JAVA
Cutting-edge productivity with RIFE

Geert Bevin, CTO, Uwyn

Who am I?

- **Geert Bevin**
- **CTO of Uwyn**, a small custom application development company (http://uwyn.com)
- **founder of RIFE** (http://rifers.org)
- **creator and contributor of many RIFE projects:** RIFE/Crud, RIFE/Jumpstart, RIFE/Continuations, Bamboo (forum), Bla-bla List (RIA todo list), Drone (information bot), Elephant (blog)
- **Java Champion**

Agenda

- Essentials
- Metaprogramming
- Q&A

Agenda

- Essentials
- Metaprogramming
- Q&A

What is RIFE?

What is RIFE?

Full-stack component framework to quickly and consistently develop and maintain Java web applications

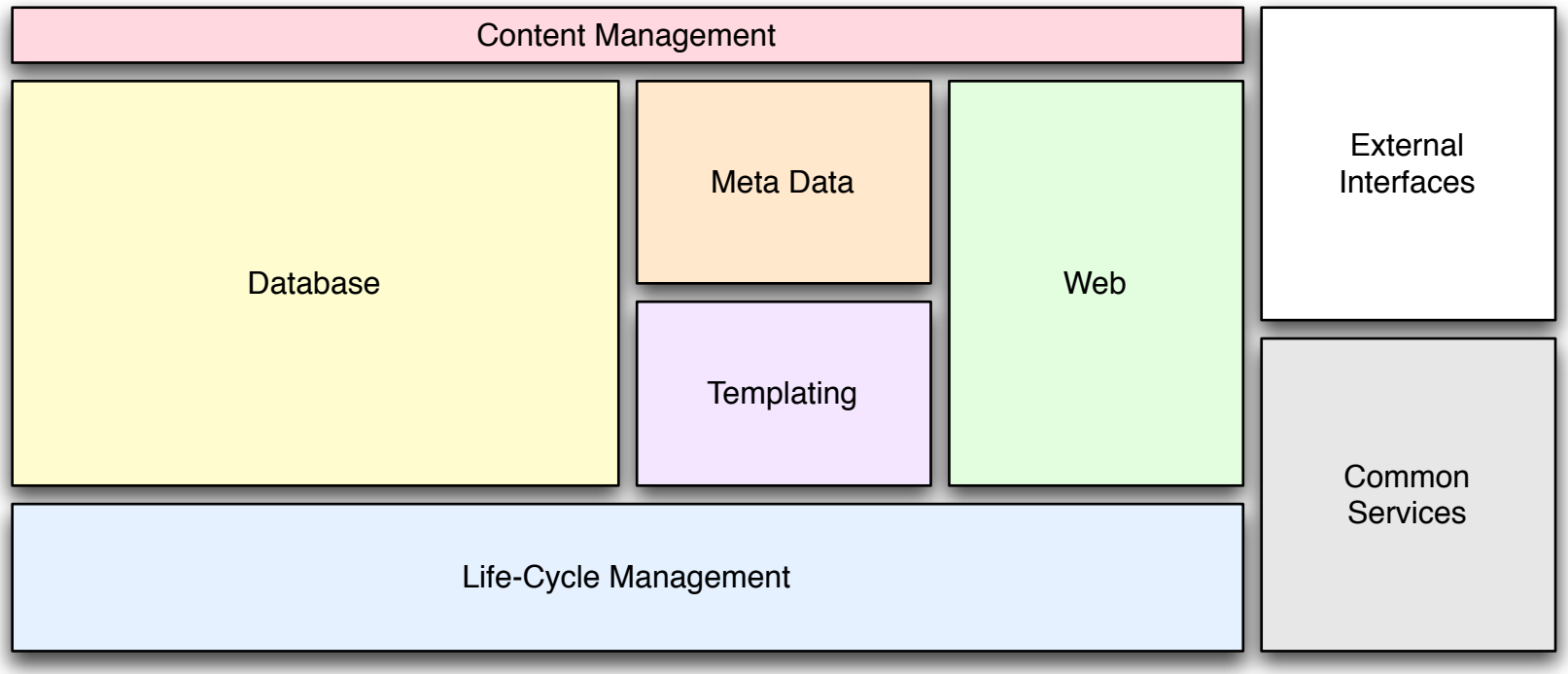
What's the state of RIFE?

What's the state of RIFE?

