GUI with Tkinter

Copyright © 2003-2021 IPT – Intellectual Products & Technologies Ltd. Licensed under the Apache 2 license

About me

Trayan Iliev

– CEO of **IPT – Intellectual Products & Technologies**
http://www.iproduct.org

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6+, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer: Spring, Java EE, Node.js, Express, GraphQL, SOA, REST, DDD & Reactive Microservices

– Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast

Where to Find The Code and Materials?

https://github.com/iproduct/intro-python

MVC Comes in Different Flavors

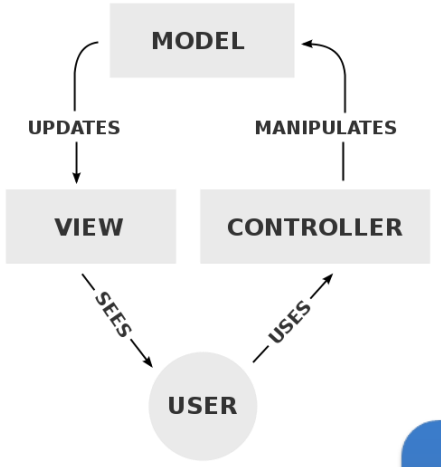
What is the difference between following patterns:

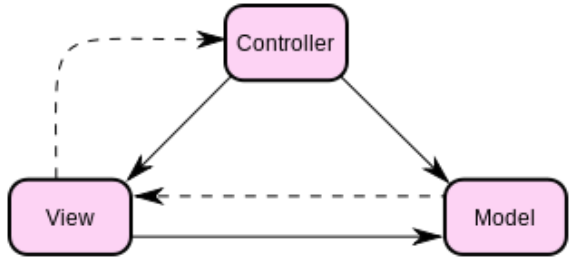
- Model-View-Controller (MVC)
- Model-View-ViewModel (MVVM)
- Model-View-Presenter (MVP)

