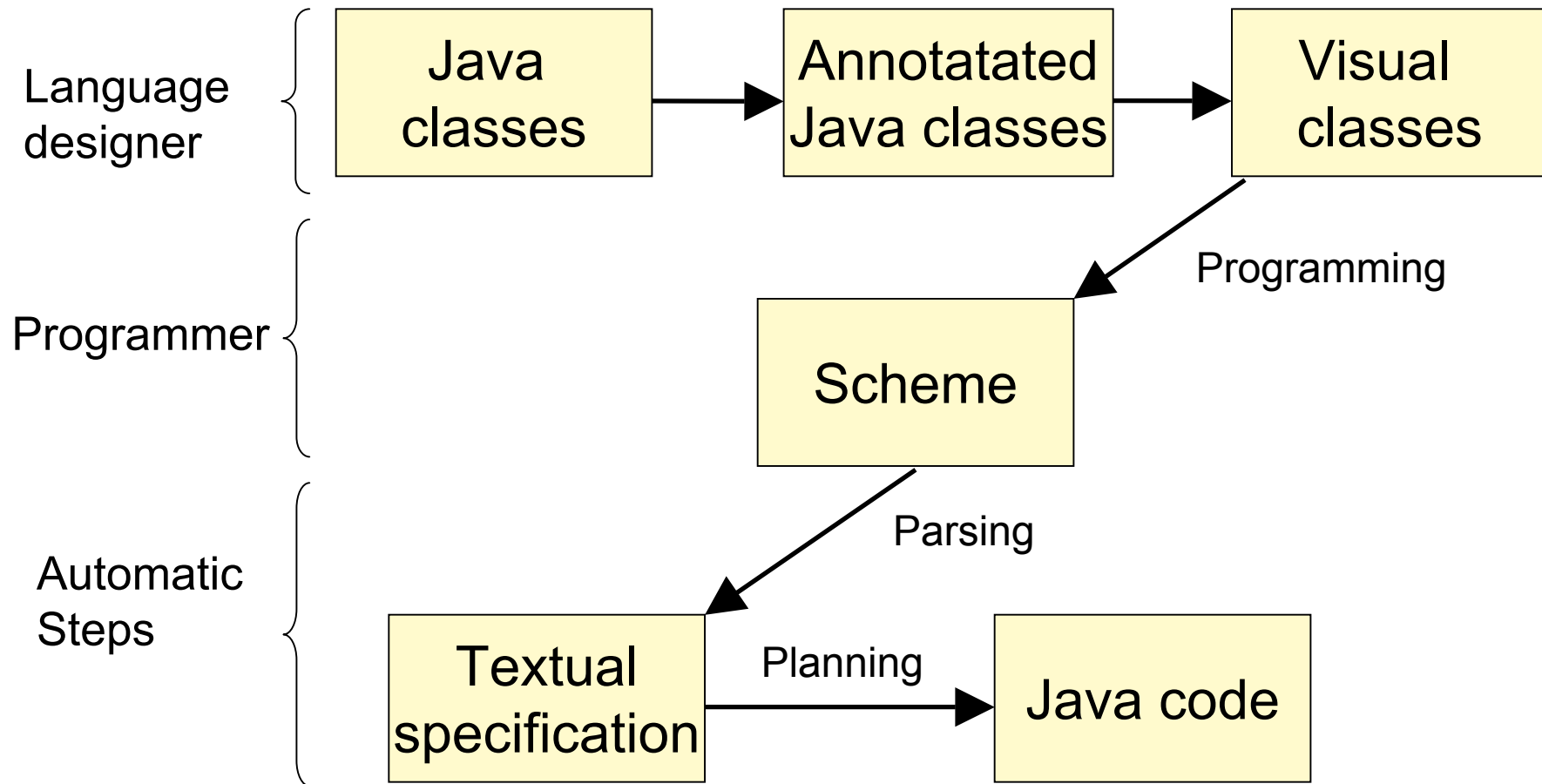

A Framework for Design and Implementation of Visual Languages

Ando Saabas,
Institute of Cybernetics

Overview

- Visual languages in general
- Steps in the framework
- Designing a simple visual language – a demo

Visual language lifecycle



Specification language

- Specifications are written either in separate files or included in Java source files between comments.

Example

```
class Sample {  
    /*@ specification Sample {  
        ...  
    }  
    @* /  
    ...  
}
```

The language core

- Variable and constant declarations

Example

```
String s;  
int a, b;  
a = 3;
```

- Bindings

```
SampleSpec x;  
x.a = x.b;
```

Axioms

- Unconditional computability statements

$$A_1, \dots, A_n \rightarrow B \mid E_1 \mid \dots \mid E_n \{M\};$$

Example

```
int in1, in2, out;
```

```
in1, in2 -> out {findResult};
```

Axioms continued

- Conditional computability statements

$$[A_{11}, \dots, A_{1n} \rightarrow B_1], \dots, [A_{m1}, \dots, A_{mn} \rightarrow B_m], \\ C_1, \dots, C_k \rightarrow D \mid E_1 \mid \dots \mid E_j \{M\}$$

Example

```
int c, b;
```

```
SampleSpec a;
```

```
double in;
```

```
[b -> c], a -> in {findInput};
```

Extensions

- Equations

```
int a, b;
```

```
double c, d;
```

```
a + b = c * d^2;
```

- Aliases

```
alias pair1 = (a, c);
```

```
alias pair2 = (b, d);
```

```
pair1 = pair2;
```

- Wildcard

```
*.in = a;
```

Specification language

```
Class Sample {
  /*@ specification Sample {
    int a, b, c;
    String s1, s2;
    [a -> b], s1 -> s2 {getS};
    b = a * c;
  }
  @*/

  String getS (Subtask subtask, String s) {
    ...
  }
}
```

The visual extension

```
<package>
  <class>
    <name>Wheel</name>
    <description>Toothed wheel</description>
    <icon>wheel.gif</icon>
    <graphics>
      <bounds x="0" y="0" width="40" height="70"/>
      <rect x="0" y="34" width="40" height="3" colour="0"
        filled="true"/>
      ...
    </graphics>

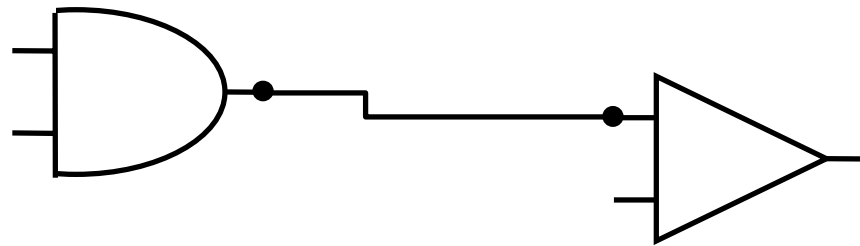
    <ports>
      <port name="tang" x="20" y="0" portConnection="area"
strict="true">
      <graphics>
      ...
```

A scheme

- A scheme is a set of objects, where the ports of objects can be connected to each other so that they form a simple graph, ie a graph with
 - no arrows
 - no loops
 - no multiple edges

- The semantics of a scheme gives a textual form of the visual specification – data structure representing the graph.

A sample scheme



```
And and_0;  
Or or_1;  
and_0.out = or_1.in1;
```

Structural synthesis rules

Conjunction introduction

$$\frac{\Gamma \vdash a : A \quad \Sigma \vdash b : B}{\Gamma, \Sigma \vdash (a, b) : A \wedge B}$$

Implication introduction

$$\frac{\Gamma, a : A \vdash b : B}{\Gamma \lambda a. b : A \rightarrow B}$$

Implication elimination

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Sigma \vdash N : A}{\Gamma \Sigma \vdash MN : B}$$

Structural synthesis rules

Double implication elimination

$$\frac{f : \underline{(A \rightarrow B)} \wedge \underline{C} \rightarrow D \quad \underline{\Gamma, a : A} \vdash b : B \quad \underline{\Sigma c : C}}{\Gamma, \Sigma \vdash f(\lambda a. b, c) : D}$$

Conjunction elimination

$$\frac{\vdash M : \underline{W} \rightarrow A \vee B \quad \underline{\Gamma} \vdash N : \underline{W} \quad \underline{\Sigma, u : A} \vdash N_1 : C \quad \underline{\Delta, w : B} \vdash N_2 : C}{\underline{\Gamma, \Sigma, \Delta} \vdash (\text{case } \underline{MN} \text{ of } \text{inl}^B u \Rightarrow N_1 | \text{inr}^A w \Rightarrow N_2) : C}$$