Code Generation
from Conceptual User Interface Patterns
The Tutorial

Pedro J. Molina, PhD.
http://pjmolina.com/cuip/
May 18th, 2007
Contents

1. Introduction
2. Patterns
3. Conceptual User Interface Patterns
4. Modelling
5. Code Generation
6. Validation
7. Open Issues
Timetable

9:00 H

Tutorial Block 1

10:05 H

Q&A

10:15 H

Tea / Coffee & Exhibition

10:45 H

Tutorial Block 2

11:50 H

Q&A

12:00 H
1. Introduction

Tutorial

*Code generation from Conceptual User Interface Patterns*
Introduction

- Tutorial Objectives
  - Show the State of the Art in MBCG for user interfaces
  - Introduce the pattern approach to UI development
  - Present an approach for the spec. & generation of code in the domain of user interfaces
A bit of history

The entire history of software engineering is that of the rise in **levels of abstraction**.

Grady Booch

Objective: Reduce the **Semantic Gap** between the human & the machine.
Novak’s Rule

“Automatic Programming is defined as the synthesis of a program from a specification.

If automatic programming is to be useful, the specification must be smaller and easier to write than the program would be if written in a conventional programming language.”

G.S. Novak.
UI Development “like other software”

- Implies moving from an abstract & general problem definition to a concrete and detailed implementation.

- OMG MDA: PIMs & PSM promotes a process to refine models for software refinement in iterations.
The working context at glance

Conceptual Model

- Structure Model
- Dynamic Model
- behavioural Model
- UI Model

Repository

Execution Model

N automatic translations

- App 1 Architecture A
- App 2 Architecture B

Solution Space

N software solutions
State of the Art

- User Interface Modelling
  - TRIDENT [Vanderdonckt93, Bodart96]
  - MOBI-D, Mecano [Puerta96]
  - OVID [Roberts98]
  - UMLi [Pinheiro00]
  - CTT, Teresa [Paternò00]
  - Genova [Genera00]

- Cons:
  - Full domain model is described as input but only User Interface is considered to obtain a tangible result
State of the Art

- CASE modelling tools
State of the Art

- TRIDENT/SEGUIA [Bodart96]
  - AIO / CIO [Vanderdonckt93]
  - Presentation Units
  - SEGUIA:
    - CIO Selection and layout manager
State of the Art

- CTT [Paternò00]
  - Task notation
State of the Art

- OVID [Roberts98]
  - Based in UML
  - View as a Class Stereotype
State of the Art

- **UMLi [Pinheiro01]**
  - UML extension for specifying UI
  - Formal Semantics
  - Low level descriptions
State of the Art

- WISDOM [Nunes01]
  - UML extension
  - Method proposed for small software companies
  - Interaction Spaces
State of the Art

GENOVA [Genera00]
→ Generates CRUD UIs
State of the Art

- UIML [Harmonia02]
- UsiXML  www.usixml.org [Vanderdonckt05]
- XAML [Microsoft05]
State of the Art

CanonSketch [Campos04]
http://dme.uma.pt/projects/canonsketch/
Domain Engineering

• Model Driven Development
  • MDA

• Software Product Line Engineering

• Program Families
  • Parnas D.L., 1976

It is a domain study to:

- Document the domain to deal with

- Identify the **common** part → enclosable in a runtime component

- Identify the **variable** part → specifiable in a model
Commonality Analysis

The document covers the following aspects:

1. Domain Scope
2. General description
3. Term Dictionary
4. Common part
5. Variabilities
6. Variation parameters
7. Critical/Open Issues
The Quest for the Lost Model…

**Parameterisation**
- Reduces common parts
- Increases family size
- Increases complexity

**Immutable**
Sample: Background color fixed to #AAFF88

**Variability**
Sample: Background color can be specified

**Standardisation**
- Increases common parts
- Reduces family size
- Simplifies the system
The Quest for the Lost Model...

