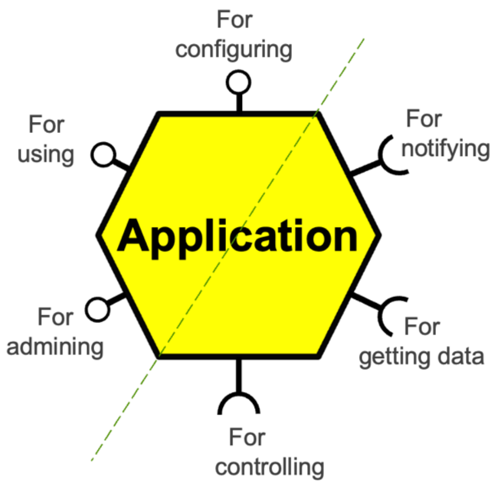
The figures from the Preview Edition of

Hexagonal Architecture Explained

How the Ports & Adapters architecture simplifies your life, and how to implement it



Alistair Cockburn

Juan Manuel Garrido de Paz

ISBN 978-1-7375197-8-2
Humans and Technology Press
32 W 200 S #504
Salt Lake City, UT 84101
V0.9a 20240507-1836 for paper books

Acknowledgements

From Alistair

I am immensely grateful to Juan Manuel Garrido de Paz, without whom this book could never have been written. Of all the people I have conversed with, Juan had the sharpest, deepest, most accurate understanding of the pattern. He saw its relationship to UML components and the *required* interface years before I did.

He was relentless in his quest to understand and describe the pattern. He provided code for me to study and include. We argued incessantly, but only ever in pursuit of the truth. Once we found it, we were once again in complete agreement.

Juan was also a relentless fan of FC Huelva:

Juan at Huelva, 2024

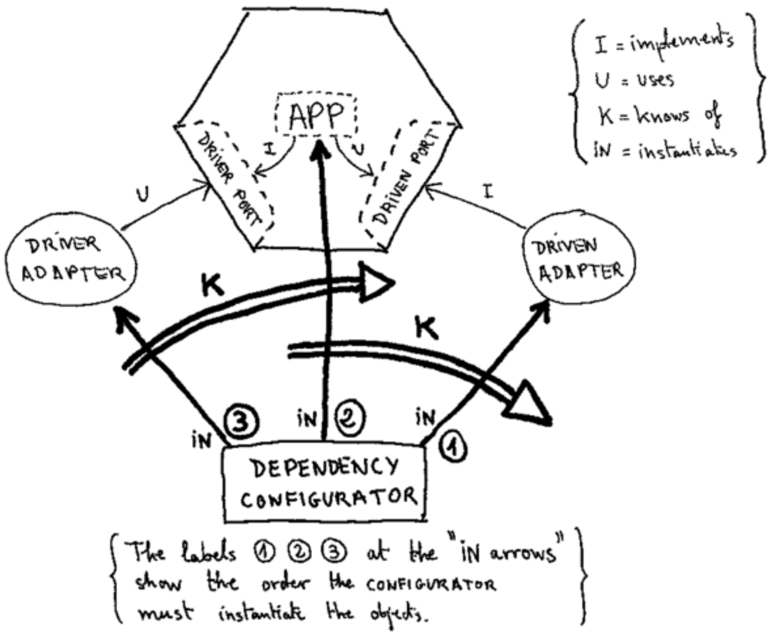


Figure 2.1. The configurator introduces the actors.

