Metawidget

“UI Generation done right”

http://metawidget.org

What we will cover

- A common requirement
- Current practices
- A better way

Common Requirement

An everyday problem

Most enterprise applications require many different data entry forms, either for collecting or displaying data

Current practices

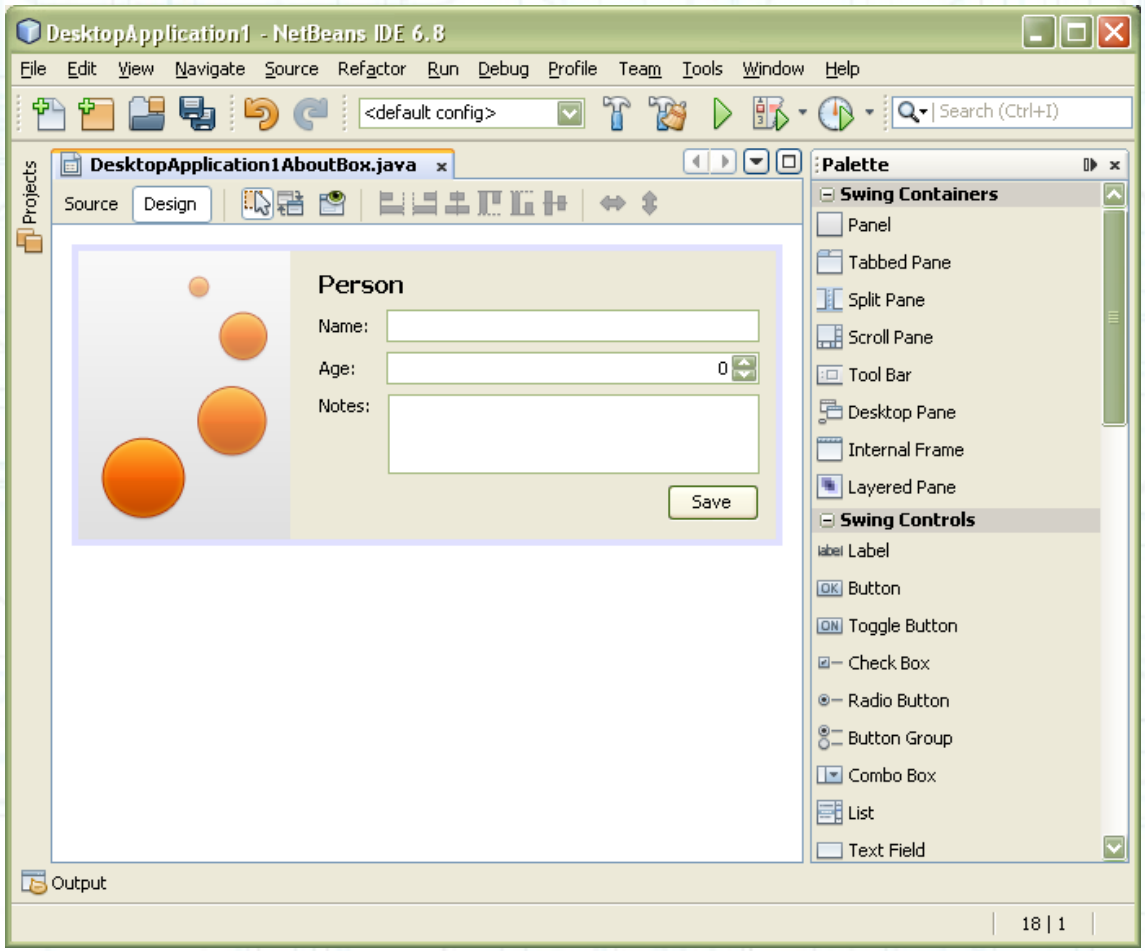
Which one are you using?

- Visual Form Designers
- UI Languages
- Code Generators

Current practices

Visual Form Designers

- Matisse
- JBoss Visual Page Editor
- etc



Current practices

UI Languages

- HTML/CSS
- Java Server Faces
- etc

```
<h:form>

    <h:inputText value="#{foo.name}"/>
    <rich:inputSpinner value="#{foo.age}"/>

</h:form>
```

Current practices

Is this you?

- ✘ Time consuming
- ✘ Duplicating definitions, error prone:

```
public String getName();
public int getAge();

<h:inputText value="#{foo.name}"/>
<rich:inputSpinner value="#{foo.age}"/>
```

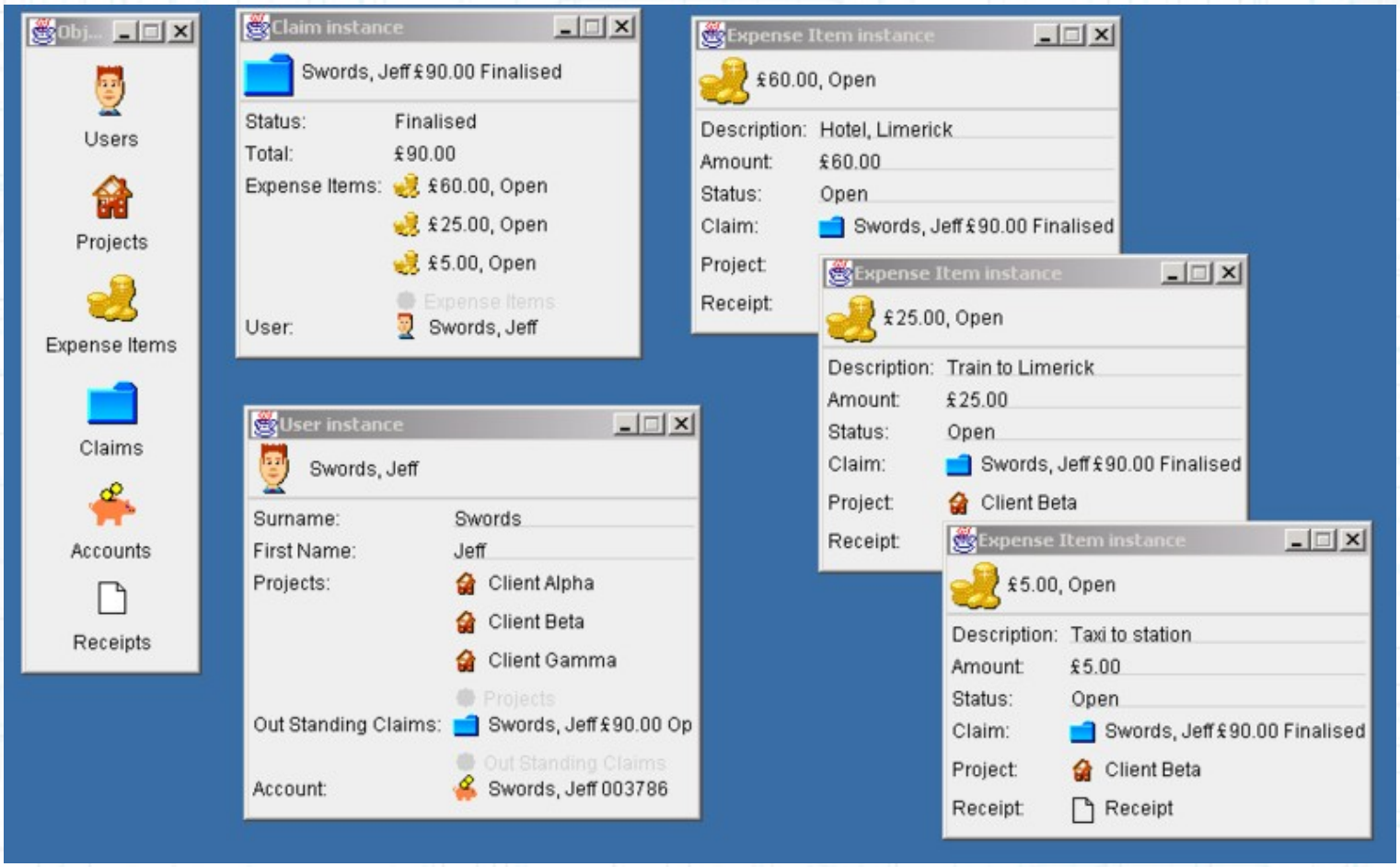
- ✘ Too laborious to do properly:

```
<h:inputText value="#{foo.name}" maxlength="30"/>
```

Current practices

Code Generators

- Naked Objects
- seam-gen
- etc



Current practices

Is this you?

