Smalltalk Frameworks for Business-Critical Aplications

presented by

Piotr Palacz

FreeFall Software Pty Ltd, Sydney

Overview

- Basic Terms and Questions

- Infrastructure Framework
- Business Model Framework
- Transformers Framework
- User Inteface Framework

- Summary: Advocated Approach

Terms: BCA

- Business Critical Application

  - critical to the business
  - many (50-100 +) simultaneous users
  - many geographical locations (2+)

  Also often:
  - high stress/fast turnover environment
  -loosely structured development
  - a number of concurrent teams
  - low average experience with OO
  - legacy factors

- Examples

  - Dealing room systems
  - Insurance policy maintenance
  - Collaterals management
  - Risk/Exposure reporting
  - Generation of advertising campaign plans

Terms: Framework

- Descriptions

  Framework is a set of cooperating classes that make up a reusable design for a specific class of software*[Gamma & al]*.

  Framework is a reusable design of a program or a part of a program expressed a s a set of classes *[Deutsch][Johnson]*.

  “The framework dictates the architecture of your applications.”
  “The framework captures the design decisions that are common to its application domain".

- Main Features

  Framework is described as:
  either a design or its expression;
  a result, rather than a process;
  a purely “technical” thing.

  In practice, each of the descriptions is useful only as a *regulative idea*.

Smalltalk Frameworks: Different?

- Language
  - Syntactically and semantically simple
- Environment
  - Consistent; rich and open
  - Reflexive
  - Permissive
  - Examples of frameworks provided: MVC + dependency; UI builders
- Consequences
  - Fast lifecycle
  - Changing roles and procedures