“Lo bueno, si breve, dos veces bueno.”
B. Gracián, (XVII century)

“Everything should be as simple as possible, but no simpler.”
A. Einstein, (XX century)
Application Modeling Language

- **DSL (Domain Specific Languages)**
  - Samples:
    - TeX
    - SQL
    - OCL
    - HTML

- **Modulable composable DSL**
  - Pros
    - Reuse
    - Scalability
    - Faster addition of features
Economics of Model Driven Development

- **Economies of Scale**
  - “The condition where few inputs like effort and time are used to produce big quantities of a unique output.” [Wit96]

- **However in Software:** each good is produced **only once**.
  - Copy Cost are = 0 €!

- **Economies of Scope**
  - “The condition where few inputs like effort and time are used to produce a big variety of outputs.
  - More added value producing different goods in the same production line.
  - Scope economy is achieved when combining two or more products in the same production line is cheaper that producing them separately. [Wit96]
Domain Engineer’s Economic Model

Traditional Cost = N * C_T

Cost with Domain Engineering = I + N * C_F

Saving S_F = C_T - C_F
2. Patterns

Tutorial

*Code generation from Conceptual User Interface Patterns*
Patterns

- Origins: C. Alexander (60s)

- First usage in Software engineering:
  - Gamma & et al. “Design Patterns”

- Pattern definition
  - Recurrent problem
    - in a given context
    - with a non trivial solution
    - that can be abstracted and reused
  - Pearl of knowledge (distilled experience)
  - Anti-patterns (didactical also)
We can explain how to make a dress specifying the scissors route over the cloth in terms of angles and cut longitudes or, we can provide a pattern.

Reading the cut specifications, nobody would have idea about what is the goal, at least not till construction.

However, the pattern anticipates the result: is the rule to create the dress. But, at the same time, is also in big amount, the dress by it self.

James Coplien, 1996.
Patterns

- Taxonomy

  - By abstraction level
    - Analysis / Conceptual
    - Design
    - Implementation (Idioms)

  - By domain (application field)
    - Software Engineering
    - User Interface
    - Architecture
    - Social Networks
    - Etc.
Design Patterns. Tidwell


UI Patterns and Techniques

Introduction
About Patterns

Organizing the Content
Overview Plus Detail
Hub and Spoke
Extras On Demand
Step-by-Step Instructions
One-Window Drilldown
Intriguing Branches
Multi-Level Help

Getting Around
Clear Entry Points
Toplevel Navigation
Color-Coded Divisions
Animated Transition
Detail View Navigation

Organizing the Page
Visual Framework
Center Stage
Titled Sections
Card Stack
Closable Panels
Movable Pieces
Progressive Disclosure
Progressive Enabling
Property Sheet
Diagonal Balance

Step-by-Step Instructions

You are designing a UI for a task that is long or complicated, and that will usually be novel for the user -- not something that they do often or want much fine-grained control over, such as the installation of a software package. You're reasonably certain that those of you who design the UI will know more than the user does about how best to get the task done.

Tasks that seem well-suited for this approach tend to be either very long and fiddly, or branched -- they consist of a series of user-made decisions that affect downstream choices.

The catch is that the user must be willing to surrender control over what happens when. In many contexts, that works out fine, since making decisions is an unwelcome burden for people doing certain things: "don't make me think, just tell me what to do next." (Think about moving through an unfamiliar airport -- it's often easier to follow a series of signs than it is to figure out the airport's overall structure.)

