An XML-based runtime user interface description language for mobile computing devices

Kris Luyten and Karin Coninx
Expertisecentrum Digitale Media
Limburgs Universitair Centrum

Design, **S**pecification and **V**erification of **I**nteractive **S**ystems
13 June 2001, Glasgow

1. Introduction

A desktop PC can not assist you everywhere you go!

But what will, and what will it take?

Mobile computing devices are invading our world!
(e.g. Personal Digital Assistants)
but. . . :

- heterogeneous environments
- physical constraints
- device constraints
- rapidly evolving technology

(Mobile Computing devices contd)

- lack "runtime extensibility"
- developers vs. designers

Mobile computing devices must assist us whenever possible, facing these problems.

Desired user interface characteristics:

- are *downloadable*
- adaptable to the user preferences
- adaptable to the target device constraints
- dynamically use services everywhere we go (*log in* on the services)

2. User Interfaces and Constraints

Two kinds of constraints:

static constraints do not evolve over time

- bodily functions

evolving constraints do evolve over time

- memory usage
- bandwidth
- processor speed

The proposed solution is using a *user interface description language*

- which is suitable for a wide range of embedded and mobile computing devices
- and works in heterogeneous environments
- utilizing this at runtime

This description language:

This description language:

1. must be *platform independent*

This description language:

1. must be *platform independent*

2. must be *declarative*

This description language:

1. must be *platform independent*
2. must be *declarative*
3. must offer us a *consistent description* (e.g. a family of products need a consistent look-and-feel)

This description language:

1. must be *platform independent*
2. must be *declarative*
3. must offer us a *consistent description* (e.g. a family of products need a consistent look-and-feel)
4. can describe *unconventional I/O*

(description language contd)

(description language contd)

5. offers *rapid prototyping*

(description language contd)

5. offers *rapid prototyping*

6. can describe *constraints*

(description language contd)

5. offers *rapid prototyping*

6. can describe *constraints*

7. is *easily extensible*

(description language contd)

5. offers *rapid prototyping*

6. can describe *constraints*

7. is *easily extensible*

8. is *reusable*

3. Describing a User Interface

What could help us to do this?

3. Describing a User Interface

What could help us to do this?

eXtensible Markup Language

Why?

Why?

1. platform independent
2. declarative
3. consistent
4. unconventional I/O
5. rapid prototyping
6. constraints
7. extensible
8. reusable

Why?

1. platform independent	√
2. declarative	
3. consistent	
4. unconventional I/O	
5. rapid prototyping	
6. constraints	
7. extensible	
8. reusable	

Why?

1. platform independent	√
2. declarative	√
3. consistent	
4. unconventional I/O	
5. rapid prototyping	
6. constraints	
7. extensible	
8. reusable	

Why?

1. platform independent	√
2. declarative	√
3. consistent	√
4. unconventional I/O	
5. rapid prototyping	
6. constraints	
7. extensible	
8. reusable	

Why?

1. platform independent	√
2. declarative	√
3. consistent	√
4. unconventional I/O	√
5. rapid prototyping	
6. constraints	
7. extensible	
8. reusable	

Why?

1. platform independent	√
2. declarative	√
3. consistent	√
4. unconventional I/O	√
5. rapid prototyping	√
6. constraints	
7. extensible	
8. reusable	

Why?

1. platform independent	√
2. declarative	√
3. consistent	√
4. unconventional I/O	√
5. rapid prototyping	√
6. constraints	√
7. extensible	
8. reusable	

Why?

1. platform independent	√
2. declarative	√
3. consistent	√
4. unconventional I/O	√
5. rapid prototyping	√
6. constraints	√
7. extensible	√
8. reusable	

Why?

1. platform independent	$\surd$
2. declarative	$\surd$
3. consistent	$\surd$
4. unconventional I/O	$\surd$
5. rapid prototyping	$\surd$
6. constraints	$\surd$
7. extensible	$\surd$
8. reusable	$\surd$

The process for *Downloading* a UI:

The process for *Downloading* a UI:

1. a working user interface

The process for *Downloading* a UI:

1. a working user interface
2. client asks for user interface

The process for *Downloading* a UI:

1. a working user interface
2. client asks for user interface
3. user interfaces serializes into an XML ui description

The process for *Downloading* a UI:

1. a working user interface
2. client asks for user interface
3. user interfaces serializes into an XML ui description
4. the XML ui description is transferred

The process for *Downloading* a UI:

1. a working user interface
2. client asks for user interface
3. user interfaces serializes into an XML ui description
4. the XML ui description is transferred
5. the XML ui description is adapted (this could happen on the server-side)

The process for *Downloading* a UI:

1. a working user interface
2. client asks for user interface
3. user interfaces serializes into an XML ui description
4. the XML ui description is transferred
5. the XML ui description is adapted (this could happen on the server-side)
6. the XML ui description is converted into a working user interface

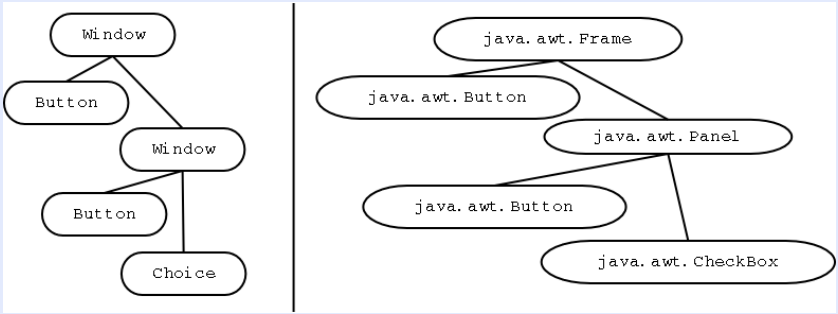

Figure 1: left:Abstract Interaction Objects (AIO) – right:Concrete Interaction Objects (CIO)

The user interface will be abstracted in the XML UI description

4. Runtime User Interface Conversion

We want the user interface to give its XML description at runtime. Java seems a good programming language to implement downloadable user interfaces:

- has a rich user interface toolkit
- supports a Reflection Mechanism (the program structure can be interrogated at runtime)
- Sun seems to push Java towards XML

- Java supports dynamic class loading using its virtual machine
- Java has a good support for Remote Method Invocation

Some considerations:

- we do not want to limit ourselves to Java
- a programmer should be able to use its "weapon of choice" (most appropriate programming language for the problem)
- programming languages without a reflection mechanism need a supporting framework

Our current implementation

- includes:
  - Java AWT widgets to XML serialization (ad-hoc CIO to AIO mapping)
  - network transport of the XML
  - deserialization of XML into Java (AIO to CIO mapping)
  - experimental Remote Method Invocation
  - basic layout algorithm

- to be done:
  - optimizing bandwidth
  - user profiling
  - support for device constraints

5. Downloading the User Interface

- mapping AIO to CIO
- taking the device constraints into account
  - currently expressed in XML
- taking the user profile into account
  - currently expressed in XSLT (with XPath)

6. Questions and Remarks

?