

## Pattern Languages & EFPL

- Look at two topics:
  - **Pattern Languages**
    - collections of patterns that used together lead to solutions for a particular domain area
  - **Evolving Frameworks pattern language**
    - a pattern language for developing frameworks together with its use in the evolution of MViews/JViews

## Pattern Languages

- "A pattern language defines a collection of patterns and the rules to combine them into an architectural style. Pattern languages describe software frameworks or families of related systems."
  - Cope, Patterns Home Page
- "A collection of patterns forms a vocabulary for understanding and communicating ideas. Such a collection may be skillfully woven together into a cohesive "whole" that reveals the inherent structures and relationships of its constituent parts toward fulfilling a shared objective. This is what Alexander calls a pattern language. If a pattern is a recurring solution to a problem in a context given by some forces, then a pattern language is a collective of such solutions which, at every level of scale, work together to resolve a complex problem into an orderly solution according to a pre-defined goal."
  - Appleton, "Patterns and Software: Essential Concepts and Terminology"

## Pattern Languages

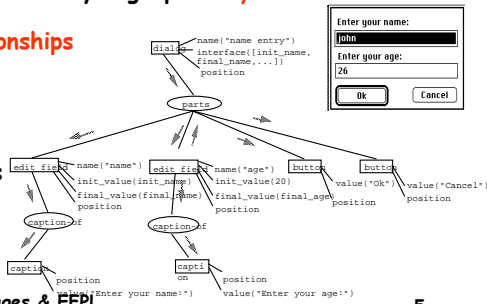
- Provide lexicon of patterns + "grammar" for threading them together
  - useful patterns
  - rules and orderings to apply them to achieve some goal
- "Good pattern languages guide the designer toward useful architectures and away from architectures whose literary analogies are gibberish or unartful writing."
  - Appleton, "Patterns and Software: Essential Concepts and Terminology"
- Illustrate with a pattern language for evolving frameworks developed by Don Roberts and Ralph Johnson
  - D. Roberts, R.Johnson "Evolving Frameworks"  
<http://st-www.cs.uiuc.edu/users/droberts/evolve.html>
- In turn will illustrate application of this pattern language by our experience in developing our MViews/JViews framework for constructing multiple view graphical environments

## MViews/JViews

- Developed over close to 10 years
  - initially from John Grundy's PhD thesis
- **Aim:** to support design and implementation of visual environments supporting multiple views with different representations
  - Eg a CASE TOOL supporting various types of UML diagram
- **Support for specification and implementation of:**
  - underlying shared repository
  - information represented in views
  - consistency management/mappings between views
  - visual representation and manipulation of elements in the views

# CPRGs

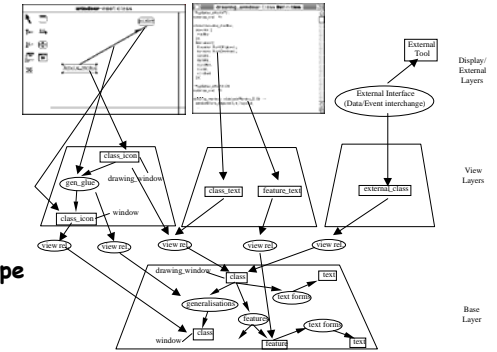
- Underlying abstraction of MViews/JViews: **change propagation and response graphs**
  - discrete change description propagation along inter-object relationships,
  - response to and storage of these change descriptions
- Each item of data is represented by a graph **component**
- Components linked via **relationships**
- Components have **attributes** representing state
- Relationships are themselves components



COMPSCI 732 §10. Pattern Languages & EFPL

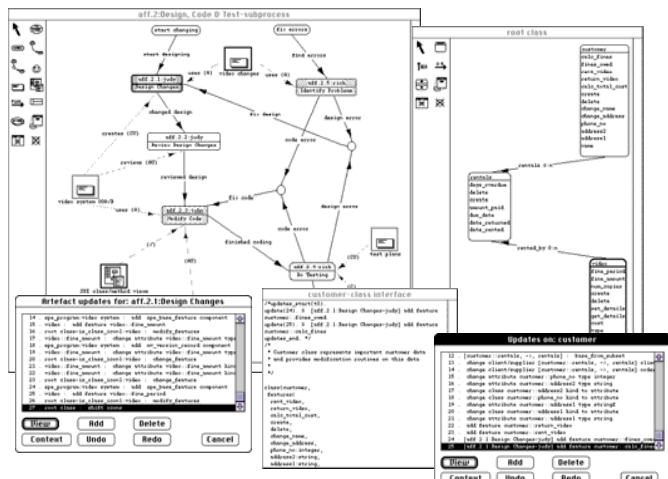
# MViews/JViews

- Framework implementing CPRG model with support for constructing multiple view - multiple representation design environments (~10 year development)
- 3-layer architecture
  - Base
  - View
  - Display
- Used to implement many of our visual tools & environments
- Eg Orion Mapper prototype



