Metadata and aspect evolution

Experiences in Aspicere

Bram ADAMS
Software Engineering Lab, INTEC, UGent

Aspicere

- What's in a name?
  - aspicere ≡ "to look at" (Latin)
  - Here: aspect language for C
- Characteristics:
  - Prolog-based pointcut language
  - Source code weaver
  - Currently only statically determinable joinpoints
  - Likewise no weaving within advices
- Future:
  - Merging into GCC 4.0 ("heterogeneous AOP")
  - cflow, sequence, ...
  - Weaving inside advices

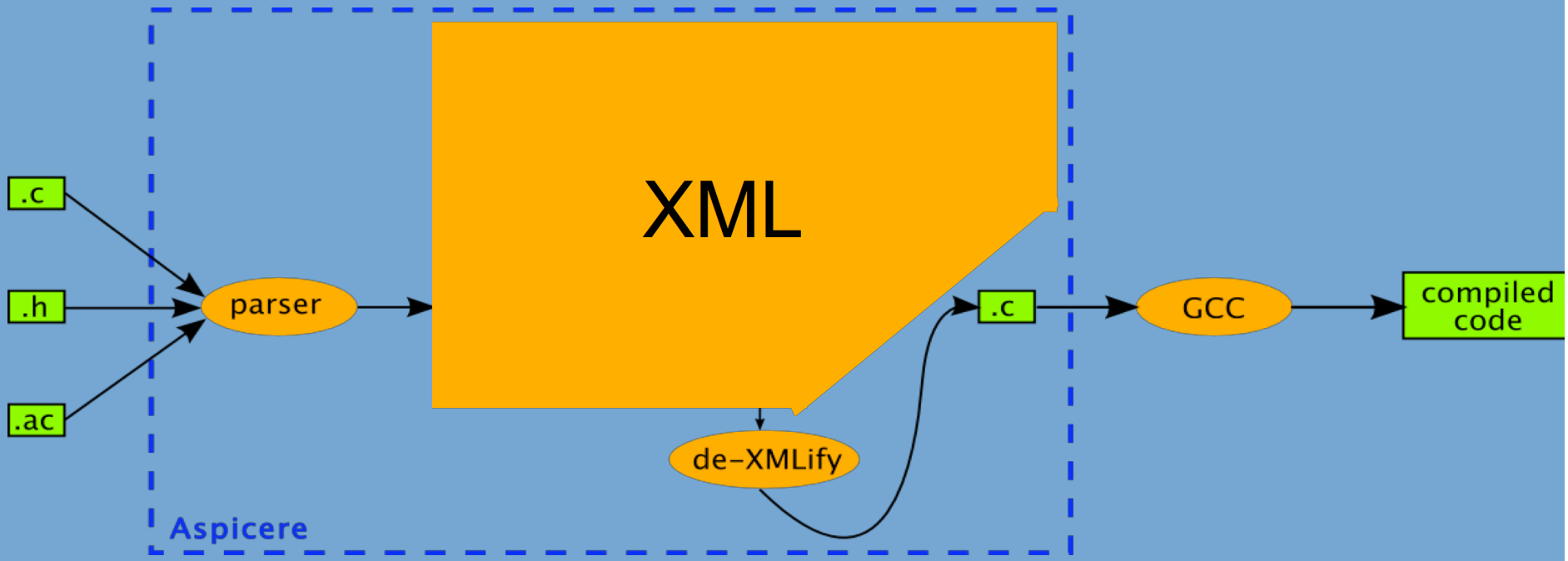
Outline

1. Aspicere, a short introduction
2. Metadata
3. Demonstration

Outline

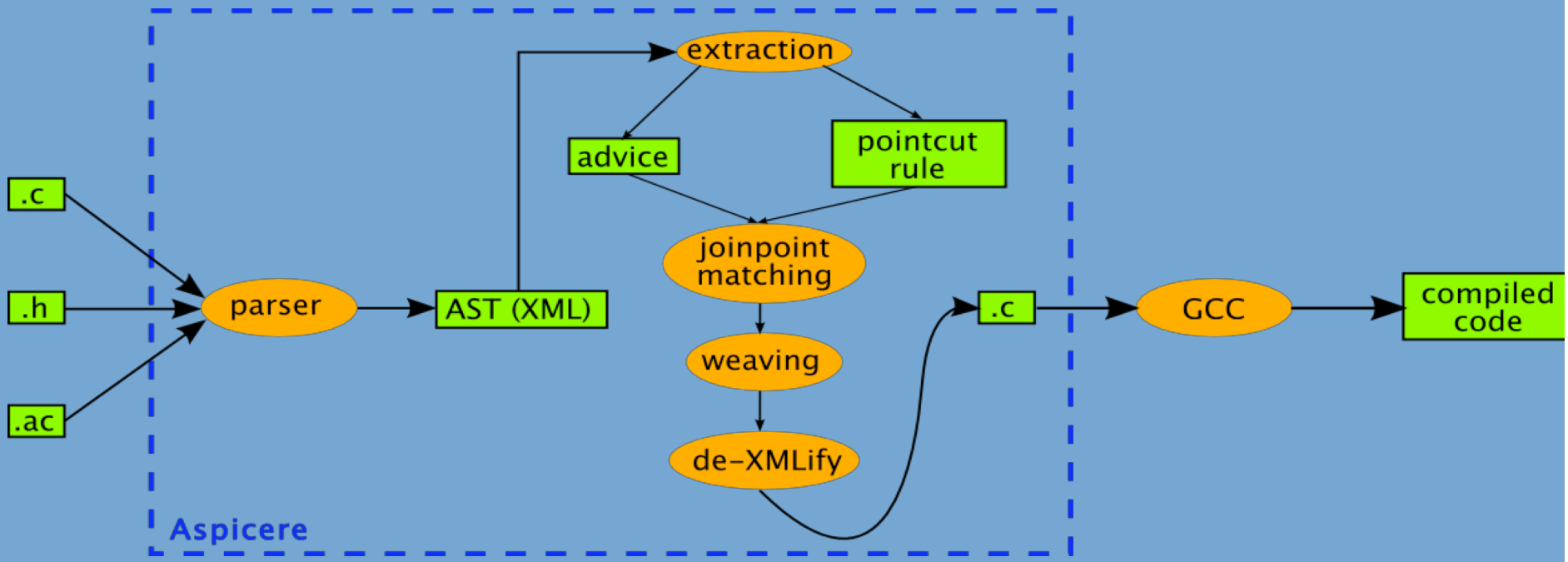
1. Aspicere, a short introduction

General architecture



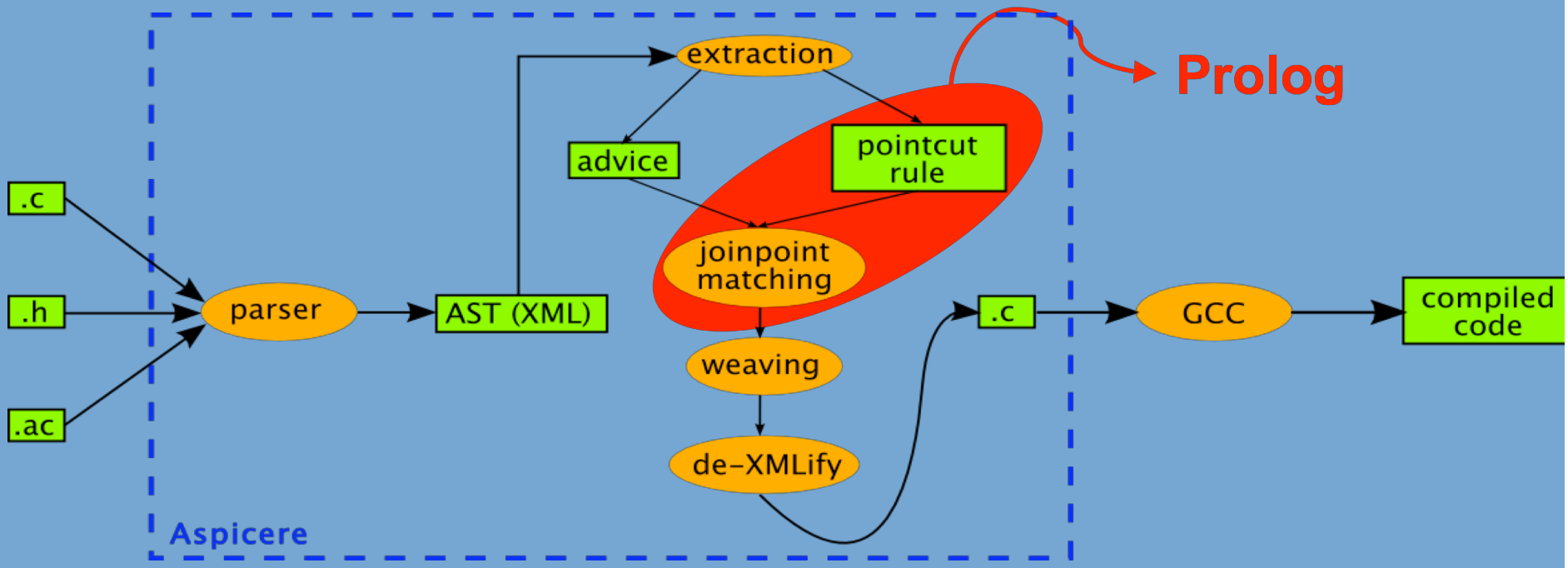
- Weaver ≡ Source-to-source transformer ≡ preprocessor for GCC

General architecture



- Weaver ≡ Source-to-source transformer ≡ preprocessor for GCC

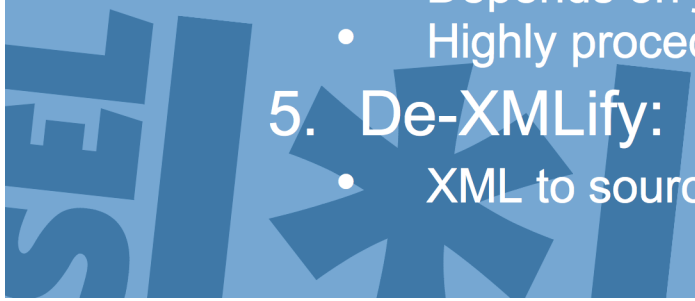
General architecture



- Weaver ≡ Source-to-source transformer
  ≡ preprocessor for GCC

More details

1. Parser:
   - btyacc (backtracking): slowwwwwww ...
   - Antlr: very fast + type-checking
2. Extraction:
   - XSLT + XPath (cached)
3. Joinpoint matching (Prolog):
   - Backward chaining
   - Instantiate joinpoints as needed
   - **Bind** weave-time properties
4. Weaving:
   - Depends on joinpoint type
   - Highly procedural
5. De-XMLify:
   - XML to source code