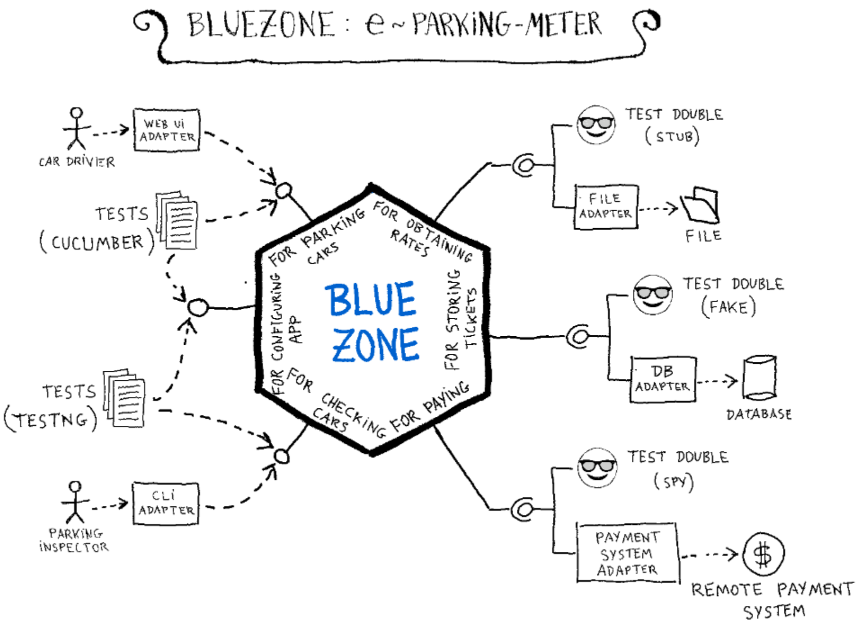


Figure 3.1. The actors in the BlueZone example

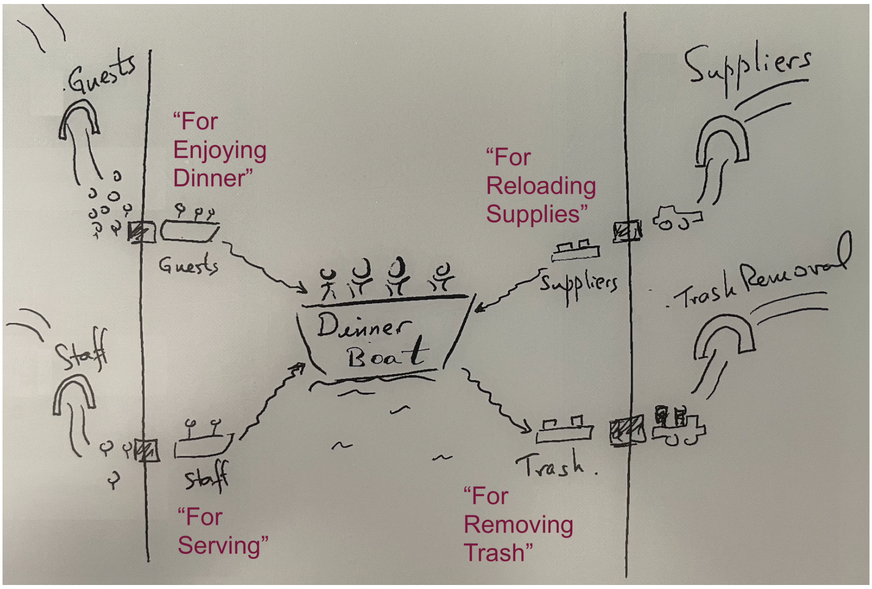


Figure 4.1. The floating restaurant analogy

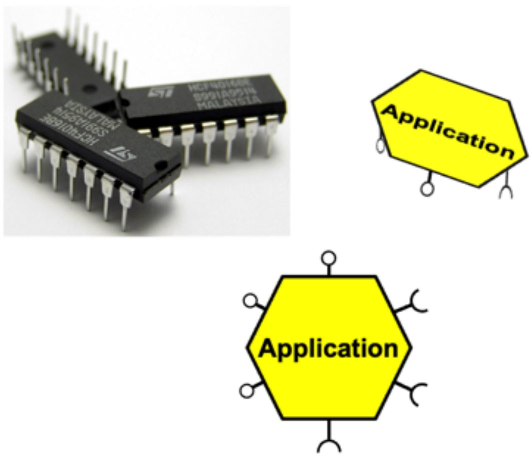


Figure 4.2. The hardware chip analogy

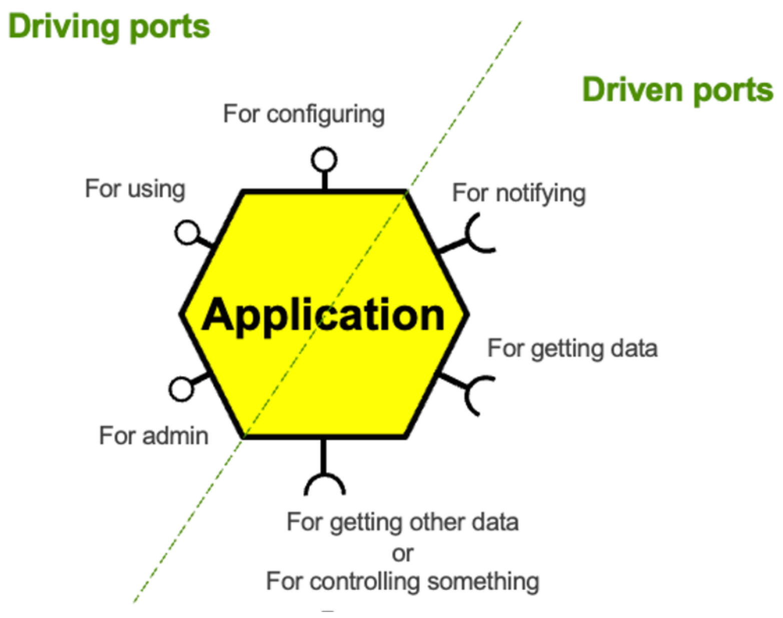


Figure 4.3. Ports are like input and output pins on the chip

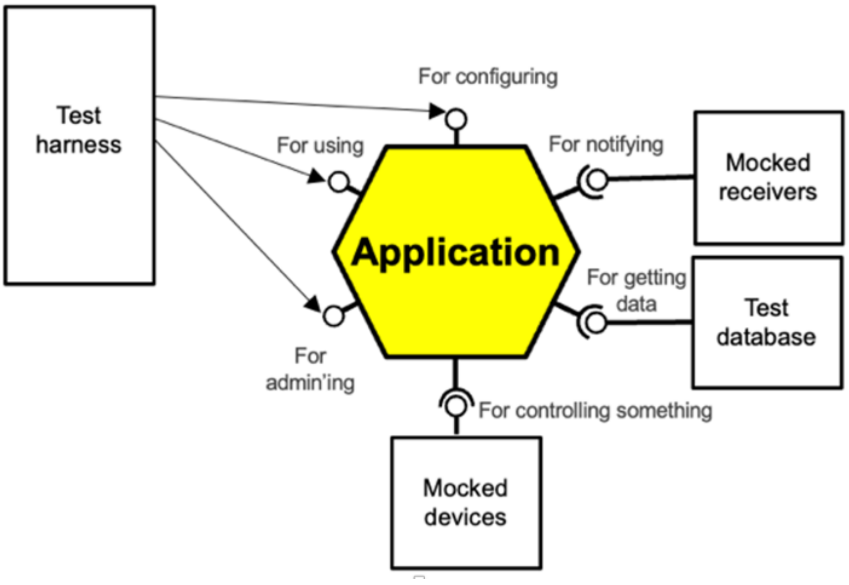


Figure 4.4. Hooking up the connections for testing

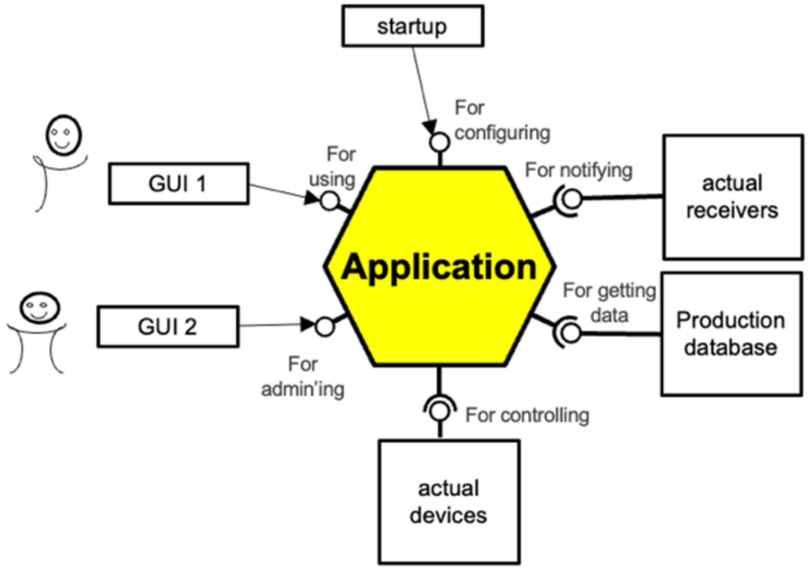
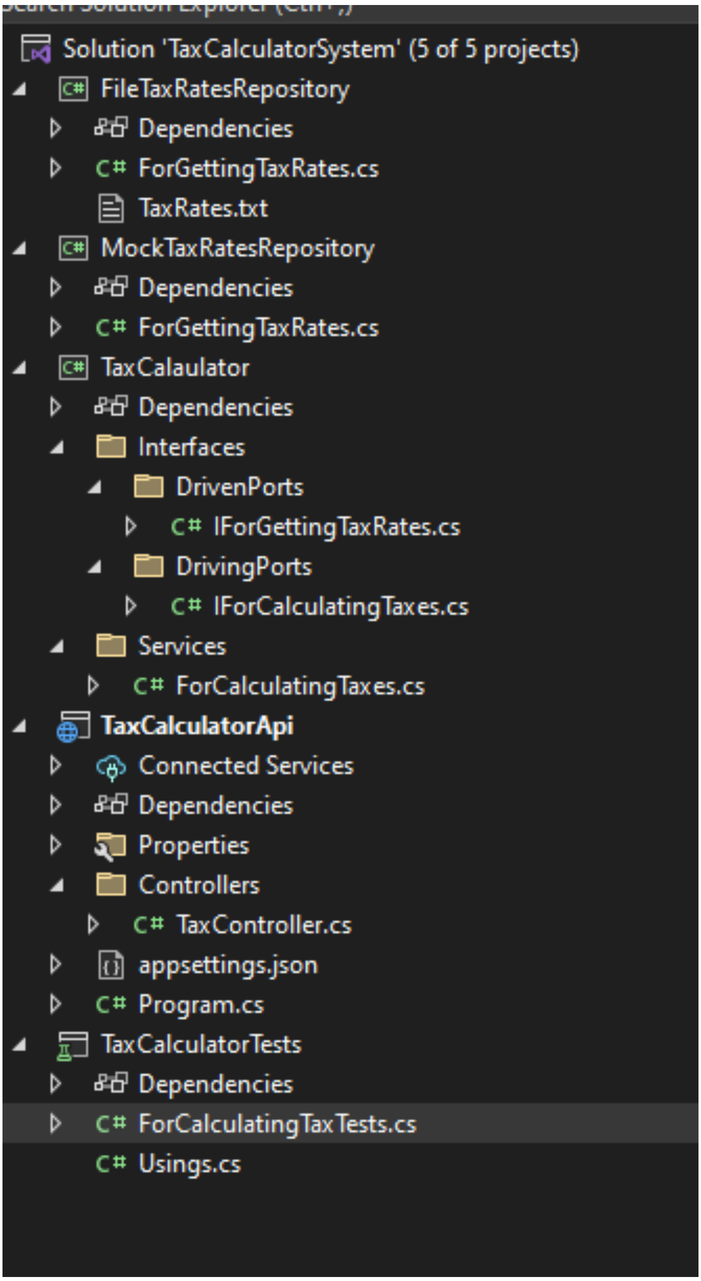


Figure 4.5. Hooking up the connections for production



```
Solution 'TaxCalculatorSystem' (5 of 5 projects)
◢ FileTaxRatesRepository
    ▷ Dependencies
    ▷ ForGettingTaxRates.cs
      TaxRates.txt
◢ MockTaxRatesRepository
    ▷ Dependencies
    ▷ ForGettingTaxRates.cs
◢ TaxCalaulator
    ▷ Dependencies
    ◢ Interfaces
        ◢ DrivenPorts
            ▷ IForGettingTaxRates.cs
        ◢ DrivingPorts
            ▷ IForCalculatingTaxes.cs
    ◢ Services
        ▷ ForCalculatingTaxes.cs
◢ TaxCalculatorApi
    ▷ Connected Services
    ▷ Dependencies
    ▷ Properties
    ◢ Controllers
        ▷ TaxController.cs
    ▷ appsettings.json
    ▷ Program.cs
◢ TaxCalculatorTests
    ▷ Dependencies
    ▷ ForCalculatingTaxTests.cs
      Usings.cs
```

Figure 4.6. The suggested folder structure.

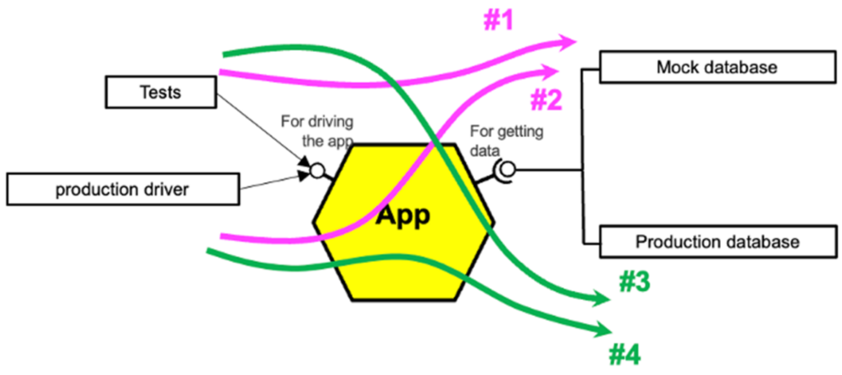


Figure 4-7. The development sequence: Tests and mocks first

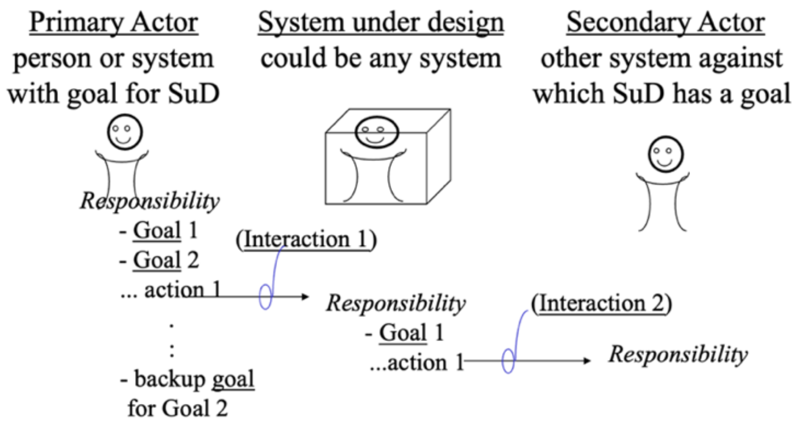


Figure 5.1. Primary and secondary actors and their goals

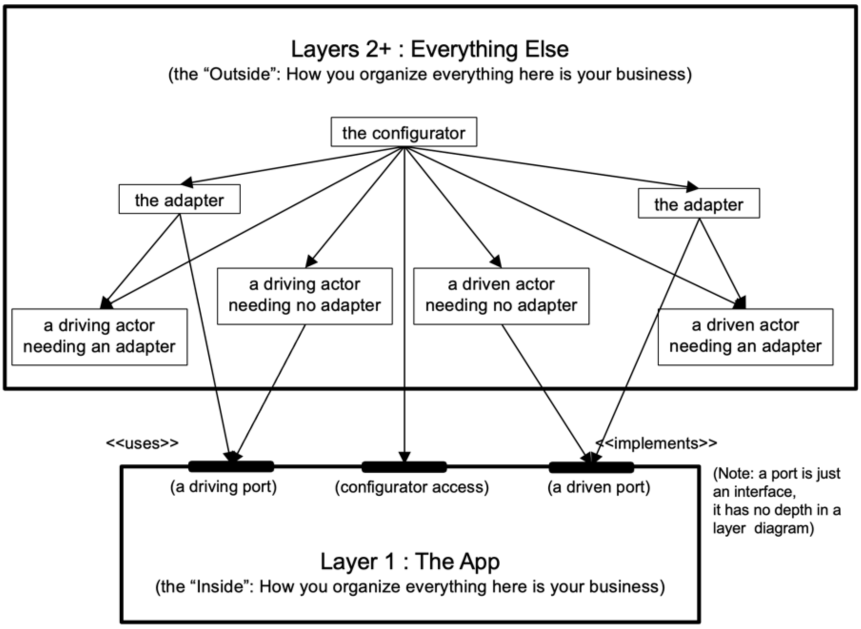

Layers 2+ : Everything Else
(the "Outside": How you organize everything here is your business)
the configurator
the adapter
the adapter
a driving actor needing an adapter
a driving actor needing no adapter
a driven actor needing no adapter
a driven actor needing an adapter
<<uses>>
<<implements>>
(a driving port)
(configurator access)
(a driven port)
(Note: a port is just an interface, it has no depth in a layer diagram)
Layer 1 : The App
(the "Inside": How you organize everything here is your business)

Figure 5.2. Ports & Adapters only specifies two layers, inside and outside.