Frameworks for BCAs

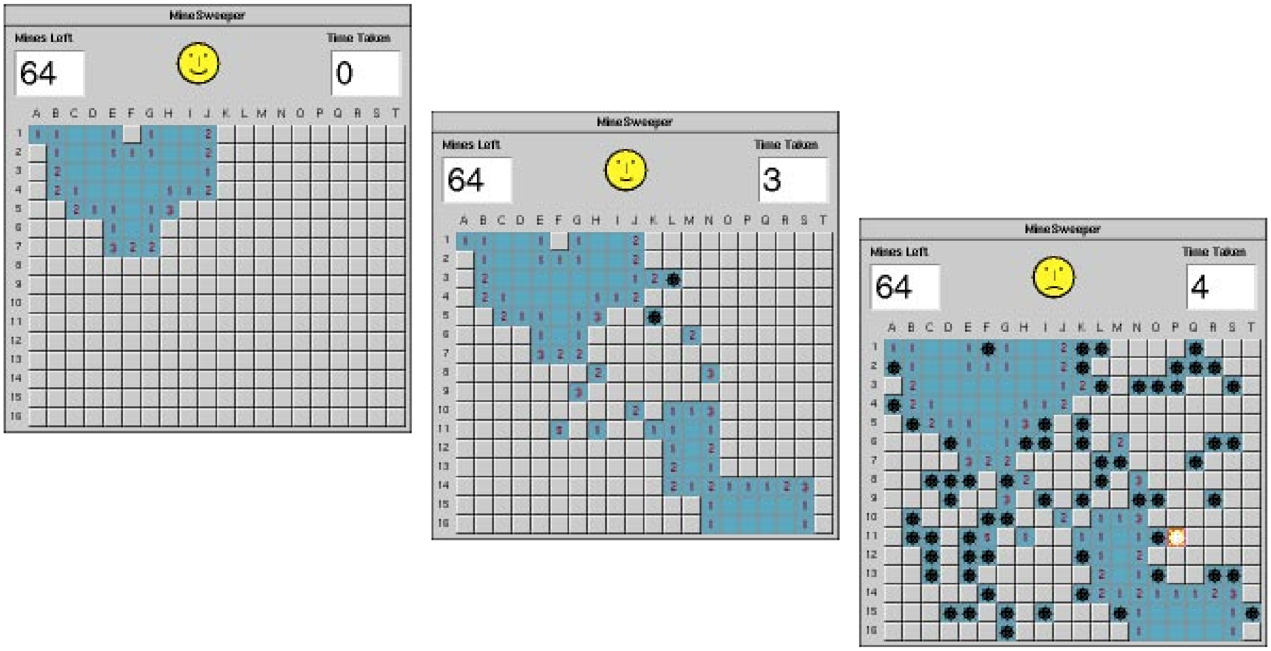
- Conflicting requirements

  - long term expectations vs short term constraints;
  - genericity and reuse vs short time available to build and deploy first iteration;
  - levels of skills required vs the actual inexperience (analysts, developers, managers);
  - delivering vs learning;
  - “old ways” vs. “new ways”;
  - creeping nad/or conflicting user requirements.

- Central Question

  What are the measures and approaches for BCAs to increase your chances for succesful delivery of a viable Smalltalk framework, given all constraints?

‘Naive’ Framework



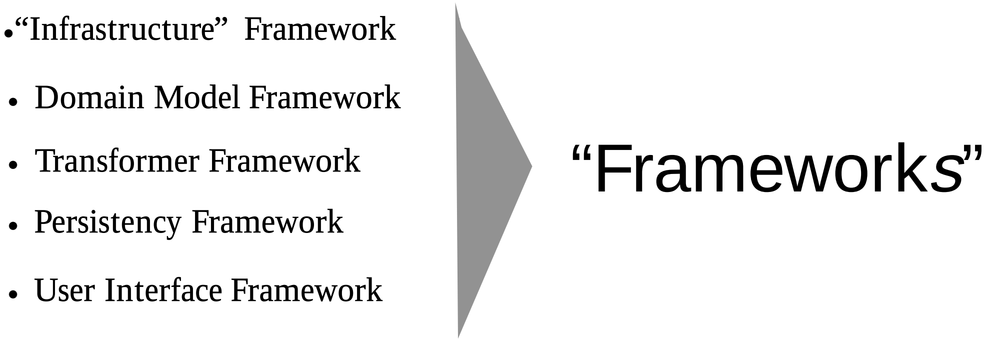
Typical Failure Causes

The development of large applications [...] is one of the most hazardous and risky business undertakings in the modern world [Capers Jones]

- Procedural
  - Poor Communicaton; No Code Reviews
  - No Standards, Conventions, Guidelines
  - Weak Configuration and Change Control
  - Weak testing procedures, esp.regression testing
- Architectural
  - Tinkering rather than (re-)Design
  - No Business Model; GUI-orientation
  - Inability to meet creeping user requirements

Frameworks for BCAs

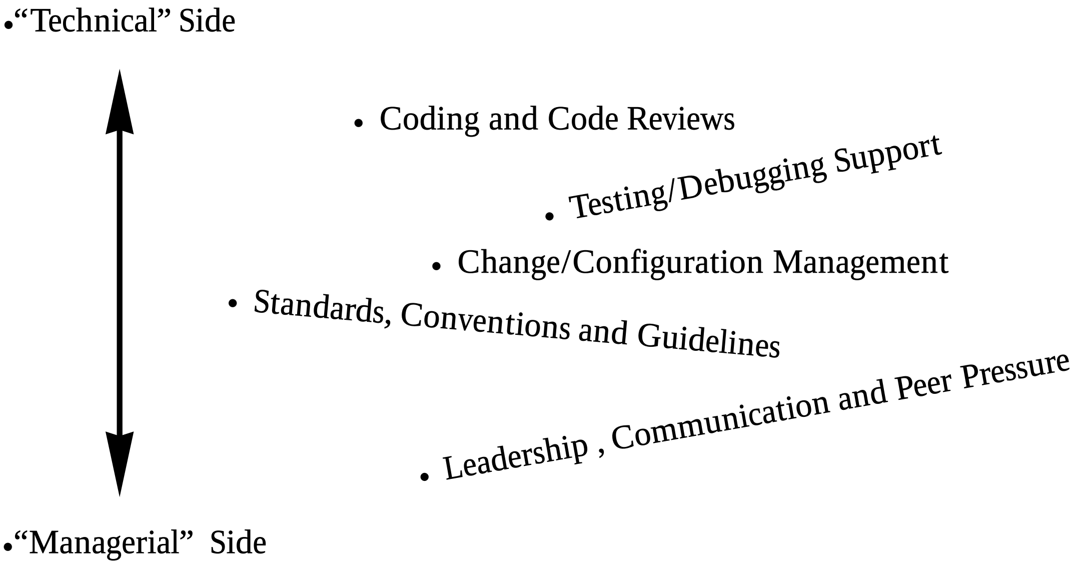
In practice, a BCA is built and maintained using a number of frameworks:



- "Infrastructure" Framework
- Domain Model Framework
- Transformer Framework
- Persistency Framework
- User Interface Framework

"Frameworks"

Infrastructure Framework



- "Technical" Side
- Coding and Code Reviews
- Testing/Debugging Support
- Change/Configuration Management
- Standards, Conventions and Guidelines
- Leadership , Communication and Peer Pressure
- "Managerial" Side

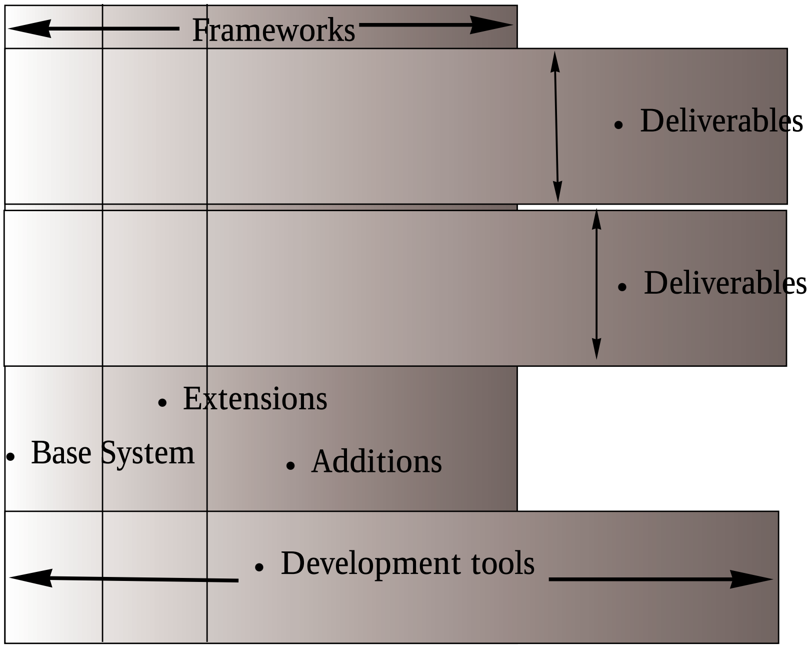
IF: Communication

- Unused Framework is only as good as a non-existent one
- Framework not understood and/or misused is as good as a bad one
- Division of Roles in Smalltalk Development: *Centralised Democracy*
- News Groups, e-mail, scheduled meetings; ad-hoc problem groups
- Talk to people before you code

IF: Standards and Guidelines

- Scope
  - APIs
  - Look and Feel
  - Coding (Mis-)Practices `(self dependents at:3 )perform:#update`
- Enforcing conventions and common style
  - Establishing peer pressure
  - Training for the needy, workshops for others
  - Permanent dissemination

IF: Configuration Management



Frameworks

- Deliverables
- Deliverables
- Extensions
- Base System
- Additions
- Development tools