More details

1. Parser:
   - btyacc (backtracking): slowwwwwww ...
   - Antlr: very fast + type-checking
2. Extraction:
   - XSLT + XPath (cached)
3. Joinpoint matching (Prolog):
   - Backward chaining
   - Instantiate joinpoints as needed
   - **Bind** weave-time properties
4. Weaving:
   - Depends on joinpoint type
   - Highly procedural
5. De-XMLify:
   - XML to source code

} WHY?

Even more details ...

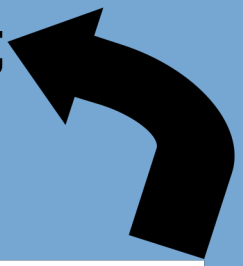
source code

```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

Even more details ...

source code

```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

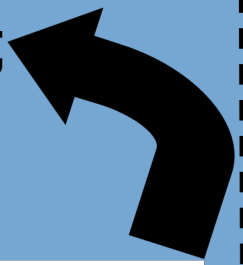


Even more details …

source code

generated code

```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
```



```
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

Even more details …

source code

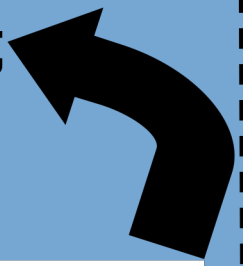
```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
```

```
int advice log() on(Jp):
    ... { ... }
```

```
int f(int* a,double b){
    ...
}
```

generated code

```
int f_caller_proxy(int* a,double b){
    ...
}
```



Even more details ...

source code

```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

generated code

```
int f_caller_proxy(int* a,double b){
    ...
}
```

Even more details ...

source code

```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

generated code

```
int f_caller_proxy(int* a,double b){
    ...
}

void log(thisJoinPoint* jp){
    ...
}
```

Even more details ...

source code

```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

generated code

```
int f_caller_proxy(int* a,double b){
    ...
}

void log(thisJoinPoint* jp){
    ...
}

void f_callee_proxy(thisJoinPoint* jp){
    ...
}
```

Even more details ...

source code

```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

generated code

```
int f_caller_proxy(int* a,double b){
    ...
}

void log(thisJoinPoint* jp){
    ...
}

void f_callee_proxy(thisJoinPoint* jp){
    ...
}
```

Even more details ...

source code

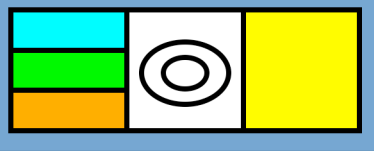
```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
```

```
int advice log() on(Jp):
    ... { ... }
```

```
int f(int* a,double b){
    ...
}
```

generated code

```
int f_caller_proxy(int* a,double b){
    ...
}
```



```
void log(thisJoinPoint* jp){
    ...
}
```

```
void f_callee_proxy(thisJoinPoint* jp){
    ...
}
```

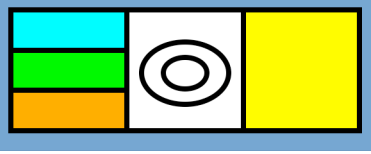
Even more details ...

source code

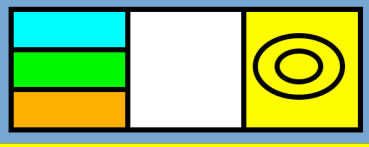
```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

generated code

```
int f_caller_proxy(int* a,double b){
    ...
}
```



```
void log(thisJoinPoint* jp){
    ..
}
```



```
void f_callee_proxy(thisJoinPoint* jp){
    ...
}
```

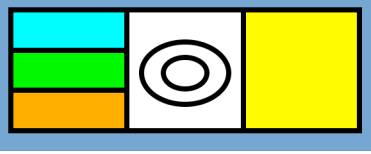
Even more details ...

source code

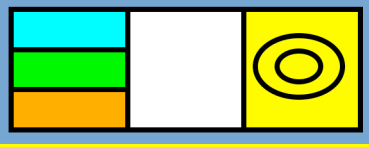
```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

generated code

```
int f_caller_proxy(int* a,double b){
    ...
}
```



```
void log(thisJoinPoint* jp){
    ..
}
```



```
void f_callee_proxy(thisJoinPoint* jp){
    ...
}
```

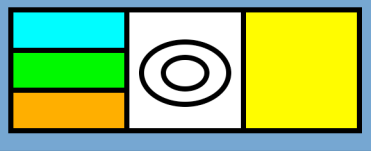
Even more details ...

source code

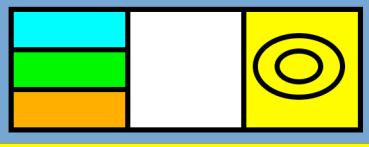
```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

generated code

```
int f_caller_proxy(int* a,double b){
    ...
}
```



```
void log(thisJoinPoint* jp){
    ..
}
```



```
void f_callee_proxy(thisJoinPoint* jp){
    ...
}
```