http://csl.ensm-douai.fr/noury/uploads/20/ModelViewController.mp3

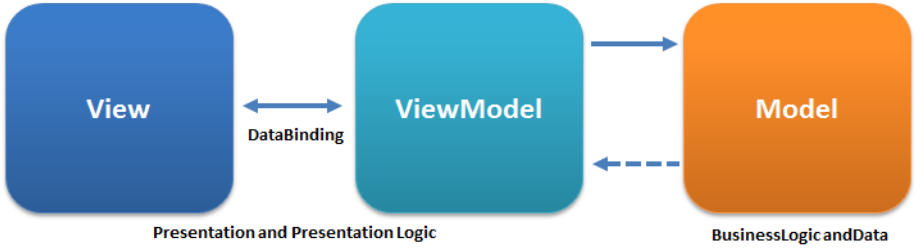
MVC Comes in Different Flavors - II

- MVC

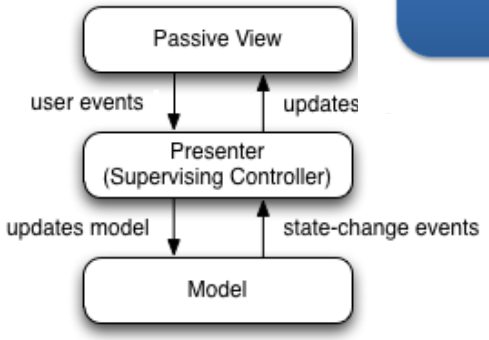




- MVVM

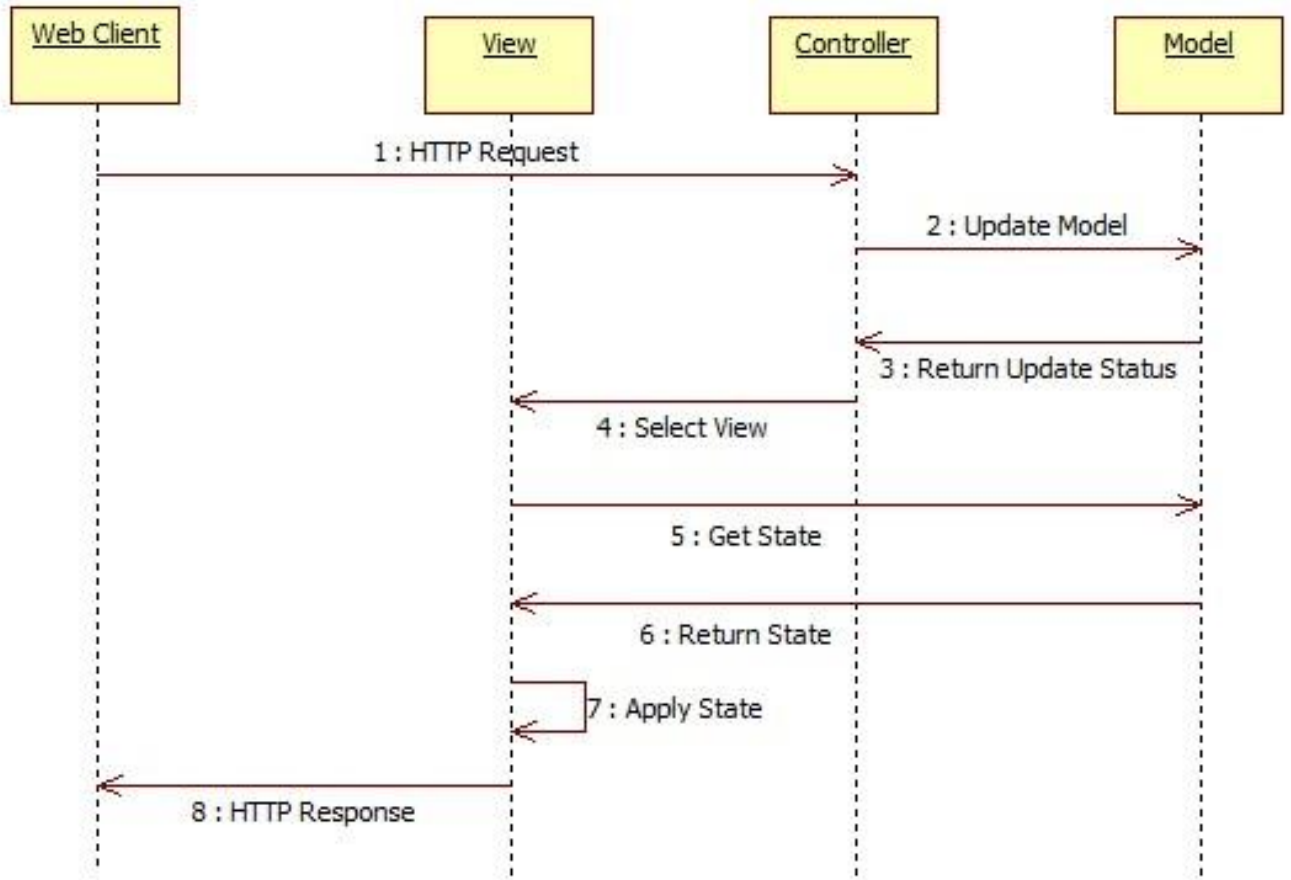


- MVP



Sources:https://en.wikipedia.org/wiki/Model_View_ViewModel#/media/File:MVVMPattern.png,
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter#/media/File:Model_View_Presenter_GUI_Design_Pattern.png
License: CC BY-SA 3.0, Authors:Ugaya40, Daniel Cardenas

Web MVC Interactions Sequence Diagram



SOLID Design Principles of OOP

- Single responsibility principle - a class should only have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the class.
- Open–closed principle - software entities should be open for extension, but closed for modification.
- Liskov substitution principle - Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
- Interface segregation principle - Many client-specific interfaces are better than one general-purpose interface.
- Dependency inversion principle - depend upon abstractions, not concretions.

Domain Driven Design (DDD)

We need tools to cope with all that complexity inherent in robotics and IoT domains.

Simple solutions are needed – cope with problems through divide and concur on different levels of abstraction:

Domain Driven Design (DDD) – back to basics:
domain objects, data and logic.

Described by Eric Evans in his book:
Domain Driven Design: Tackling Complexity in the Heart of Software, 2004

Domain Driven Design (DDD)

Main concepts:

- ❖ Entities, value objects and modules
- ❖ Aggregates and Aggregate Roots [Haywood]:

value < entity < aggregate < module < BC

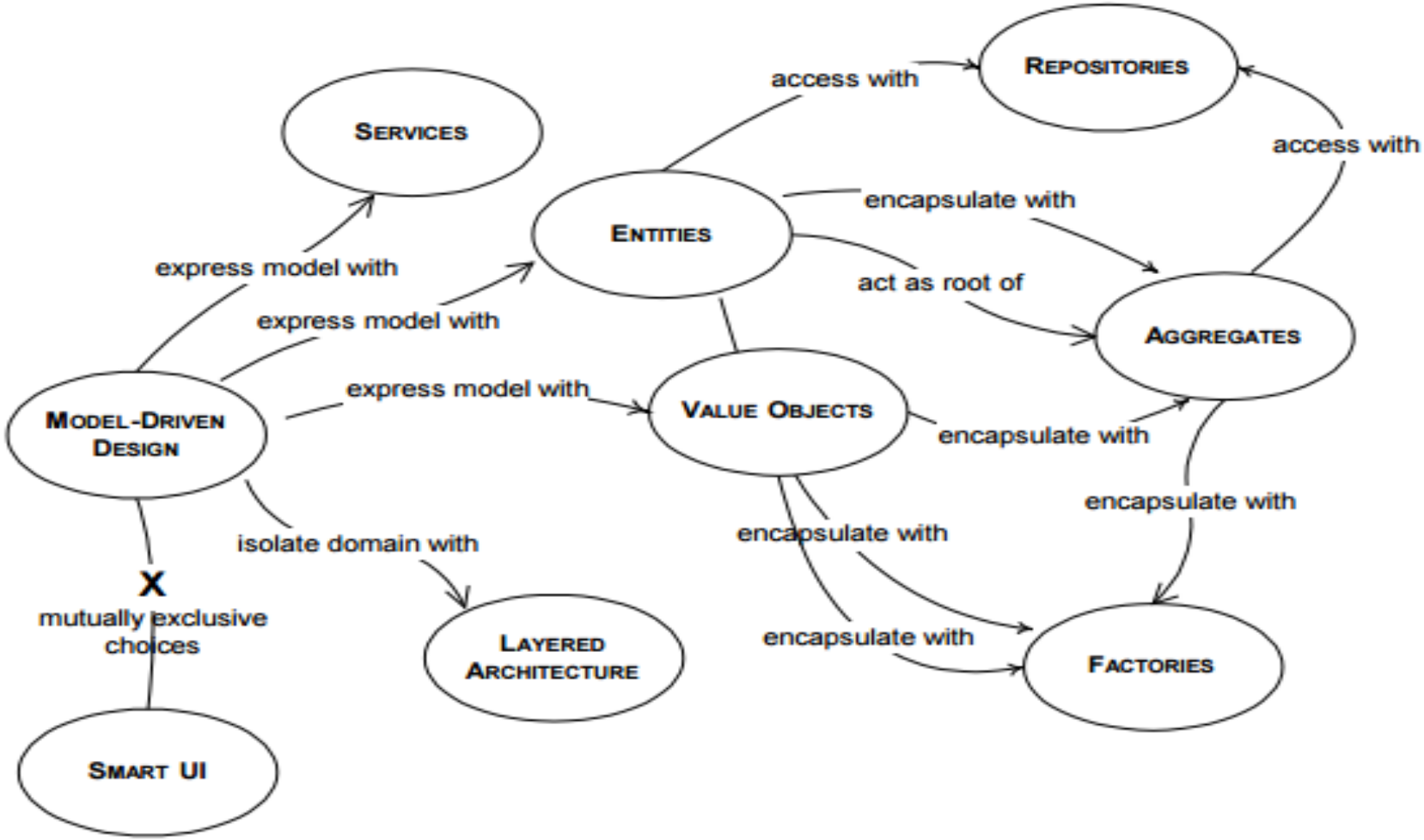
- ❖ Repositories, Factories and Services:

application services <-> **domain services**

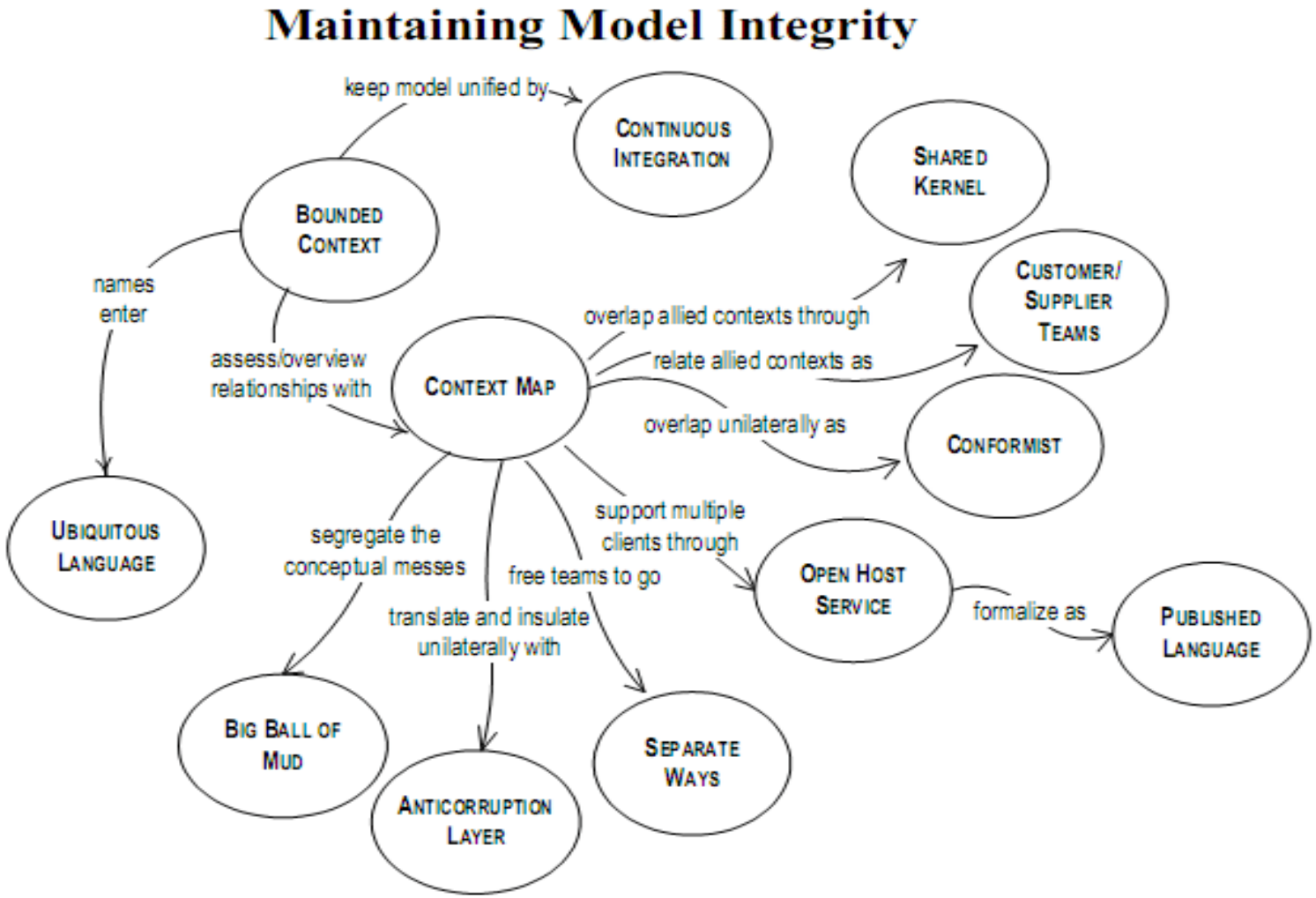
- ❖ Separating interface from implementation