IF: Change Management

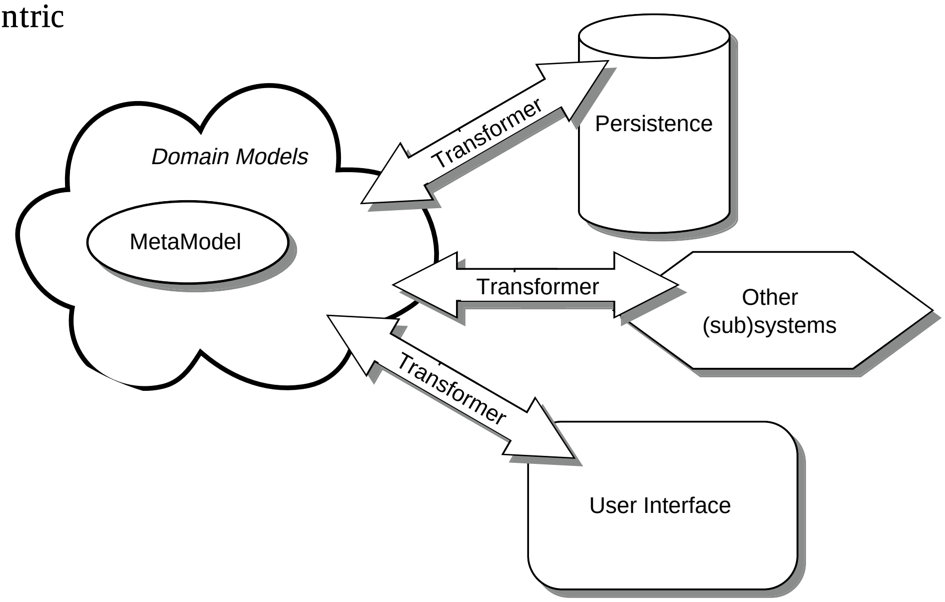
- Requirements
  - Change as an object
  - Arbitrary unit of change
  - Support for recursive prerequisites
  - Detection & resolution of conflicts
- Tools
  - System “ as is”
  - System “tinkered”
  - In-house tools
  - Commercial Tools
  - Customised Commercial Tools

IF: Testing

- Testing
  - Tracing
  - External Configuration
  - Test Methods
  - Logging
  - Basic Functionality Test
  - Regression Tests
  - Interface-driven testing?
- Tools?
  - For Spying
  - For Profiling
  - For Documentation

IF: Architecture

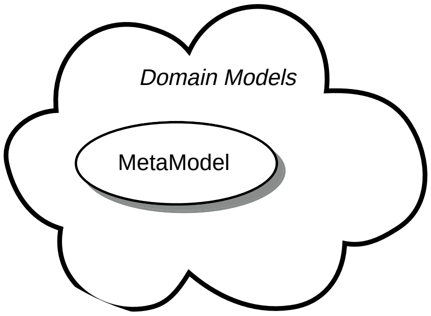
- Types:
  - Interface-centric
  - Ego-centric
  - Model-Centric



Domain Model Framework

Domain Models (aka Enterprise Object):
representations of the structure and behaviour
of the entities involved.

- Meta-description
- Base Responsibilities
- Common Patterns
- Auxiliary Classes



DMF: Meta-description

- What ?

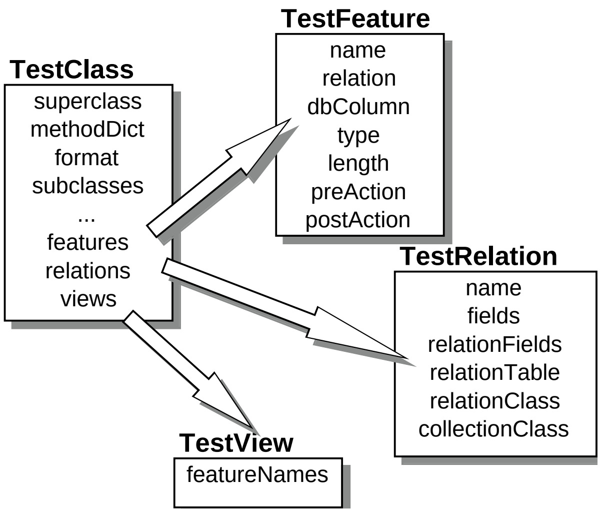
  Central specification of basic properties of an object
  (instances of a domain model class).
  Available at runtime.