- ✘ Static code generation
  - doesn't help much beyond early stages of development
- ✘ Generic UI
  - basic CRUD
  - isn't enough metadata to do as good a job as a human designer
- ✘ Dictate the architecture
  - if you build your app our way, we'll generate a UI for you

A better way

Metawidget

Designed to address each of these shortcomings

A better way

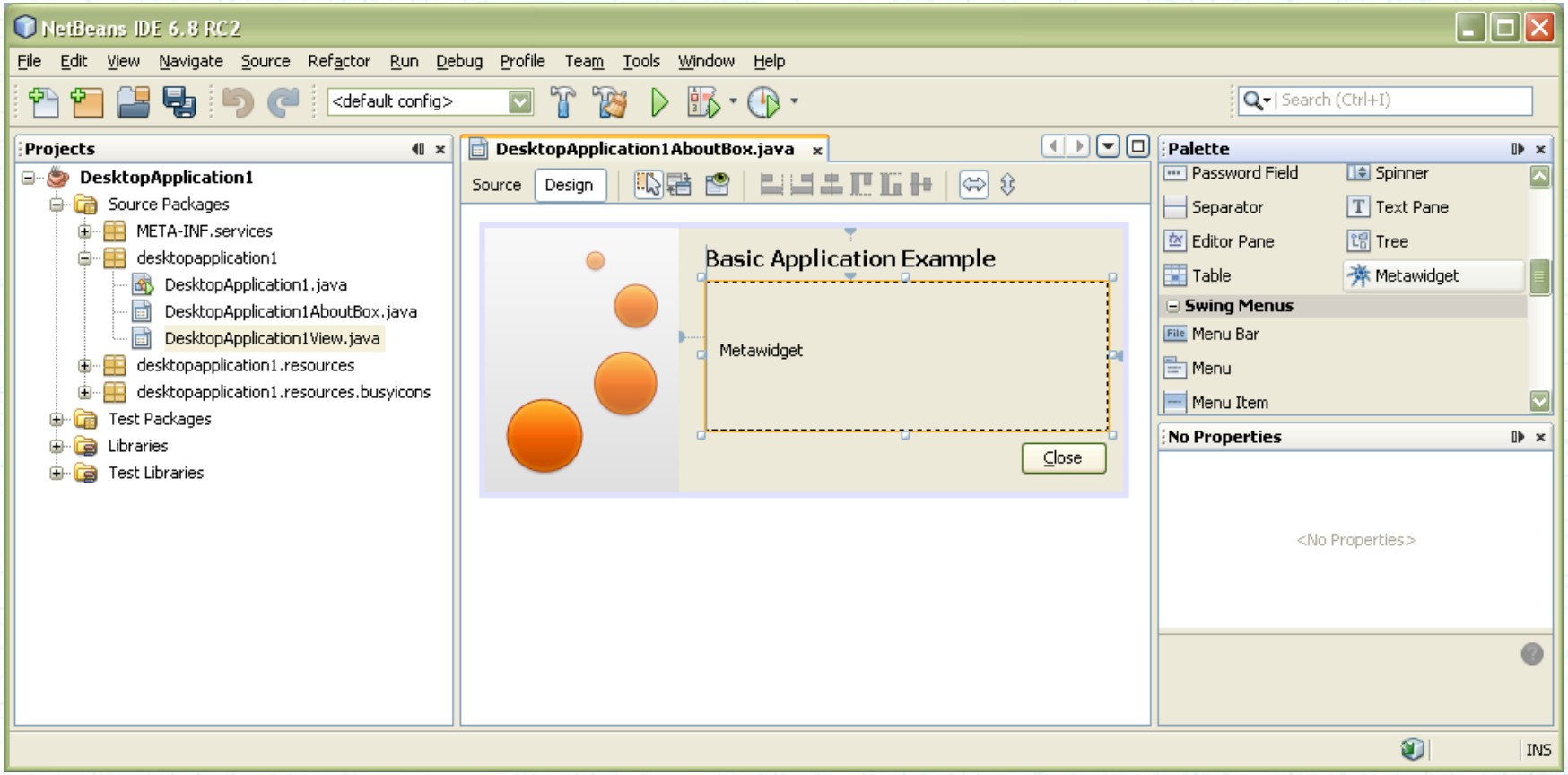
- ✔ Uses your existing architecture
  - *your existing annotations, XML files, business rules*
  - *your existing UI toolkit, third-party libraries, custom components*
  - *easy to mix technologies, or plug-in your own*
- ✔ Doesn't try and 'own' the entire UI
  - *only tries to generate the 'inside' of forms*
  - *doesn't hide your existing UI toolkit*
  - *just another widget in your toolbox*
- ✔ No static code generation
  - *inspects business objects at runtime*

A better way

- ✔ Automatically applies constraints
  - existing validation libraries, easy to add your own
- ✔ No duplicated definitions
  - reads names, types, constraints already defined in your architecture
- ✔ No time at all
  - once configured, changes to screens are free