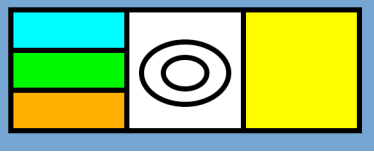
Even more details ...

source code

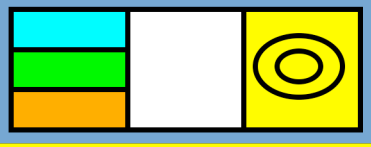
generated code

```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

```
int f_caller_proxy(int* a,double b){
    ...
}
```



```
void log(thisJoinPoint* jp){
    ...
}
```



```
void f_callee_proxy(thisJoinPoint* jp){
    ...
}
```

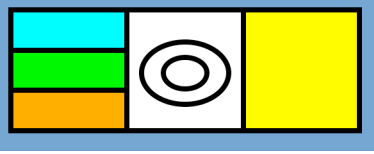
Even more details ...

source code

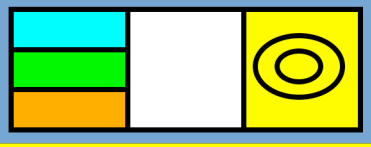
```
int f(int* a,double b);
int main(void){
    ...
    res=f(ptr,5.0);
    ...
}
int advice log() on(Jp):
    ... { ... }
int f(int* a,double b){
    ...
}
```

generated code

```
int f_caller_proxy(int* a,double b){
    ...
}
```



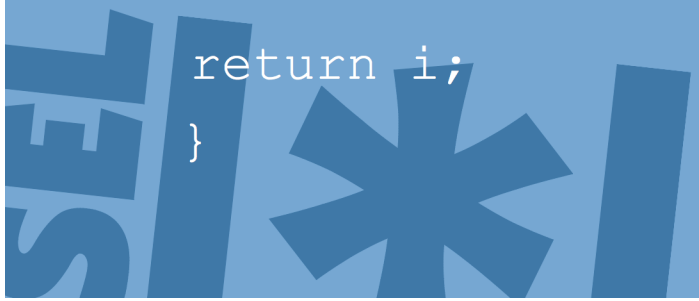
```
void log(thisJoinPoint* jp){
    ...
}
```



```
void f_callee_proxy(thisJoinPoint* jp){
    ...
}
```

Example

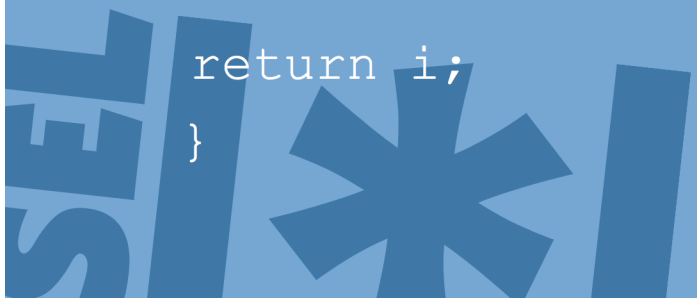
```
ReturnType advice tracing_nonvoid(ReturnType) on (Jp):
  call(Jp,_)
  && type(Jp,ReturnType)
  && !str_matches("void",ReturnType)
    {
    ReturnType i;
    /* Tracing code */
    i = proceed ();
    /* Tracing code */
    return i;
    }
```



Example

