

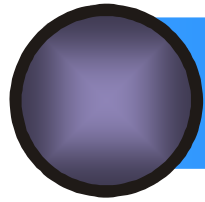


Aeronautical Institute of Technology (ITA)
Federal University of São Paulo (UNIFESP)

2nd Workshop on Assessment of Contemporary
Modularization Techniques (AcoM.08) – OOPSLA 2008

Using Metadata in Aspect-Oriented Frameworks

***Eduardo M. Guerra
Jefferson O. Silva
Fábio F. Silveira
Clóvis T. Fernandes***

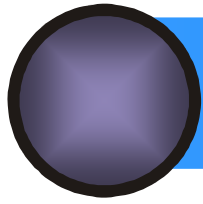


Starting with an Example...

Logging is the “classical” crosscutting concern and there are many examples that use aspects to modularize this functionality.

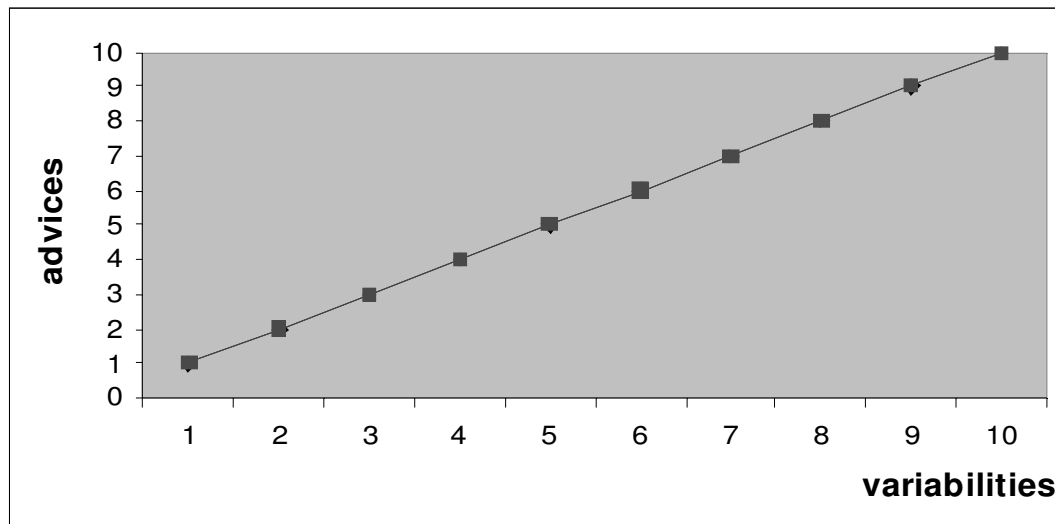
But one can imagine an application that has variations on how each method should be logged:

- *What? Method name; declared exceptions; thrown exception; arguments; parameters types; return type; return ...*
- *Where? Database; Files; Socket; Encrypted File ...*

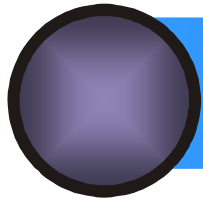


Using Aspects to Solve this Problem

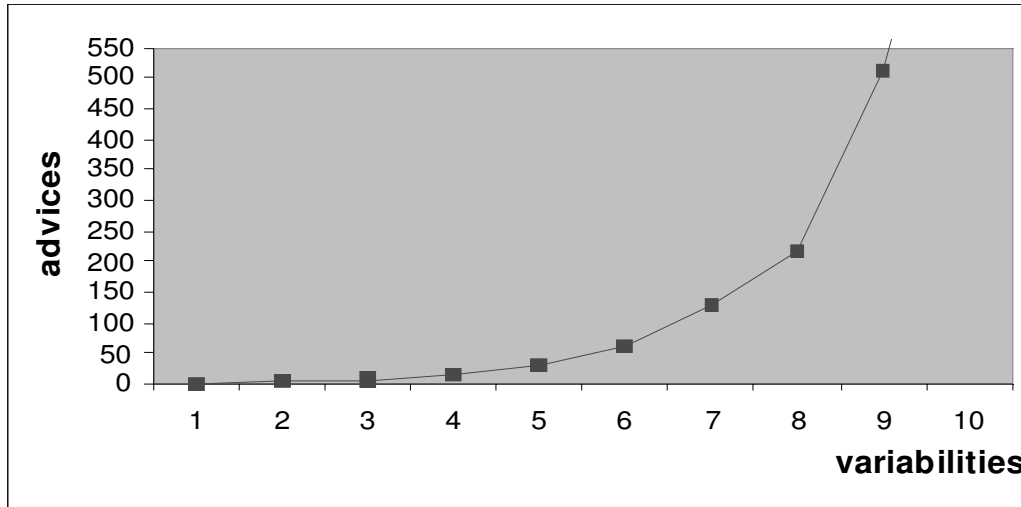
In a traditional approach, each variation will have an advice to describe what will be the behaviour for each method.



In the case that only one variation is possible per method, for each one, an advice will be needed.

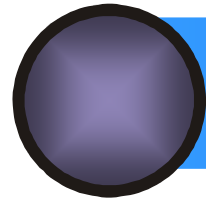


Big problem!!!



In the case that it is possible to combine variations, the number of advices needed grows exponentially!!!

This solution is also not flexible, because the addition of a new variation has a great impact in the number of advices.

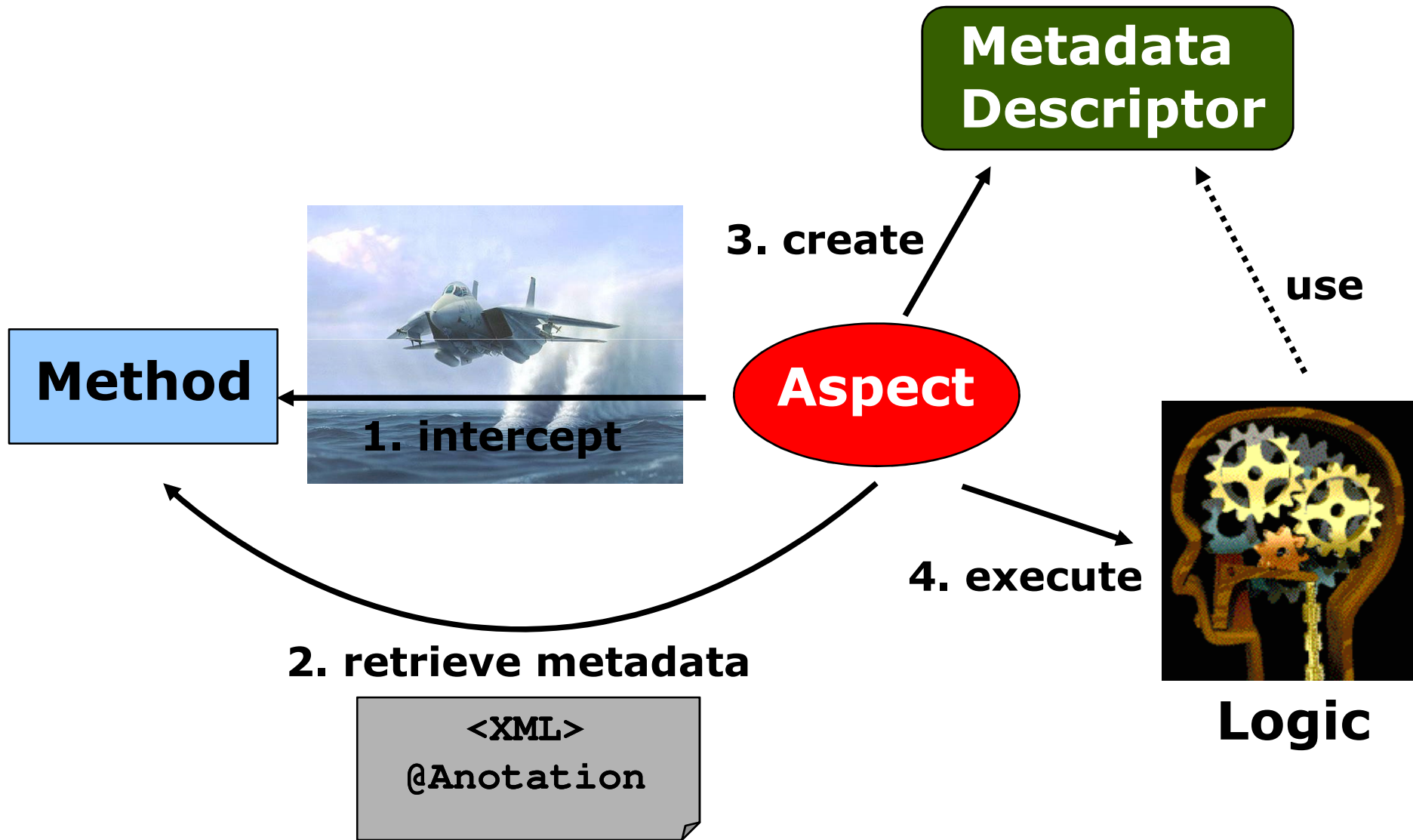


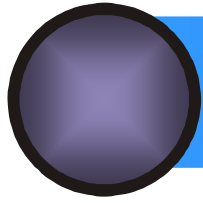
Using Metadata in Aspects

This paper proposes the use of metadata in application classes to define variability information to be used by the aspect behaviour for runtime adaptation.

In other words: there will be only one advice that reads class metadata and decides what/how will be the behaviour for such an situation.

How this approach works...