9

Domain Driven Design (DDD)



Domain Driven Design (DDD)



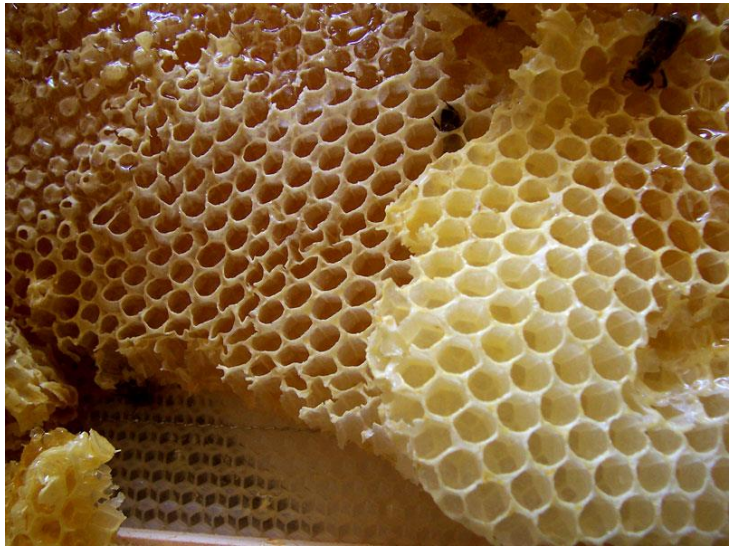
Domain Driven Design (DDD)

- Ubiquitous language and Bounded Contexts
- DDD Application Layers:
- Infrastructure, Domain, Application, Presentation
- Hexagonal architecture :