But in other contexts, it backfires. Expert users often find Step-by-Step Instructions (especially in wizard form) to be frustratingly rigid and limiting. This is particularly true for software that supports creative processes -- writing, art, etc. Know your users well!
### Magnetism

<table>
<thead>
<tr>
<th>Author</th>
<th>Martijn van Welie</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>Users need to position objects precisely.</td>
</tr>
<tr>
<td>Principle</td>
<td>Task conformance</td>
</tr>
<tr>
<td>Context</td>
<td>Applications that use direct manipulation. The application involves graphical manipulations of item where the relative or absolute positions are important.</td>
</tr>
<tr>
<td>Forces</td>
<td>The users want precision but the display resolution is relatively low. The users see positions as a &quot;final destinations&quot; but dragging usually does not involve &quot;walls&quot;.</td>
</tr>
<tr>
<td>Solution</td>
<td>Make the objects magnetic towards certain positions or other objects.</td>
</tr>
<tr>
<td>Rationale</td>
<td>When the users are positioning the objects, the objects get drawn to other objects or positions when they are close i.e. typically within several pixels. The destination object should act as a &quot;wall&quot; that keeps the moving object from passing it. In other cases, a &quot;bump&quot; is better than a &quot;wall&quot;. The &quot;magnetism&quot; helps users position the objects because they do not the high level of dexterity needed without the presence of &quot;magnetism&quot;. No precision is needed to reach the destination. This decreases the time needed for positioning tasks.</td>
</tr>
<tr>
<td>Examples</td>
<td><img src="http://www.welie.com" alt="Image of WINAMP UI" /></td>
</tr>
</tbody>
</table>

1. (Santana) Oye Como Va  
4:15
Pattern Templates

- Traditionally patterns are described following a template
- There is no universal template

- Each domain has its own templates
- Each author can use what he considers he needs
- The most used: Gamma et al.

- We tried to standardize a common factor as a guide for new patterns in the field of User Interface Patterns and provide a very simple XML DTD: PLML [CHI2003]
Pattern Language

- Set of patterns
  - In a given domain
  - Strongly interrelated
  - Provides a set of primitives to build solutions in the domain
  - No necessarily complete!!!
    - A pattern is an empiric approximation
3. Conceptual User Interface Patterns / CUIP

Tutorial

*Code generation from Conceptual User Interface Patterns*
Just-UI Method

- A pattern language for the specification and generation of business user interfaces

- Objectives:
  1. Improve an OO Conceptual Model with a model for the specification of user interface requirements
  2. In a practical way: looking for a high degree of code generation
  3. Increasing the level of abstraction of the specification in the User Interface
  4. Increasing the quality and productivity of the whole development process

- Very concrete domain
  - User Interfaces for Information Systems / Business software (nevertheless a big market)
Just-UI Method

Proposal:

• Extension of a Conceptual Model

• Introducing an Abstract User Interface Model

• Using Conceptual User Interface Patterns as basic primitives (building blocks)

In a feasible way for enabling reliable UI Code Generation
Advantages:

• A **Unique Specification** for the system (Funct + UI)
  – Making easier the maintenance phase

• **Abstract User Interface Specification**
  – Device independence: Desktop, Web, Mobile, Pervasive Devices, Voice…

• **Rapid Prototyping** of User Interface
  – Running prototypes in early stages of Requirements Gathering & Analysis
Pattern Languages

Requirements
- Analysis Patterns [Fowler96]
- UI Conceptual Patterns [Molina98-03]

Specification

Logical Design
- Design Patterns [Gamma95]
- Common Ground Collection [Tidwell92]
- Usability Collections [vanWelie00, Traetteberg00, Javahery02]

Physical Design
- Idioms [Coplein98]

Implementation

Structural

User Interface

Usability
Delta Effect

Requirements

Analysis

Design

Implementation

$\Delta$

\(\Delta\) Delta Effect.
Conceptual User Interface Patterns

• Captures a recurrent concept in the process of user interface requirement gathering

• Provides an homogeneous solution that allows to solve the requirement in the implementation in an automatic way
Principles / Requirements:

- Easy to use
- Compact Model
- Minimal set of Primitives
- Maximal Expressiveness Power
- Provide a Default Behaviour for frequent cases
- Saves time/effort (following Novak’s Rule)
Conceptual User Interface Patterns

Patterns are arranged in three abstraction levels: from general to particular aspects.