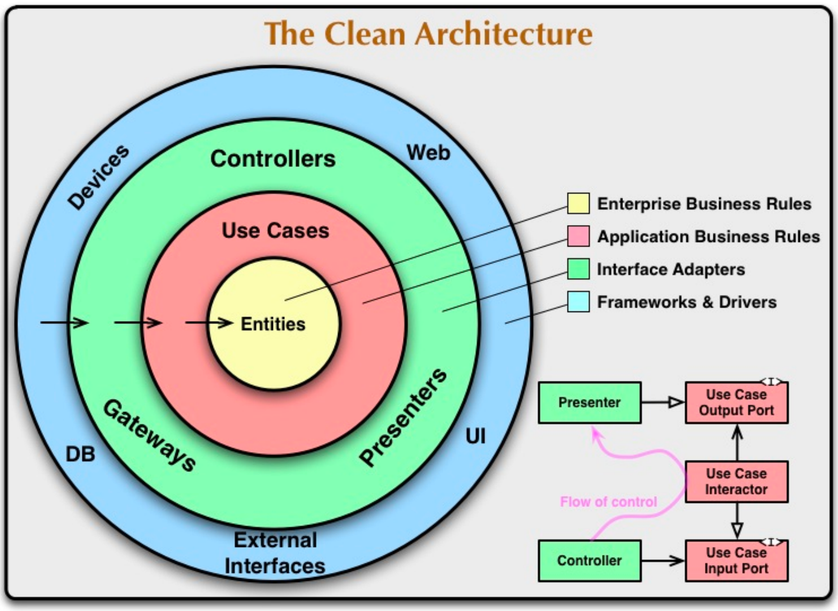


Figure 5.3. Clean architecture
https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

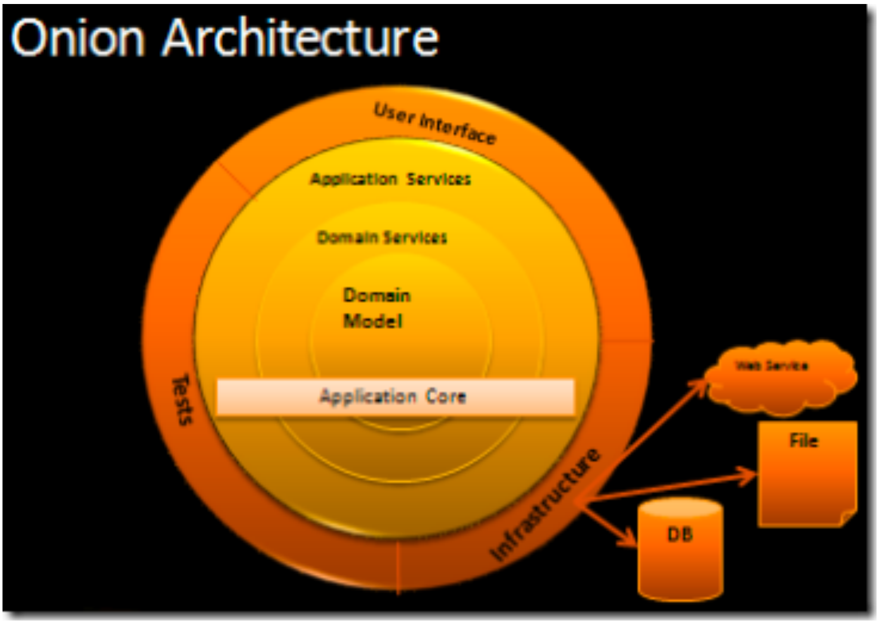
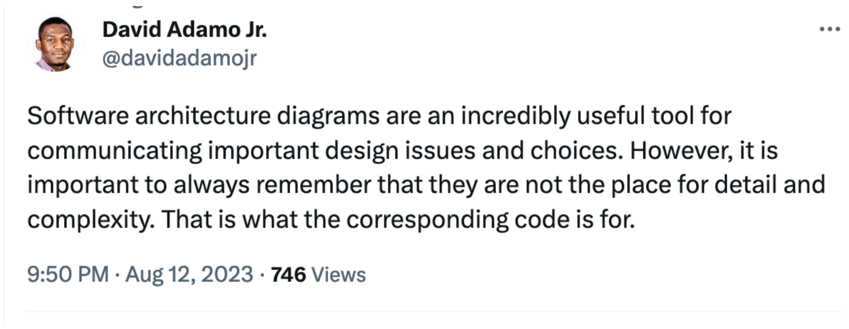


Figure 5.4. Onion architecture
https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/


David Adamo Jr.
@davidadamojr

Software architecture diagrams are an incredibly useful tool for communicating important design issues and choices. However, it is important to always remember that they are not the place for detail and complexity. That is what the corresponding code is for.

9:50 PM · Aug 12, 2023 · **746** Views

Figure 5.5. Architecture drawings are not code: https://twitter.com/davidadamojr/status/1690541235918753792

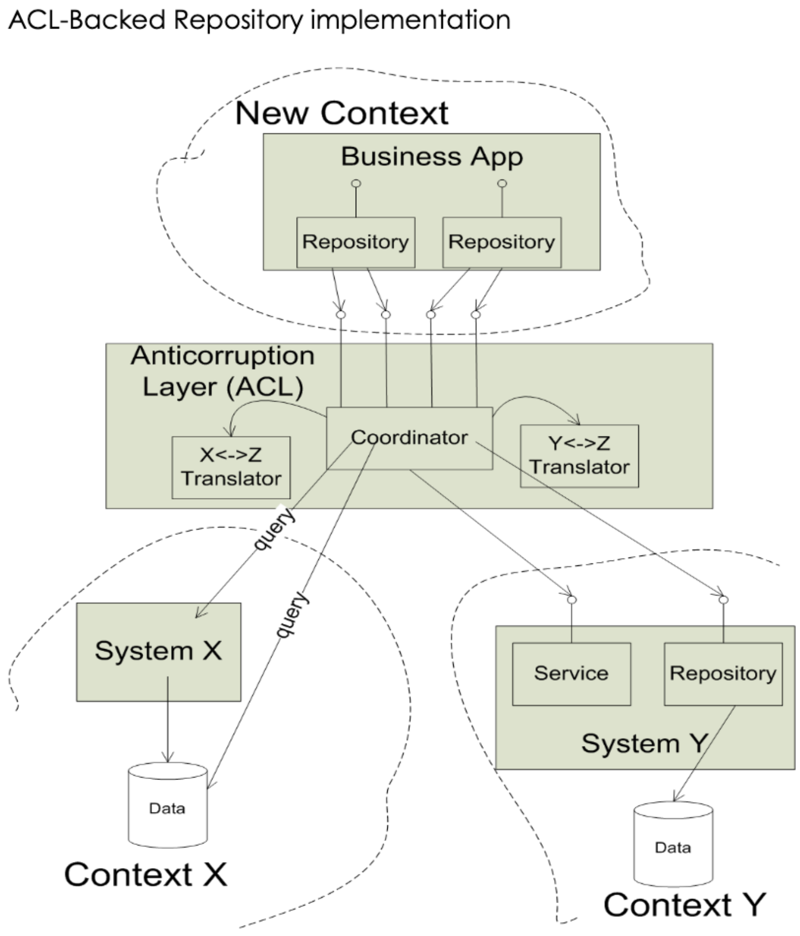


Figure 5.6. Example of an ACL with several responsibilities (Evans, E, 2013)

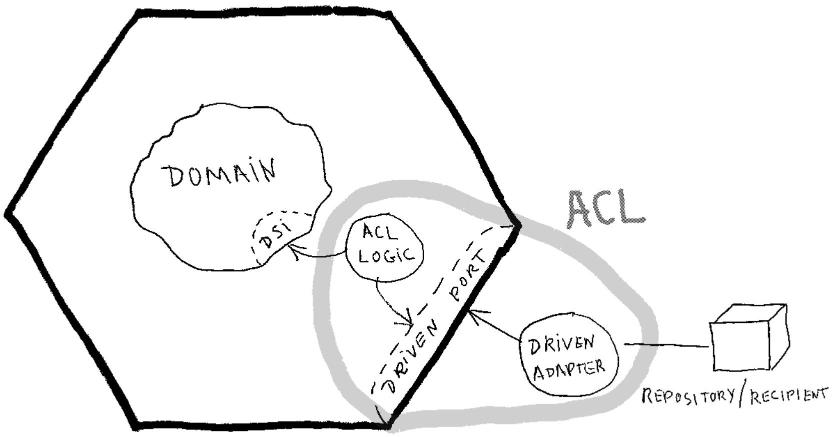


Figure 5.7: Anti-corruption layer blending over the hexagon boundary

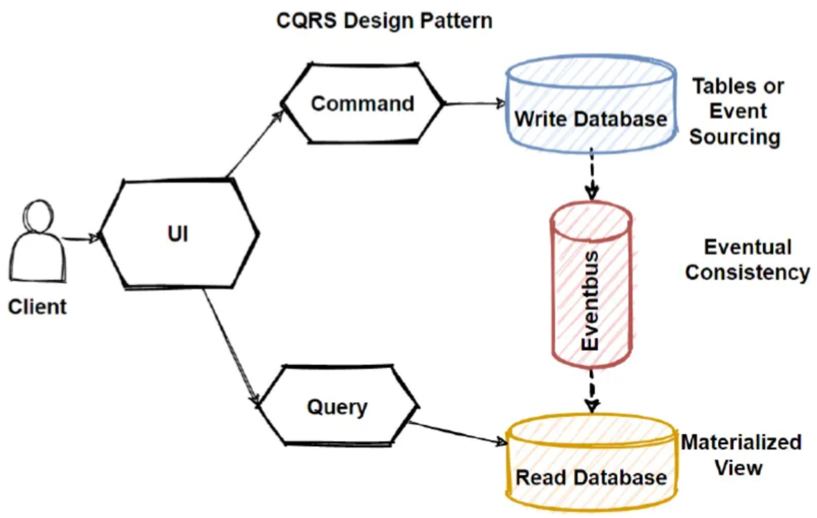


Figure 5.8. The CQRS architecture, courtesy of Mehmet Ozkaya [https://medium.com/design-microservices-architecture-with-patterns/cqrs-design-pattern-in-microservices-architectures-5d41e359768c}

