Migratable User Interface Descriptions in Component-Based Development

Kris Luyten Chris Vandervelpen
Karin Coninx

Expertisecentrum Digitale Media
Limburgs Universitair Centrum

12 June, DSV-IS 2002, Rostock

Presentation Overview

1. Introduction
2. The SEESCOA Component System
3. User Interface Descriptions
4. Available Interactors
5. The UI Component
6. The UI Component Architecture
7. Components Example Case
8. The Rendering Engine
9. The Layout Management Problem

10. Future Work
11. Conclusion
12. Questions

1. Introduction

Goal: Dynamically generating user interfaces for embedded systems using a high level UI description

- Building system and device independent UIs
- Enhance component-based development for embedded systems with high-level UI descriptions
- Easy to build for non-programmers

2. The SEESCOA Component System

- SEESCOA: **S**oftware **E**ngineering for **E**mbedded **S**ystems using a **C**omponent **O**riented **A**pproach.
- Components reusable for embedded systems
- Specific properties:
  - Asynchronous
  - Dynamically composable
  - Replacable at runtime
- Components annotated by UI description

3. User Interface Descriptions

- Focus on structure UIs
- XML syntax (platform independent, declarative, consistency, constraints, extensible, reusable, transformations)
- Linked to individual components, like functional component interface definition
- Only "interaction" components should export description

```
<interactor>
  <range name="focusrange">...</range>
</interactor>
<interactor>
<button name="camera1_onoff">
 <action>
  <func service="Surveillance.Controls">
     switch
  </func>
  <param name="camera2"/>
  <param name="camera1_onoff"/>
 </action>
</button>
</interactor>
```

4. Available Interactors

- Button
- Range
- Choice (Single, Multiple)
- Text input
- Video
- Label
- Actions

Enough to describe simple dialogs, forms only

5. The User Interface Component

- Special Purpose component
- UI description can be "submitted"
- The same interface, other rendering backends
- Can be replaced at runtime
- **input**: one or more high-level UI descriptions, **output**: a working User Interface

6. The UI Component Architecture

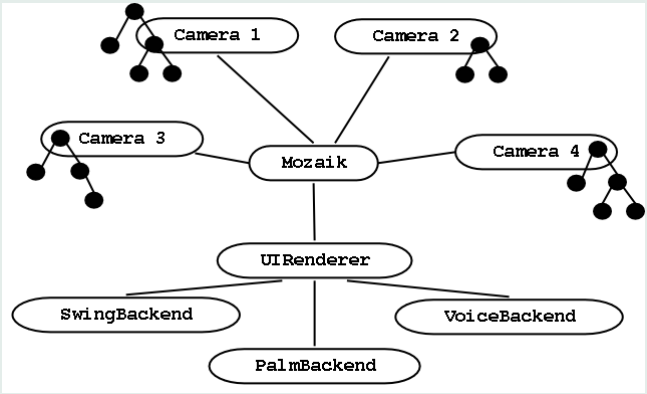
- 2 kinds of parsing available:
  - DOM: more memory, easier to experiment with
  - SAX: less memory, faster, more coding
- Available widgets dynamically loaded
- Availabe output formats/backends are modular and self-contained

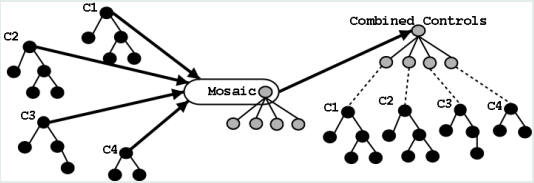
7. Components Example Case

The camera surveillance system

- Description: a guard has to check four places using the surveillance camera's
- Solution: dynamically generate a mosaic interface for viewing them all at once
- Each camera is represented by a Camera component
- Each Camera component is annotated with a high-level UI description
- The application logic requires a Mosaic component

- Mosaic component merges the camera component descriptions and submits them to a UI component




C1
C2
C3
C4
Mosaic
Combined_Controls
C1
C2
C3
C4


Camera Mozaik <2>
Zoom:
value : 0
Brightness:
value : 0
Zoom:
value : 0
Focus:
value : 0
Zoom:
value : 0
Focus:
value : 0
Zoom:
value : 0
Contrast:
value : 0

8. The Rendering Engine

- Independent, self-contained backends
  - Java Swing, Java AWT, HTML, Palm,...
- Eats XML, produces an adapted User Interface
- Action triggers
  - Interactors can have "action" childs
  - Action child = func child (target + method) + zero, one or more param childs

9. The Layout Management Problem

- Focusing on 2D dialog-based UIs
- Radically changing screen space and resources
- Define *spatial contracts* on the interactor layout
- Use hierarchical groups to allocate screen space

10. Future Work

- At the low end: Rendering engines for AWT, Swing, Applet (AWT) and HTML are finished, PDA in progress
- Conversion to other XML-based UI descriptions: UIML, XForms,...
- Adding sort of Domain Model: defining set of interactors for a specific context
- Introducing a Task Model on top
  - e.g. ConcurTaskTree (using XML output)
- ...

11. Conclusions

- The proposed approach offers
  - flexibility
  - adaptibility
  - reusability
- Dynamically show new interfaces (webservices)

12. Questions, Remarks,...

?