1. User Access
   - HAT: Application Access

2. Interaction Units
   - Service IU: Service
   - Instance IU: Object interaction
   - Population IU: Set of objects
   - Master/Detail IU: Composition
3. Elemental Pattern

- Introduction  Constraints input
- Defined Selection  Error prevention
- Population Selection  Selection Helper
- Supplementary Information  Selection feedback
- Dependency  Fields interdependencies
- Filter  ¿How to limit the output?
- Order Criterion  ¿How to order?
- Display Set  ¿What to show?
- Actions  Commands & actions
- Navigation  Exploration of the system
Just-UI
Conceptual Pattern Language

Level 1 (1)
Level 2 (4)
Level 3 (11)
Hierarchical Action Tree (HAT)

- A task oriented tree providing the user access to the system.
  - Intermediate nodes: grouping labels
  - Leaf nodes: links to interaction units

```
Purchases
  - Purchase article
  - Purchase to supplier
  - Pending orders

Sales
  - Sell article
  - New order
  - Pending orders

Store
  - ...
Interaction Unit

- It is an abstraction of a window, a scenario where user interacts with the system
  - Based on: AIO & Presentation Unit [Vanderdonckt93, Bodart95]
  - Extended with behaviour semantics

Interaction Unit

Name
Alias
Help message
...
Service Interaction Unit

- Represents an IU for a service.
  - Gathers input arguments and launches the service.
Instance Interaction Unit

- Abstracts object presentation.

- Composed of:
  - Display Set
  - Actions
  - Navigation
Population Interaction Unit

- Represents a set of objects
  - Composed of:
    - Filters
    - Order Criteria
    - Display Set
    - Actions
    - Navigation
Master/Detail Interaction Unit

- Represents a composed pattern with head-details semantics.

  - Composed of:
    - A Master Component
    - One or many Detail Components

![Diagram showing master-detail interaction unit](image)
Filter

- **Definition**: Expresses a search condition to locate information about a object population in a given class.
  - It is defined as an open logic expression. It can contain variables to allow the user to provide values for such variables in runtime.

- **Example**: For the class vehicle, it is need to search for free vehicles of a given type.

```
Busy = False AND VehicleType= V_{Type}
```
Order Criteria

- **Definition**: Order mechanism for a population of objects, based on class's attributes.
- Formed by a series of pairs
  
  `<expression, sense>`

- **Example**: Ordering customers by surname & name.

  `<Surname, ASC>, <Name, ASC>`
Display Set

- **Definition**: Order list of object’s properties to be observed.
  - It is specified as an order list of properties’ expressions (class’s attributes & properties reachable by means of associations navigation)

- **Example**: The properties relevant to display an invoice in a given context are: invoice number, total amount and name of the customer.

```plaintext
InvoiceNumber,
TotalAmount,
Customer.Surname,
Customer.Name
```
Navigation

- **Definition**: Is a subset of the semantic relations defined between classes. It allows to cross from one object to related ones by means of semantic relations.

- **Example**: Given a customer, it is convenient to navigate to invoices, payments and current open contracts.