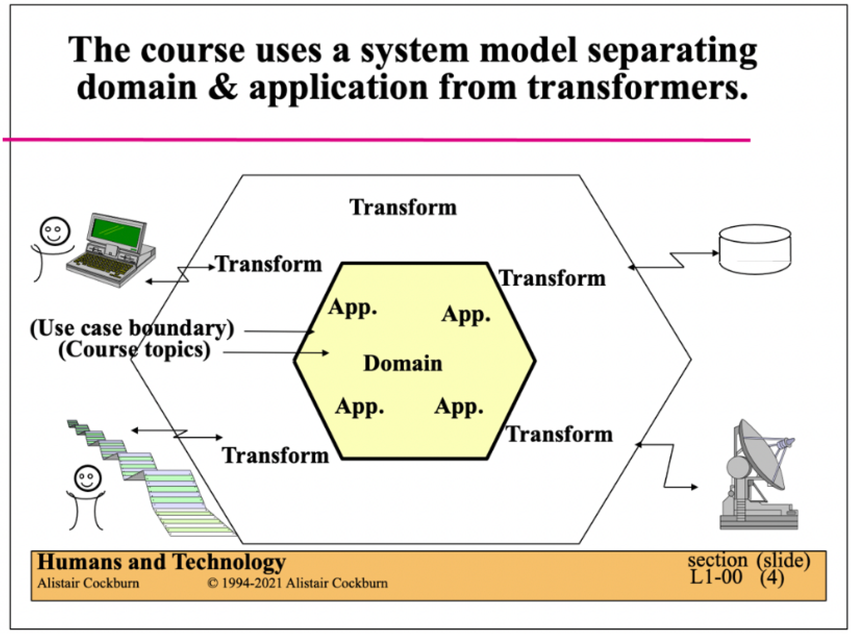


Figure 6.1. The earliest hexagonal picture, from 1994

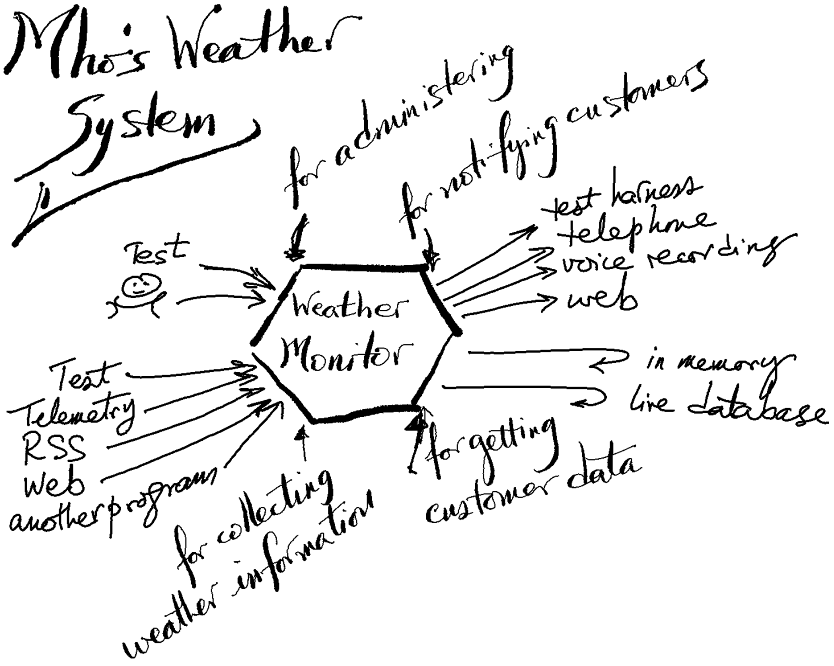


Figure 6.2. Mho's weather system in the hexagon

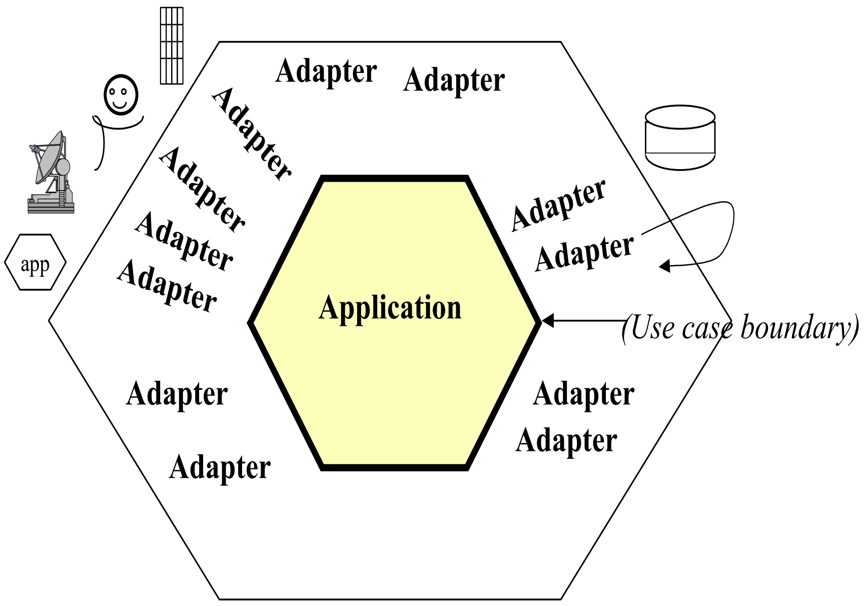

Adapter
Adapter
Adapter
Adapter
Adapter
Adapter
Adapter
Adapter
Adapter
Adapter
Adapter
Adapter
Application
app
(Use case boundary)

Figure 6.3

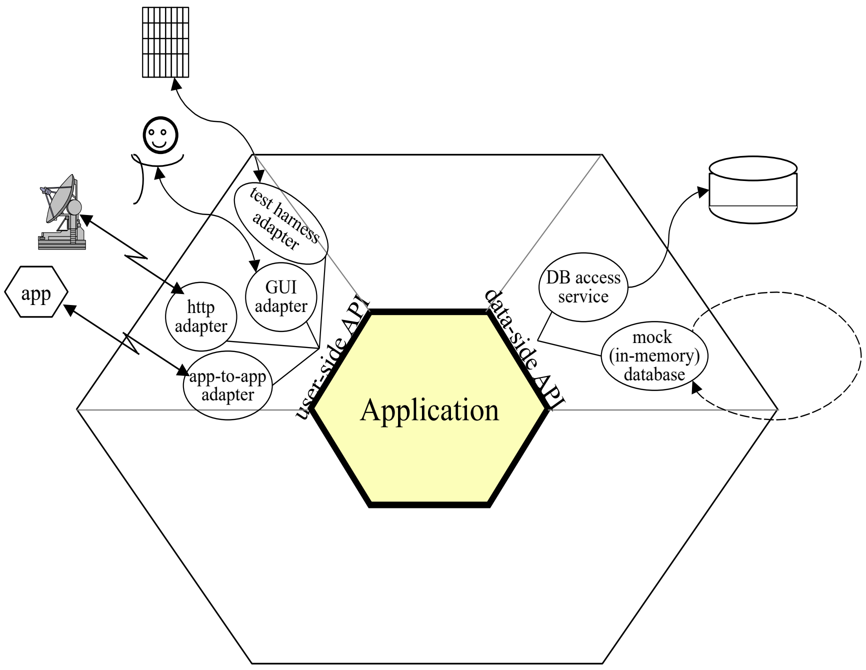

test harness adapter
GUI adapter
http adapter
app-to-app adapter
app
user-side API
Application
data-side API
DB access service
mock (in-memory) database