COMPSCI 732 §10. Pattern Languages & EFPL

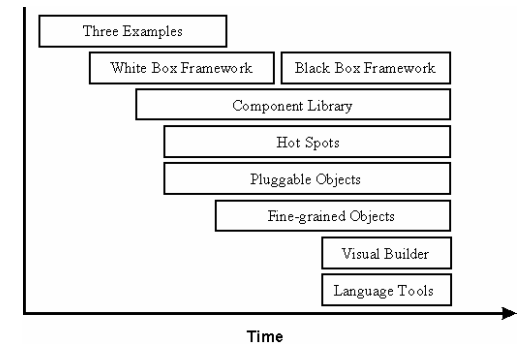
# Example use: SPE/Serendipity



COMPSCI 732 §10. Pattern Languages & EFPL

# Evolving frameworks

- The patterns in this pattern language are not design patterns in the usual sense, rather they are patterns describing useful processes and tasks that software developers perform when developing frameworks
- Names and temporal interaction of the patterns are shown in the following figure



COMPSCI 732 §10. Pattern Languages & EFPL

## 3 Examples

- **Context:** You've decided to develop a framework for a domain
- **Problem:** How do you start designing a framework
- **Forces:**
  - people work best by abstracting from examples
  - developing examples can pay for the costs of developing framework
- **Solution:** Develop three applications that you believe the framework should help you build

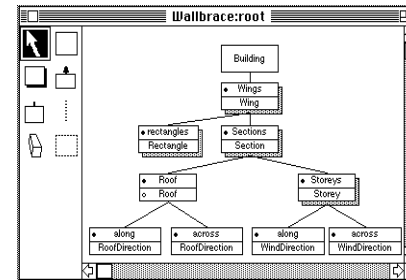


COMPSCI 732 §10. Pattern Languages & EFPL

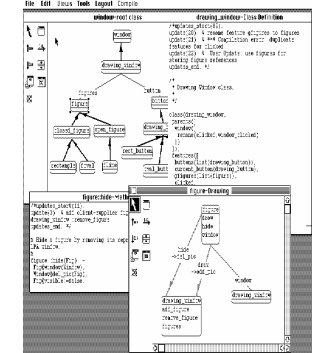
9

## MViews/JViews application

- Initially developed a tool for constructing multiple view class diagrams (Ispel)
- Then developed a programming environment for programming in Smart, an OO Declarative Language (SPE)
- Then developed a multiple view ER modeller (MViews-ER)



COMPSCI 732 §10. Pattern Languages & EFPL



## White-box Framework



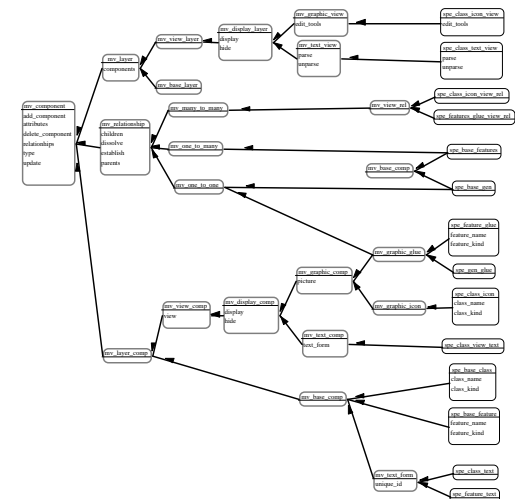
- **Context:** You are building your second application
- **Problem:** How to choose between using inheritance or composition as the basis for using the framework
- **Forces:**
  - Inheritance gives strong coupling between components, but allows reused components to be modified/extended
  - Making a new class requires programming
  - Composition is simpler, but you need to know in advance what can be changed via parameterisation etc
  - Compositions can be dynamic, inheritance is static
- **Solution:** use inheritance to build a white box framework by generalizing from classes in the initial application
- **Why:** inheritance is most expedient way of allowing users to change code in an OO environment - inherit and override. After using this approach for a while it will become clearer as to what changes and what doesn't

COMPSCI 732 §10. Pattern Languages & EFPL

11

## MViews

- MViews was developed by abstracting from experience with Ispel
- Framework of classes for multiple view graphical and textual environments
- Reused via inheritance and overriding of framework classes - ie a white box framework