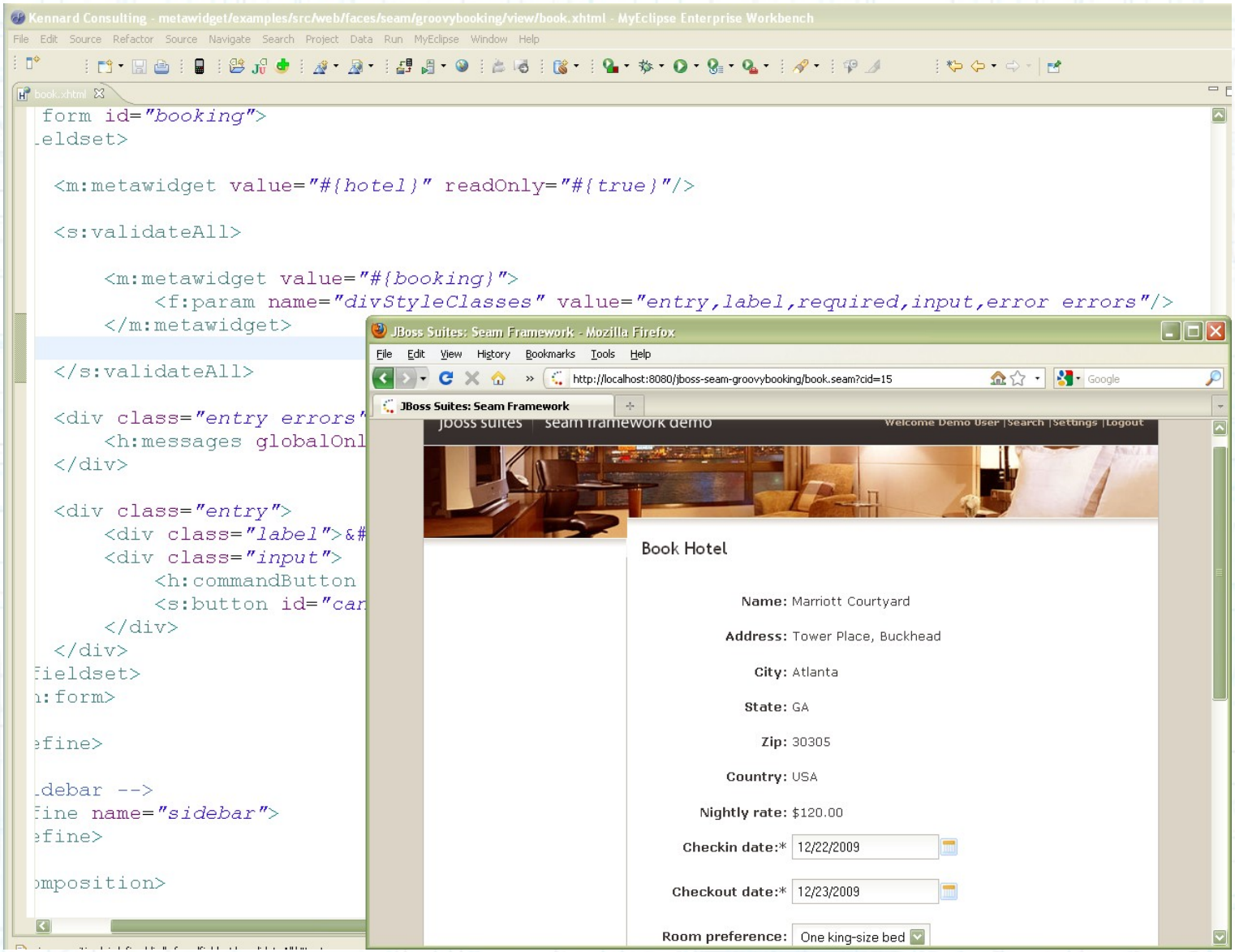
A better way

Doesn't 'own' the UI



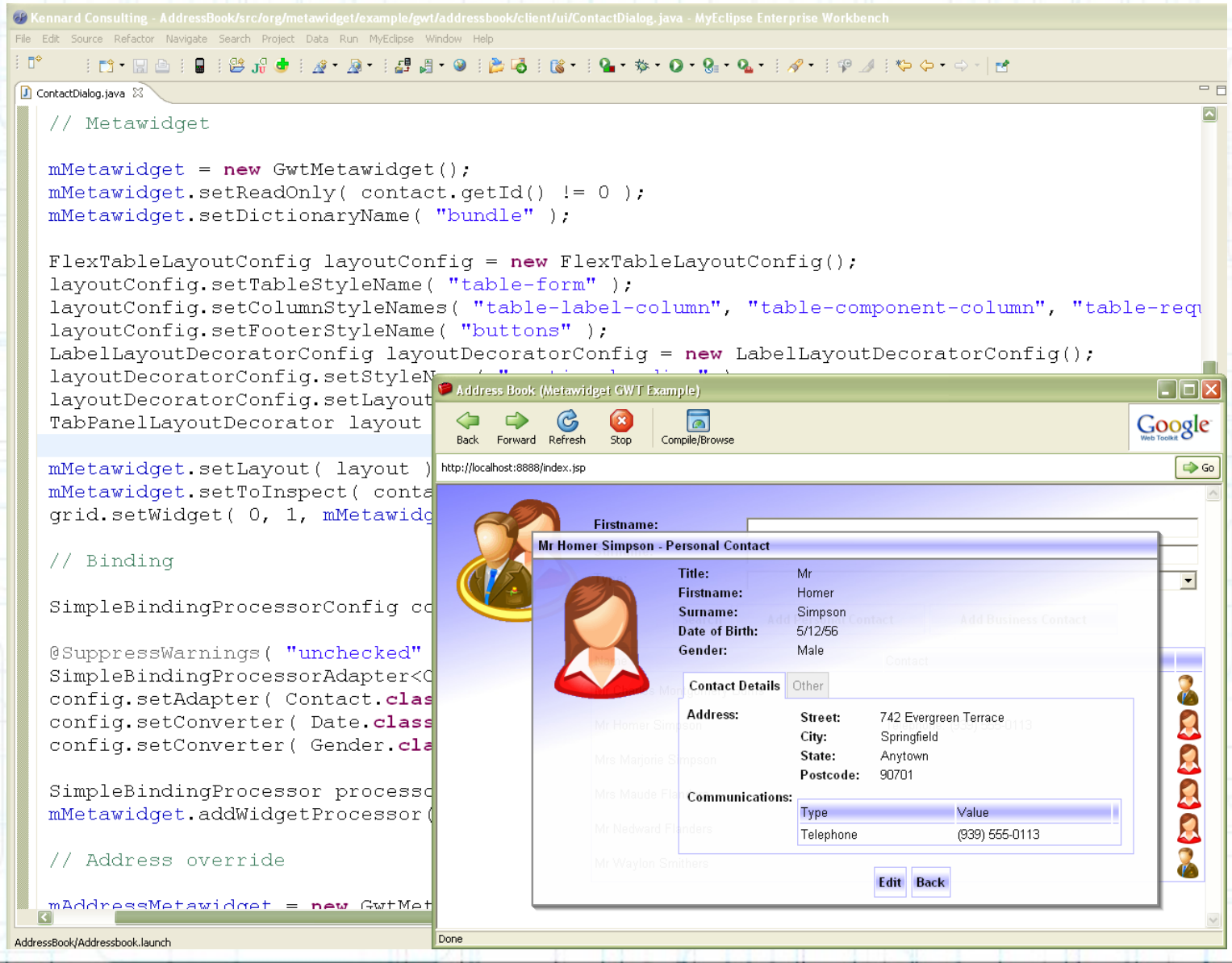
A better way

Doesn't 'own' the UI



A better way

Doesn't 'own' the UI



Conclusion

- Everyday requirement
- Unsatisfactory current practices
- A better way

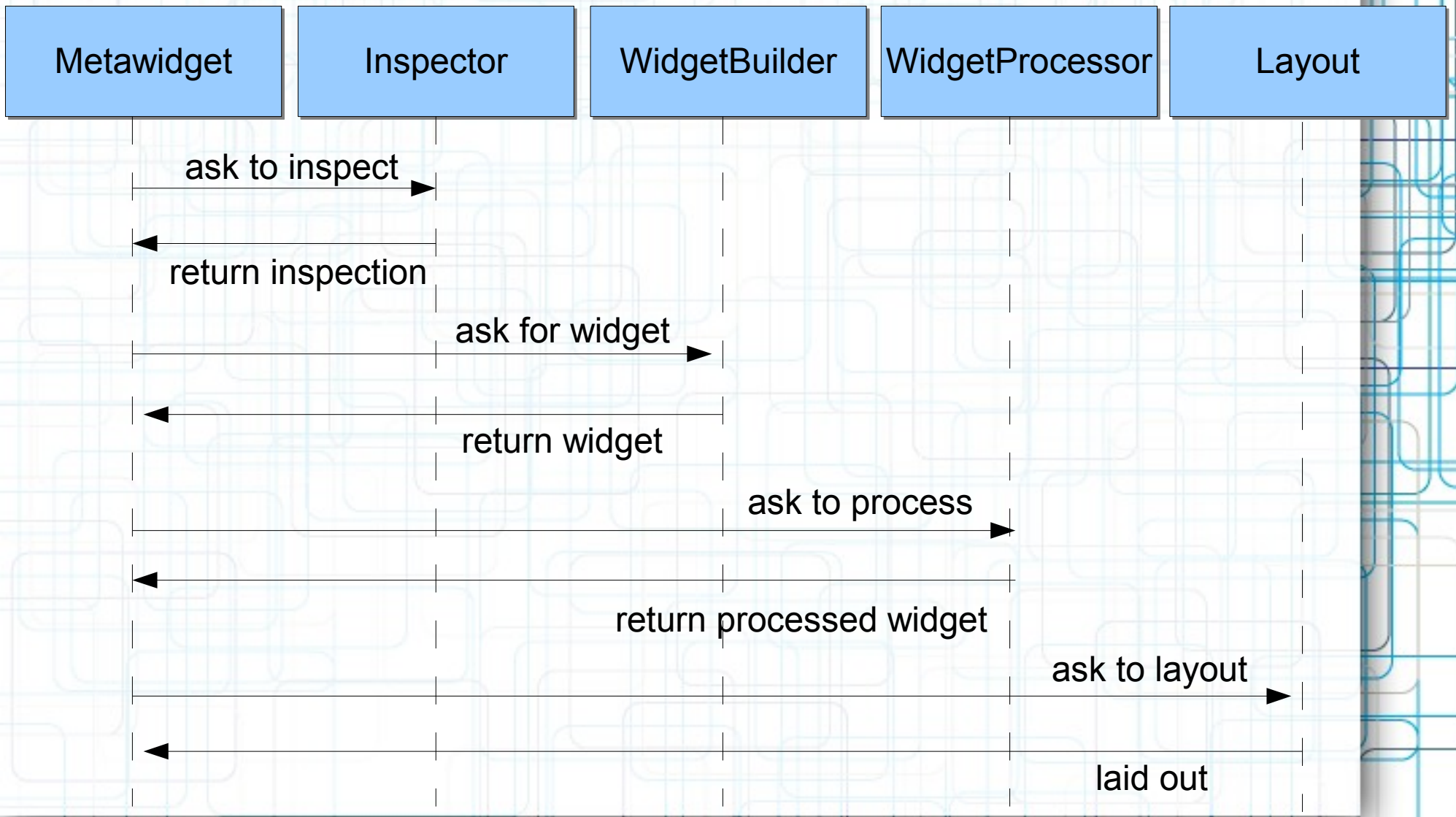
Thank You!

Questions?

