Robust

Need to update pointcuts consistently

What if library API methods change?

Motivating example in AspectJ

Safe Enumeration:

"No calls to `nextElement()` of an Enumeration allowed if source object has been changed"

- 1) First intercept creation of an *Enumeration* object for each *Vector* object

```
after (Vector ds) returning (Enumeration e) :  
  call (Enumeration+.new(..)) && args (ds)  
  { ...remember enumeration object creation... }
```

- 2) Then intercept an update of the same *Vector* object

```
after (Vector ds) :  
  vector_update () && target (ds)  
  { ...remember vector object update... }
```

- 3) If there is a call to *nextElement* of an *Enumeration* and *Vector* object has changed raise an exception!

```
before (Enumeration e) :  
  call (Object Enumeration.nextElement ()) && target (e)  
  { if (...related vector object was updated...)  
    throw new ConcurrentModificationException ();  
  }
```

A pointcut to intercept changes of a *Vector* object requires listing of all methods that could modify it:

```
pointcut vector_update () :  
  call (* Vector.add* (..)) ||  
  call (* Vector.clear ()) ||  
  call (* Vector.insertElementAt (..)) ||  
  call (* Vector.remove* (..)) ||  
  call (* Vector.retainAll (..)) ||  
  call (* Vector.set* (..));
```

Aspects Require Better Pointcuts!

Common answer: Prolog

```
vector_update_method(M) :-  
  // M is a method in  
  // a class V named Vector  
  methodDecl(M,_,V,_,_),  
  typeDecl(V,'Vector',_,_),  
  // N is a method named nextElement  
  // in an implementation I of the  
  // Enumeration interface  
  methodDecl(N,'nextElement',I,_),  
  implements(I,E),  
  typeDecl(E,'Enumeration',_,_),  
  // N may read field F  
  // (also via a chain of calls)  
  mayRead(N,F),  
  // and M may write field F  
  mayWrite(M,F).
```

Some systems that use this architecture:
Alpha, Aspicere, Carma, Compose,
LogicAJ, Sally, Soul, etc.*

Disadvantages of Prolog:

