ui2go Design Decisions

Dirk Struve
phylofriend at projectory.de
https://github.com/yogischogi/ui2go/

March 16, 2015

Contents

1 Why ui2go?

1.1 Programming Situation

1. Started learning Korean and wrote a small language course.

2. Needed several programs to automate language course building.

3. No time for programming, wanted to learn Korean (still want to, spent too much time on programming ;-).

4. As I just spent a few hours every month on programming, the following issues became most important:

   - Readability: After a month I forgot most things I had done before.
   - Productivity: I wanted to spend time on programming, not on programming languages and associated tools.
   - Incremental programming: I had no time to do big changes at once.
   - Stability of programming environment: Widely adopted programming tools often change and require lots of maintenance. This usually goes unnoticed when using them daily.

5. Go turned out to be a good choice for the task.

6. But I needed a simple UI for some tasks sometimes.

7. Existing UI technologies left me frustrated.

1.2 What I Needed

1. Fast creation of simple UIs.

2. No fully fledged UI toolkit.

2 Events

2.1 Event Propagation

1. Event system should be useful for arbitrary programs.

2. Event package contains universal event sender and receiver classes.

3. Every class can be made an event sender or receiver just by mixing in event sender or receiver.

4. Event senders and receivers support function calls and channels for event propagation.

5. Programmers can use push or pull style.

6. In GUI programs: circular control flow metaphor
    - Events flow from input device to GUI, from there to the program logic and back again.
    - automatic event propagation from window down to the controls and up again
    - automatic event propagation done by function calls (no concurrency issues, much faster than channels)

2.2 Event Structure

1. Traditional event systems are created for performance and hard to use.
2. Try to create easy-to-use and semantically meaningful events.

3 Layout Management

1. Inspired by MiG-Layout, but simpler and easier.
2. Layout is done like printing lines (a bit like like Printf).
3. Layout manager tries to take the burden from the programmer.
4. Mock-up mode for testing a layout without creating widgets.
5. No pixel accurate layout (layout manager tries to automate as much as possible).

4 Drawing Model

1. All drawing operations are based upon go-cairo.
2. Cairo is stable, widely used and well documented.
3. But cairo is not "goish", feels a bit strange sometimes.
4. Drawing of widgets (not layout) like CSS box model.

5 Cross Platform

5.1 Pros

1. nice to have the same API on different platforms
2. results in larger user group
3. will increase project popularity

5.2 Cons

1. A cross platform library does not make a program cross platform.
   - lots of subtle platform specific details (apart from the library), that often require an extreme amount of work for simple tasks
   - truly platform independence results in middle-ware OS
2. Library will become more complex (error prone and slow).
3. Dependencies on extra libraries introduce lots of problems (bugs, version changes).
4. Cross platform design puts restrictions on the project that may turn out as big problems later.
5. Burden to the programmer.
   - longer training period because of extra complexity
   - Programmers need to know the details of the cross platform API and the underlying platform (marketing people always make different claims, but...).
6. Platform specific programs tend to be smaller, faster and more usable.

5.3 Solution

1. The need for an easy-to-use event system turned the decision in favour of a cross platform solution.
2. Window abstraction layer directly on top of the system libraries.
   - small code base
   - fast