Appendix A

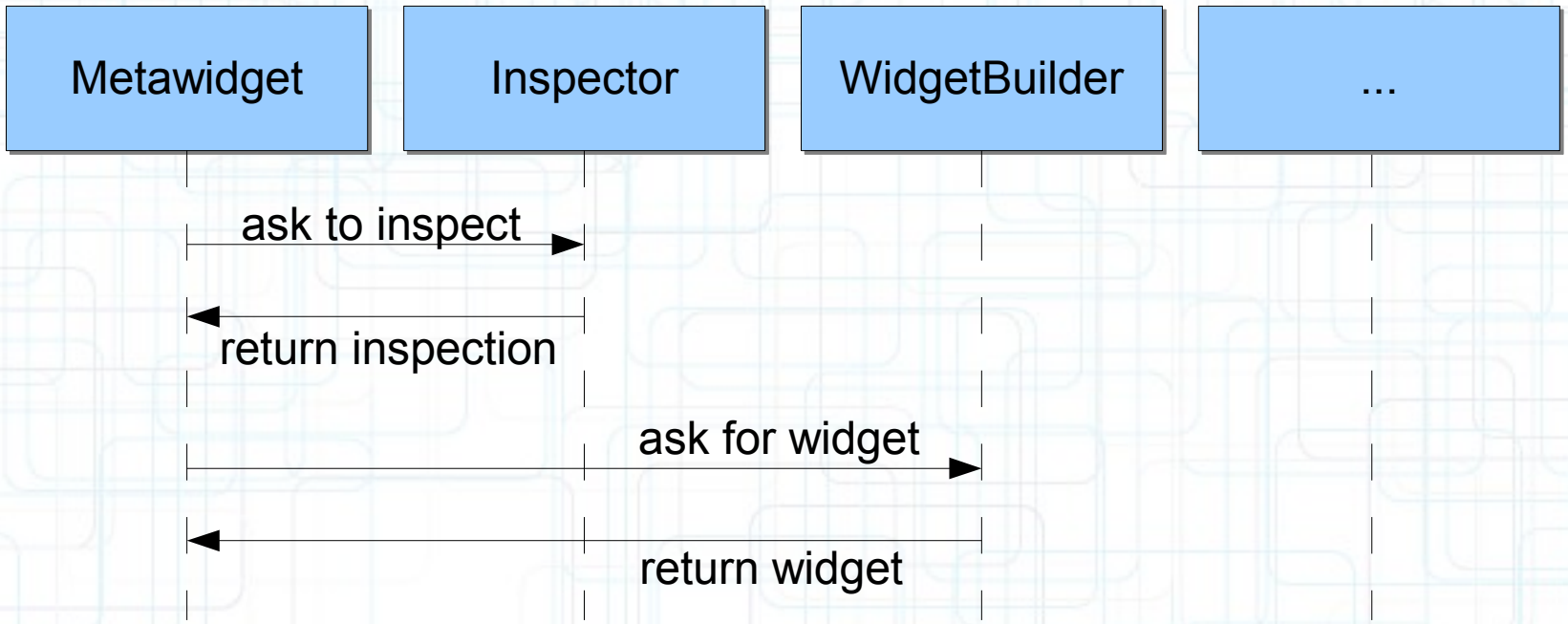
A better way

Uses your existing architecture



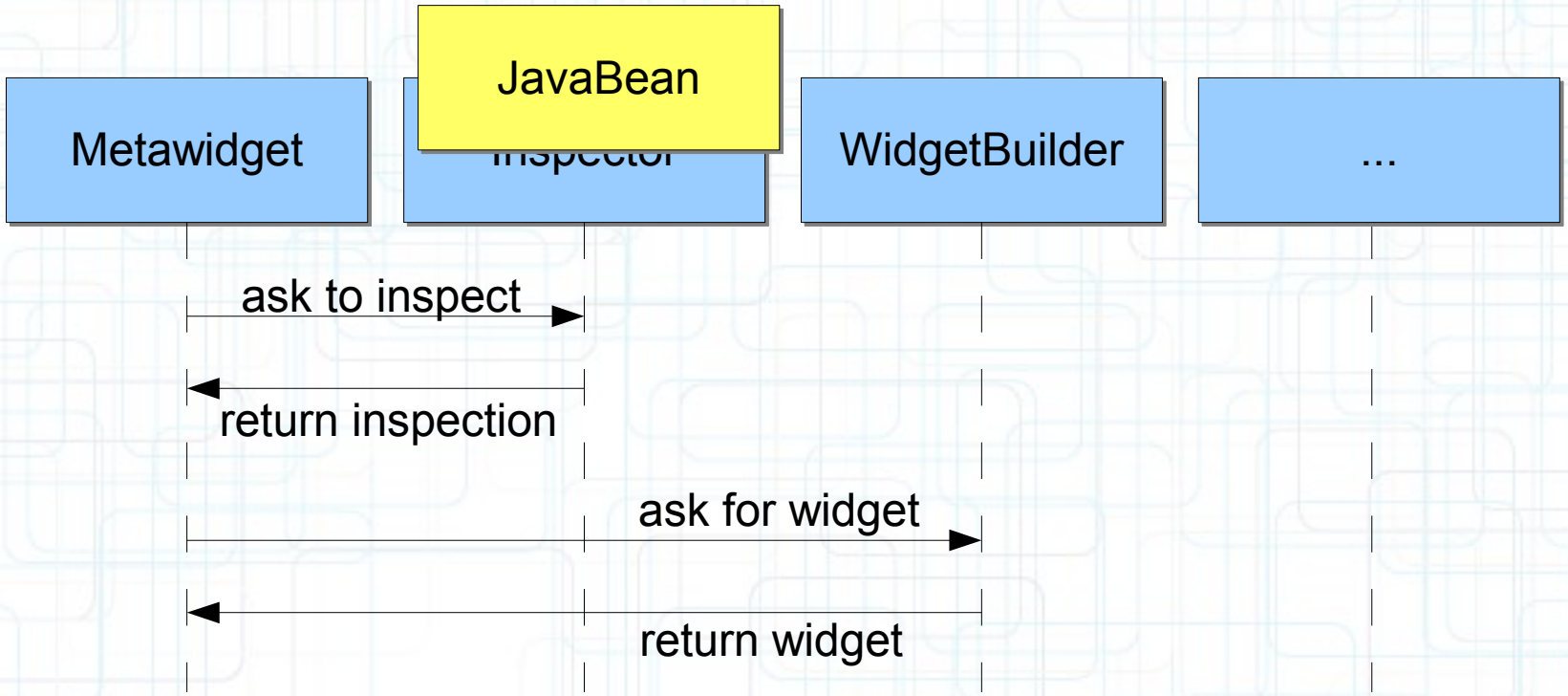
A better way

Uses your existing architecture



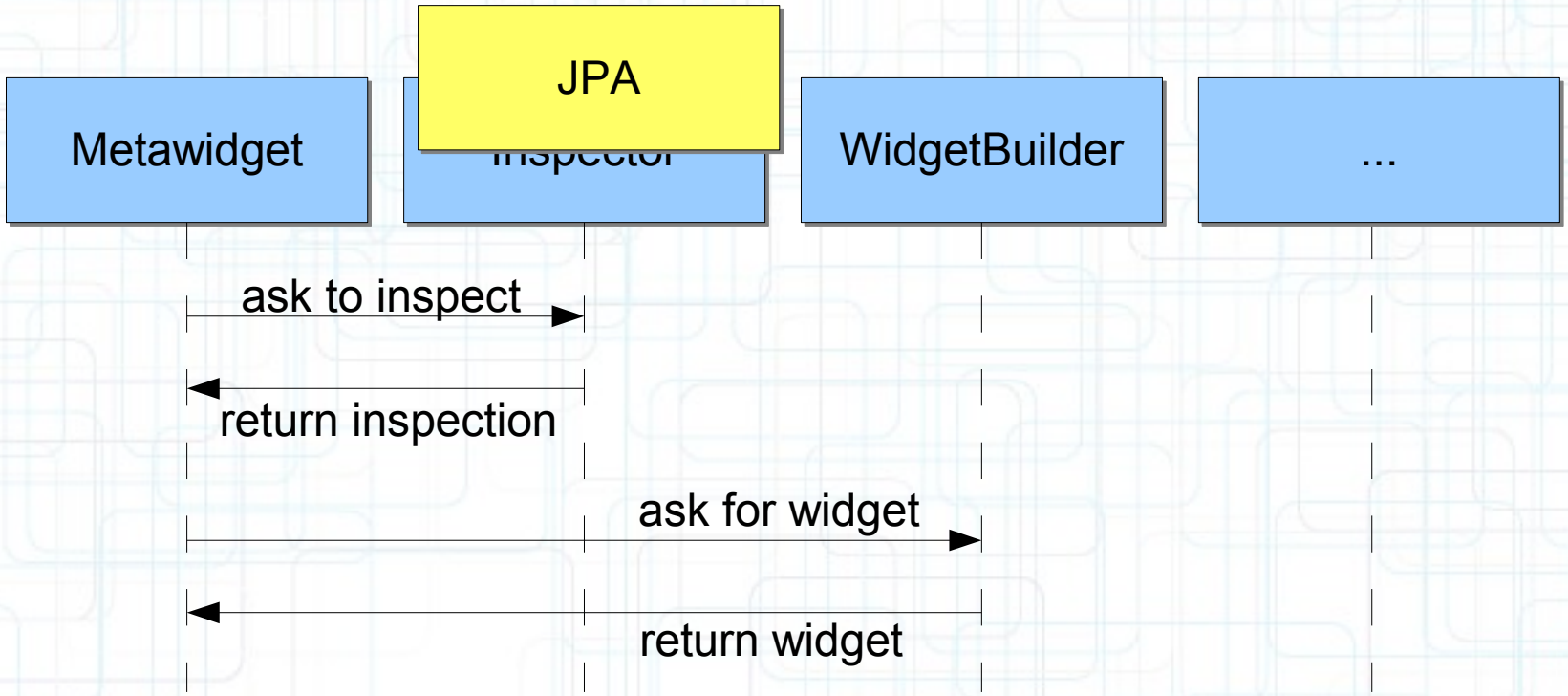
A better way