Figure 6.4

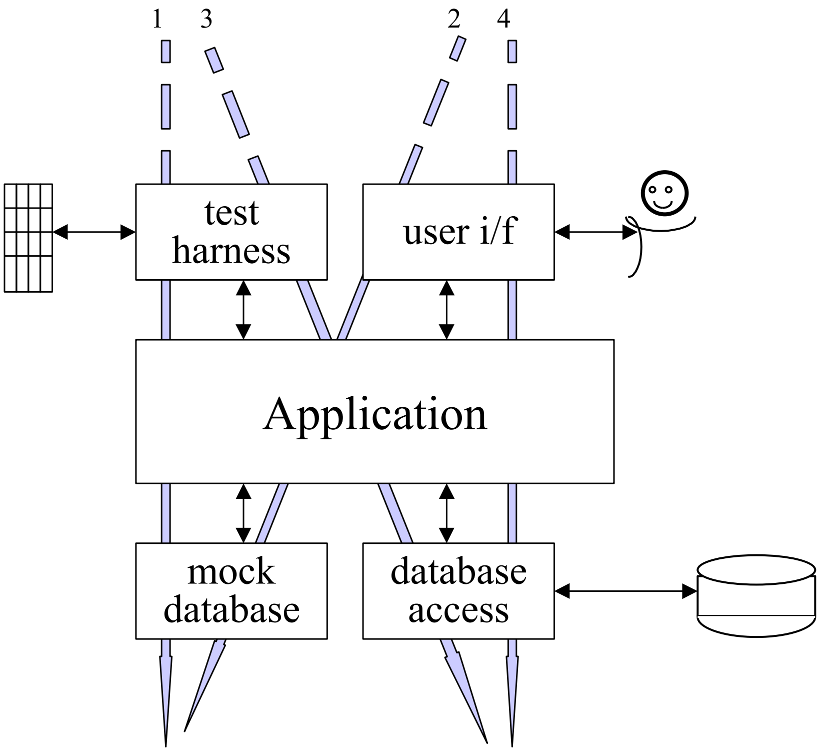

1
3
2
4
test harness
user i/f
Application
mock database
database access

Figure 6.5

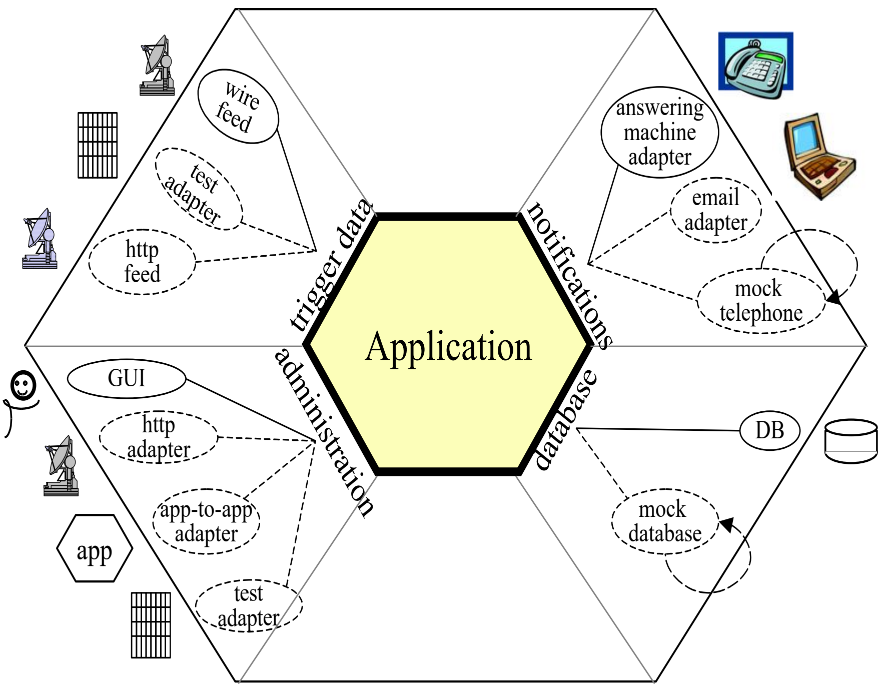

wire feed
test adapter
http feed
trigger data
answering machine adapter
email adapter
mock telephone
notifications
Application
GUI
http adapter
app-to-app adapter
test adapter
administration
app
database
DB
mock database

Figure 6.6

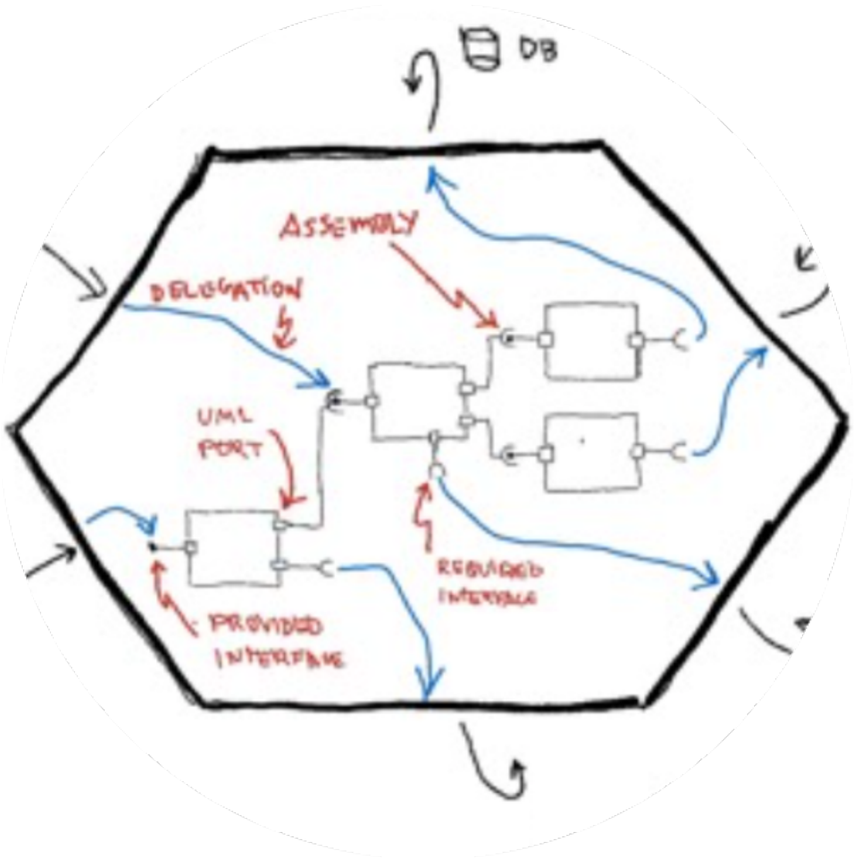

DB
ASSEMBLY
DELEGATION
UML PORT
REQUIRED INTERFACE
PROVIDED INTERFACE

Figure 6.7

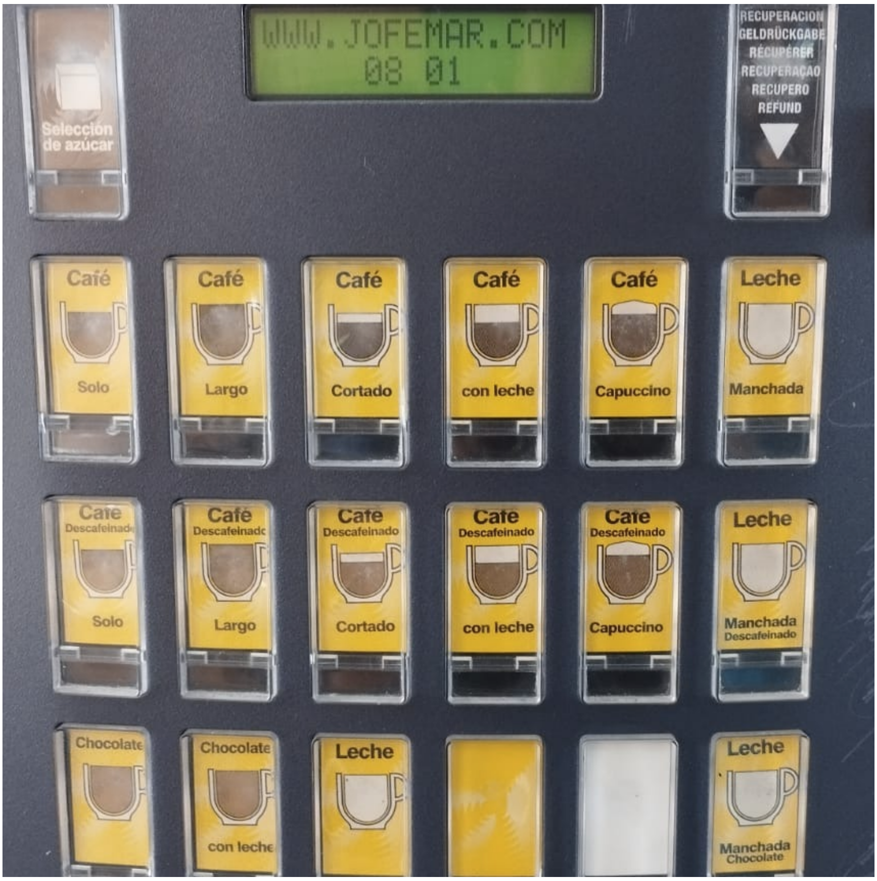


Figure 6.8: The inevitable coffee machine

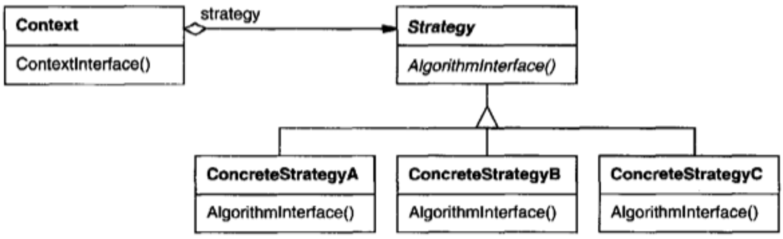


Figure 6.9: The *Strategy* pattern

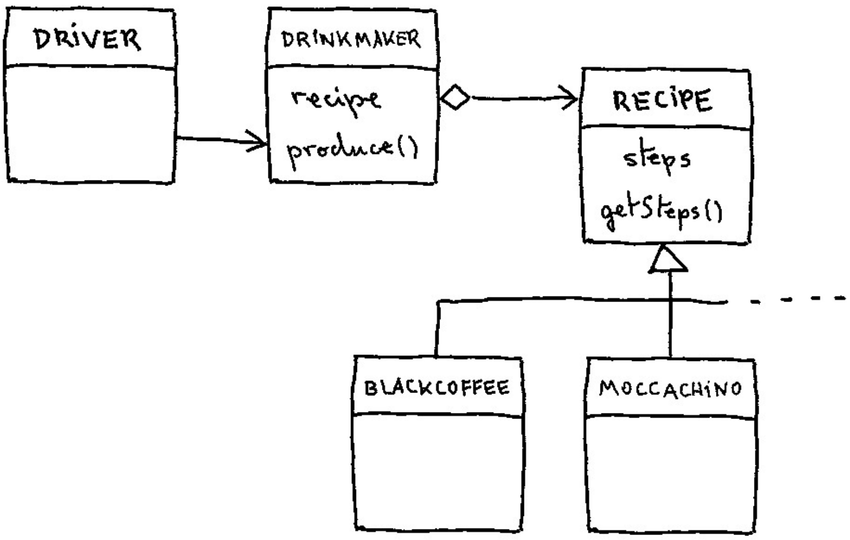


Figure 6.10: The drinkmaker example
(Image courtesy of Juan Manuel Garrido de Paz)

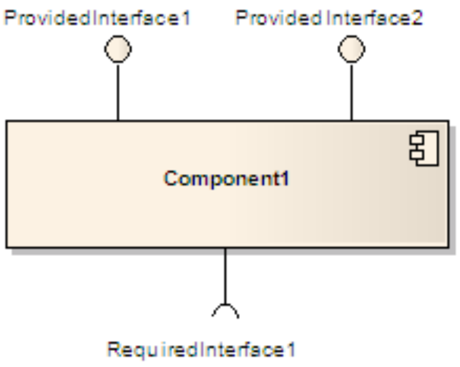


Figure 6.11: A UML Component with Provided and Required interfaces

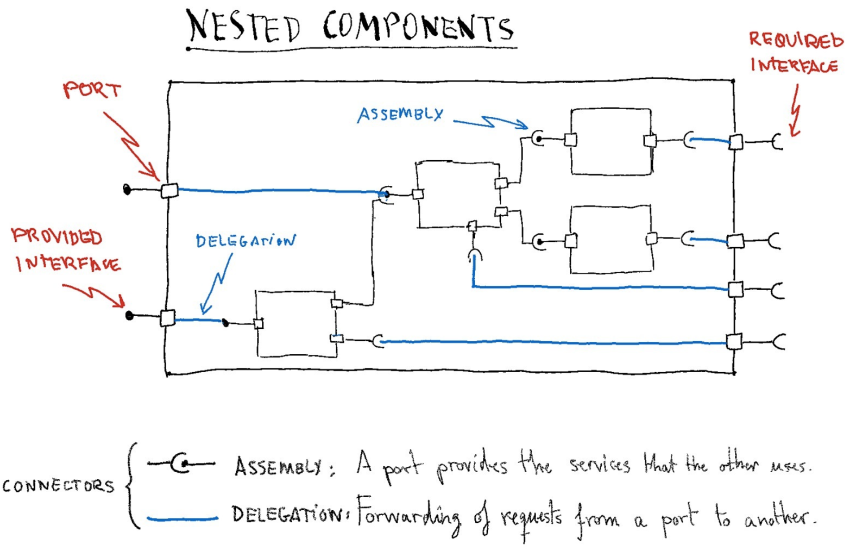

NESTED COMPONENTS
PORT
REQUIRED INTERFACE
ASSEMBLY
PROVIDED INTERFACE
DELEGATION
CONNECTORS
ASSEMBLY: A port provides the services that the other uses.
DELEGATION: Forwarding of requests from a port to another.