```plaintext
P_Invoices, P_Payments, P_OpenContracts
```
Actions

- **Definition**: Set of services reachable to change the state of a given object.

- **Example**: Cars in an ‘rent a car’ service can be rented, returned or finally sold.

  
  \[ S_{Rent}, S_{Return}, S_{Sell} \]
Introduction

Example

- The user describes certain fields with a well-known format: postal codes, telephones, mail addresses, etc.

- The user interface should prevent errors in this data entry fields.

- The Introduction Patterns [Molina98] addresses this input data problem.
**Introduction Pattern template**

![Introduction Pattern template](image)
Sample generated
Defined Selection

**Nuevo Cliente**

<table>
<thead>
<tr>
<th>Código:</th>
<th>AUTO</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Nombre:</td>
<td>Silvia</td>
<td></td>
<td>Apellido:</td>
<td>Estébanez</td>
</tr>
<tr>
<td>Dirección:</td>
<td>Avda. de Castilla</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teléfono:</td>
<td>949-230-001</td>
<td></td>
<td>Fax:</td>
<td>-</td>
</tr>
<tr>
<td>Población:</td>
<td>Guadalajara</td>
<td></td>
<td>Provincia:</td>
<td>Guadalajara</td>
</tr>
<tr>
<td>CP:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sexo:</td>
<td>Mujer</td>
<td></td>
<td>Fecha de nacimiento:</td>
<td></td>
</tr>
</tbody>
</table>

**Defined Selection sample**
Dependency

- Models interdependences between service arguments.

- Values & states of UI fields change depending on other values & states.

- Dynamic is expressed using event-condition-action rules (ECA).
Dependency pattern example
Dependency Pattern specification window.
State Recovery Sample
State Recovery

State Recovery Sample

![Image of a state recovery sample form]

- Cliente: Estébanez Silvia
- Nombre: Silvia
- Apellido: Estébanez
- Fecha de nacimiento: 11/06/75
- Sexo: Mujer
- Estado Civil: Casado
- Dirección: Avda. de Castilla
- Teléfono: 949-230-001
- Población: Guadalajara
- CP: 23159
- Provincia: Guadalajara

[Buttons: Aceptar, Cancelar]
Argument Grouping
4. Modelling

Tutorial

*Code generation from Conceptual User Interface Patterns*
Just-UI / Diagramming

- Inspired in the principles of
  - Ruble “A&D of O-O applications”
  - Constantine & Lockwood “Software for Use”

- Simple, easy of use. Based in paper, post-its & pens

- Easy to discuss with the user (promotes early validation)

- Allows to involve the user in the design of the system
Graphical primitives in Just-UI.
Just-UI / Diagramming

Just UI / VISIO Prototype
Just-UI / Diagramming

BizUI Prototype / MS DSL Tools

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
Just-UI / Diagramming

BizUI Prototype / MS DSL Tools
Example of graphical language for UI specification.
Example of graphical language for UI specification.
Formalization

HAT

Interaction Units

Connected with actions and navigations
Formalization

- Visibility Semantics

- View
  - Defines a User Interface in the Model
  - It is a Collection of Interfaces
Visibility Semantics

- Attributes
- Services
- Roles

Effective Items = IU Defined Items \( \cap \) Visible Items

- Name, Surname, Salary
- Create, Delete, Edit
- Accounts, Invoices, Payments

- Name, Surname
- Create, Edit
- Invoices, Payments
Meta-model

Class

1:1

0:M

1:1

Service

Attribute

0:M

1:1

Argument

Concept

1:1

0:M

Pattern

definition

application

0:M

0:M
Meta-model

Class

1:1

0:M

Service

1:1 definition

0:M

ServiceIU

Auxiliary Patterns

0:M application

Class

0:M

1:1 definition

ServiceIU

Master/Detail IU

1:1

0:M

Definition

0:M

Application

Class

1:1 definition

0:M

Instance IU

Population IU

Filter

Order Criterium

Display Set

Actions

Navigation

Instance IU

0:1

0:1

0:1

0:1

Population IU

0:M

0:M

0:M

0:M
5. Code Generation

Tutorial

Code generation from Conceptual User Interface Patterns
Code Generation

Rapid Prototyping based on:
- Code Generation