- What for?

  - Mechanised generation of the code
  - Pre- & post-conditions in testing
  - High-level interaction with transformers, eg:
    - validation
    - construction of SQL

- How?

  - Feature descriptions
  - Relationship descriptions
  - View descriptions



DMF: Model Responsibilities

- Administratrivia
- State control
- Auto-validation
- Concurrency control
- Security support
- Support for transformers
- Mappings and codesets

DMF: Common Patterns

- Objectives

  -Avoiding hard-coding (class names, requests)
  -Avoiding tight dependence (local API, expression of an algorithm)
  [Gamma &al]

- Patterns
  - Factory — Provides an interface for creating families of related or dependent objects without specifying their concrete class
  - Bridge — Decouples an abstraction from its implementation so that the two can be changed independently.
  - Policy (Strategy) — Defines a family of encapsulated and interchangeable algorithms
  - Builder — Separates the construction of an object from its representation.

DMF: Auxiliary Classes

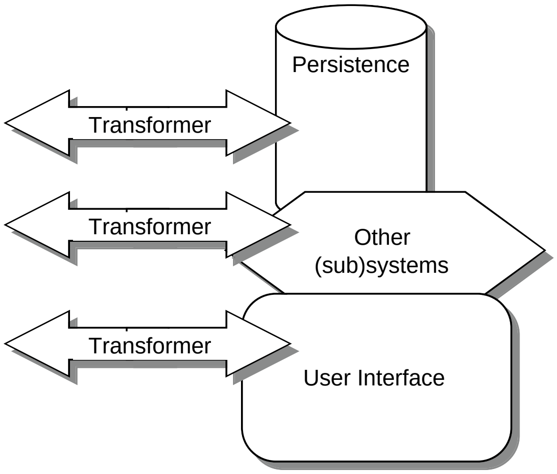
- State machines
- Condition objects
- Filter objects
- Relation objects (incl. trees and containers)
- Input simulators
- SQL generators
- Registries (incl. code sets & object caches)

Transformer Framework

- Transformers (aka adaptors)

  Active APIs encapsulated as pluggube objects

- Types
  - Dependency transformers
  - Slot- and aspect- adaptors
  - Object-Relational transformers
  - UI-bindings



TF: Patterns

- Proxy

  Provides a surrogate or placeholder for another object to control access to it.

- Chain of Responsibility

  Requests between two objects are handled by an intermediary.

- Mediator

  Defines an object that encapsulates how a set of objects interact.

- Visitor

  Represents an operation to be performed on the elements of an object structure.

- Command

  Encapsulates a request as an object.

TF: Example

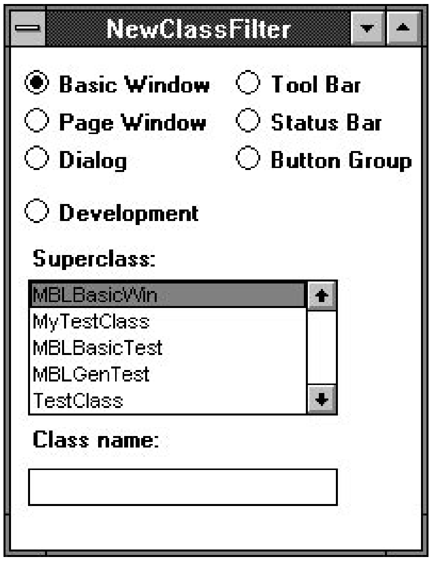
```
Object
     Model (’dependents’)
          ValueModel ()
               ComputedValue (’cachedValue’‘eagerEvaluation’)
                    BlockValue (’block’‘arguments’‘numArgs’)
               PluggableAdaptor (’model’‘getBlock’‘putBlock’‘updateBlock’)
                    TypeConverter ()
               ProtocolAdaptor (’subject’‘subjectSendsUpdates’‘subjectChannel’‘accessPath’)
                    AspectAdaptor (’getSelector’‘putSelector’‘aspect’)
                    DomainAdaptor (’aspect’‘getBlock’‘putBlock’)
                    IndexedAdaptor (’index’)
                         SlotAdaptor ()
                              RangeAdaptor (’subject’‘rangeStart’‘rangeStop’‘grid’)
               ValueHolder (’value’)
                    BufferedValueHolder (’subject’‘triggerChannel’)
```