Figure 6.12: Components can be nested
(Image courtesy of Juan Manuel Garrido de Paz)

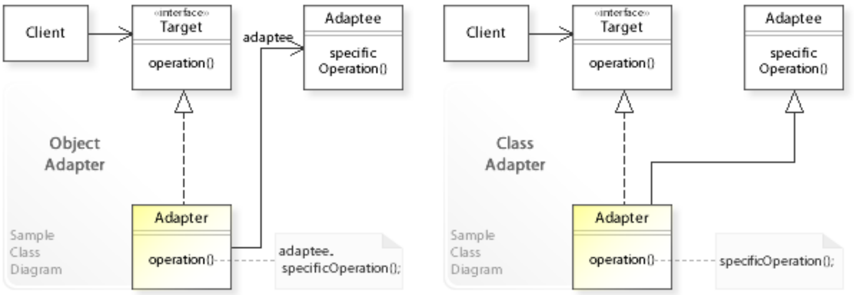


Figure 6.13: The *Adapter* pattern

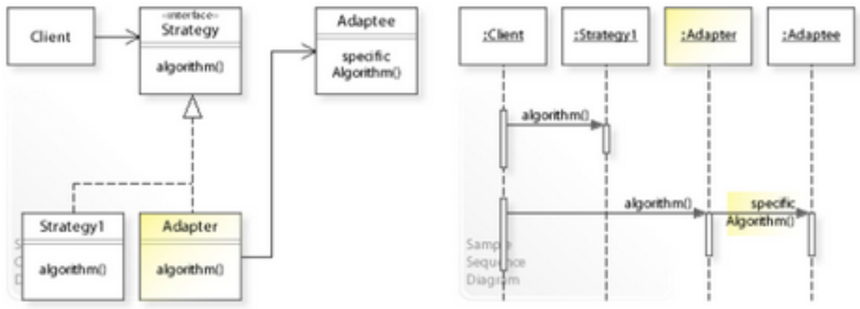


Figure 6.14: Using *Strategy* and *Adapter* together
http://www.w3sdesign.com/GoF_Design_Patterns_Reference0100.pdf

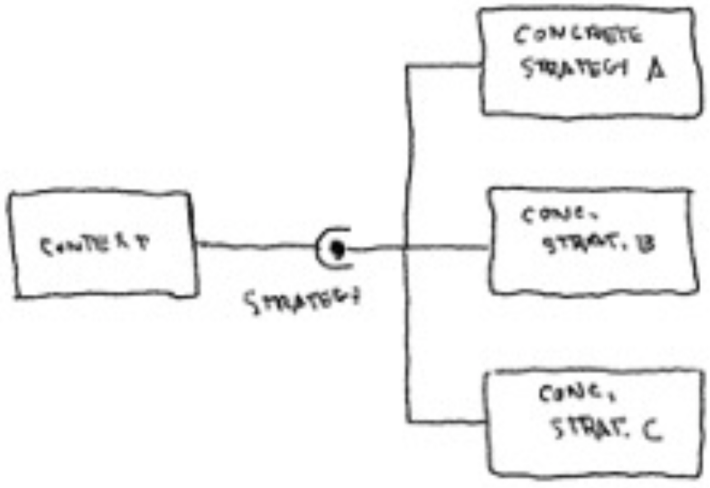


Figure 6.15: *Strategy* as a component diagram
(Image courtesy of Juan Manuel Garrido de Paz)

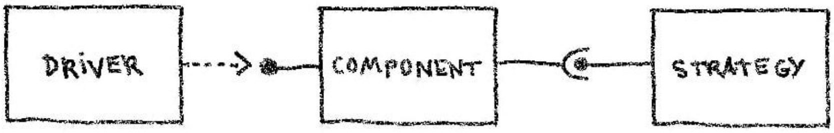


Figure 6.16: *Component + Strategy* as a component diagram
(Image courtesy of Juan Manuel Garrido de Paz)

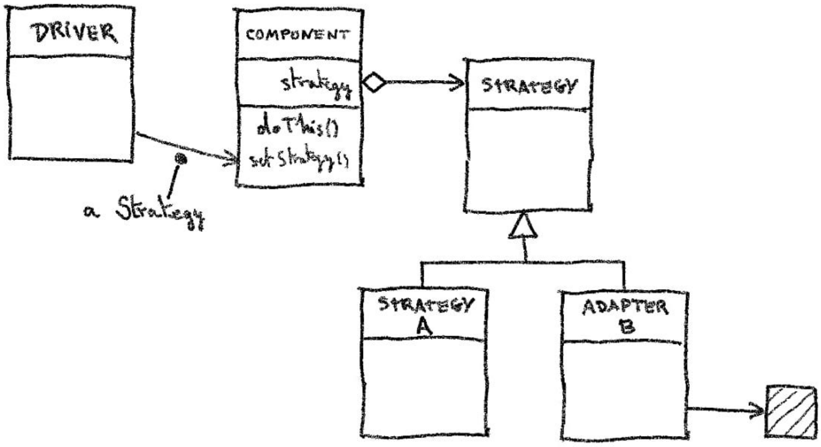


Figure 6.17: *Component + Strategy* as a class diagram
(Image courtesy of Juan Manuel Garrido de Paz)

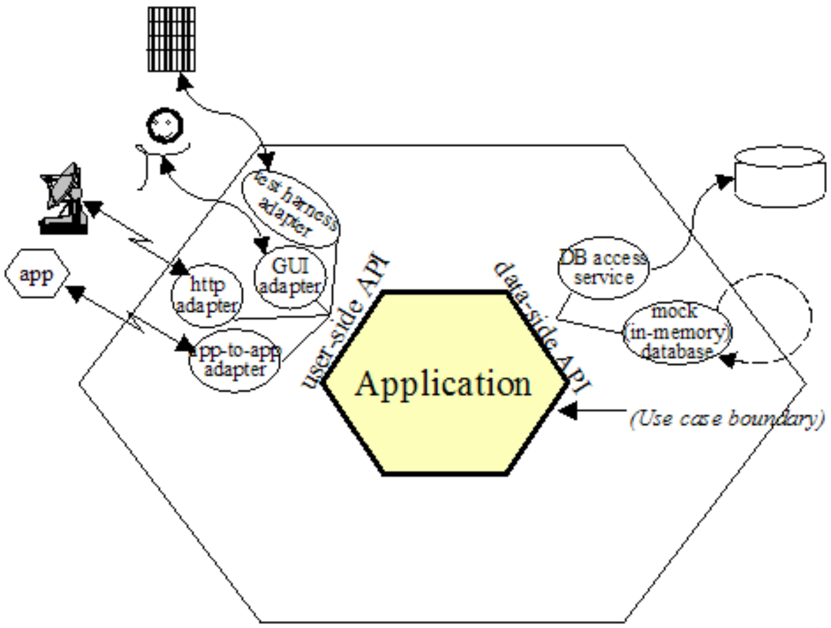


Figure 6.18: Ports & Adapters aka Hexagonal Architecture

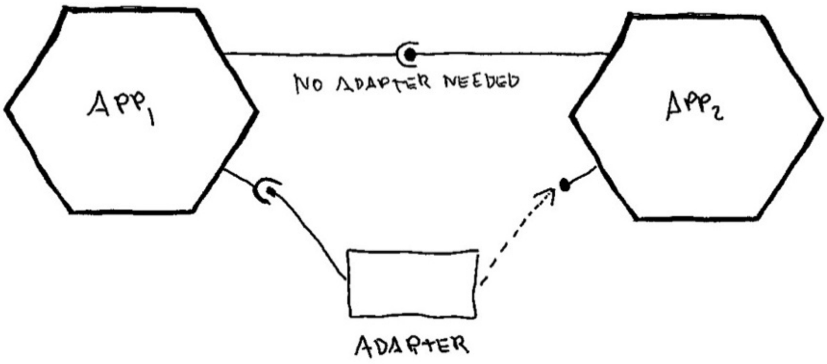


Figure 6.19: *Apps interacting with and without needing adapters*
(Image courtesy of Juan Manuel Garrido de Paz)

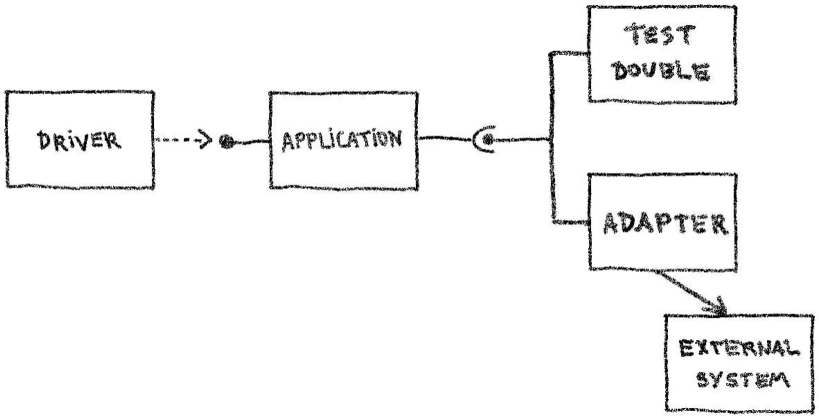


Figure 6.20: *Ports & Adapters* as component diagram showing test double
(Image courtesy of Juan Manuel Garrido de Paz)

