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 $\equiv$ Source-to-source transformer
  $\equiv$ preprocessor for GCC
General architecture

- **Weaver** = Source-to-source transformer preprocessor for GCC

Diagram: [Diagram of the General Architecture]
More details

1. Parser:
   - btyacc (backtracking): slowwwwwwww ...
   - 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): slowwwwwwwww ...
   - 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 ...

```
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){
    ...
}
```
source code

```c
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):
    ... { ... }
```

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

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

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

generated code

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

```c
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

```c
int f_caller_proxy(int* a, double b){
    ...
}
void log(thisJoinPoint* jp){
    ...
}
```
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_caller_proxy(thisJoinPoint* jp) {
    ...
}

void f_callee_proxy(thisJoinPoint* jp) {
    ...
}
int f(int* a, double b);
int main(void) {
    ...
    res = f(pir, 5.0);
    ...
}

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

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

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

int advice log() on(Jp) {
    ...
    {...} ...
}
Even more details ...

source code

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

generated code

```c
int f_caller_proxy(int* a, double b){
    ...
}
void log(thisJoinPoint* jp){
    ...
}
void f_caller_proxy(thisJoinPoint* jp){
    ...
}
```
int f(int* a, double b);
int main(void){
    ...
    res = f(pir, 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 ...
Even more details ...

source code

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

generated code

```c
int f_caller_proxy(int* a, double b) {
    ...
}
void log(thisJoinPoint* jp) {
    ...
}
int f_callee_proxy(thisJoinPoint* jp) {
    ...
}
void f_callee_proxy(thisJoinPoint* jp) {
    ...
}```
Even more details ...

source code

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

generated code

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

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

void f_callee_proxy(thisJoinPoint* jp) {
    ...
}
```
int f(int* a, double b);
int main(void) {
    ...
    res = f(pir, 5.0);
    ...
}

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

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

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

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

int f_caller_proxy(int* a, double b) {
    ...
}
Even more details ...

source code

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

generated code

```c
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;
}
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;
    }

Aspect = normal compilation unit enhanced with advice
Bindings

• What?
  • Logic variables which are bound and can be used freely throughout advice code
  • $\approx$ 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 $\Rightarrow$ reusable, robust pointcuts
  • NECESSITY $\Rightarrow$ no Object-class, nor template parameters

generic aspect language
Outline

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

✅ loose coupling
✅ no fixed metadata source
❌ delocalized
Metadata supply and consumption

ReturnType advice serialize(ReturnType) on (Jp):

call(Jp,Name)

&& type(Jp,ReturnType)

&& transaction(Name)

{ /*...*/ }
ReturnType advice serialize(ReturnType) on (Jp):

call(Jp,Name)

&& type(Jp,ReturnType)

&& transaction(Name)

{ /* ... */ }
Metadata supply and consumption

DB  XML  property-file  Prolog-facts  user input

Prolog interface

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?