COMPSCI 732 §10. Pattern Languages & EFPL

12

## Component library



- **Context:** You are developing the second and subsequent examples based on the white box framework
- **Problem:** Similar objects must be implemented for each problem the framework solves. How do you avoid writing similar objects for each instantiation of the framework
- **Forces:**
  - Bare-bones frameworks require a lot of effort to reuse. Things that work out of the box are much easier. A good library of concrete components makes a framework easier to use
  - Its hard to tell initially what components will be reused. Some will be problem specific - some will be reused most times
- **Solution:** Start with a simple library of concrete components and add extra ones as you need them.
  - Add all components initially and later remove ones that never get reused. These are still useful as they give examples of how to use the framework
- In MViews many concrete classes were implemented for use in SPE
- These were adapted or generalised for use in MViewsER

COMPSCI 732 §10. Pattern Languages & EFPL

13

## Hot Spots

- **Context:** You are adding components to the component library
- **Problem:** As you develop applications similar code gets reused over and over again. These code locations are called "hot spots". How do you eliminate this similar code?
- **Forces:**
  - If changeable code is scattered it's difficult to trace and change
  - if changeable code is in a common place flow of control can be obscure
- **Solution**
  - Separate code that changes from code that doesn't - encapsulating the changing code in objects. Composition can then be used to select the appropriate behaviour rather than having to subclass
  - use appropriate design patterns to encapsulate changes eg:
    - algorithm changes => Strategy, Visitor
    - Actions => Command
    - Implementations => Bridge
    - etc



COMPSCI 732 §10. Pattern Languages & EFPL

14

## Pluggable Objects



- **Context:** You are adding components to your component library
- **Problem:** Most of the subclasses differ in trivial ways (eg only one method overridden). How do you avoid having to create trivial subclasses?
- **Forces:**
  - New classes increase system complexity
  - Complex sets of parameters make classes difficult to understand and use
- **Solution**
  - Design adaptable subclasses that can be parameterised with messages to send, code to evaluate, colours to display, buttons to hide, etc
- Check what it is that is changing between subclasses and make an instance variable or whatever to hold the state associated with the change.
- MViews was ported to Java. At the same time many classes were turned into JavaBeans components with settable properties for customisation

COMPSCI 732 §10. Pattern Languages & EFPL

15

## Fine grained objects



- **Context:** You are refactoring your component library to make it more usable
- **Problem:** How far should you go in dividing objects into smaller ones
- **Forces**
  - The more objects in the system the harder it is to understand
  - Small objects allow applications to be constructed by composing small objects together so little programming is required
- **Solution:**
  - Continue breaking objects up into smaller pieces until it doesn't make sense to divide further - ie decide what the "atomic" level is for this domain
  - Frameworks will ultimately be used by domain experts so tools will be developed to compose objects automatically, so it's more important to avoid programming than to avoid lots of objects.
- In JViews, graphics components were reduced in scope to permit design by composition. This led to the development of BuildByWire, a GUI element construction tool

COMPSCI 732 §10. Pattern Languages & EFPL

16

# Black Box Framework



- **Context:** You are developing pluggable objects by encapsulating hot spots and making fine-grained objects
  - **Problem:** How to choose between using inheritance or composition as the basis for using the framework?
  - **Forces:** as per White Box framework
  - **Solution:**
    - Use inheritance to organise your component library and composition to combine components into applications. Inheritance taxonomies support part browsing; composition allows for maximum flexibility.
  - A black-box framework is one where you can reuse components by plugging them together and not worrying about how they accomplish their individual tasks. In contrast, white-box frameworks require an understanding of how the classes work so that correct subclasses can be developed.
  - JViews evolved into a black box framework, with some parts (notably GUI development) more black box than other parts
- COMPSCI 732 §10. Pattern Languages & EFPL

# Visual Builder

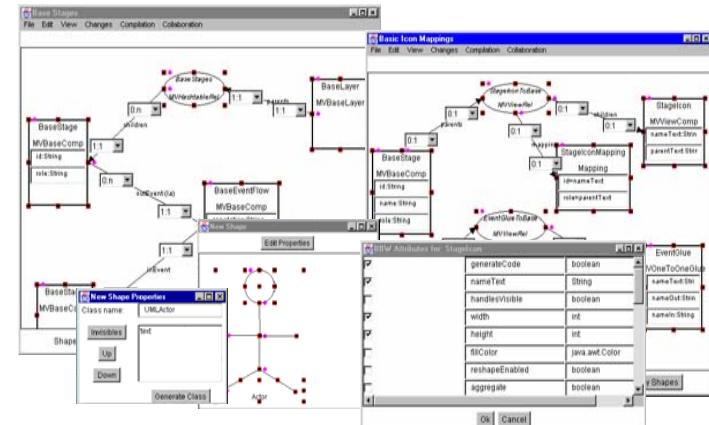
- **Context:** You have a black box framework. Applications are made by composing objects. Behaviour now determined entirely by interconnection of components. Application is now in two parts:
  - Script to connect parts and turn them on
  - Behaviour of parts (provided by framework)
- **Problem:** the connection script is very similar between applications. How do you simplify its construction?
- **Forces**
  - Compositions are complex and difficult to understand
  - Building tools is costly, but domain experts don't want to be programmers
- **Solution:**
  - Construct a visual language and environment to construct the script. This generates the code for the application