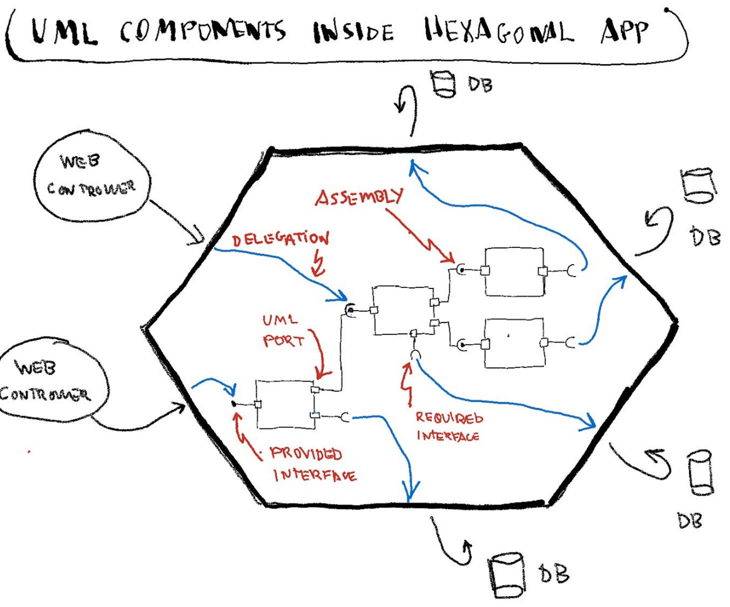


Figure 6.21: *Components* within *Ports & Adapters*
(Image courtesy of Juan Manuel Garrido de Paz)

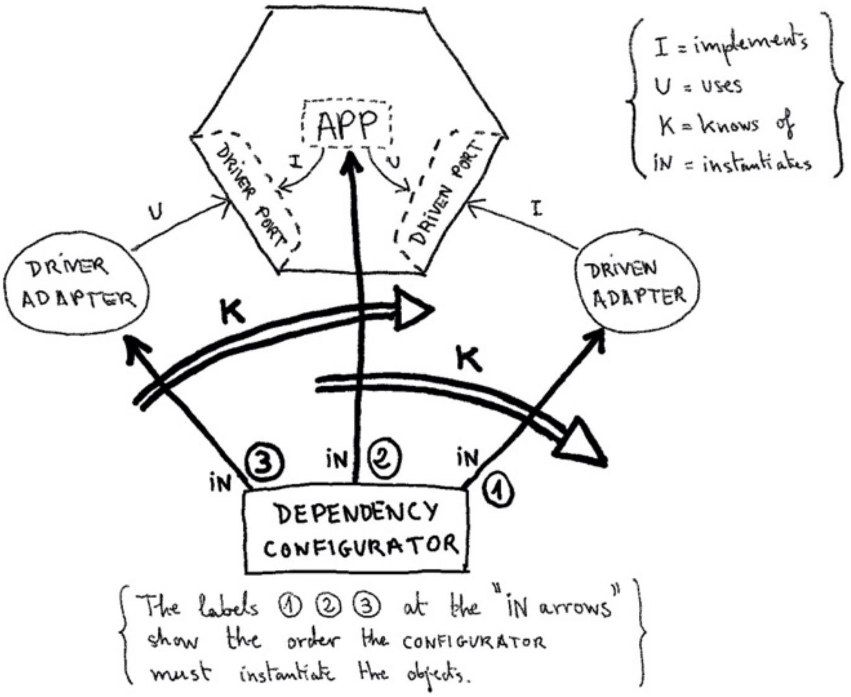


Figure 6.22: The Configurator sets up the knowledge paths (Image courtesy of Juan Manuel Garrido de Paz)

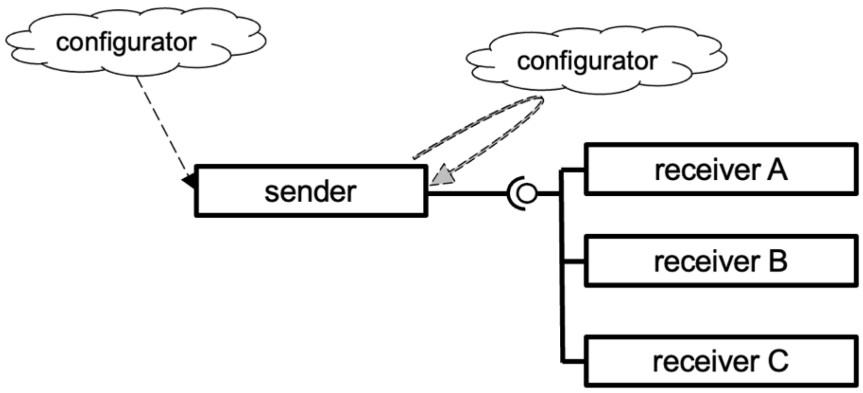


Figure 6.23. Informal view of Configurable Receiver, showing two choices for the placement of the configurator.

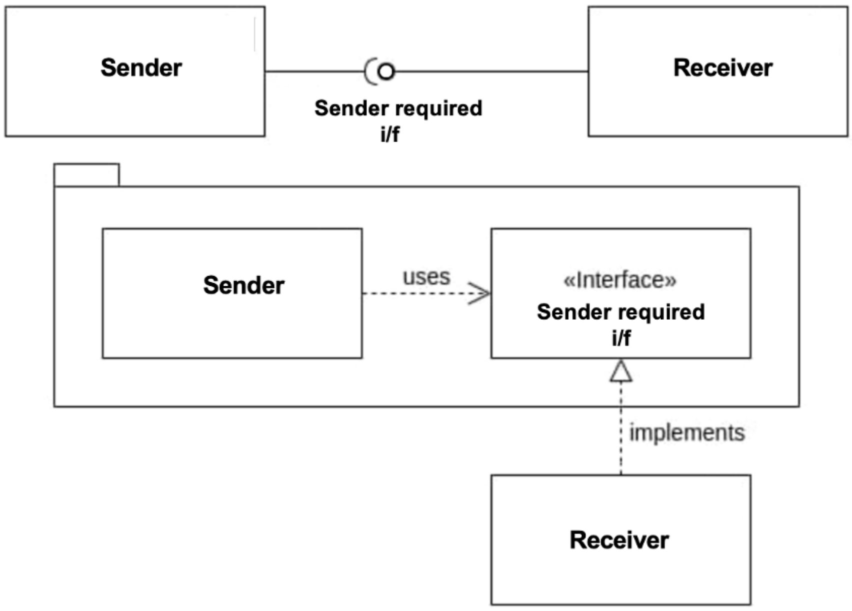


Figure 6.24. The sender owns the interface; receivers can be in different modules.

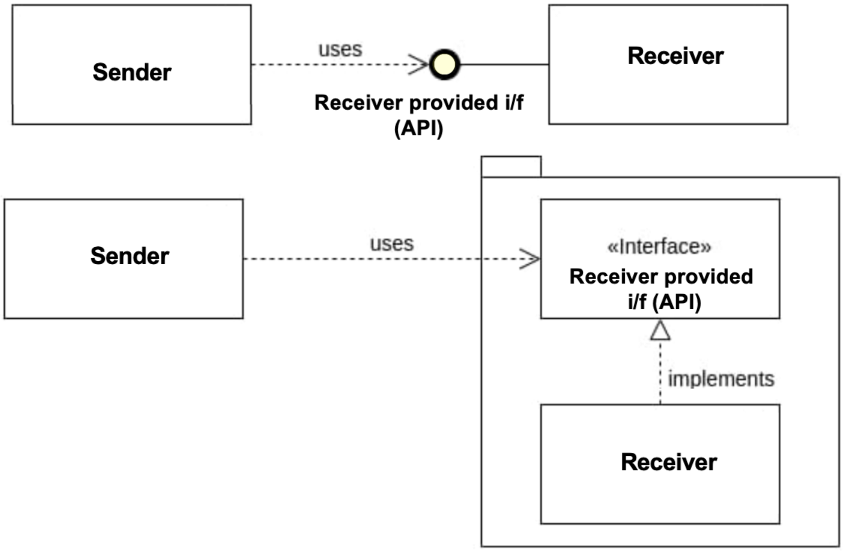


Figure 6.25. Not what we are after just now: The receiver owns the interface, the sender has a compile-time dependency on the receiver.

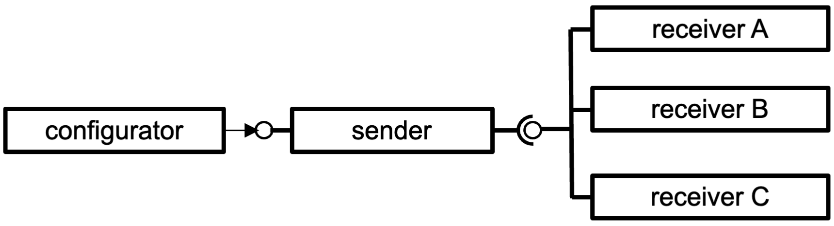


Figure 6.26a. The configurator tells the sender which receiver to use

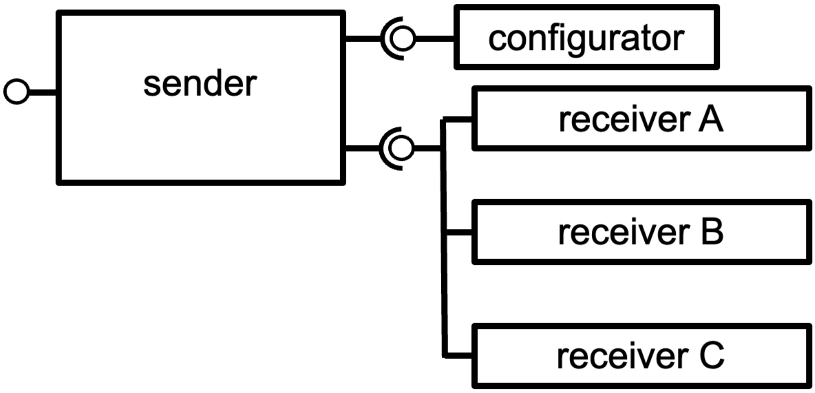


Figure 6.26b. The sender asks the configurator which receiver to use

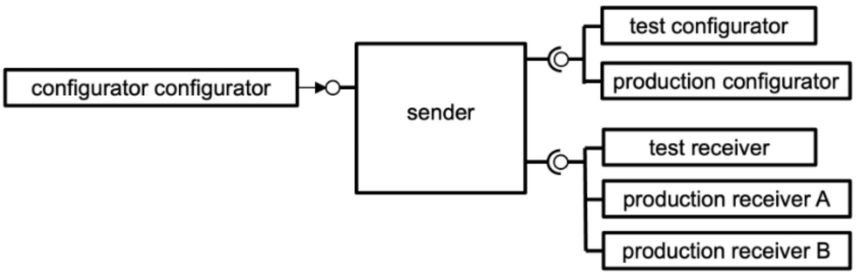


Figure 6.27. The configurator-configurator sends in a configurator to use as a service locator or broker for which receiver to use.