OUTSIDE <-> transformer <->

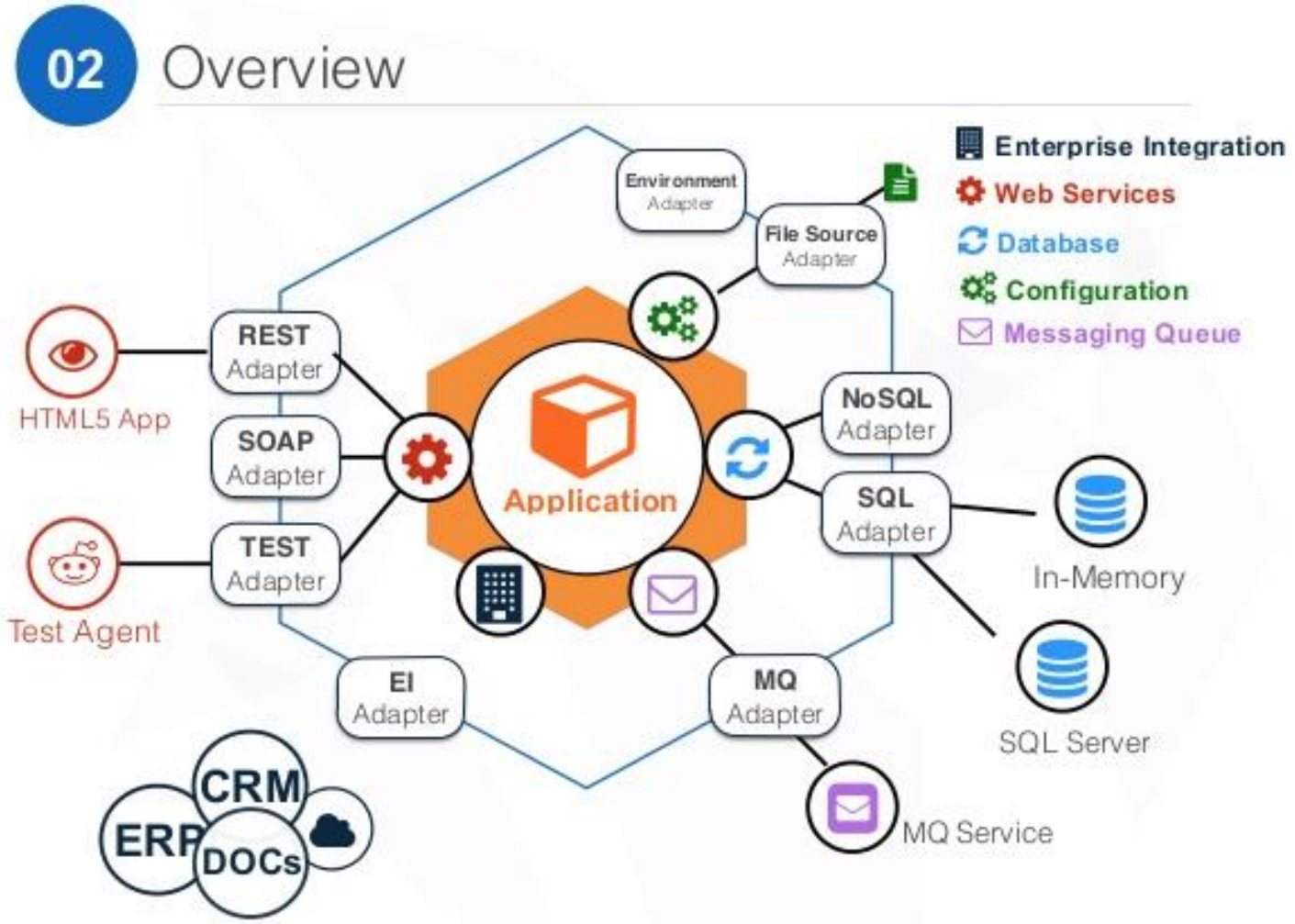
(application <-> domain)

[A. Cockburn]



12

Hexagonal Architecture



Source: https://www.slideshare.net/fabricioepa/hexagonal-architecture-for-java-applications

13

Hexagonal Architecture Principles

- Allows an application to equally be driven by users, programs, automated test or batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases.
- As events arrive from the outside world at a port, a technology-specific adapter converts it into a procedure call or message and passes it to the application
- Application sends messages through ports to adapters, which signal data to the receiver (human or automated)
- The application has a semantically sound interaction with all the adapters, without actually knowing the nature of the things on the other side of the adapters

Source: http://alistair.cockburn.us/Hexagonal+architecture

Tkinter Concepts

- widgets - a Tkinter user interface is made up of individual widgets. Each widget is represented as a Python object, instantiated from classes like ttk.Frame, ttk.Label, and ttk.Button.
- widget hierarchy - widgets are arranged in a hierarchy. The label and button were contained within a frame, which in turn was contained within the root window. When creating each child widget, its parent widget is passed as the first argument to the widget constructor.
- configuration options - widgets have configuration options, which modify their appearance and behavior, such as the text to display in a label or button. Different classes of widgets will have different sets of options.
- geometry management - widgets aren't automatically added to the user interface when they are created. A geometry manager like grid controls where in the user interface they are placed.
- event loop - Tkinter reacts to user input, changes from your program, and even refreshes the display only when actively running an event loop. If your program isn't running the event loop, your user interface won't update.