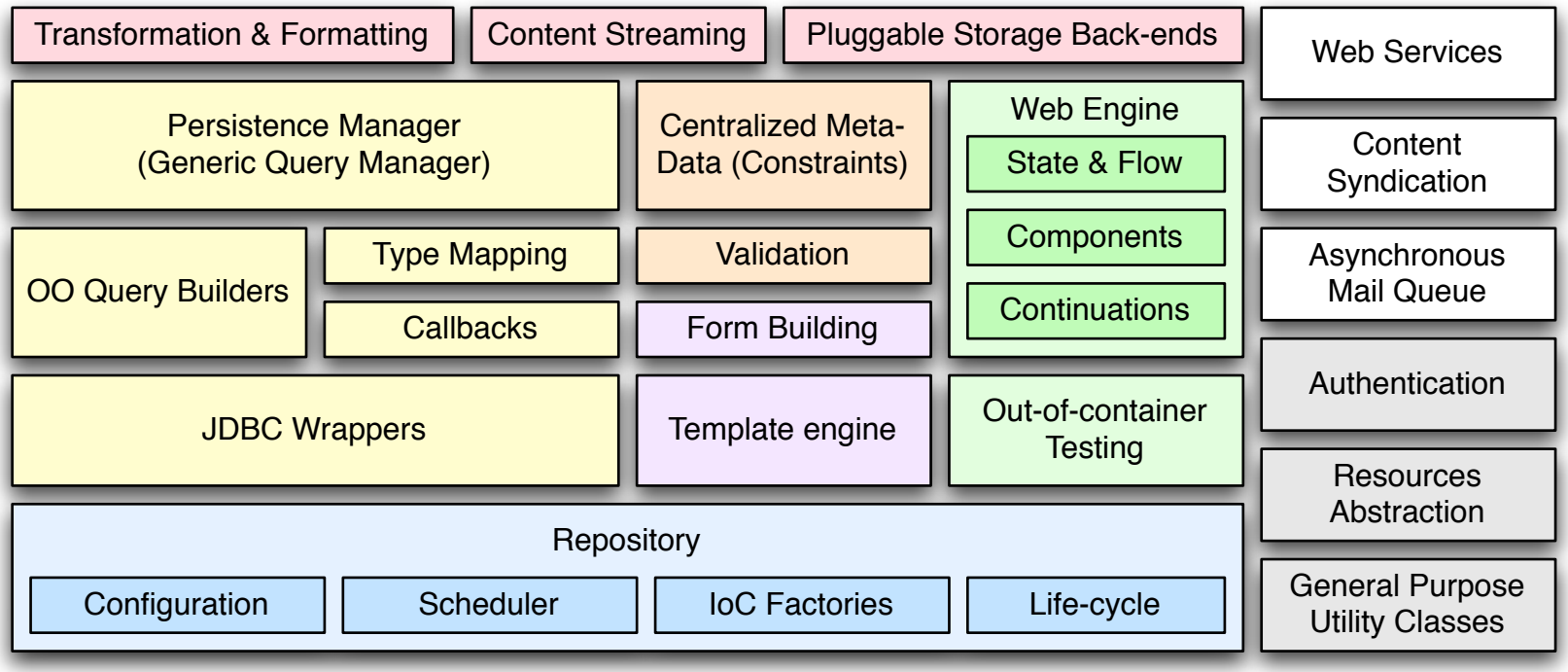
- In development **since the year 2000**
- Currently around **3,500** downloads per month
- **Production-ready**
- Used by **big companies**
  - Facebook
  - Greenpeace

What's inside RIFE?

What's inside RIFE?



What's inside RIFE?



Agenda

- Essentials
- Metaprogramming
- Q&A

Agenda

- Essentials
- Metaprogramming
- Q&A

What is metaprogramming?

What is metaprogramming?

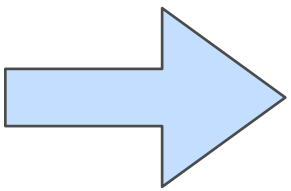
Writing programs that write or manipulate other programs.

Benefits of metaprogramming

- You work with a **domain specific API** or language that allows you to build with larger blocks
- **High-level approach** to easily achieve otherwise complex or time-consuming tasks
- Still **access to the underlying framework** for customizations or finer-grained implementations

Benefits of metaprogramming

- You work with a **domain specific API** or language that allows you to build with larger blocks
- **High-level approach** to easily achieve otherwise complex or time-consuming tasks
- Still **access to the underlying framework** for customizations or finer-grained implementations



You can concentrate on your business logic and waste less time on trivialities

Highest level abstraction

RIFE/Crud

Demo 1

RIFE/Crud Summary

- **Automatically generates** administration functionalities for tedious, repetitive CRUD operations
- **No code generation, runtime-based**, reloads on-the-fly
- Driven by your **domain model beans**
- **Constraints** gives total control over the data and content formats
- Builds on RIFE's **component model**

Introducing RIFE/Crud

- **Can be integrated** into any other RIFE site
- Fully customizable:
  - built-in support for **localization**
  - layout through **well-structured CSS**
  - markup is driven by **templates**,
    they can be replaced in different levels of granularity
  - **CRUD API** for reusing most of the automated
    run-time features during customization
  - **replacement** of individual CRUD components
  - customizable **menu generation**

Introducing RIFE's Constraints

Introducing RIFE's Constraints

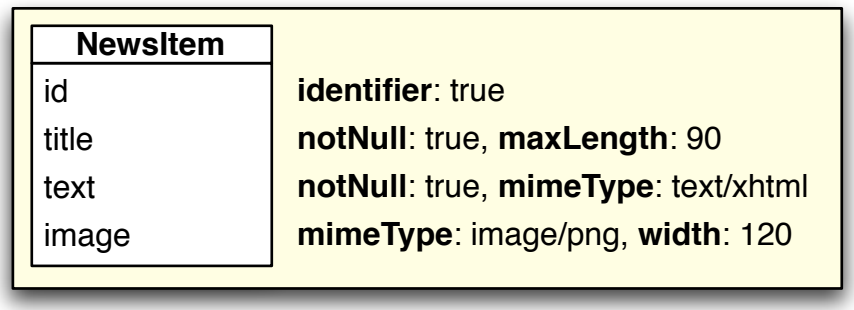
Rich dynamic meta-data API for Java bean instances and their properties