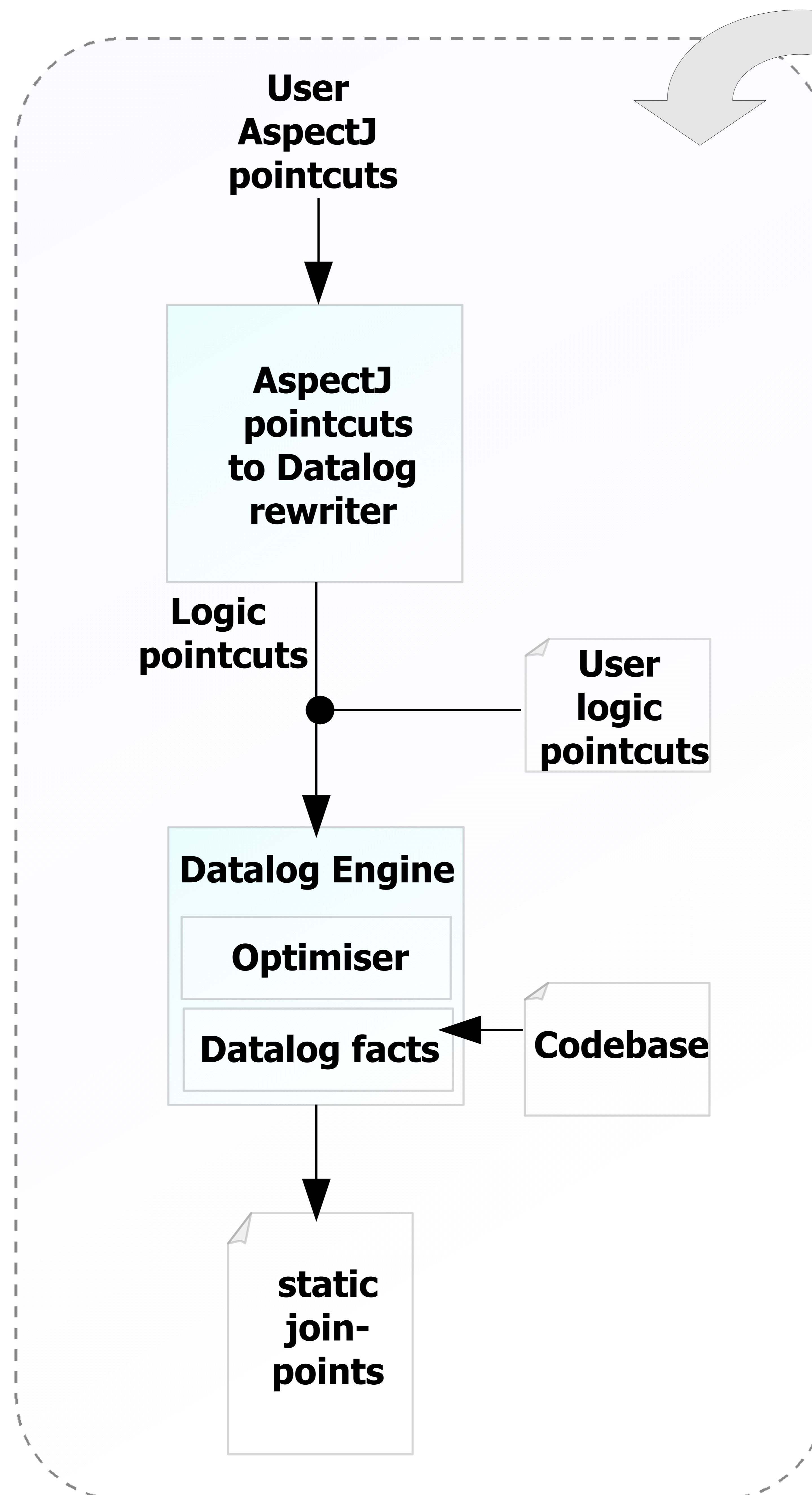
- Require mode annotations, cuts
- Queries may still not terminate
- Facts in memory do not scale

Datalog is the right solution!

- DataLog = Prolog - data structures
- Expressive enough, efficient and scalable
- Queries are always guaranteed to terminate

But...

- Name and signature patterns expressed in a logic language are too verbose and unreadable
- Not every developer is immediately ready to learn Datalog



We propose:

- More semantic pointcuts using Datalog together with purely syntactic AspectJ pointcuts
- Where the latter are merely a syntactic sugar of the former
- Novice developers can define pointcuts in a more familiar style
- Experts can resort to the full power of logic and still enjoy the conciseness of name and signature patterns

Rewriting AspectJ pointcuts into Datalog:

- Separate static and dynamic pointcuts
- Static pointcuts are converted into Datalog using a set of 90 simple term rewrite rules
- We have constructed an automated AspectJ 1.2.1 to Datalog rewrite system
- Full set of rules is available online at: <http://www.aspectbench.org/>

Semantics of Pointcuts!

A rule to rewrite get (or set) pointcuts:

```
aj2d1(get(fieldpat), C, S)
```

⇒

```
∃ F,R: (fieldpat2d1(fieldpat,C,R,F),  
  getShadow(S, F, R))
```

The fieldpat2d1 in its turn gets rewritten to:

```
fieldpat2d1(fmodpat typepat membpat,C,R,F)
```

⇒

```
∃ T: (fmodpat2d1([fmodpat],F),  
  typepat2d1(typepat, C, T),  
  fieldmembpat2d1(membpat, C, R, F),  
  field(F), hasType(F, T))
```

Similar rules are used to rewrite terms

fmodpat2d1, typepat2d1 and fieldmembpat2d1

And so on, rules are applied exhaustively until all rewriting is complete.