```
ReturnType advice tracing_nonvoid(ReturnType) on (Jp):
  call(Jp,_)
  && type(Jp,ReturnType)
  && !str_matches("void",ReturnType)
    {
    ReturnType i;
    /* Tracing code */
    i = proceed ();
    /* Tracing code */
    return i;
    }
```

Prolog predicates

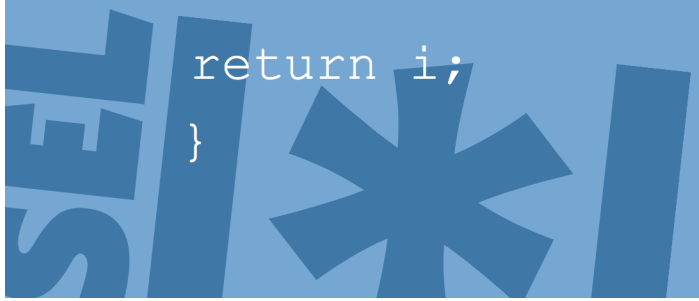


Example

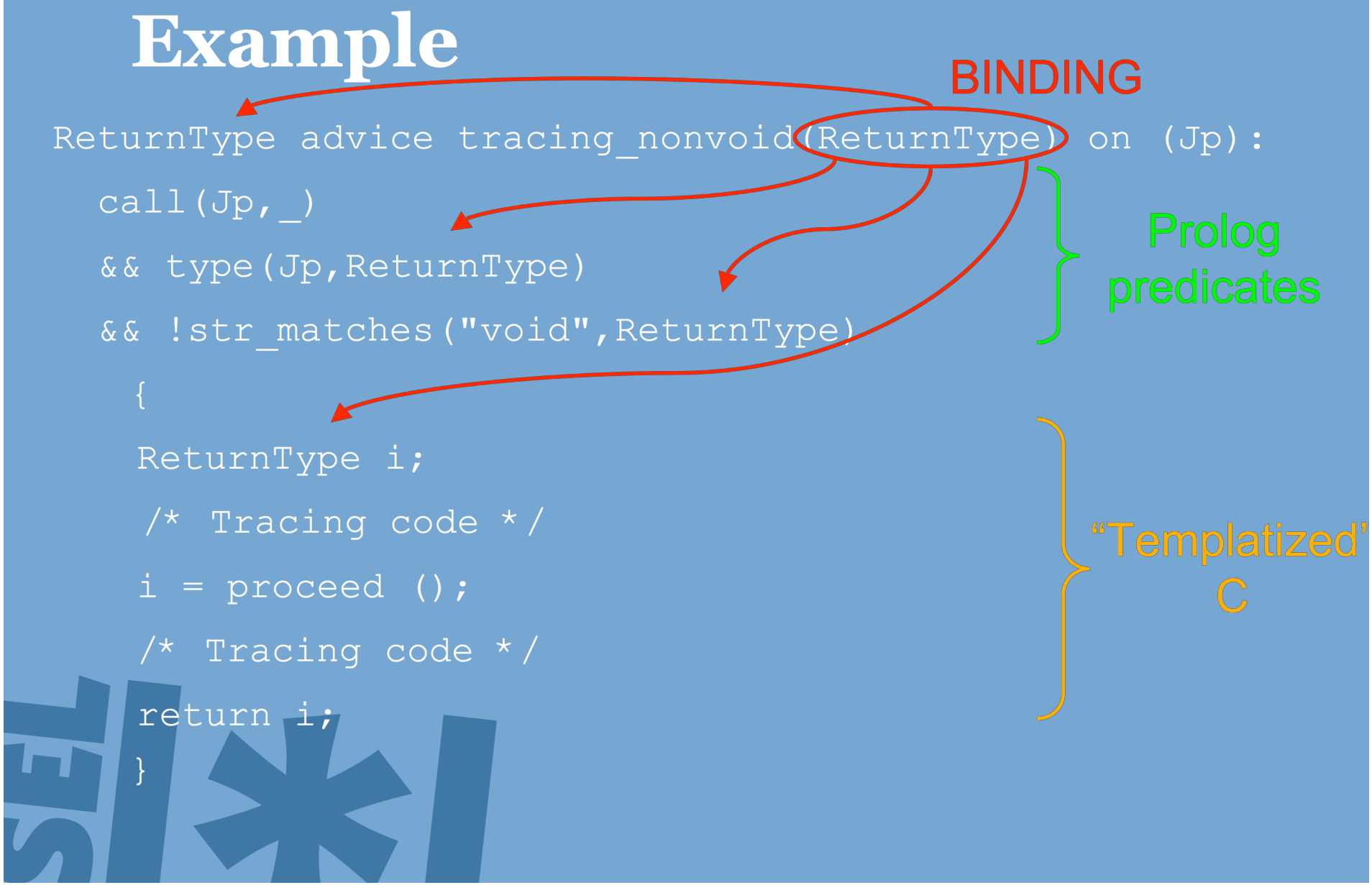
```
ReturnType advice tracing_nonvoid(ReturnType) on (Jp):
  call(Jp,_)
  && type(Jp,ReturnType)
  && !str_matches("void",ReturnType)
    {
    ReturnType i;
     /* Tracing code */
    i = proceed ();
    /* Tracing code */
    return i;
    }
```

Prolog predicates

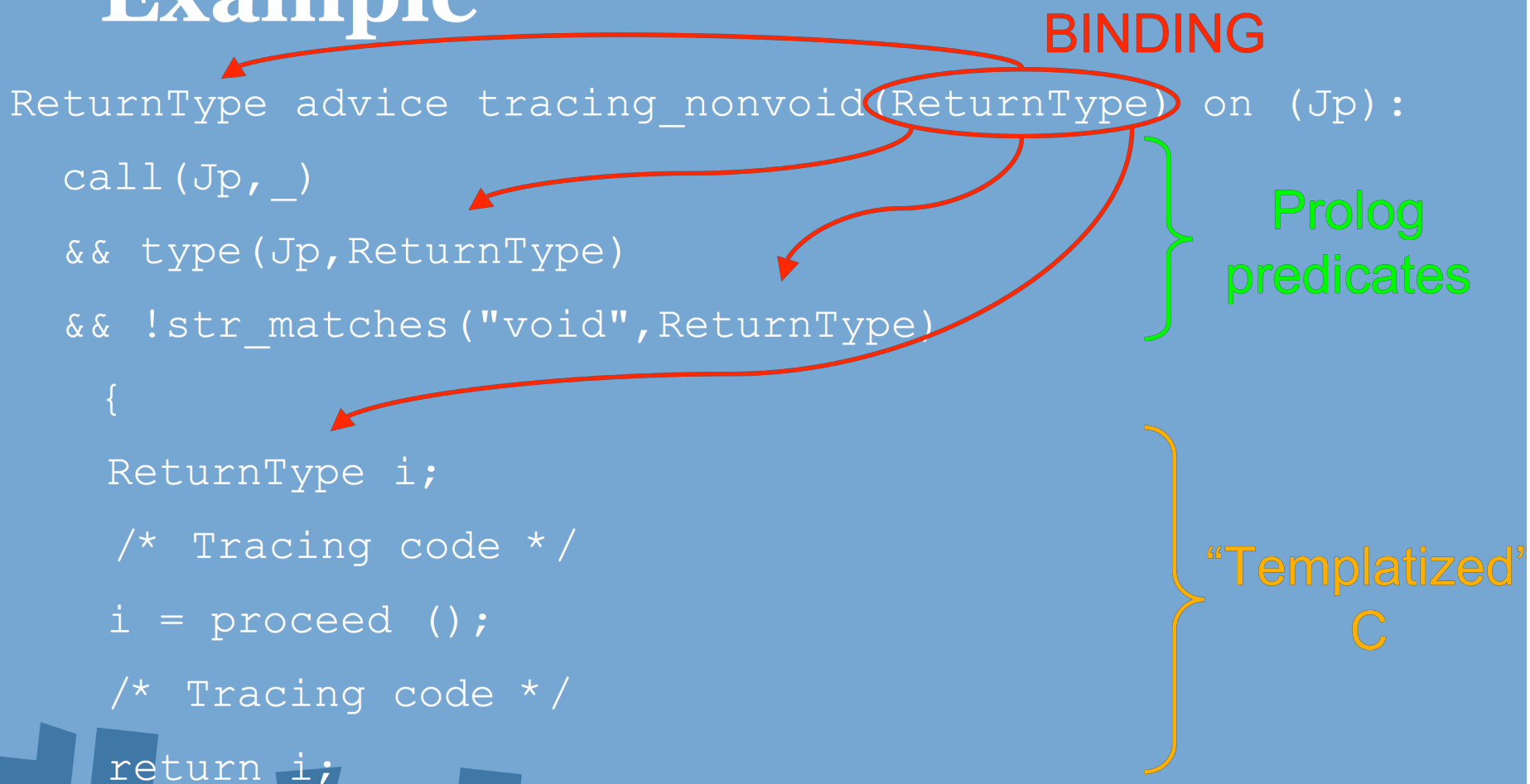
"Templatized" C



Example


BINDING
ReturnType advice tracing_nonvoid(ReturnType) on (Jp):
call(Jp,_)
&& type(Jp,ReturnType)
&& !str_matches("void",ReturnType)
{
ReturnType i;
/* Tracing code */
i = proceed ();
/* Tracing code */
return i;
}
Prolog predicates
"Templatized" C

Example



➡ Aspect ≡ normal compilation unit enhanced with advice

