Garfixia Software Architectures

Ports-And-Adapters / Hexagonal Architecture

The main aim of this architecture is to decouple the application's core logic from the services it uses. This allows different services to be "plugged in", and it allows the application to be run without these services.

The core logic, or business logic, of an application consists of the algorithms that are essential to its purpose. They implement the use cases that are the heart of the application. When you change them, you change the essence of the application.

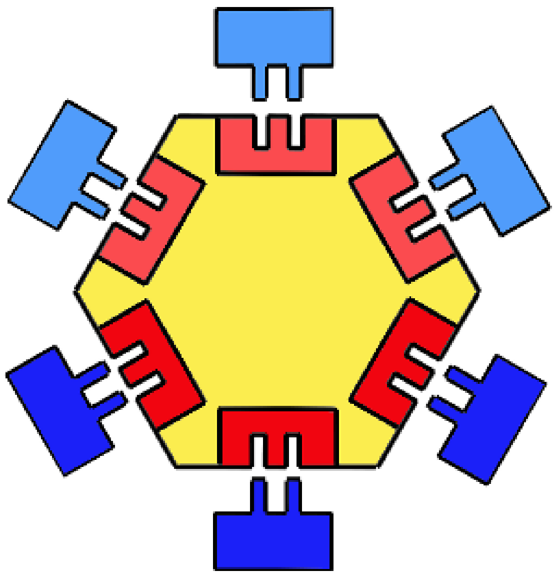
The services are *not* essential. They can be replaced without changing the purpose of the application. Examples: database access and other types of storage, user interface components, e-mail and other communication components, hardware devices.

In a strict sense of this architecture even the application's framework is a set of services. The core logic of an application should not depend on these services in this architecture (so that it becomes "framework agnosic").

Advantages of this architecture:

- The core logic can be tested independent of outside services.
- It is easy to replace services by other ones that are more fit in view of changing requirements.

An overview of Ports-And-Adapters:



yellow:	core logic
light red:	primary ports
light blue:	primary adapters
dark red:	secondary ports
dark blue:	secondary adapters

The number of ports depends on the application. The shown number of 6 ports nicely matches the name of the architecture. But the point of the architecture's name is not that the number six matters, but that *the core logic is at the center*. A realistic number of ports is about 2 to 4.

A **port** is an entry point, provided by the core logic. It defines a set of functions.

Primary ports are the main API of the application. They are *called by* the primary adapters that form the user side of the application. Examples of primary ports are functions that allow you to change objects, attributes, and relations in the core logic.

Secondary ports are the interfaces for the secondary adapters. They are *called by* the core logic. An example of a secondary port is an interface to store single objects. This interface simply specifies that an object be created, retrieved, updated, and deleted. It tells you nothing about the way the object is stored.

An **adapter** is a bridge between the application and the service that is needed by the application. It fits a specific port.

A **primary adapter** is a piece of code between the user and the core logic. One adapter could be a unit test function for the core logic. Another could be a controller-like function that interacts both with the graphical user interface and the core logic. The primary adapter calls the API functions of the core logic.

A **secondary adapter** is an implementation of the secondary port (which is an interface). For instance, it can be a small class that converts application storage requests to a given database, and return the results of the database in a format requested by the secondary port. It can also be a mock database object needed to unit tests certain parts of the core logic. The core logic calls the functions of the secondary adapter.

Where does it come from?

Alistair Cockburn invented it in 2005. It is a response to the desired to create thoroughly testable applications. As Cockburn says: "Allow 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."

When should you use it?

If 100% unit-test code coverage is important to your application. Also, if you want to be able to switch your storage mechanism, or any other type of third-party code. The architecture is especially useful for long-lasting applications that need to keep up with changing requirements.

How does it work?

The application can be used by different types of users. Each of these can create their own variant of the application, by plugging in custom adapters.

- An instance of the application is created, as well as the adapters.
- The secondary adapters are passed to the core logic (dependency injection).
- The primary adapters receive a link to the core logic. They start to drive the application.
- User input is processed by one or more primary adapter(s) and passed to the core logic.
- The core logic interacts with the secondary adapters only.
- Output of the core logic is returned to the primary adapters. They feed it back to the user.

Common implementation techniques

Check Alistair Cockburn's article and Matteo Vaccari's blog for some code examples.

Some details:

- Dependency injection is used to pass the secondary adapters to the core logic
- Secondary ports are implemented as interfaces. Secondary adapters implement these interfaces.
- You could create a factory for adapters for a given service.

Links

- Hexagonal Architecture Alistair Cockburn's original article on this architecture.
- The birthday greetings kata Matteo Vaccari's blog that provides an example Java implementation of the port and adapter.
- Visualising Test Terminology Nat Price's information on testing using Ports-And-adapters.