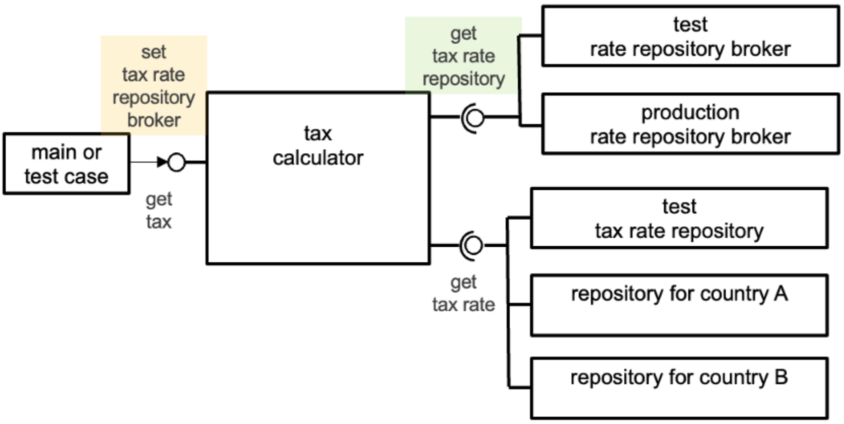


Figure 6.28. Main provides a broker to use to look up receivers.

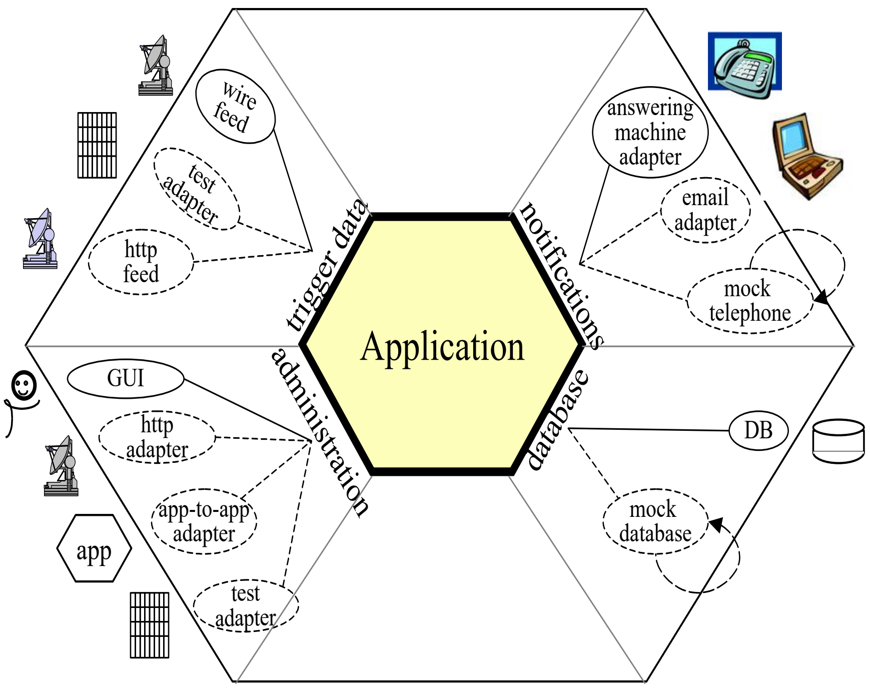


Figure 6.29. Ports & Adapters as known use of Configurable Receiver

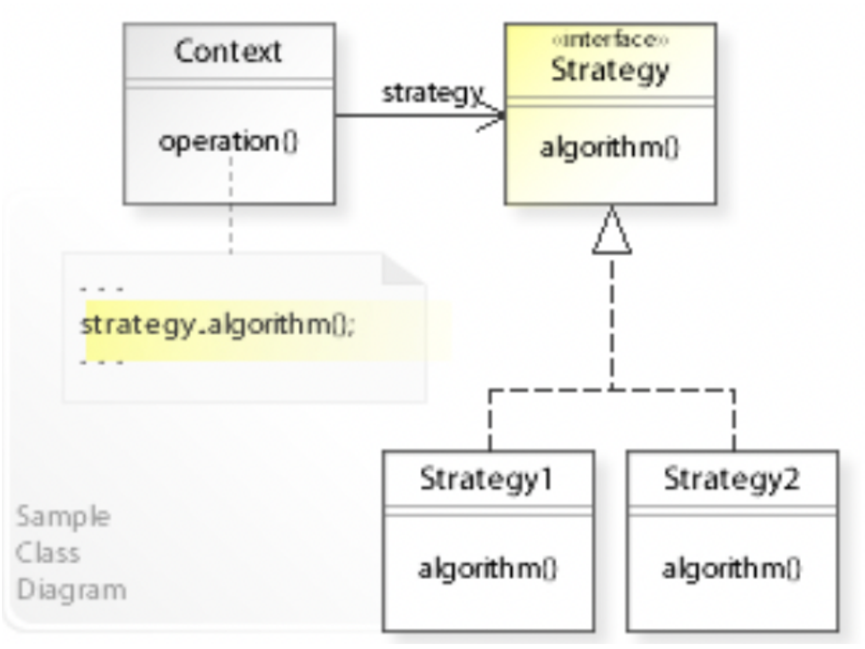


Figure 6.30: The Strategy pattern
(source: https://en.wikipedia.org/wiki/Strategy_pattern)

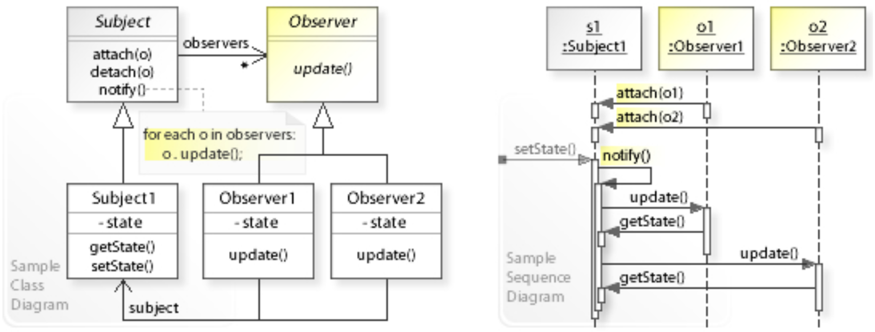


Figure 6.31. The Observer pattern (source: https://en.wikipedia.org/wiki/Observer_pattern)

-

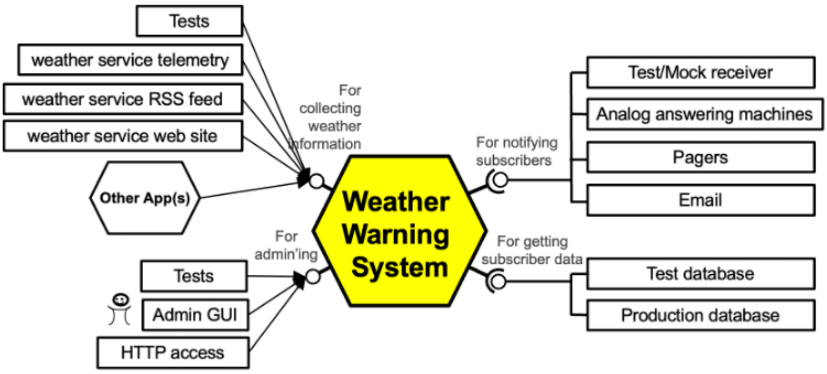


Figure 7.1. A Sample of The Pattern in Action

R.I.P. Juan Manuel Garrido de Paz. Thank you.

About the Authors

Dr. Alistair Cockburn (pronounced CO-BURN), known for his wild hair photo on LinkedIn, was named as one of the “42 Greatest Software Professionals of All Times" in 2020, as a world expert on object-oriented development, software architecture, project management, use cases and agile development. Since 2015 he has been working on expanding agile to cover every kind of initiative, including social impact project, governments, and families. For his latest work, see https://alistaircockburn.com/.

Juan Manuel Garrido de Paz (August 3, 1970 - April 18, 2024) won his Bachelor in Software Engineering at the Polytechnic University of Madrid. He became the world's other leading authority on the Ports & Adapters pattern by probing and interacting with Dr. Alistair Cockburn over years. A senior developer for the government of Andalucía, his two passions were Hexagonal Architecture and Recreativo de Huelva Football Club. Sadly, Juan passed away just weeks before this book went to print. This book is dedicated to him and his life.

"Looking at the screen of my laptop, I realized that it was full of code that didn't let me understand what it did regarding business logic. From that moment I began to search until I discovered the architecture that decouples the business logic from the frameworks: Hexagonal Architecture, more correctly called Ports & Adapters. From that moment until now, I haven't stopped reading and learning about this pattern."

Used by giants like Netflix and Amazon, the Hexagonal or Ports & Adapters architecture simplifies testing, protects against business logic leakage, supports changing technologies in long-running system, and lets you apply Domain Driven Design.

In this definitive book on the subject, pattern author Dr. Alistair Cockburn and Juan Manuel Garrido de Paz lay bare all of the intricacies of the pattern, providing sample code and answering your many frequently asked questions.

Hexagonal Architecture Explained

Alistair Cockburn & Juan Manuel Garrido de Paz

$40.00
ISBN 978-1-7375197-8-2
54000>
9 781737 519782