Bindings

- What?
  - Logic variables which are bound and can be used freely throughout advice code
  - ≈ C++ template parameter
  - cf. Kris Gybels' and Johan Brichau's work, Cobble, LogicAJ, ...
- How?
  - Consider tuple of bindings $L=(L_1,...,L_n)$
  - Instantiate advice once for all solutions for L
- Why?
  - Leverage power of Prolog ➔ reusable, robust pointcuts
  - NECESSITY ➔ no Object-class, nor template parameters

generic aspect language

Outline

1. Aspicere, a short introduction
2. Metadata
3. Demonstration

Outline

2. Metadata

Metadata

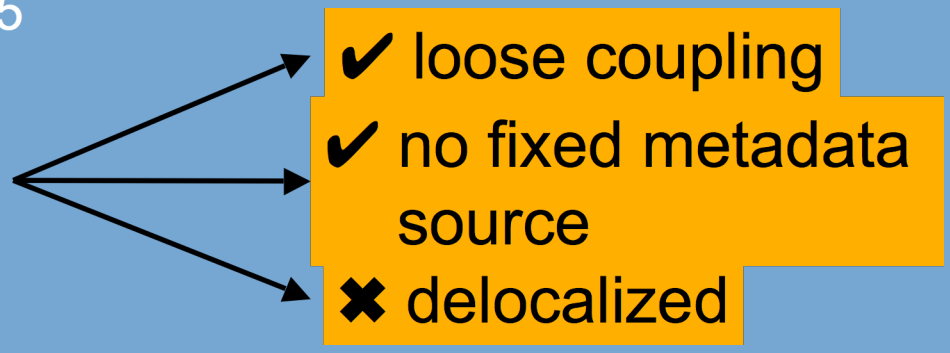
- What?
  - "data about data": semantics, design decisions, conventions, ...
- Why?
  - automated (aspectized) evolution, aspect mining, ...
- How?
  - Documentation ➔ Javadoc, Doxygen, ...
  - Separate file ➔ property files, ...
  - Language support ➔ Java 5 annotations, C# custom attributes
  - AOP introduction ➔ AspectJ 5
- In Aspicere:
  - Prolog facts & rules ≡ ... ∩ ...
- Future:
  - What about annotations in C?

Metadata

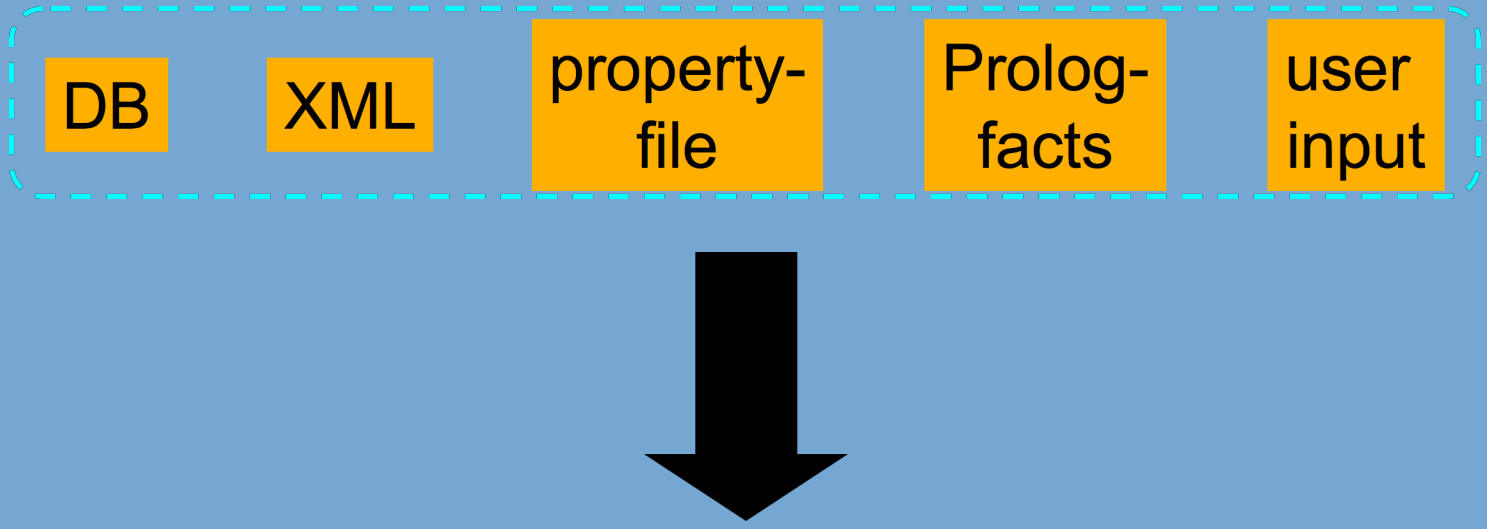
- What?
  - "data about data": semantics, design decisions, conventions, ...
- Why?
  - automated (aspectized) evolution, aspect mining, ...
- How?
  - Documentation ➔ Javadoc, Doxygen, ...
  - Separate file ➔ property files, ...
  - Language support ➔ Java 5 annotations, C# custom attributes
  - AOP introduction ➔ AspectJ 5
- In Aspicere:
  - Prolog facts & rules ≡ ... ∩ ...
- Future:
  - What about annotations in C?

Metadata

- What?
  - "data about data": semantics, design decisions, conventions, ...
- Why?
  - automated (aspectized) evolution, aspect mining, ...
- How?
  - Documentation ➔ Javadoc, Doxygen, ...
  - Separate file ➔ property files, ...
  - Language support ➔ Java 5 annotations, C# custom attributes
  - AOP introduction ➔ AspectJ 5
- In Aspicere:
  - Prolog facts & rules ≡ ... ∩ ...
- Future:
  - What about annotations in C?

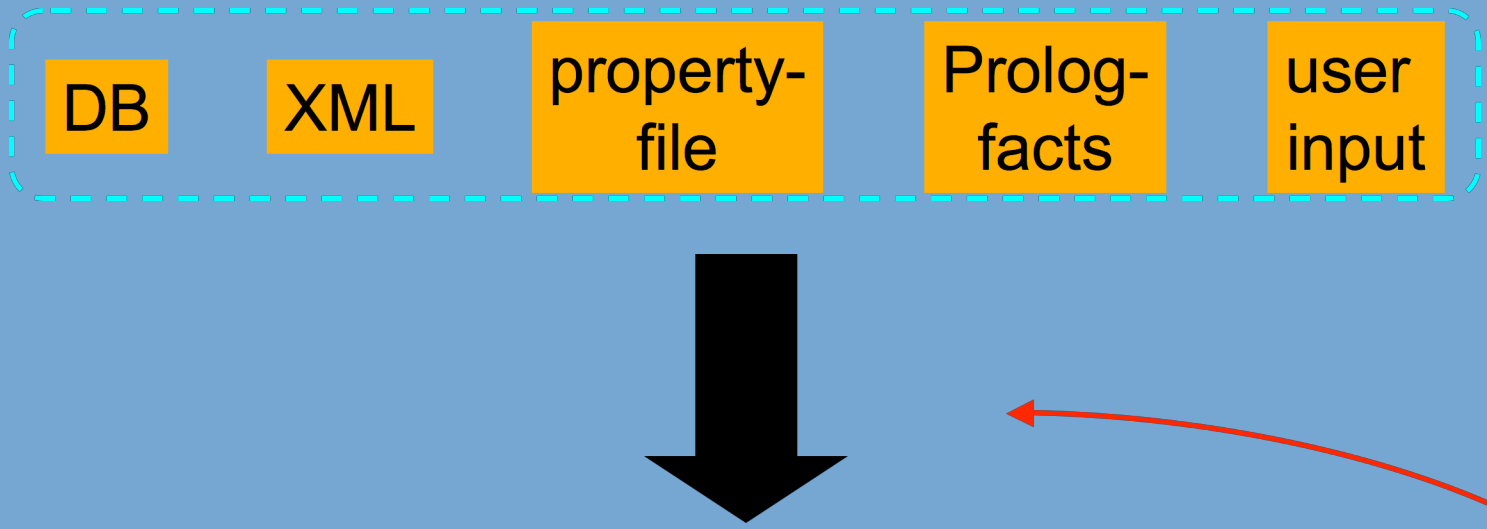


Metadata supply and consumption