Uses your existing architecture



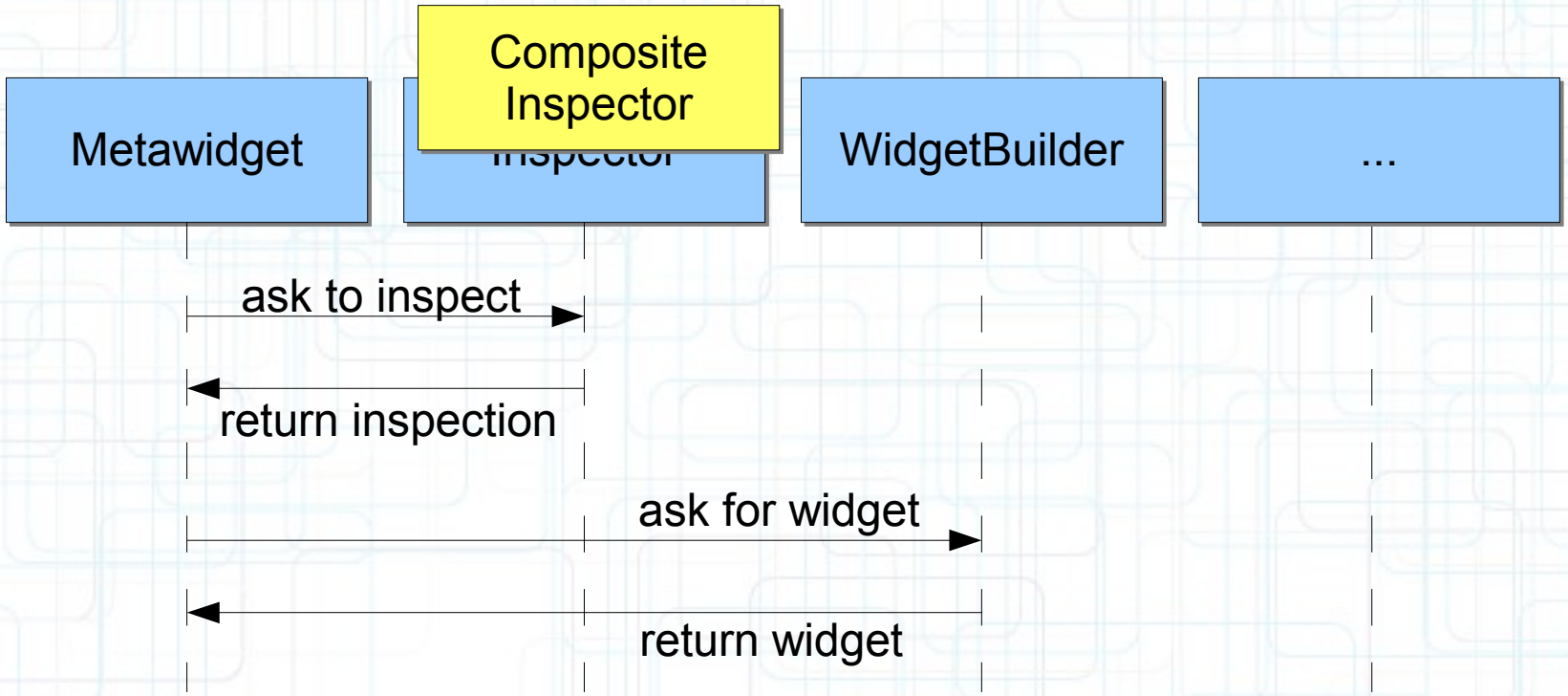
A better way

Uses your existing architecture



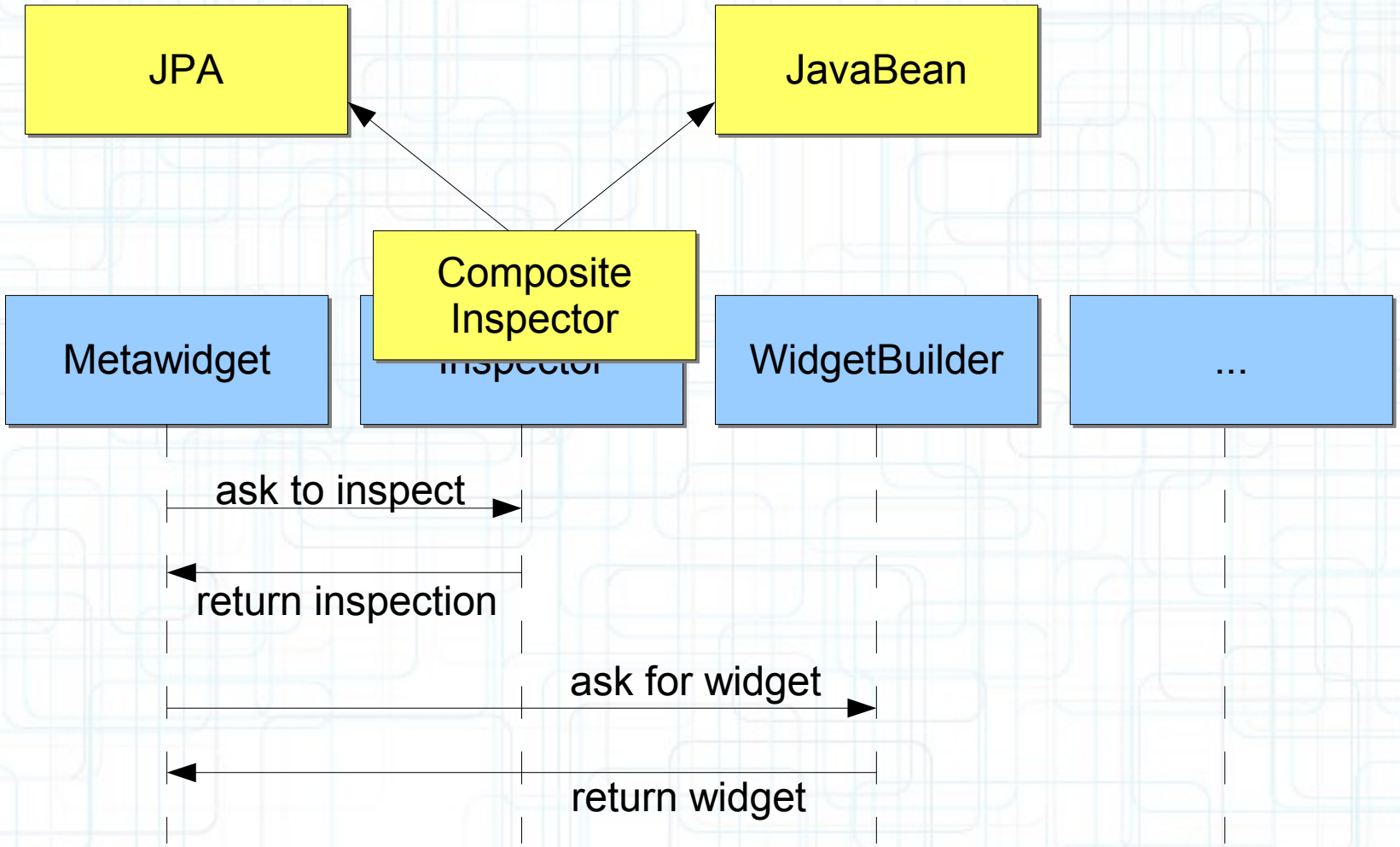
A better way

Uses your existing architecture



A better way