Source: VisualWorks 2.0, base image

User Interface Framework

- Reusable UI components
- 4GL-ish UI tools
- Linking Models and UIs

- Typical Elements



NewClassFilter

● Basic Window ○ Tool Bar
○ Page Window ○ Status Bar
○ Dialog ○ Button Group

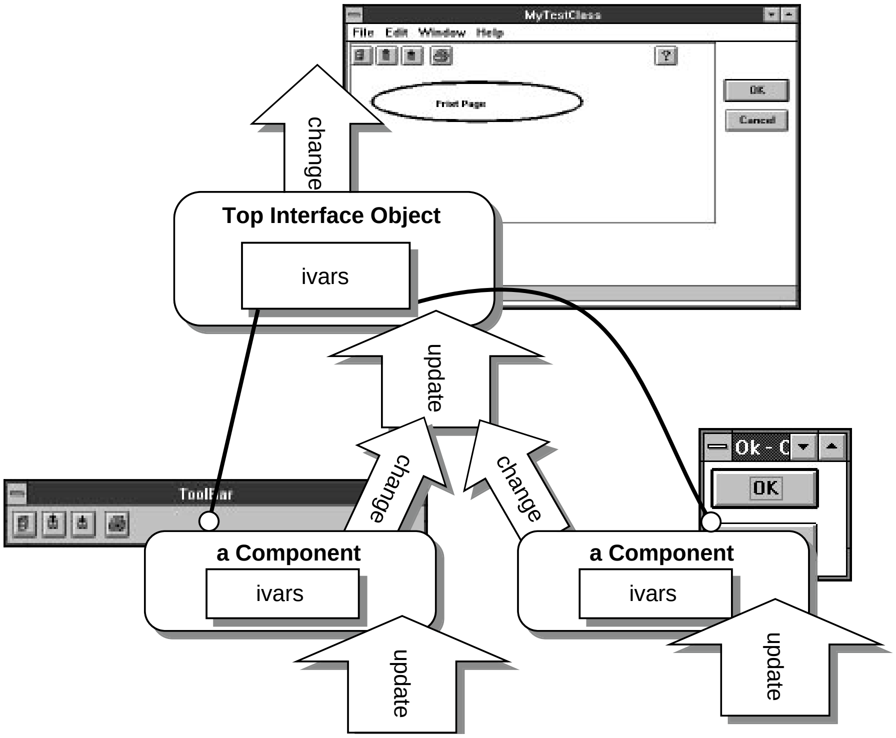
○ Development

Superclass:

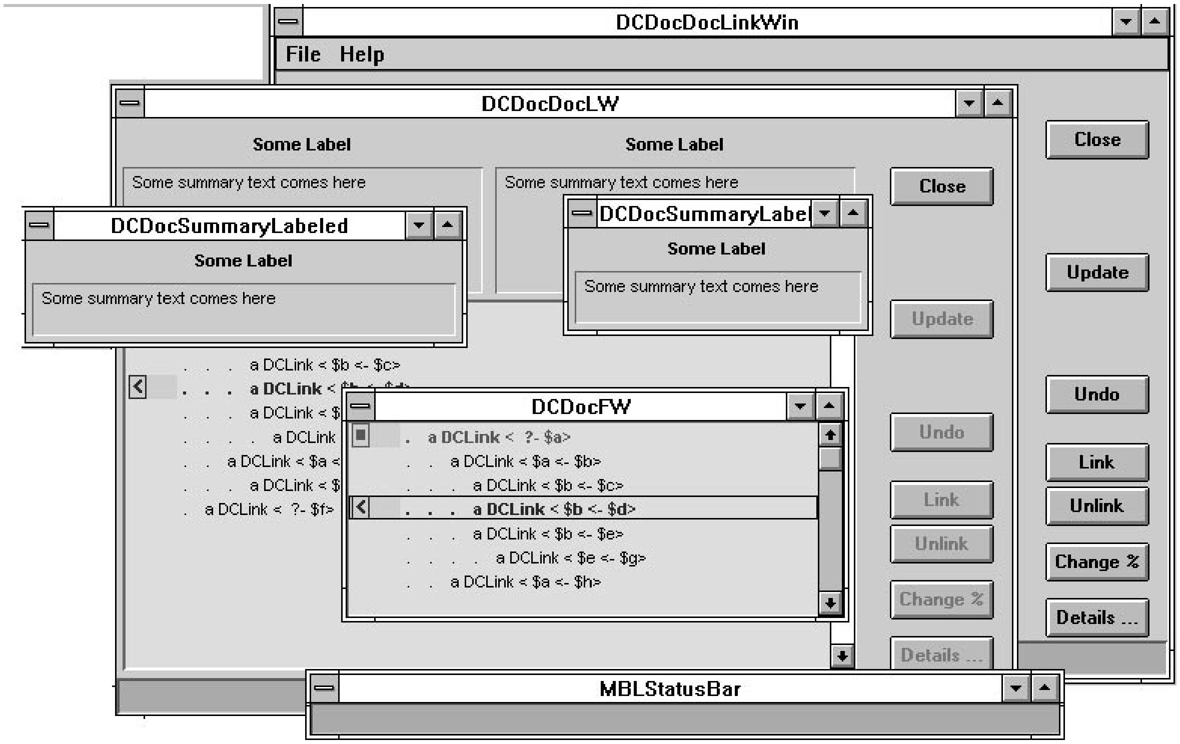
MBLBasicWin
MyTestClass
MBLBasicTest
MBLGenTest
TestClass

Class name:

UIF: Components



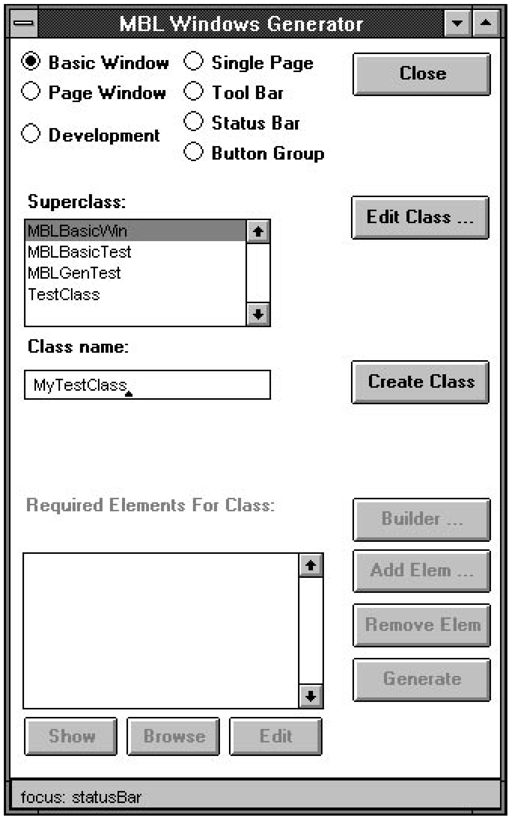
UIF: An Example



```
. . . a DCLink < $b <- $c>
. . . a DCLink < $b <- $d>
. . . a DCLink < $
. . . . a DCLink
. . a DCLink < $a <
. . . a DCLink < $
. a DCLink < ?- $f>
```

```
. a DCLink < ?- $a>
  . . a DCLink < $a <- $b>
  . . . a DCLink < $b <- $c>
  . . . a DCLink < $b <- $d>
  . . . a DCLink < $b <- $e>
  . . . . a DCLink < $e <- $g>
  . . a DCLink < $a <- $h>
```