```
ReturnType advice serialize(ReturnType) on (Jp):
   call(Jp,Name)
   && type(Jp,ReturnType)
   && transaction(Name)
    { /* ... */ }
```

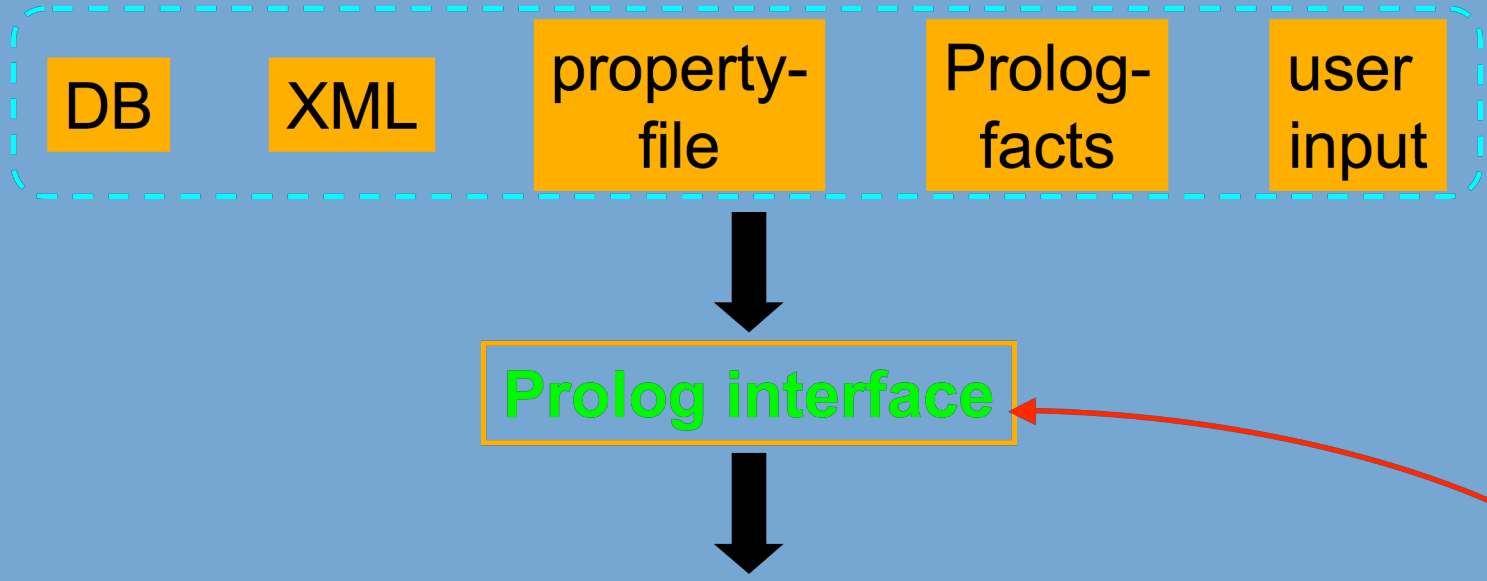
Metadata supply and consumption