- Inference / Default Values / Scaffolding

Advantages
- Repeatable Process, Quality, Resource Saving, Time to Market, Productivity

Shortcomings
- Fixed Design, Specialized Domain
- Customization → Round Trip Problem
Code Generation

- General Techniques
  - File Cloning
  - Templates
    - Rewriting rules (a la Haskell, Prolog, Lisp, etc.)
  - String Concatenation
    - Grammar based transformations
    - STDs to Finite Automata

- UI oriented Techniques
  - UI Componentization
  - AIO to CIO mappings
  - Design patterns & runtime architectures: MVC, PAC, SmartControls, Webparts
Code Generation Technologies

- Xml de/serialization
  - DOM / SAX

- Parsers
  - ANTLR

- Templates
  - “Enforcing Strict Model-View Separation in Template Engines” [Parr04]
  - StringTemplate
Code Generation: Conceptual Map

- Template
- Meta-model
- Programs / Models
- Mappings
- Transformation Algorithms
- Abstraction
- Instanciation
- Source Code
- Specification
- Classes / Types
- Instances
- Reification
- Reverse Engineering
- Source Code
- Specification
- Template
- Meta-model
- Programs / Models
- Mappings
- Transformation Algorithms
- Abstraction
- Instanciation
- Source Code
- Specification
- Template
- Meta-model
- Programs / Models
- Mappings
- Transformation Algorithms
- Abstraction
- Instanciation
- Source Code
- Specification
Round Trip Problem / Tweaking

- Tweaking
- Round Trip Problem [Bergman02]

![Diagram showing the process of round trip problem and tweaking]

Requirements changes
Spec. evolves

Re-apply can be done automatically?
Development workbench architecture

Modeller / Editor

Validator / Checker

Code Generator

Repository
Tool Support

- ON Modeller
- ON Validator
- ON DTD Meta-model
- ON Transformation Engines
  - Business Logic
  - Client
  - Documentation
  - Function Points Measure
Code Generation Architecture

1. Load
2. Inference
3. Generation

Explicit Specification
Memory Structures

Reification

Source Code

Specifications

Transformations

Templates
Algorithms
### N-layered Architectures

**Pros**
- Scalable
- Multi-channel
- Interoperable
  - Connection of heterogeneous systems

**Cons**
- Performance penalty due to layer over layer cost

**System Components**
- PDA
- Desktop GUI
- Web UI
- 3rd apps
- Web Server
- Exposed Web Services
- Business Logic
- Consumed Web Services
- Consumed Web Services
- 3rd Web Services
- DB (Database)

**Exposed Web Services**
- 3rd Web Services
- 3rd Web Services
- 3rd Web Services

**Consumed Web Services**
- Business Logic
- Business Logic
- Business Logic

**Application Layers**
- Business Logic
- Consumed Web Services
- Exposed Web Services
- Web Server
- PDA
- Desktop GUI
- Web UI
- 3rd apps

**Pros**
- Scalable
- Multi-channel
- Interoperable

**Cons**
- Performance penalty due to layer over layer cost
Mappings: AIO → CIO WinForms

- Hierarchical Action Tree → Menu
- Interaction Unit → Form
- Action → Button
- Navigation → Button
- Display Set
  - An object → Labels
  - Many objects → Grid
- Defined Selection → Combo-boxes
- ...
Examples: VB 6.0/Windows
Mappings: AIO → CIO Web / HTML

- Hierarchical Action Tree → Dynamic Menu
- Interaction Unit → HTML page
- Action → <A> Link
- Navigation → <A> Link
- Display Set → HTML Labels
- Defined Selection → <Table>
  - An object
  - Many objects
- Defined Selection → <Option>
  - ...

Capgemini Spain / Technology Services
Code Generation 2007, May 18th
© 2007 Pedro J. Molina. All rights reserved
### Customer

#### By Name

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Surname</th>
<th>License plate</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>1 Sofia</td>
<td>Marti</td>
<td>4444-GAN</td>
</tr>
<tr>
<td>☐</td>
<td>2 Pedro J.</td>
<td>Molina</td>
<td>1303-CHN</td>
</tr>
<tr>
<td>☐</td>
<td>4 Edy</td>
<td>Malkovic</td>
<td>1234-AHA</td>
</tr>
<tr>
<td>☐</td>
<td>5 Paul</td>
<td>Monz</td>
<td>1111-MNZ</td>
</tr>
<tr>
<td>☐</td>
<td>6 Mary</td>