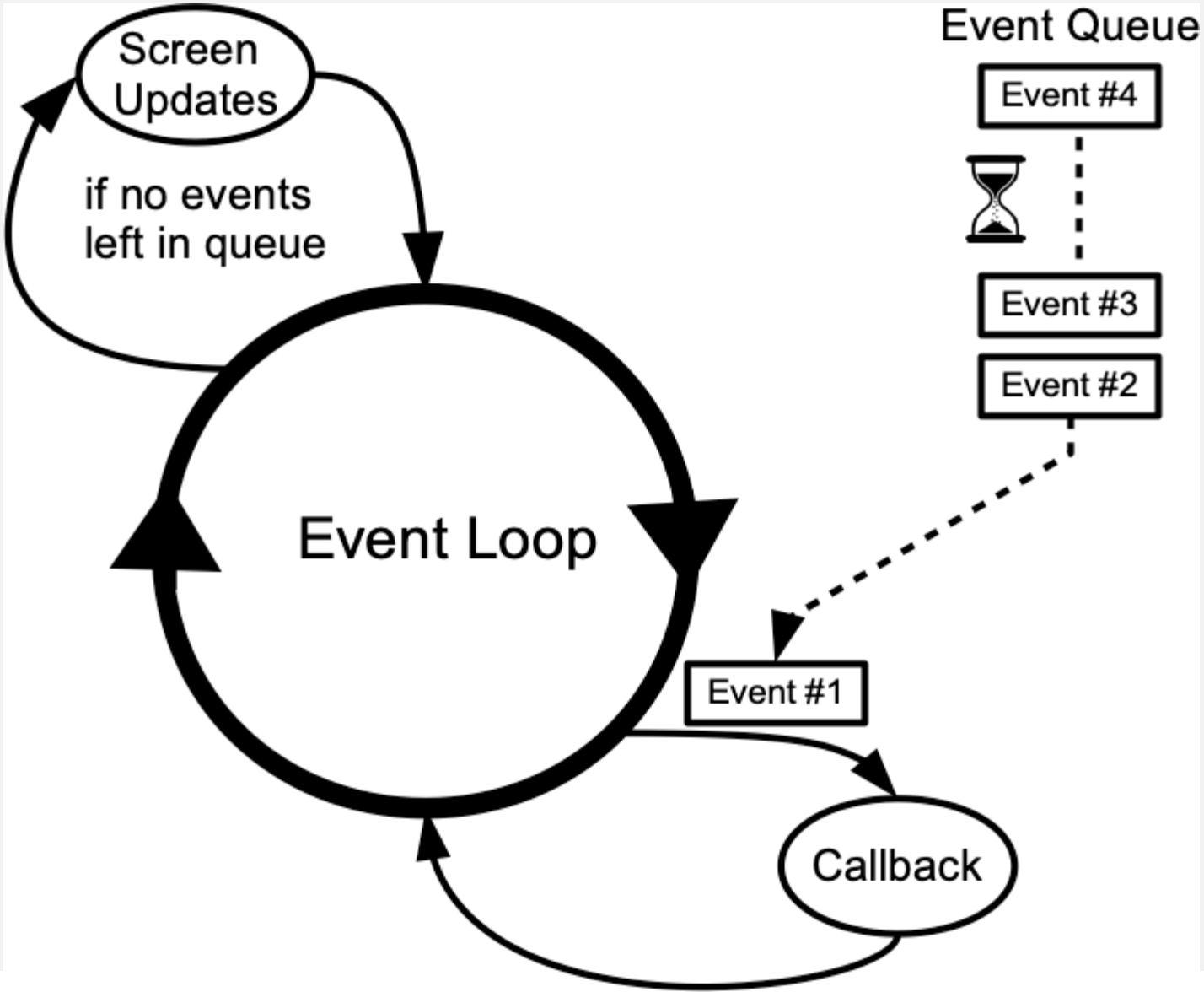
Tkinter Minimal Application

```
import Tkinter as tk

class Application(tk.Frame):
    def __init__(self, master=None):
        tk.Frame.__init__(self, master)
    self.grid()
    self.createWidgets()

    def createWidgets(self):
        self.quitButton = tk.Button(self, text='Quit', command=self.quit)
        self.quitButton.grid()


app = Application()
app.mainloop()
```

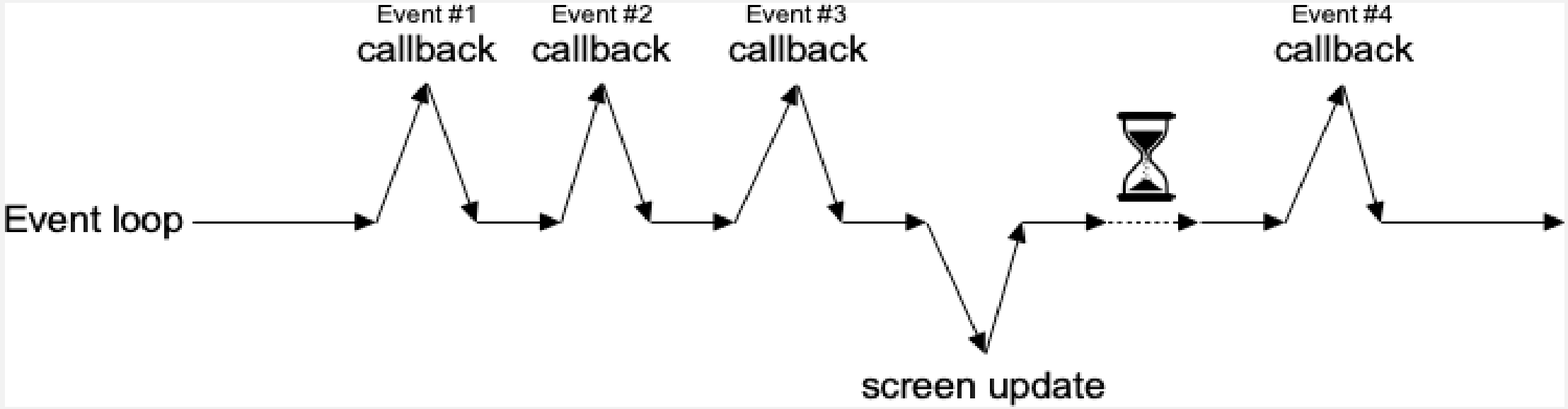
Tkinter Widget Main Concepts

- Window - has different meanings in different contexts, but in general it refers to a rectangular area somewhere on your display screen.
- Top-level window - a window that exists independently on your screen. It will be decorated with the standard frame and controls for your system's desktop manager. You can move it around on your desktop. You can generally resize it, although your application can prevent this.
- Widget – a generic term for any of the building blocks that make up an application in a graphical user interface. Examples: buttons, radiobuttons, text fields, frames, and text labels.
- Frame - the basic unit of organization for complex layouts. A frame is a rectangular area that can contain other widgets.
- Child, parent - when any widget is created, a parent-child relationship is created. For example, if you place a text label inside a frame, the frame is the parent of the label.