```
ReturnType advice serialize(ReturnType) on (Jp):
   call(Jp,Name)
   && type(Jp,ReturnType)
   && transaction(Name)
    { /* ...*/ }
```

metadata

Metadata supply and consumption




```
ReturnType advice serialize(ReturnType) on (Jp):
   call(Jp,Name)
   && type(Jp,ReturnType)
   && transaction(Name)
    { /* ... */ }
```

Outline

1. Aspicere, a short introduction
2. Metadata
3. Demonstration

Outline

3. Demonstration

Conclusion and questions

- Conclusion:
  - Prolog facts and rules enable transparent storing of metadata
  - Aspicere's use of Prolog-like pointcuts allows easy exploitation of metadata
- Questions:
  - Does direct language support for metadata (a.k.a. annotations) yield better evolution opportunities than other mechanisms?
  - What about availability of metadata?

- Brichau, J., Mens, K. and De Volder, K. (2002). *Building composable aspect-specific languages with logic metaprogramming.* In GPCE '02: The ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering, pages 110–127. Springer-Verlag.
- Gybels, K. and Brichau, J. (2003). *Arranging language features for more robust pattern-based crosscuts.* In AOSD '03: Proceedings of the 2nd international conference on Aspect-Oriented Software Development, pages 60–69. ACM Press.
- Kniesel, G., Rho, T. and Hanenberg, S. (2004). *Evolvable Pattern Implementations Need Generic Aspects.* In ECOOP '04: Proceedings of Workshop on Reflection, AOP and Meta-Data for Software Evolution.
- Laddad, R. (2005): "AOP and metadata: A perfect match, Part 1 and 2" (IBM developerWorks)
- Lämmel, R. and De Schutter, K. (2005). *What does aspect-oriented programming mean to Cobol?* In AOSD '05: Proceedings of the 4th international conference on Aspect-Oriented Software Development, pages 99–110 . ACM Press.
- Loughran, N. and Rashid (2003). *Supporting Evolution in Software using Frame Technology and Aspect-Orientation*. Workshop on Software Variability Management, Groningen, The Netherlands.