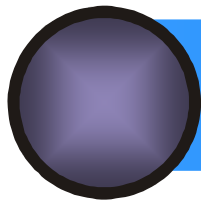




Benefits from using Metadata

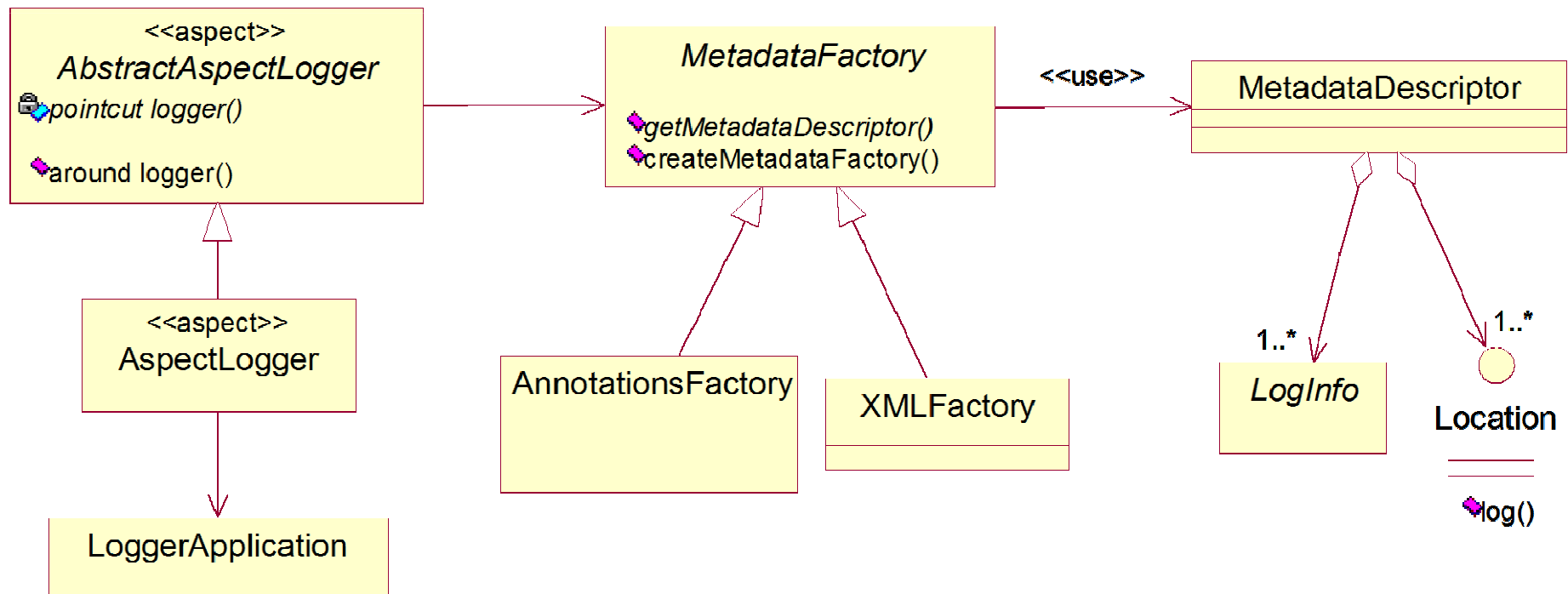


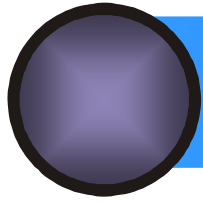
- ***Eliminate syntactic coupling:*** is not needed to inherit abstract aspects to implement a specific behaviour for each method.
- ***Provide extensibility on variabilities:*** new variabilities can be created without the creation of new advices.
- ***Reduce the number of advices:*** the framework deals with combinations without exponential explosions.
- ***Simplify the pointcut management:*** pointcuts will not be so granular and can be defined in a general way.



Metadata-Based Logger

The Metadata-Based Logger is a case study for the construction of an aspect-oriented framework based on metadata.



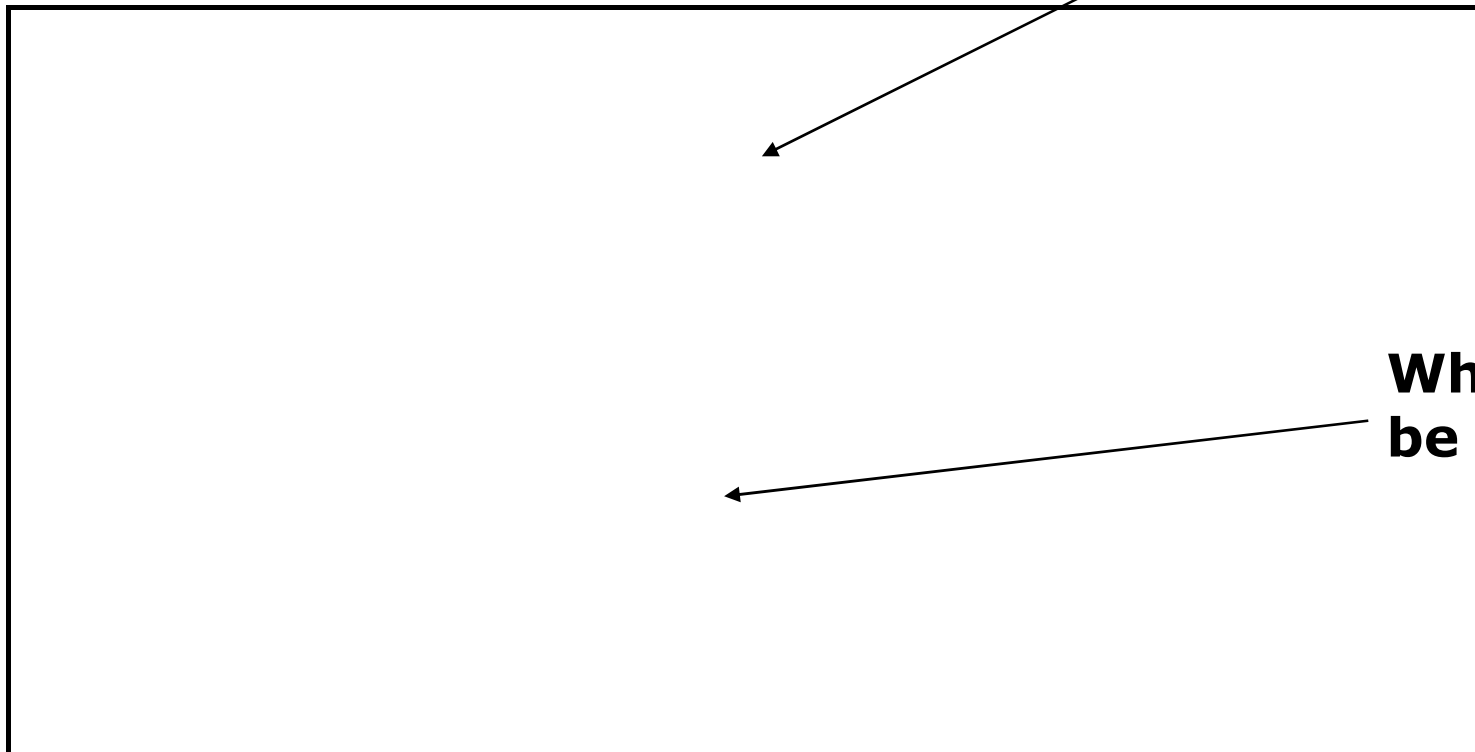


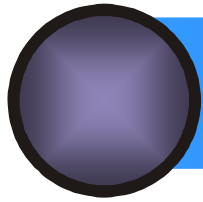
Annotations Metadata

**What have to
be logged?**



**Where have to
be logged?**





XML Metadata

**Method
Information**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<logmarker>
  <method signature="void logging.LoggerApplication.testLog(Integer)">
    <info>
      <value>Method</value>
      <value>Parameter</value>
      <value>Return Type</value>
      <value>Exception</value>
    </info>
    <type>
      <value>DB</value>
      <value>File</value>
    </type>
  </method>
</logmarker>
```

**What does it
have to be
logged?**

**Where does it
have to be
logged?**

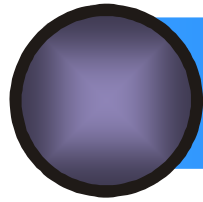


Table-based Calculation Framework

```
public pointcut obtainValue():
    call (* BaseSalary.getValue())
    && withincode (* Employee.calculateMySalaryBasedOnEvents(int, int));
public String getNameOfTableClassWithPackage() {
    return "tableBasedCalculation.instantiation.MealTicket";
}
public String getValueType() { return "float"; }
```

Table-based Calculation is a AOF that has the objective to increment or decrement intercepted values from the base code and provides composition alternatives, which are implemented in abstract aspects that contain the logic for the extraction of values from the base code.

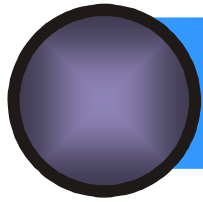
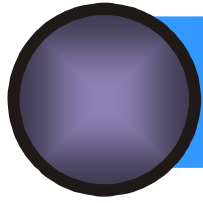


Table-based with Metadata

```
@Calculation(  
    tableClass = {"MealTicket"},  
    valueType = {"float"}  
)  
public float getValue() {  
    return value;  
}
```

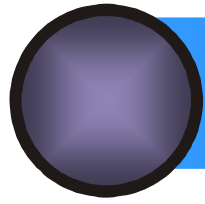
Using the proposed approach, references to the framework classes do not need to be into the aspect. Base code methods could have been annotated or described in XML, which contains these references.



Future Works

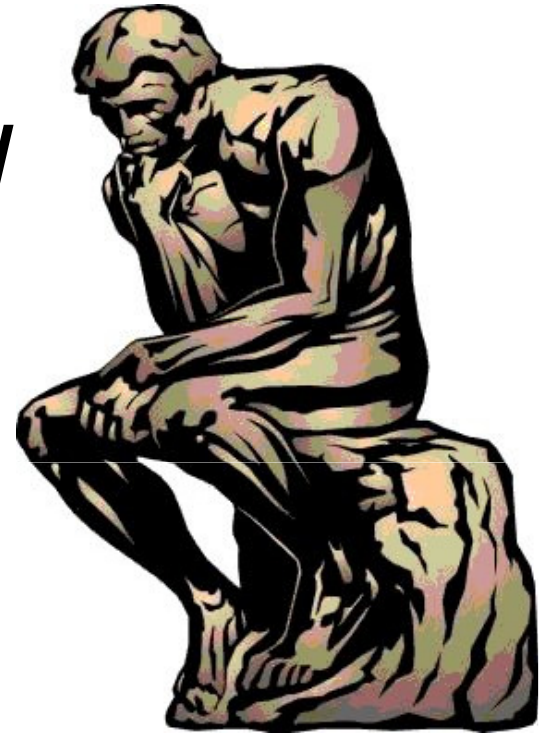


- *Create another aspect-oriented framework based on metadata for another crosscutting concern (maybe for calling notification).*
- *Experiment the use of those frameworks in an application and made a comparative study with the classical approach.*



Final Words

*A framework is not good
for what it does, but
when it can be used for
things that it doesn't
do.*



Thanks for your attention!

Questions???