# JComposer and BuildByWire

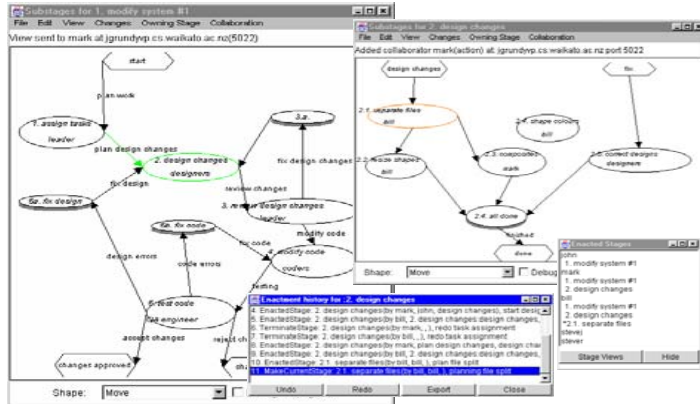
- Developed two visual tools for use in constructing Jviews-based environments:
  - **JComposer:** a tool to visually define most of the "back end" structure
    - further structure filled in by programming using class templates generated by JComposer
  - **BuildByWire:** a tool to visually define the GUI front end
    - defines GUI elements (including interaction points and behaviour) and GUI editing windows
    - generates components that can be used by JComposer to construct complete applications

# BBW and JComposer



## Applications

- Many applications built using JViews
- Eg Serendipity, a process modelling environment



COMPSCI 732 §10. Pattern Languages & EFPL

21

## Language Tools

- **Context:** You have created a builder
- **Problem:** Visual builders create complex composites. How do you inspect and debug these
- **Forces:**
  - Existing tools are inadequate as they don't provide information at the right abstraction level
  - Building good tools takes time
- **Solution:**
  - Create specialised debugging and inspection tools

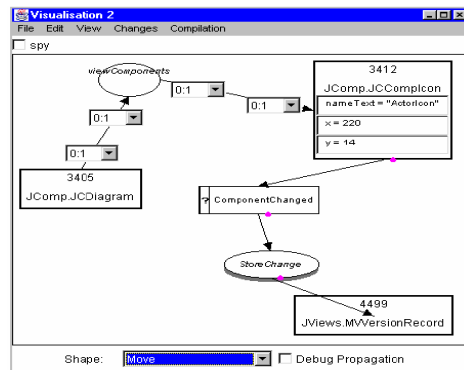


COMPSCI 732 §10. Pattern Languages & EFPL

22

## JVisualise

- JVisualise allows execution state of JViews-based systems to be queried, visualised, and dynamically modified
- Visualisations use abstraction levels equivalent to those used by the JComposer tool



COMPSCI 732 §10. Pattern Languages & EFPL

23

## Extensions to EFPL

- Our experience with developing MViews/JViews has led us to propose several extensions to EFPL (see our paper):
  - **Platform migration**
    - Deals with need to change underlying implmn platforms as lifetime of a framework extends beyond typical impmn technology cycle
  - **Integrating applications**
    - Deals with need/desire to integrate together multiple applications developed using the framework and third party applications
  - **Reflective framework and Self Extending Framework**
    - Dealing with the need to be able to extend the framework "on the fly" using a meta model approach (cf Pounamu)
- The new patterns were workshopped at KoalaPlop 2001

COMPSCI 732 §10. Pattern Languages & EFPL

24



## Application to other frameworks

- **Eclipse**
  - Has a mixture of whitebox and blackbox architectures
  - Has handled integrating applications as core business and has aspects of reflective and self extending framework
  - Some development of visual builder tools (eg PDE) but this is rudimentary.
  - Expect to see significant energy going into visual builder and language support tools to make plugin construction/debugging easier
- **Argo**
  - Very similar to Eclipse, but arguably at a less mature stage
  - Momentum of development lost with the rise of Eclipse
- **Pounamu**
  - A further application of the platform migration pattern to MViews/JViews
  - Rich set of visual builder tools, very much black box

## Summary

- **Framework programming** uses a different style than does conventional software development
  - becoming more the standard approach with the proliferation of application frameworks
- **Pattern Languages** are collections of patterns with rules for combining them to solve problems in a particular domain
  - the "Language" is not a language in the usual programming language sense
- **Evolving frameworks** is a useful Pattern Language for developing a new framework
  - we didn't know about this pattern language when developing MViews/JViews, but in retrospect we used it almost exactly