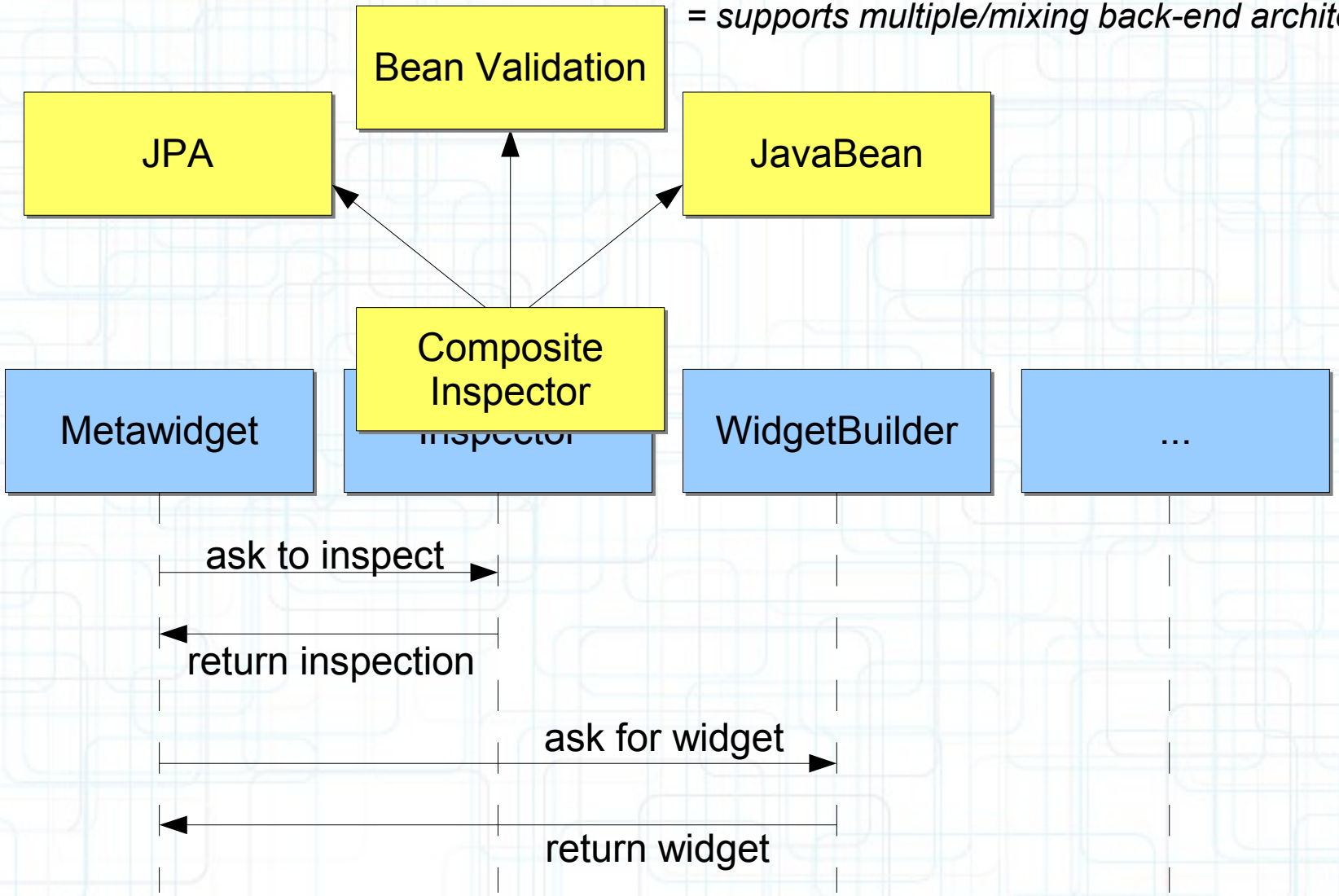
Uses your existing architecture



A better way

Uses your existing architecture

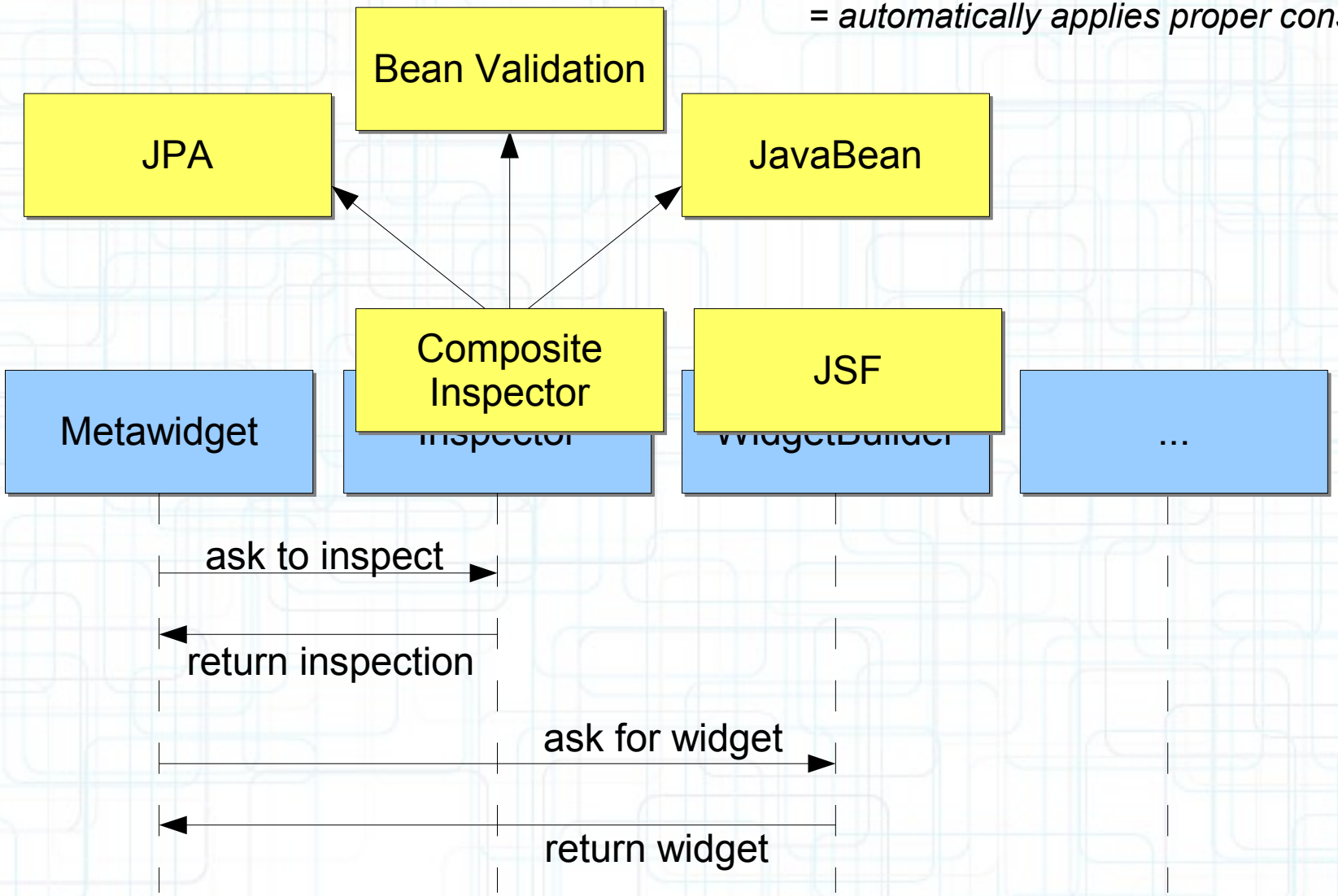
= no duplicate definitions from other layers
= supports multiple/mixing back-end architectures