Regular JavaBean instance

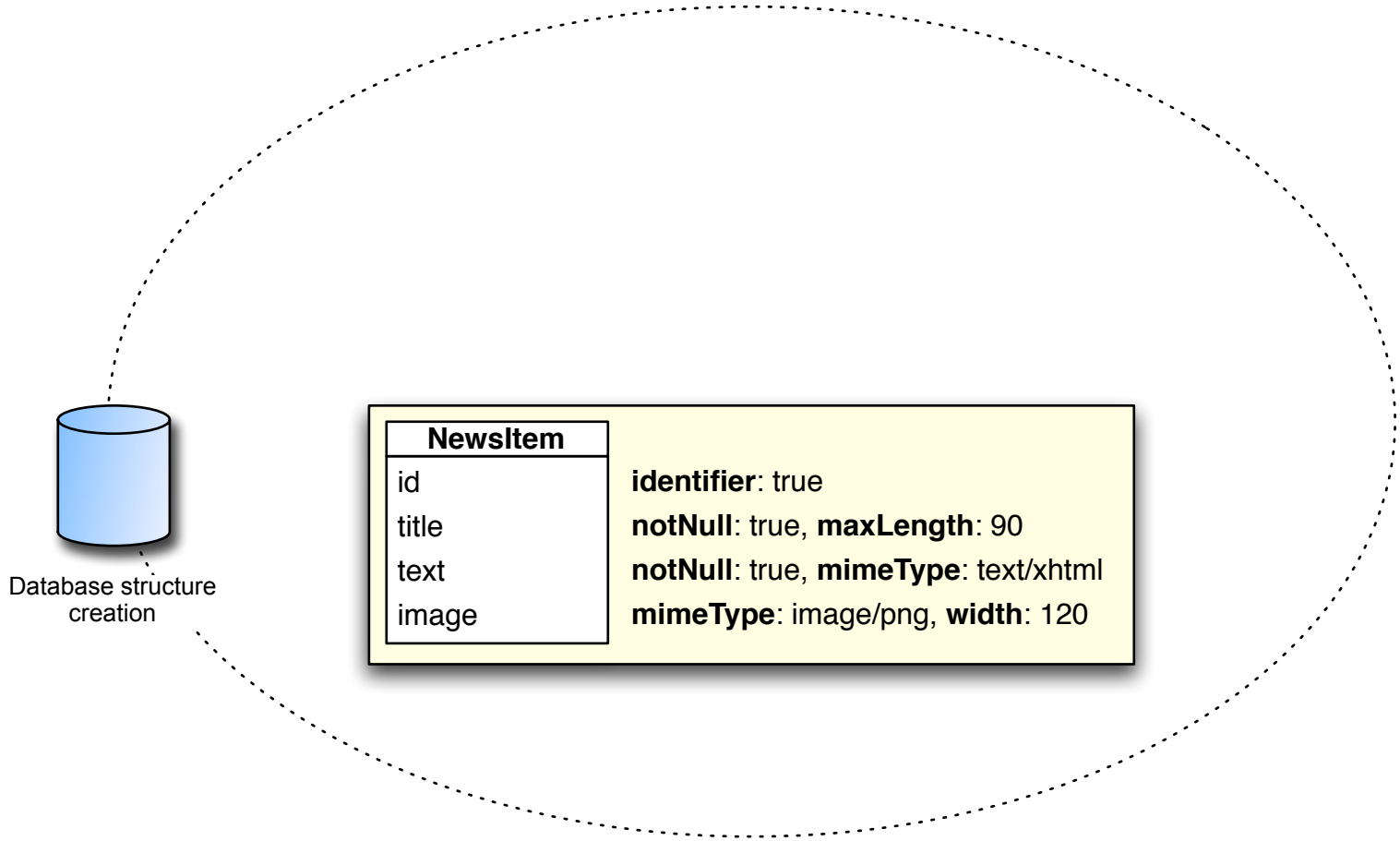
- **Regular JavaBean** instance

Introducing Constraints

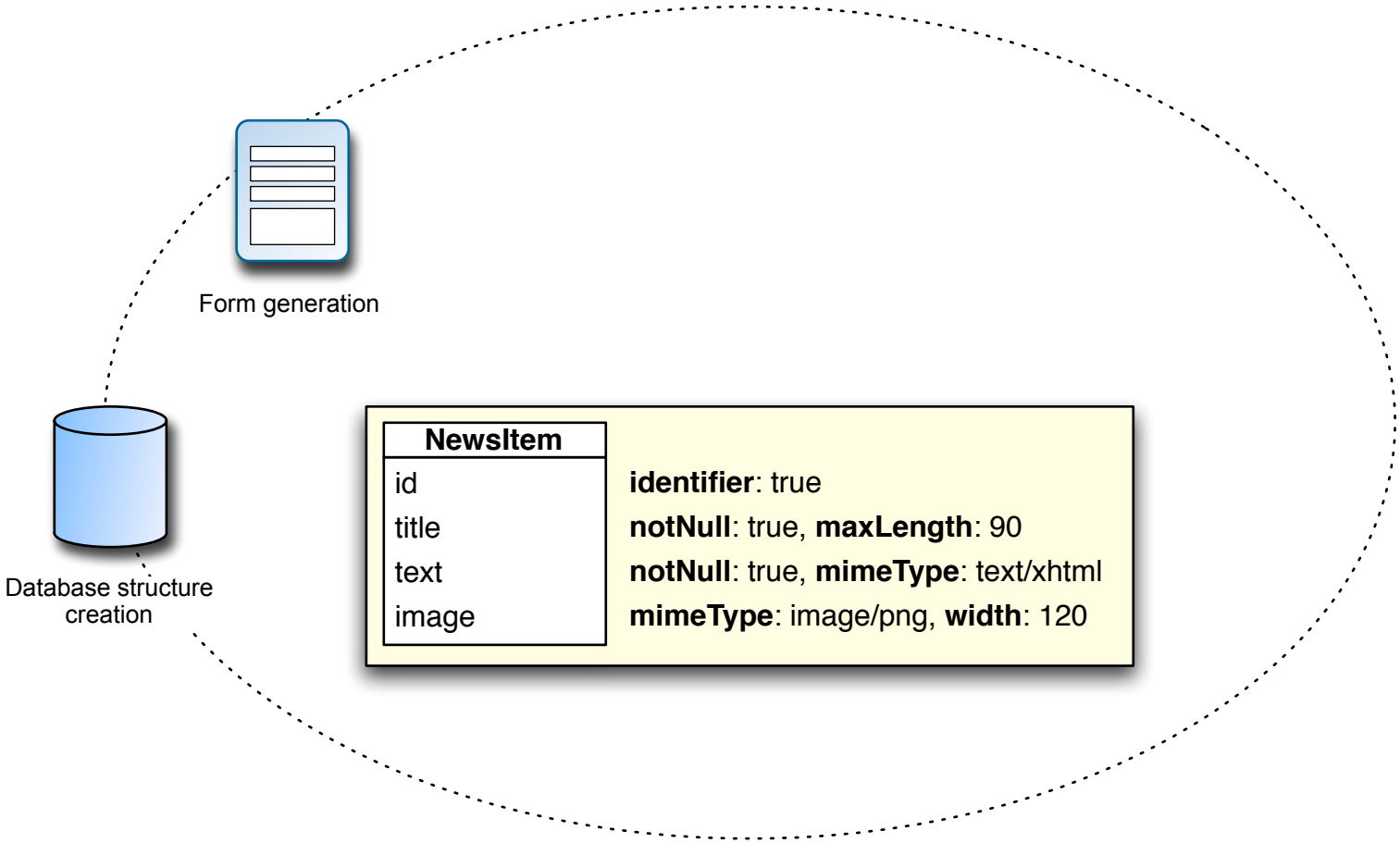
- **Regular JavaBean** instance
- **Additional** Constraints that provide **rich meta-data**



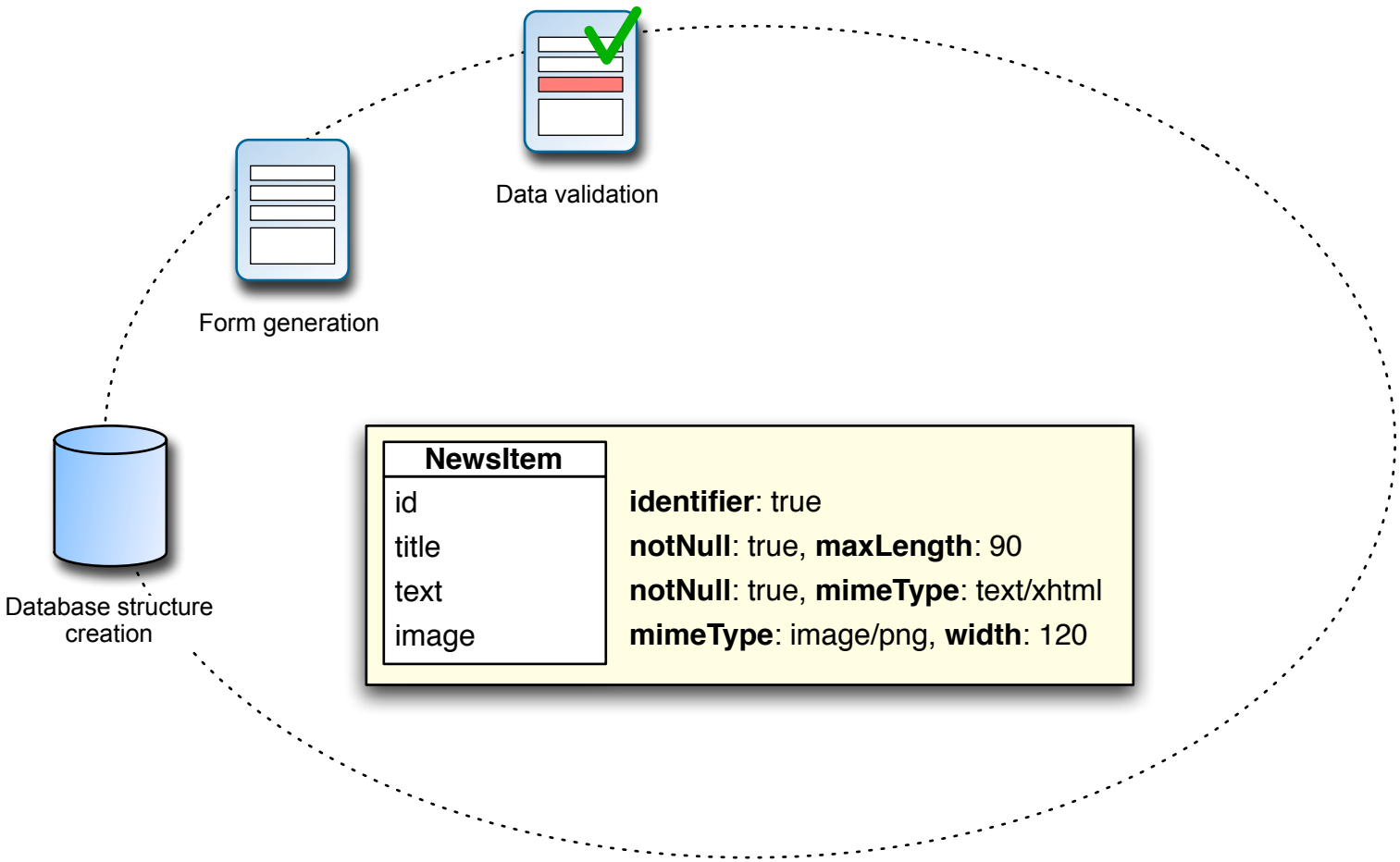
Entire framework adapts



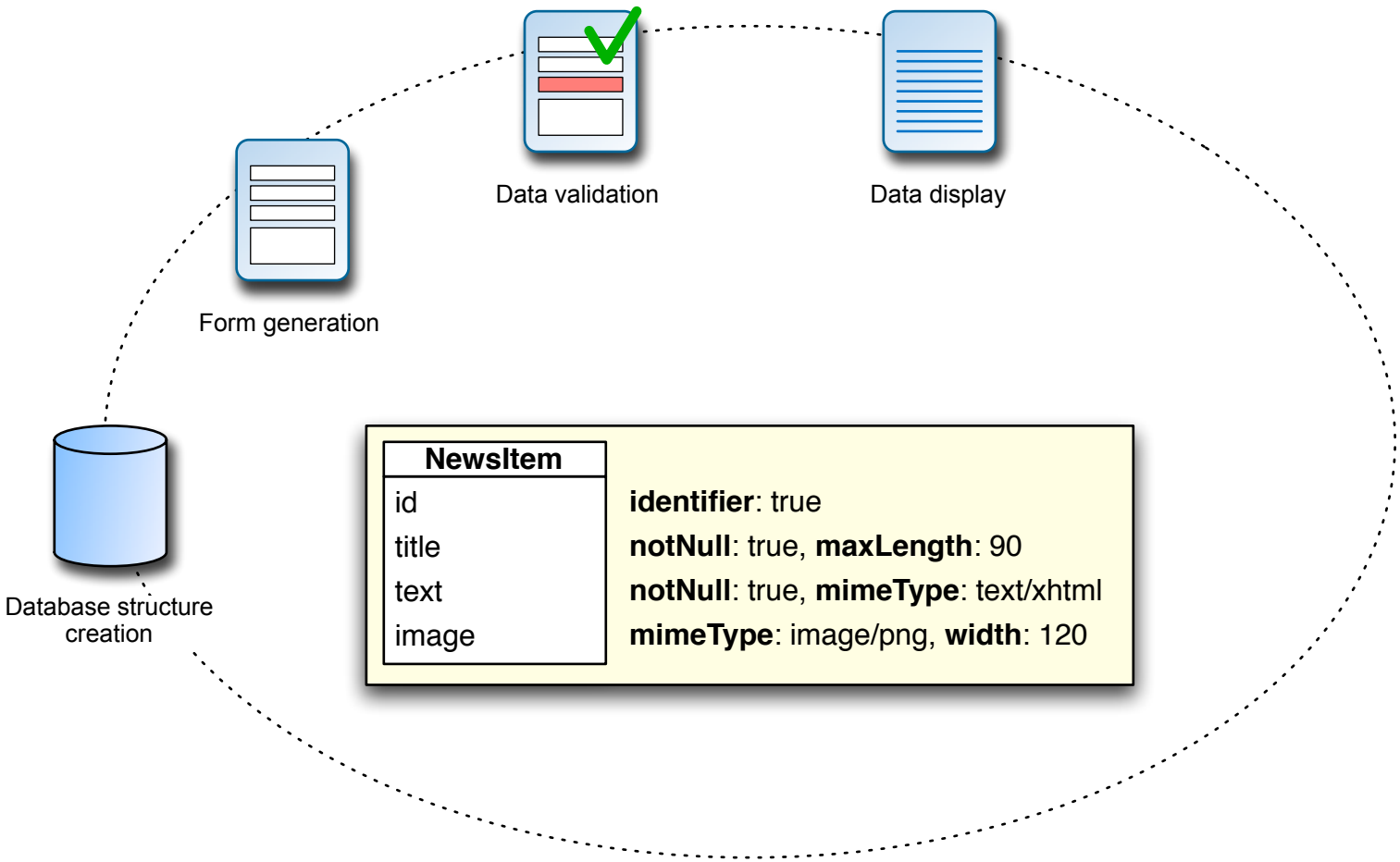
Entire framework adapts



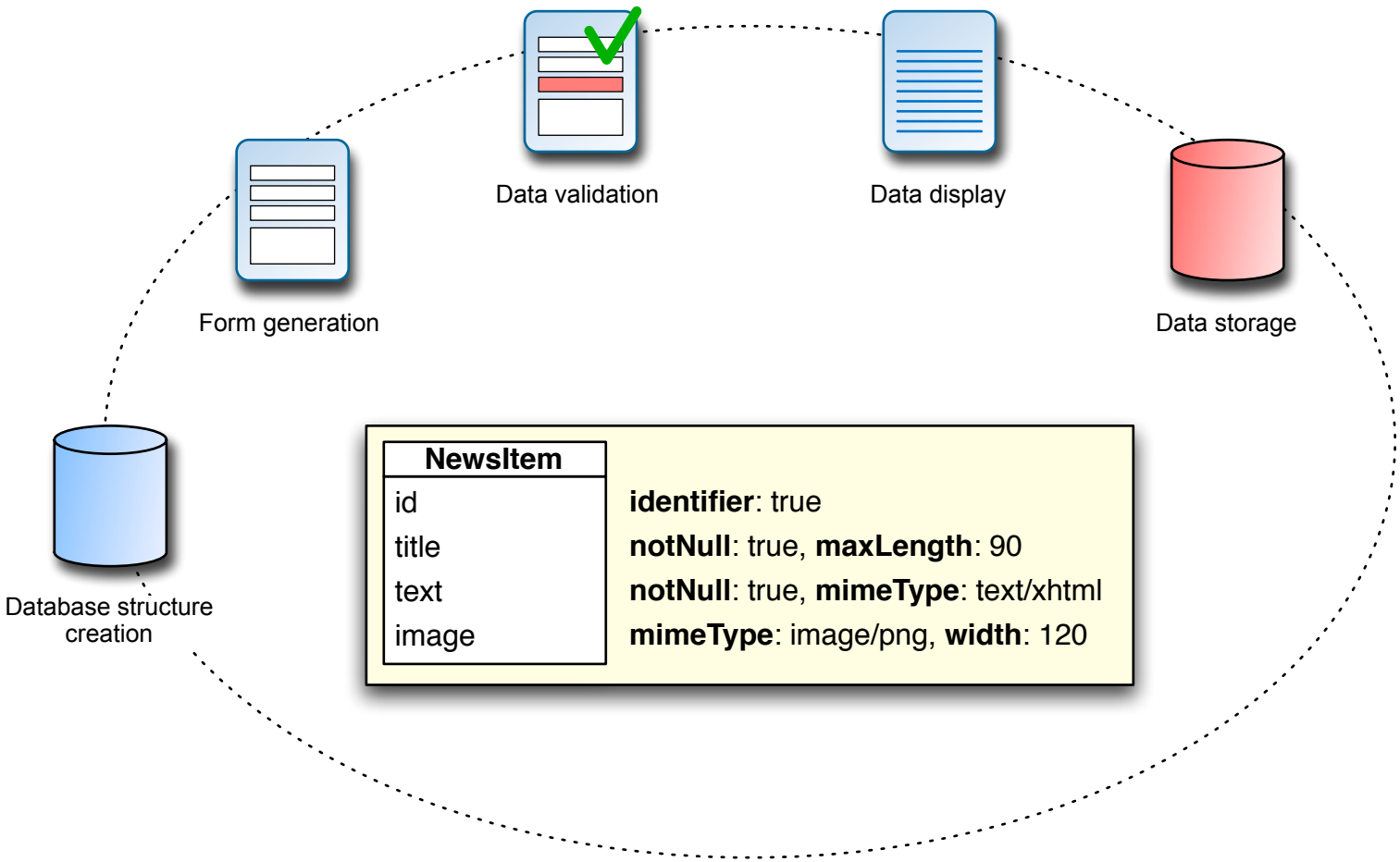
Entire framework adapts



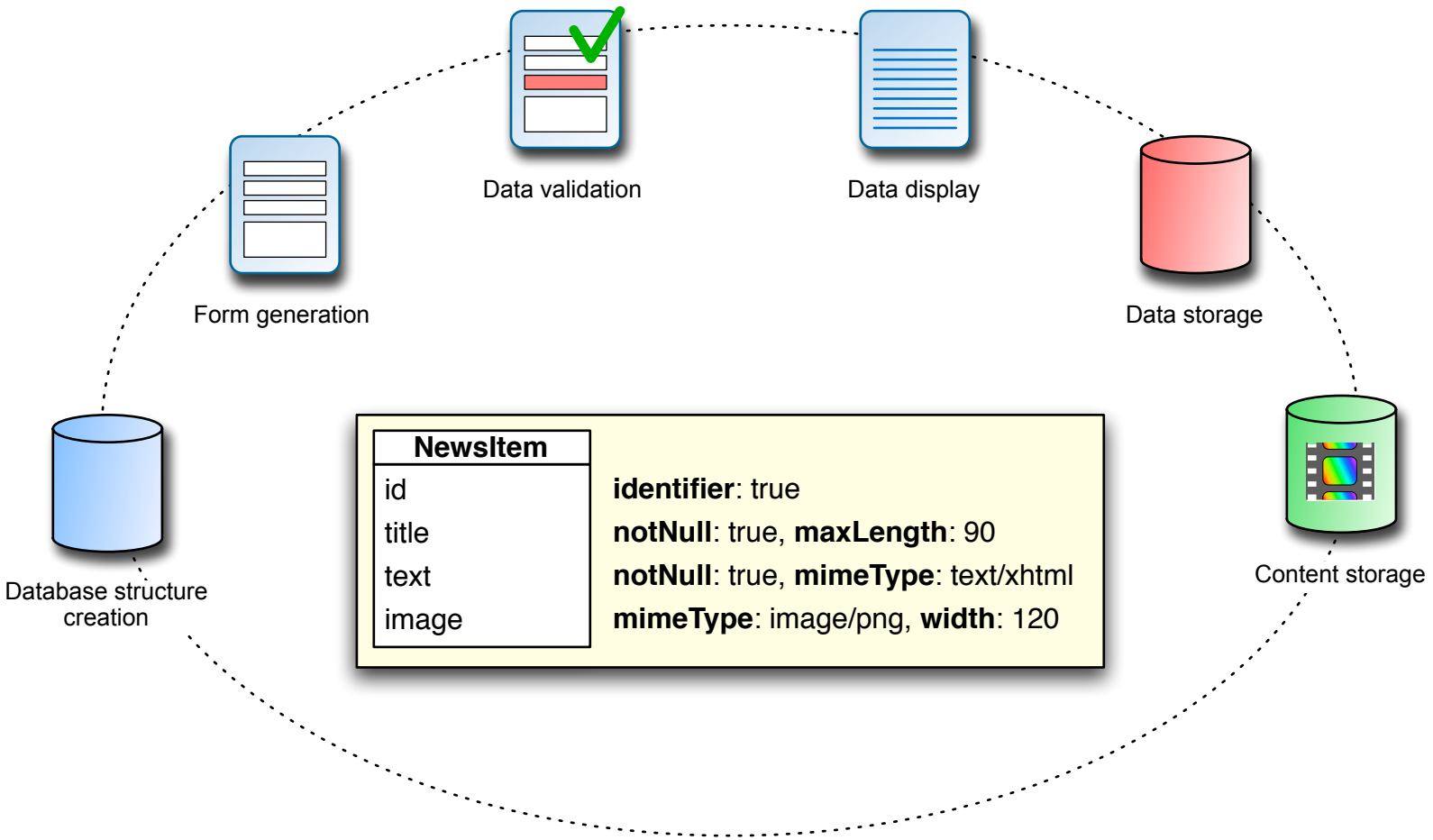
Entire framework adapts



Entire framework adapts



Entire framework adapts



Introducing Constraints

- **Unobtrusive:** applies to any existing POJO
- When quickly writing a simple application, you can **combine** the bean and the constraints **in one class**
- **Instance and runtime based:** they can dynamically change at the appropriate moment *(ie. callbacks)*

What do Constraints look like?

What do constraints look like?

- **Regular POJO** JavaBean with a number of properties

```
public class NewsItem {

    private Integer   id;
    private String    title;
    private String    text;
    private byte[]    image;

    public void    setId(Integer id)       { this.id = id; }
    public Integer getId()                 { return id; }
    public void    setTitle(String title)  { this.title = title; }
    public String  getTitle()              { return title; }
    public void    setText(String text)    { this.text = text; }
    public String  getText()               { return text; }
    public void    setImage(byte[] image)  { this.image = image; }
    public byte[]  getImage()              { return image; }
}
```

What do constraints look like?

- **Regular POJO** JavaBean with a number of properties

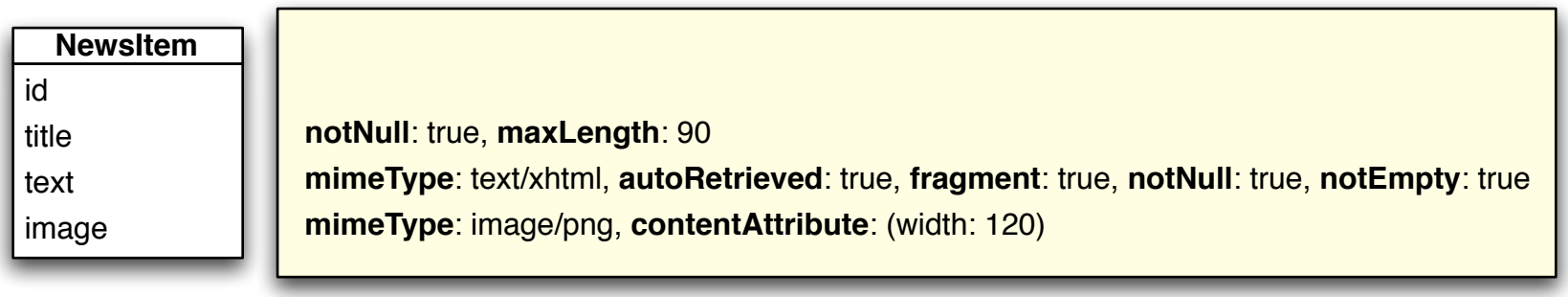
What do constraints look like?

- **Regular POJO** JavaBean with a number of properties
- Constraints as **additional rich meta-data**

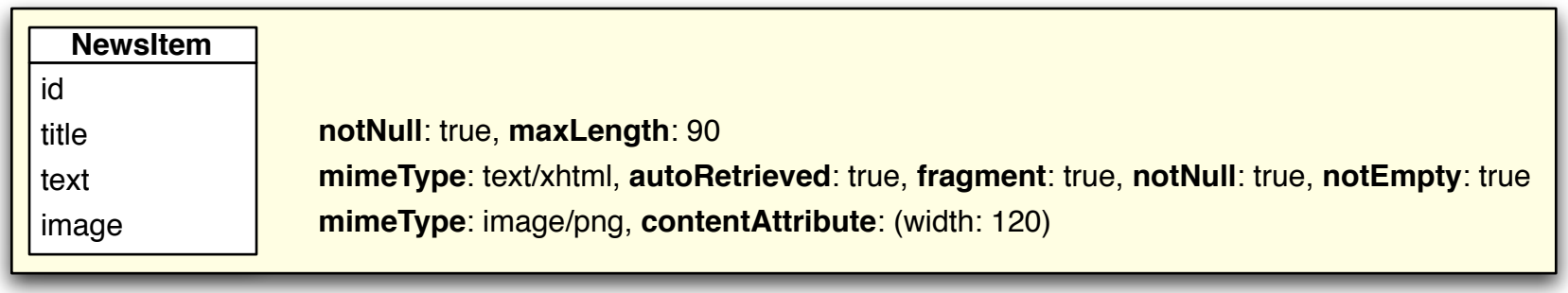

```
public class NewsItemMetaData extends MetaData {
    public void activateMetaData() {

        addConstraint(new ConstrainedProperty("title")
            .notNull(true)
            .maxLength(90));
        addConstraint(new ConstrainedProperty("text")
            .mimeType(MimeType.APPLICATION_XHTML)
            .autoRetrieved(true)
            .fragment(true)
            .notNull(true)
            .notEmpty(true));
        addConstraint(new ConstrainedProperty("image")
            .mimeType(MimeType.IMAGE_PNG)
            .contentAttribute("width", 120)

    }
}
```

What do constraints look like?

- **Regular POJO** JavaBean with a number of properties
- Constraints as **additional rich meta-data**



RIFE automatically combines both behind the scenes



Demo 2

Flow declaration

Flow declaration

```
<site>
    <globalexit name="home" destid="Home"/>

    <arrival destid="Home"/>

    <element id="Home" implementation="elements.pub.Home" url="home">
        <exit name="admin"/>
        <flowlink srcexit="admin" destid="^Admin"/>
    </element>

    <element implementation="elements.pub.Order"/>

    <subsite id="News" file="crud:model.NewsItem" urlprefix="news"/>
    <subsite id="Category" file="crud:model.Category" urlprefix="category"/>
</site>
```

Flow declaration

- **Declaration** of all **state transitions**
- Clear **overview** of your application's building blocks
- RIFE's **web engine** will use this information to give you **very advanced features** *(eg. three-dimensional flow, portlet-like embedded elements, stateful components, total relocatability, URL localization, complete reusability, ...)*
- XML, Java, Annotations, Groovy and Janino **builders**
- Declaration can be **automated**: RIFE/Crud

Templating

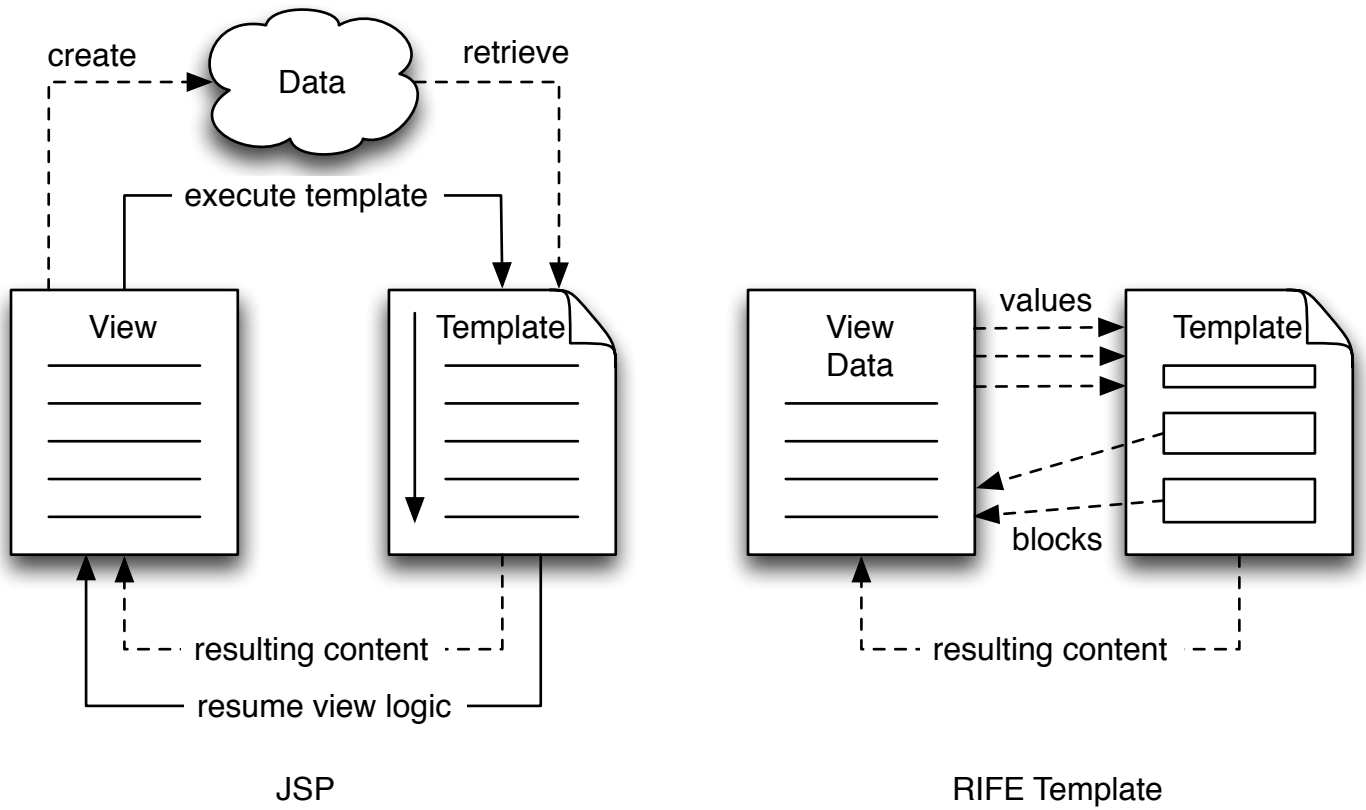
Templating

```
<form action="${v SUBMISSION:FORM:order/}" method="post">
    <!--V 'SUBMISSION:PARAMS:order'/-->

    <div class="form_field">
        <div><label for="shippingMethod">Method</label></div>
        <div>
            <div class="${v MARK:shippingMethod}${/v}">
                <!--V 'ERRORS:shippingMethod'--><!--/V-->
                <!--V 'FORM:RADIO:shippingMethod'-->id="shippingMethod"<!--/V-->
            </div>
        </div>
    </div>
</form>

<!--B 'MANDATORY:*'-->mandatory<!--/B-->
<!--B 'WRONGLENGTH:*'-->wrong length<!--/B-->
<!--B 'INVALID:*'-->invalid<!--/B-->
<!--B 'ERRORMESSAGE:*'--><!--V 'ERRORMESSAGE'/--><br/><!--/B-->
<!--B 'MARK:ERROR'-->mark_error<!--/B-->
```

Templating

- **Don't worry**, it's **very intuitive** once you get used to the concepts and the notation *(if you must, you can change it)*
- **Logic-less, unintrusive** template engine that's not bound to the format of the language *(eg. DOM)*
- You mark up existing layouts and indicate **content placeholders** and **design blocks**
- You **assemble the templates** in plain Java
- A **blueprint** to build the final layout

Comparison with JSP



Templating

- **Only four tags:** V, B, BV and I
- Template **language independent**, can be used for XHtml, Html, Xml, Text, SQL, Java, ... templating
- **Invisible** markup that keeps the language valid
- Alternative **compact** markup that's less convoluted
- **Hierarchical** dynamic templates, using includes
- **Bi-directional:** from code to template and vice-versa
- **No logic!**

Component implementation (Element)

Component implementation (Element)

```
@Elem(
    submissions = {
        @Submission(name = "order", beans = {@SubmissionBean(beanclass = OrderData.class)})
    }
)
public class Order extends Element {
    public void processElement() {
        Template t = getHtmlTemplate("pub.order");

        if (hasSubmission()) {
            OrderData order = getSubmissionBean(OrderData.class);
            if (!((Validated)order).validate()) {
                generateForm(t, order);
            } else {
                t.setBean(order);
            }
        }
        print(t);
    }
}
```

Component implementation (Element)

- Very **intuitive**
- Doesn't have to worry about state handling, form submission, application flow, ... since the **engine takes care of it all**
- Regular Java with **few artifacts**
- Can be written in **scripting languages:** Beanshell, Groovy, Janino, Jython, Pnuts, Rhino *(javascript)* and Tcl
- Each element is **totally decoupled** from any other element and the wiring happens in the site-structure

Agenda

- Essentials
- Metaprogramming
- Q&A

More information

- **RIFE website:** http://rifers.org
- **Documentation cookbook:** http://rifers.org/wiki/display/RIFE/Cook+Book
- **Live users guide:** http://rifers.org/wiki/display/RIFE/LiveGuide
- **Upcoming books:** series of 3 short books from O'Reilly
- **Many real-life open-source projects:** forum, blog, IRC logger, to-do list tracker, pastebin, order checkout process, ...

Remember!

Enter the evaluation form and be a part of making Øredev even better.

You will automatically be part of the evening lottery