<td>McNamara</td>
<td>7878-MCN</td>
</tr>
<tr>
<td>☐</td>
<td>7 Ann</td>
<td>Min</td>
<td>0001-MIN</td>
</tr>
</tbody>
</table>

6 item(s).

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Surname</th>
<th>License plate</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>4</td>
<td></td>
<td>11 true</td>
</tr>
<tr>
<td>☐</td>
<td>2</td>
<td></td>
<td>12 True</td>
</tr>
</tbody>
</table>

12 item(s).
Examples: PocketPC/Windows CE

- UICP to Pocket/PC Study [Belenguer02]
6. Validation

Tutorial

*Code generation from Conceptual User Interface Patterns*
Empiric Validation

- FAST [Weiss99]

Domain Engineering Life-Cycle

- Domain Analysis
- Domain Model
- Domain Implementation
- Tools & Domain Workbench

Application Engineering Life-Cycle

- Req. Analysis
- Application Spec.
- Application Development
- Application

Feedback

Investment

Release Development Suite

Payback
Applications Contexts

- Model Execution Environments
  - 1996-1999 / OO-Method Project
    - Research & Development in Conceptual Modelling
  - 1999-2004 / CARE Technologies / Olivanova products
    - Model Execution for business applications
  - 2005-2006 / Bancaja / Pisa Project
    - Financial Terminal for a bank
  - 2006-2007 / Pársec / Capgemini
    - Model Driven Development for improving software factories
Empiric Validation

- Validation in an industrial context
  - Ideas implemented in commercial CARE Technologies products: Olivanova
  - Used for the development of several heterogeneous projects:
    - Golf Tournament Tracking System
    - Sales Dpt. of a Building Co.
    - Furniture Warehouse
    - Web-based Chat for a Telecom Co.
    - e-Profile tracking for a USA University
    - In-house developments (intranet)

- Full MDD workbench implemented by Capgemini for a Spanish Bank: Bancaja (PISA Project)
Empiric Validation / Defect Cost

Defects %

- Traditional SW Life Cycle
- MDD Live Cycle

Defects Cost

Snow ball Effect

Analysis, Design, Build, Maintenance
Empiric Validation

Time redistribution:
- Less time for Design & Coding
- More time for Requirements, Analysis & Testing

Traditional Development

- Requirements 10%
- Analysis 20%
- Design 10%
- Coding 40%
- Testing 15%
- Deployment 5%

Model Execution Development

- Requirements 25%
- Analysis 40%
- Design 4%
- Coding 6%
- Testing 20%
- Deployment 5%

Capgemini Spain / Technology Services
Code Generation 2007, May 18th
© 2007 Pedro J. Molina. All rights reserved
Productivity Measure

- **Developer Effort**
  - As stated by [Myers92]
  - Conceptual Modeling

- **Using Inference**
  - Phase 1. Rapid Prototyping
  - Phase 2: Final UI

Using Inference

- **Phase 1. Rapid Prototyping**
  - UI 40%  Func. 60%
  - 15%  85% Saving

- **Phase 2: Final UI**
  - UI 50%  Func. 50%
  - 50%  50% Saving
User Interface Code Generation

- Rapid Prototyping based on:
  - Inference (Scaffolding a la Ruby on Rails)
  - Code generation

- Pros
  - Repeatable quality processes
  - Resource saving
  - Better *Time to Market*
  - Productivity gain

- Cons
  - Fixed design
  - Applicability at fixed and closed domains
  - Customization $\rightarrow$ Round Trip Problem
Process review

Pros
- Forward engineering → unique source → one way
  - Solution can be anticipated looking at the model
  - Value is in the model
- Robust, proven
- Assured quality by construction
- Fast: prototyping and rapid development

Cons
- Rigid
- Default Values are not always the optimal ones
- Exceptions have to be handled with manual code / tweaking and round-tripping
7. Open Issues

Tutorial

*Code generation from Conceptual User Interface Patterns*
Guideline compliance

- Windows / Mac / X11 Guidelines
- Usability / Accessibility Guidelines
- US Section 508 / Accessibility
- WAI / AAA Recommendations

  • Conformance Level A ➔ Priority 1. must
  • Conformance Level Double-A ➔ Priority 2. should
  • Conformance Level Triple-A ➔ Priority 3. may