A better way

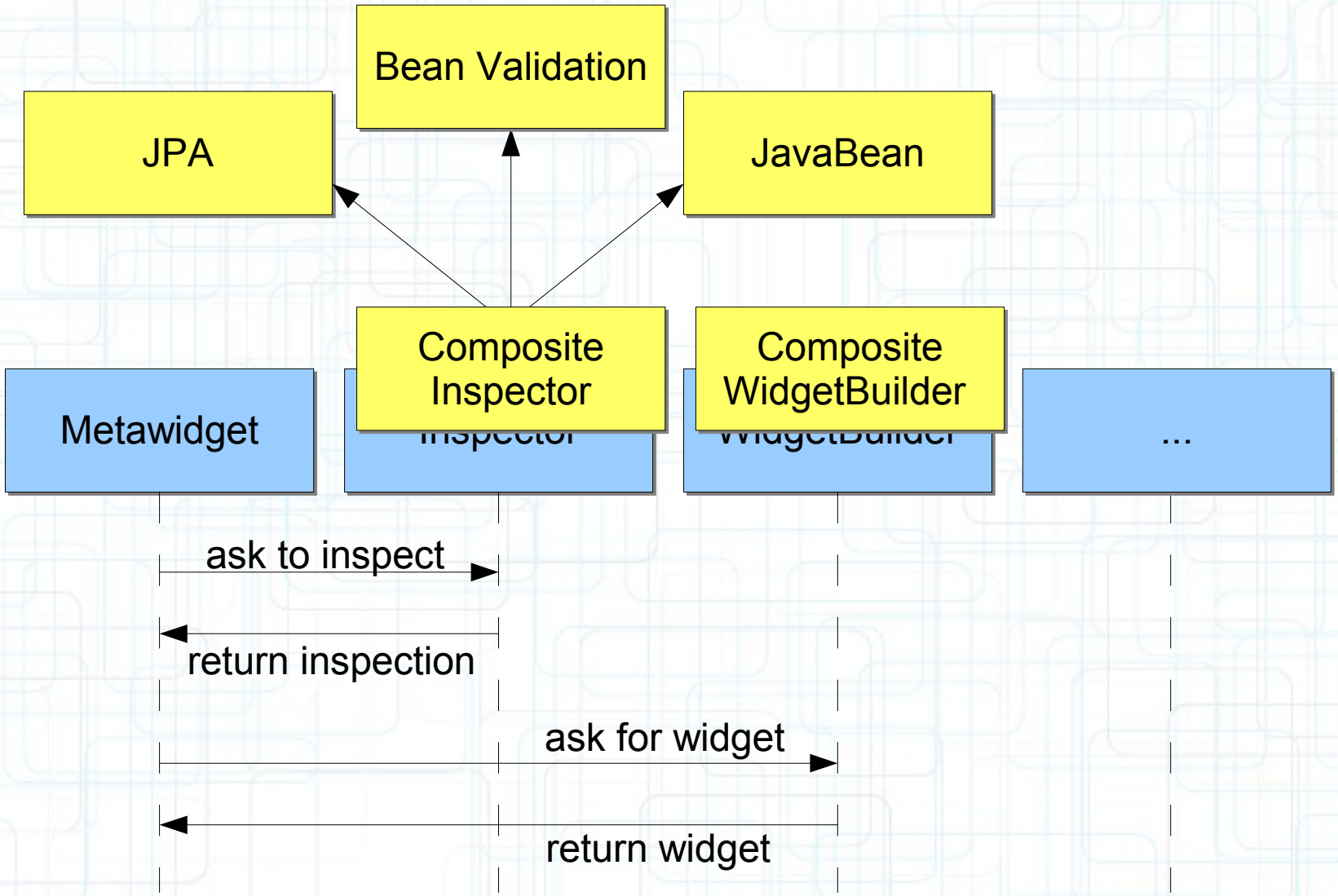
Uses your existing architecture

= automatically applies proper constraints



A better way

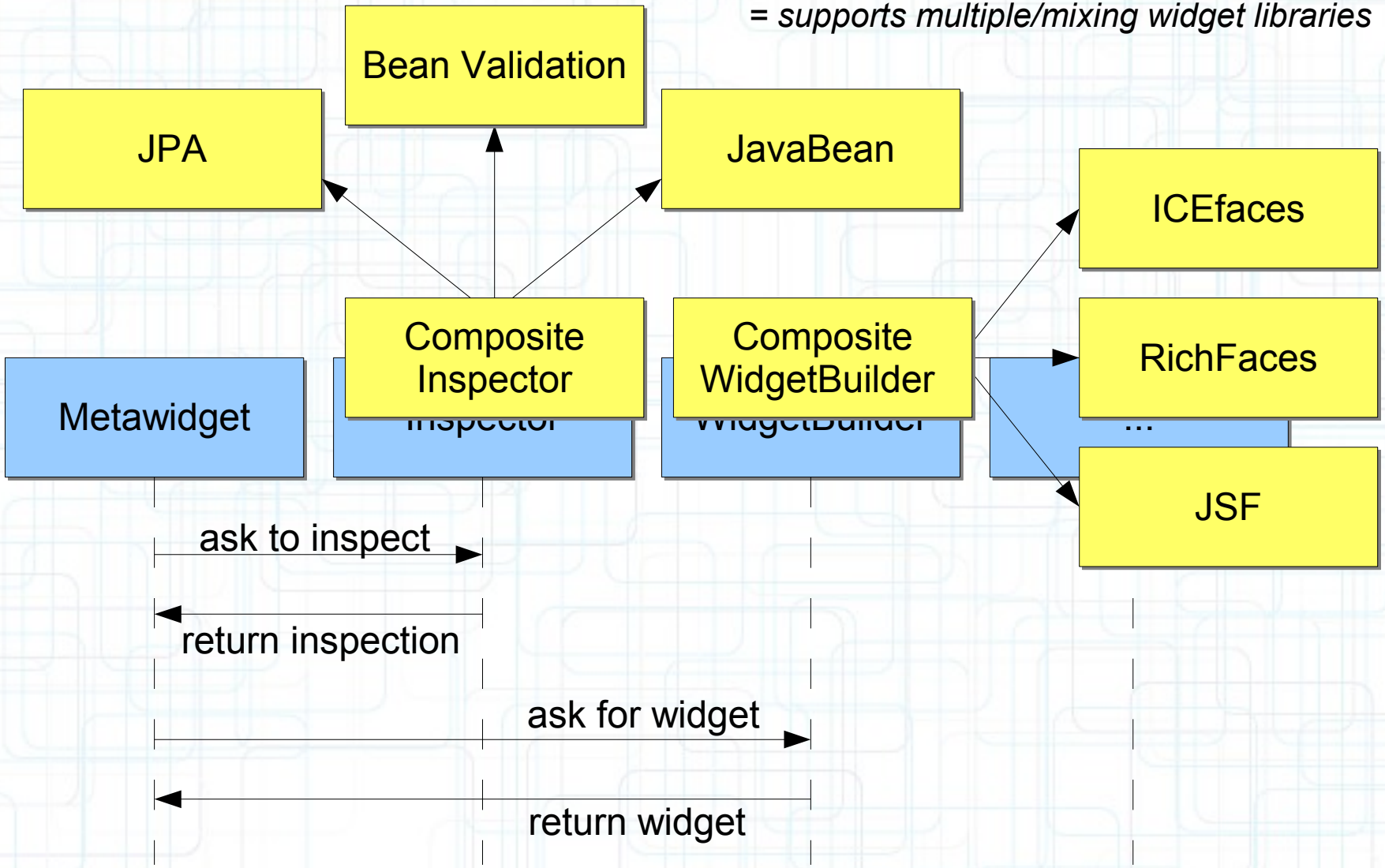
Uses your existing architecture



A better way

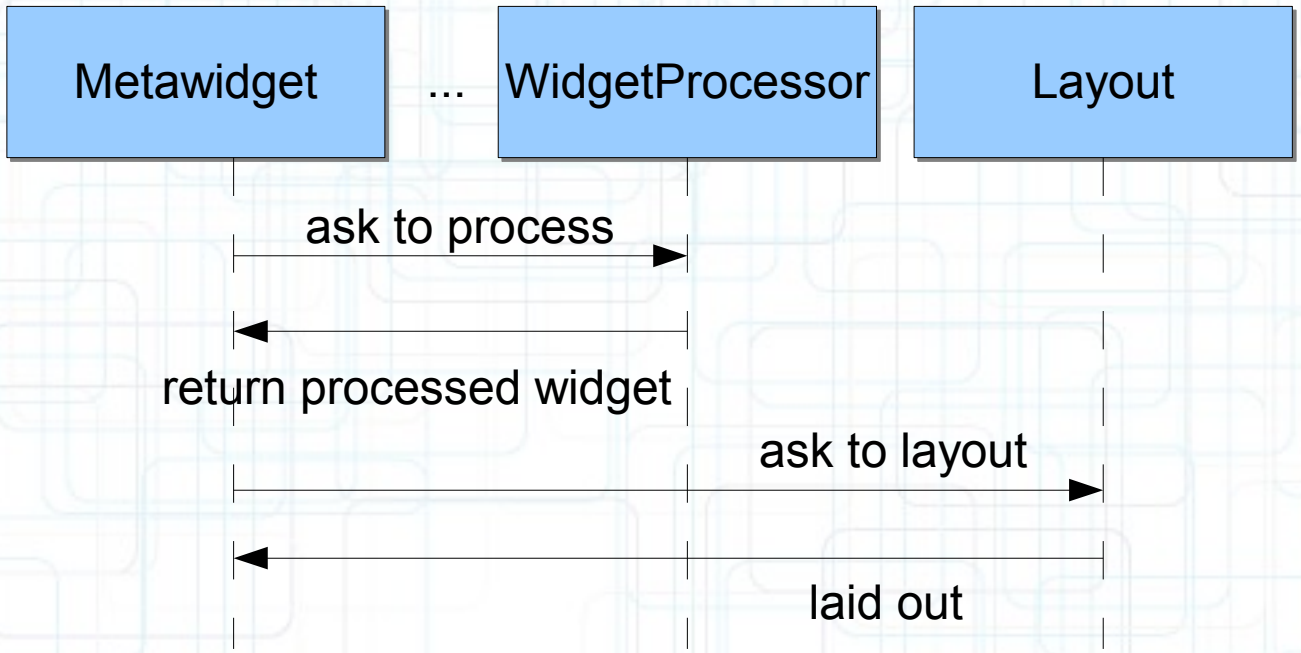
Uses your existing architecture

= new widgets can be swapped in en masse
= supports multiple/mixing widget libraries



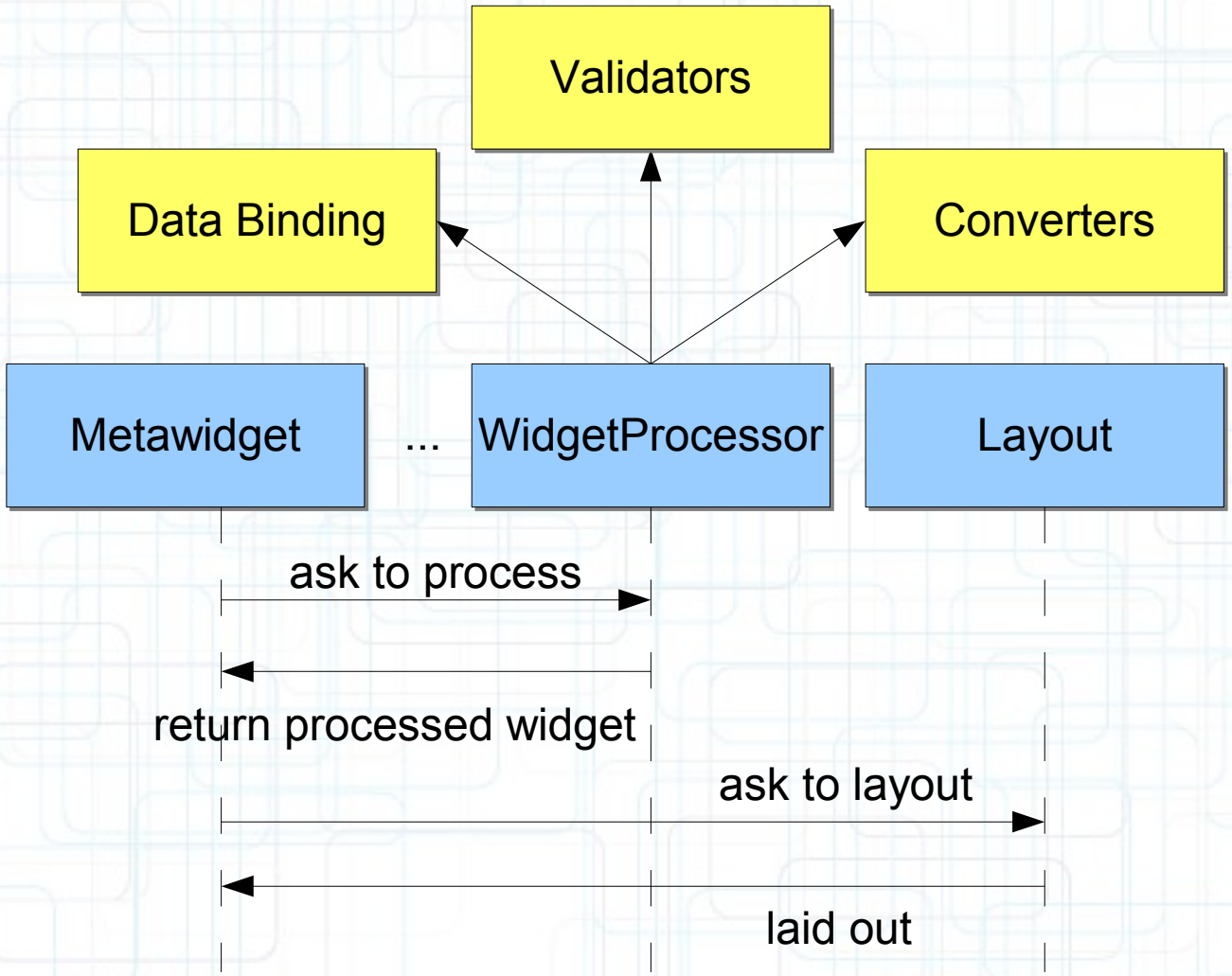
A better way

Uses your existing architecture



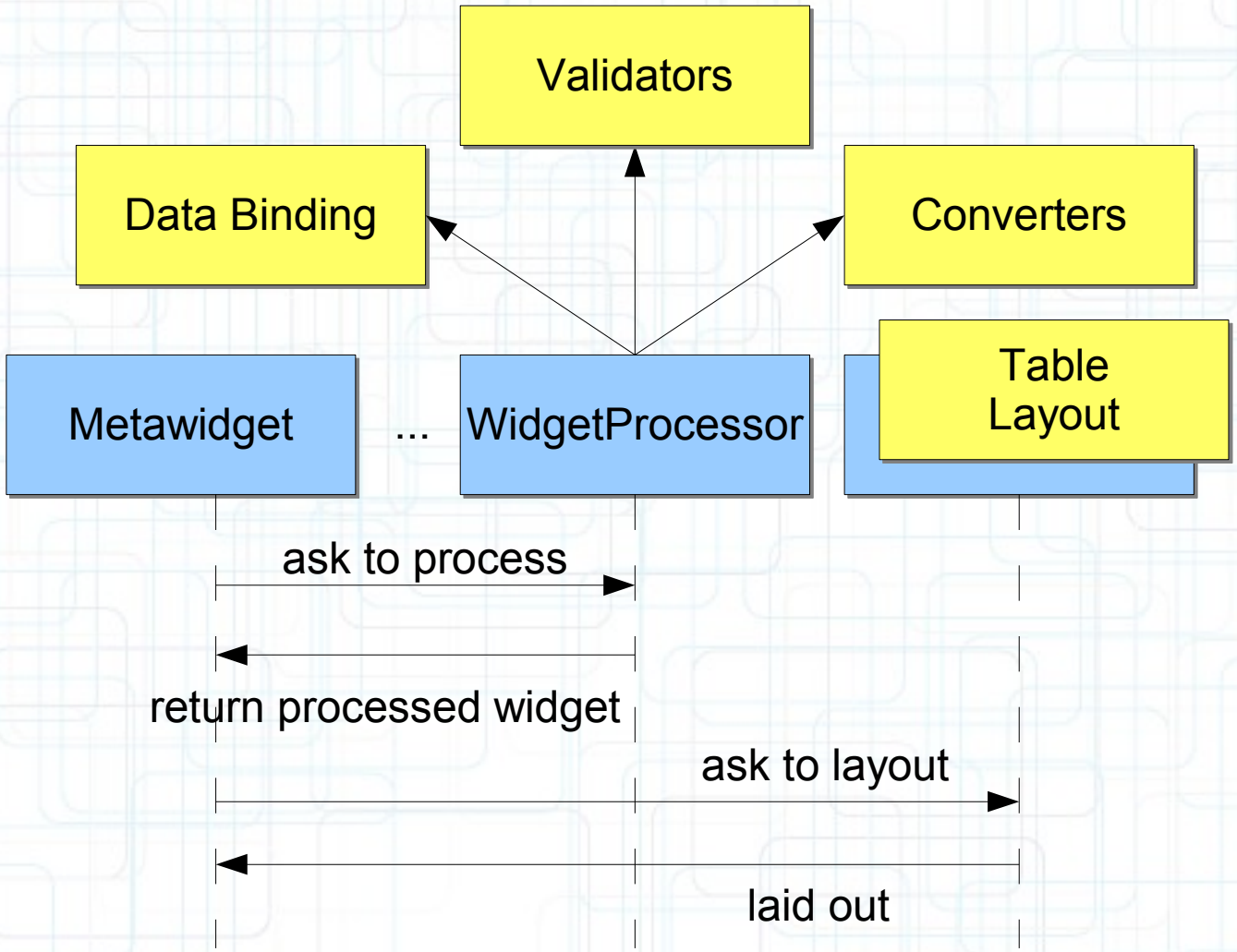
A better way

Uses your existing architecture



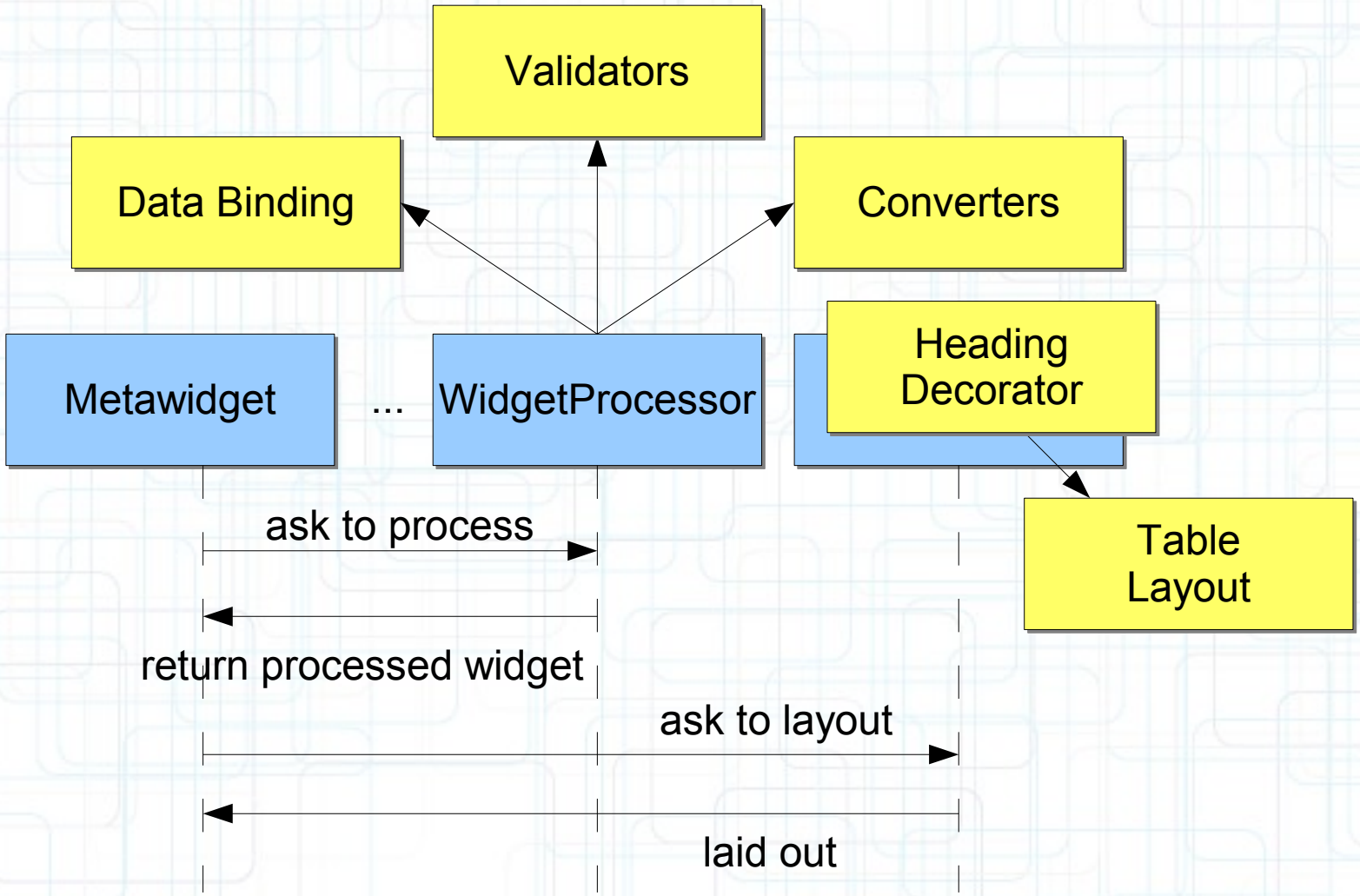
A better way

Uses your existing architecture



A better way

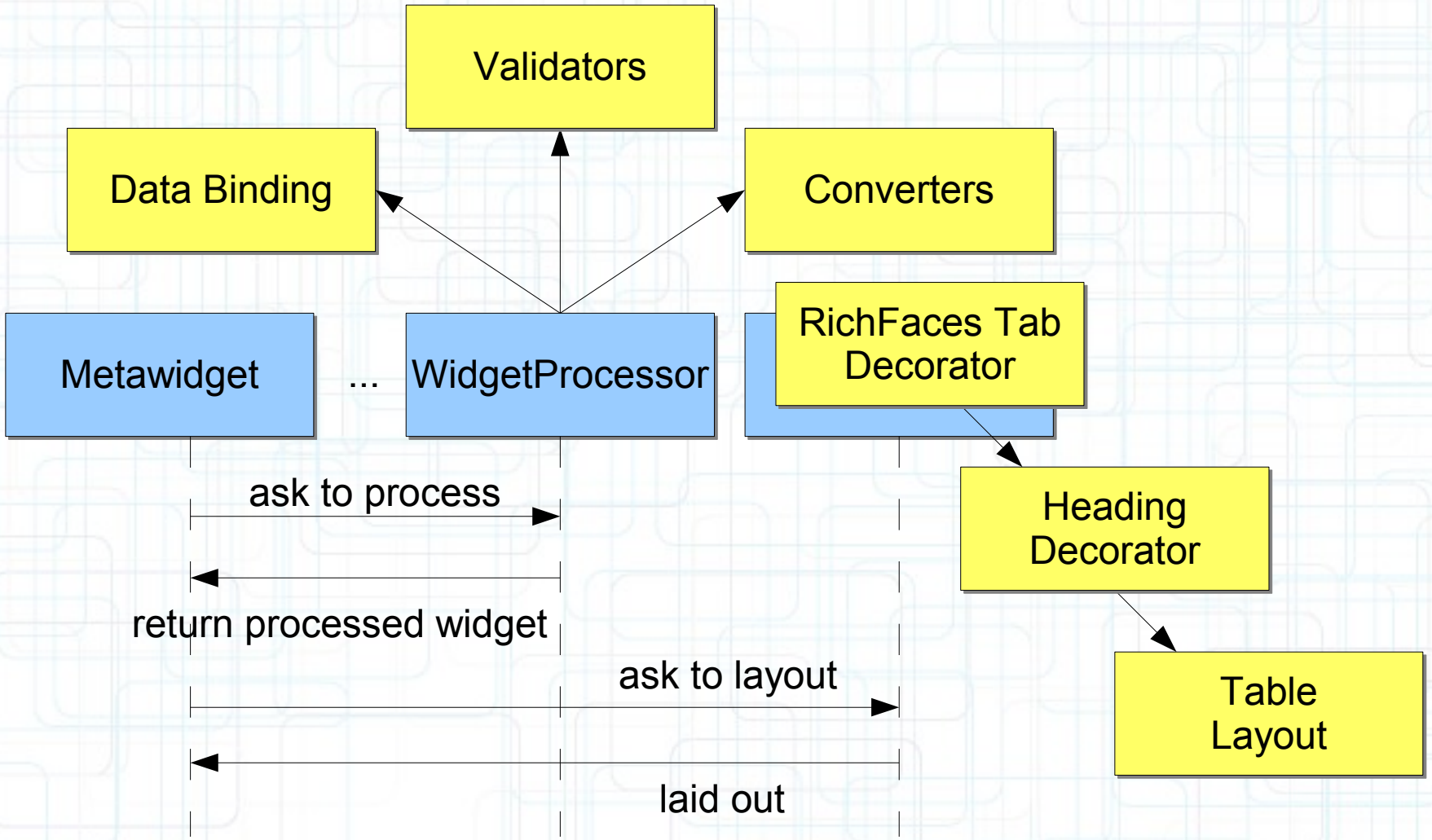
Uses your existing architecture



A better way

Uses your existing architecture

= automatic consistency across forms



Appendix B

A better way

Acid Test:

Retrofitting
an existing app