UIF: Tools



● Basic Window ○ Single Page
○ Page Window ○ Tool Bar
○ Status Bar
○ Development ○ Button Group

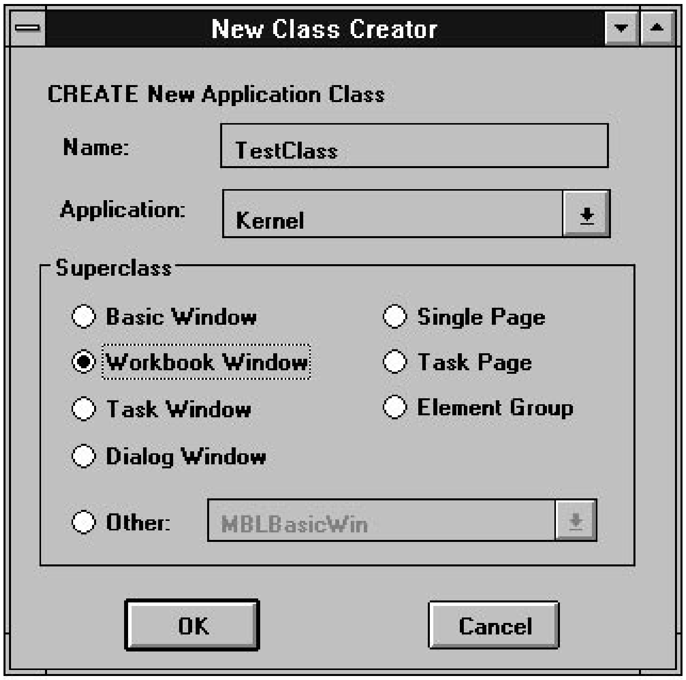
Superclass:
MBLBasicWin
MBLBasicTest
MBLGenTest
TestClass

Class name: MyTestClass

Required Elements For Class:

focus: statusBar

- 4GL-sh GUI tools



CREATE New Application Class

Name: TestClass

Application: Kernel

Superclass

○ Basic Window ○ Single Page
● Workbook Window ○ Task Page
○ Task Window ○ Element Group
○ Dialog Window
○ Other: MBLBasicWin

Persistency Framework

- Physical layer
  - Connections
  - Sessions
  - RDBM API encapsulation
- Logical layer
  - Transactions
- Object layer
  - Decomposition & construction of objects
  - Validation & exception handling

The Future?

- Frameworks as products
- More framework elements available as component-ware
- Emergence of standard types of transformers
- Domain Models driven by high level OOAD tools
- OO host APIs
- First signs: *NeXTSTEP*

Summary: Advocated Approach

- Processual rather than Reistic

  Standard approaches concentrate on the strictly technical
  and on the outcome rather than the process.
  Procedure begets results, and not vice versa.

- Pragmatic rather than Theoretical

  Too much confidence in methodologies (actually, methods) and authority is not productive.
  Magic spells cannot replace invention, although adopting any reasonable rules increases your chances.
  Formal rules are not always helpful in a real situation.

- Top Down rather than Bottom-Up

  Many elements can be determined a priori, eg.
  cornerstones of an architecture can be described and taken into account before concrete design takes place.
  Move first from generic assumptions to specialised ones, not vice versa.
  Foresee rather than use inductive trial and error.

Last Page!

- References

  [Gamma &al] *Design Patterns*, Addison-Wesley 1995
  [Capers Jones] *Patterns of large software systems: Failure and success*, in: Computer, March 1995.
  [Martin Griss] *Software reuse: A process of getting organized*, in: Object Magazie, May 1995

- Credits

  VisualWorks 1.0 and 2.0 were the environment used for the xamples.

  Some of the screen shots come from past development stages of the code owned by Macquarie Bank Limited.

  Thanks to Kevin Bungard of Object Oriented P/L for the early input on database transformers.

- Feedback

  - piotr@FreeFall.com.au; piotr@smalltalk.org.au
  http://WWW.smalltalk.org.au/piotr