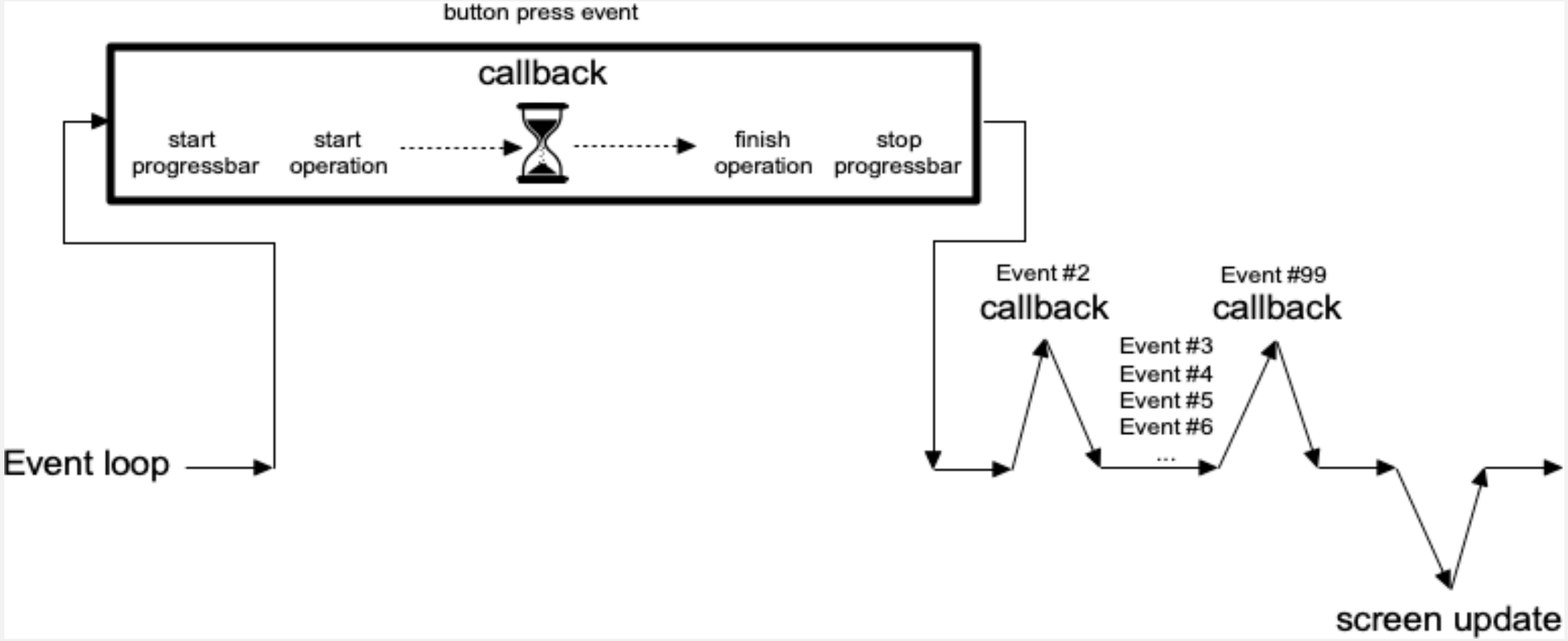
Files and Programs



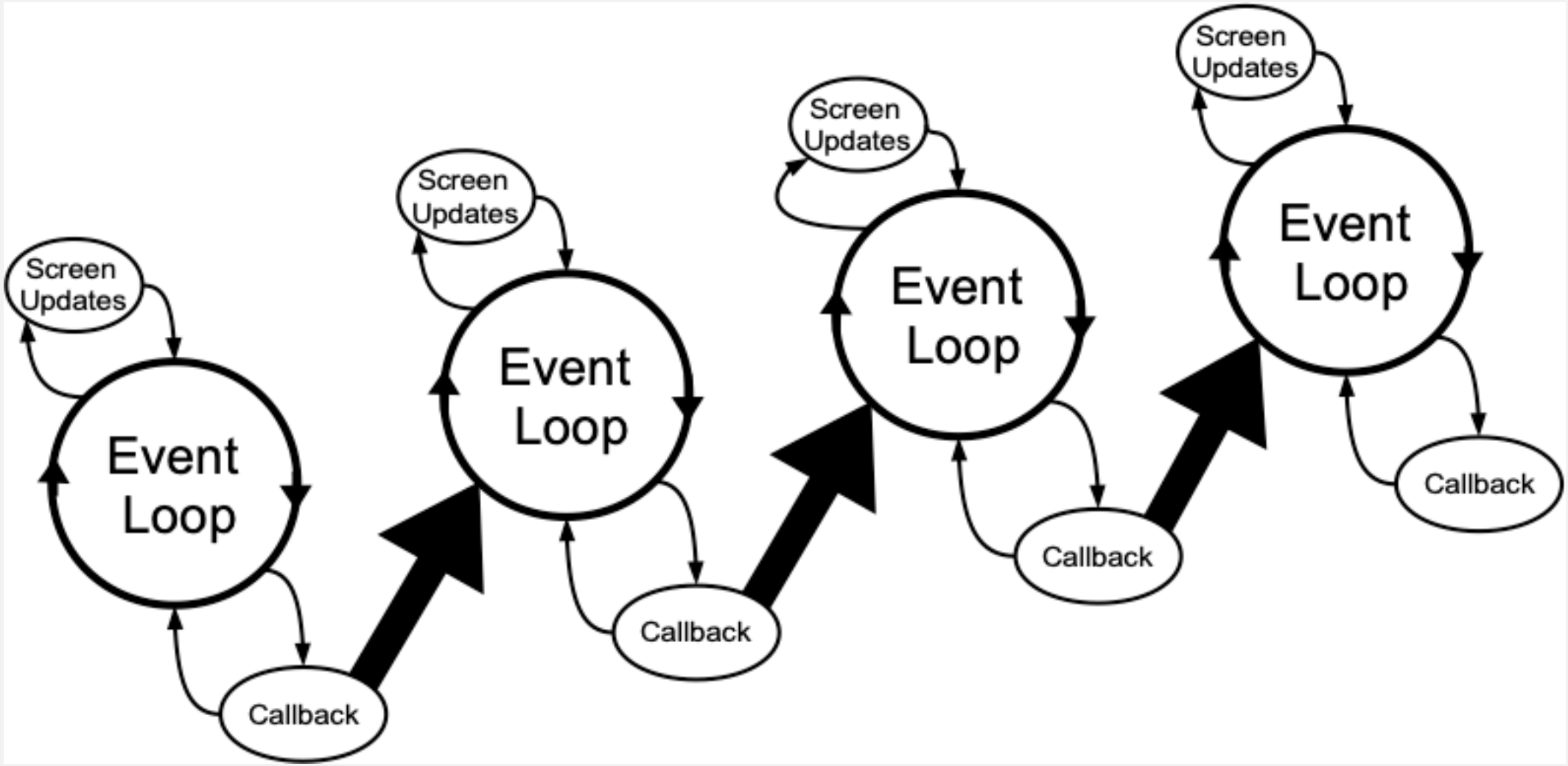
Files and Programs



Files and Programs



Files and Programs



Grid Geometry (Layout) Manager

grid() geometry manager treats every window or frame as a table - a gridwork of rows and columns.

- A cell is the area at the intersection of one row and one column.
- The width of each column is the width of the widest cell in that column.
- The height of each row is the height of the largest cell in that row.
- For widgets that do not fill the entire cell, you can specify what happens to the extra space. You can either leave the extra space outside the widget, or stretch the widget to fit it, in either the horizontal or vertical dimension.
- You can combine multiple cells into one larger area, a process called spanning.

Grid Manager Main Options: *w.grid(option=value, ...)*

column - the column number where you want the widget gridded, counting from zero, default is 0.

columnspan - you can grab multiple cells of a row and merge them into one larger cell by setting the columnspan option to the number of cells. For example, w.grid(row=0, column=2, columnspan=3) would place widget w in a cell that spans columns 2, 3, and 4 of row 0.

in_ - registers w as a child of some widget w2, use in_=w2. The new parent w2 must be a descendant of the parent widget used when w was created.

ipadx - internal x padding. This dimension is added inside the widget inside its left and right sides.

ipady - internal y padding. This dimension is added inside the widget inside its top and bottom borders.

padx - external x padding. This dimension is added to the left and right outside the widget.

pady - external y padding. This dimension is added above and below the widget.

row – the row number into which you want to insert the widget, counting from 0. The default is the next higher-numbered unoccupied row.

rowspan - can grab multiple adjacent cells of a column, however, by setting the rowspan option to the number of cells to grab.

sticky - determines how to distribute any extra space within the cell that is not taken up by the widget at its natural size.

Main Widgets

- Frame
- Text
- Scrollbar
- ScrolledText
- Separator
- Checkbox
- Radio Button
- Combobox
- Listbox
- Slider
- Spinbox
- Sizegrip
- LabelFrame
- Progressbar
- Notebook
- Treeview
- Canvas
- Toplevel

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD