# ECOOP Paper

## *Context*

Integrating components to a legacy system requires binding overlapping concepts in the legacy system and the component (JasCo paper has a similar motivation). This is usually performed by implementing adapters. Our solution is also adapter-based, but our adapters are initialized and injected with an injection framework (Spring framework provides a similar mechanism?).
We need to present our work as a framework, this requires

- figuring out the architecture, what does this framework look like
- what does a typical work flow that uses our framework look like
- which case study can we use to illustrate our contribution
- How to integrate instance pointcuts with adapter injections?
- A well rounded motivation
    - what are the shortcomings of the existing approaches, in which respect our solution is superior?

## Outline

1 Introduction
  1.1 Setting the context
    1.1.1 what is our level : implementation
    1.1.2 concept binding between components by translating from one to the other
  1.2 Brief problem statement
  1.3 What is our solution
    1.3.1 A dependency injection framework which automatically adapts objects of incompatible types
    1.3.2 Integrated with instance point-cuts this framework allows selectively adapting groups of objects
2 Motivation
  2.1 Loose coupling
  2.2 Non-intrusive integration
  2.3 Light-weight framework (this is more of a feature than a motivation point, but it can be motivated)
3 The Framework
  3.1 The basic architecture of the framework
    3.1.1 What are the individual features
      3.1.1.1 Adapter classes
        3.1.1.1.1 The @Adapter annotation
        3.1.1.1.2 how are they constructed, according to extended types and adaptee fields (what is the naming convention for adaptee fields?)
        3.1.1.1.3 The data structure of the adapter registry
      3.1.1.2 Injection language

3.1.1.2.1 new, single, method return / semantics
3.1.1.2.2 integration with instance pointcuts / selective adaptation
3.1.1.3 Adapter resolution
3.1.1.3.1 What is the logic for adapter resolution
3.1.1.3.2 Precedence delcarations
3.1.2 How do these features fit together: this will be kind of apparent from the previous explanations but this section is more about offering a way to use our framework
3.1.2.1 A methodology/guidelines for using our framework
3.2 The run-time behavior
3.3 Checks
4 Evaluation
4.1 We claim that the approach is non-intrusive
4.1.1 What kind of advantages in terms of non-intrusiveness we provide by comparing two component binding strategies?
4.1.2 Why is our solution superior?
5 Related Work
5.1 ??
6 Conclusion and future work...

Misc: It's nice to have a running example throughout the paper, Arnout is working on a Logging example, I was thinking about a GUI example with a well known GUI library however it might be a lot of work and we don't have that much time. I think we should emphasize usage-based adaptation that is elaborated in the possible use cases (number 2), however this cannot be a made-up example, so I will search for a simple real-life example that will support the the need for such an adaptation. Usage-based adaptation is also the integration point between adapter injection and instance point-cuts.

## Possible use cases

1. **Message translation and implicit routing (type-based routing)**

   Adapters are used as message translators in component integration. The routing of the translated messages as usually content based, looking at the header which contains information about the message type. With adapter injection framework this information is declared in the association declarations as types. This provides a type-based routing of messages.

2. **Usage-based adaptation**

   Instance pointcuts allow objects to be selected according to how they are used. Integrated with the adapter injection framework, this allows wrapping objects based on their uses in the system.

   *Implicit content change case*

   Some methods can change the contents of an object, which may require the object to be handled later in its life cycle. These changes may not necessarily require creating an object of different type.

   Example: A file object is passed to the addHeader() method, this method appends a header at the beginning of the file contents. This method changes the content of the file. This is an implicit event. The type of the object is still File, so how will the system know if this file has a header or not, if a flag is not present? Instance pointcuts can bookkeep such objects.

3. Object Filtering / Message Filter (redundant objects)

   Instance pointcuts can also handle this case.

~~4. Model Transformation~~

   ~~How can we use our framework to implement model transformers? Can it be integrated with a model transformation approach?~~