Guideline compliance

Sample of WAI Recommendation

• Guideline 2. *Don't rely on color alone.*
  
  – 2.1 Ensure that all information conveyed with color is also available without color, for example from context or markup. [Priority 1]

  – 2.2 Ensure that foreground and background color combinations provide sufficient contrast when viewed by someone having color deficits or when viewed on a black and white screen. [Priority 2 for images, Priority 3 for text].

• Some parts of the guideline can be enforced by a code generator.

• Others should be still guaranteed by a designer.
Guideline compliance

- Put as much as possible standardisation inside the common part.
  - Implement such common part using the guideline criterion to be compliant with.
  - Usually guidelines also include best practices in the domain → study them deeply before starting the design of the output

- Now the code generator produces compliant software by design.

- In some contexts the product must be certified to proof compliance
  - If you can proof the compliance of your code generator output you will get the certification for all the output produced by the generator.
High vs Low level UI modelling

UI Modelling

**High level Spec.**

**Pros:**
- near to the problem domain
- more general

**Cons:**
- difficult to implement,
- rigid automatic implementations

**Low level Spec.**

**Pros:**
- adapted to the target device
- very precise

**Cons:**
- device dependent
- difficult to be re-targeted to another device
Modelling & Code Gen. Technologies

- Diagramming tools
  - Eclipse GMF
  - MS DSL Tools
  - MetaEdit+
  - UML Based tools

- Textual Domain Specific Languages
  - Grammars & Parsers: ANTLR

- Templates
  - Strict separation of model, transformation & presentation (template)
  - output grammars → StringTemplate
User Interface Code Generation

- **Key concept**
  - AIO → Abstract Interface Object (technology independent)
  - CIO → Concrete Interface Object (platform dependent)
  - Mappings between them

- **Architectures**
  - MVC → Model View Controller
  - Direct data-binding
User Interface Code Generation

- **AIO → CIO Selection**
  - Based on platform mapping
  - Based on data type
  - Based on usability rules
  - Based on user choice

<table>
<thead>
<tr>
<th>AIO</th>
<th>CIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept</td>
<td>Windows</td>
</tr>
<tr>
<td>View</td>
<td>Form</td>
</tr>
<tr>
<td>List of objects</td>
<td>Grid</td>
</tr>
</tbody>
</table>

![Image of user interface code generation with examples](image)
User Interface Code Generation

- **Automatic Layout Generation**
  - There is not silver bullet
  - However you can find good heuristics to solve 80% of cases
  - Column alignment, multicolumn, grid, etc.

- **Choices depend on**
  - Domain
  - Device physical constrains

![User Interface Code Generation Example](image-url)
Creativity vs Automation. Where to stop?

**Artist & graphical designers**
- **Skills:**
  - Creativity
  - Design aesthetics
- **Tools:**
  - Dreamweaver, Illustrator, Freehand

**Programmers**
- **Skills:**
  - Analytic thinking
  - Abstraction
- **Tools:**
  - IDEs, compilers, debuggers, profilers.

- They have different skills sets, tools & languages
- It is difficult to find people with both skills
- Team work is needed to
  - Improve communication
  - Clear separation of concerns (SoC) & responsibilities
  - Degrees of freedom
Conclusions

1. User Interface Conceptual Patterns
2. UI Pattern Language for Business Applications
3. Abstract Specification Model (device independent)
4. Simple Graphical Notation
5. Inference support for Rapid Prototyping
6. Scalable/Industrial validated method
PhD Thesis (in Spanish)

User Interface Specification:
From requirements to automatic code generation.


More info about: CUIP

Index of Conceptual User Interface Patterns available on-line at:

http://pjmolina.com/cuip