



Ch25. 層與邊界 & Ch26. 主元件


Bear

Chapter25. 層與邊界

談談 Hunt the Wumpus 冒險遊戲



Hunt the Wumpus 冒險遊戲



```
HUNT THE WUMPUS
INSTRUCTIONS (Y-N)? N
YOU ARE IN ROOM 17
TUNNELS LEAD TO 7 16 18
SHOOT OR MOVE (S-M)? M
WHERE TO? 16
ZAP--SUPER BAT SNATCH! ELSEWHEREVILLE FOR YOU!
YOU ARE IN ROOM 5
TUNNELS LEAD TO 1 4 6
SHOOT OR MOVE (S-M)? M
WHERE TO? 4
YOU ARE IN ROOM 4
TUNNELS LEAD TO 3 5 14
SHOOT OR MOVE (S-M)?
```

▲ Hunt the Wumpus 遊戲截圖

Hunt the Wumpus 冒險遊戲

遊戲大致可分為以下元件：

- UI 處理
- 當前遊戲狀態



Hunt the Wumpus 冒險遊戲

假設我們想保留文字UI，
但要從遊戲規則中解耦，
以便因應地區，轉換不同語言...

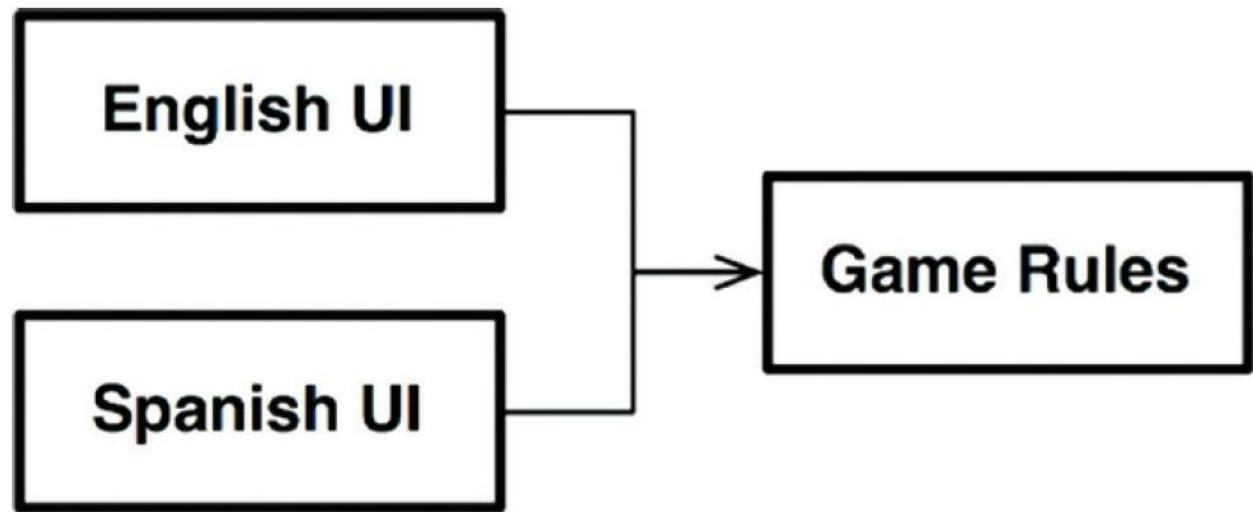


Figure 25.1 Any number of UI components can reuse the game rules

Hunt the Wumpus 冒險遊戲

接著假設遊戲狀態是，
保存在某種持久性儲存體...

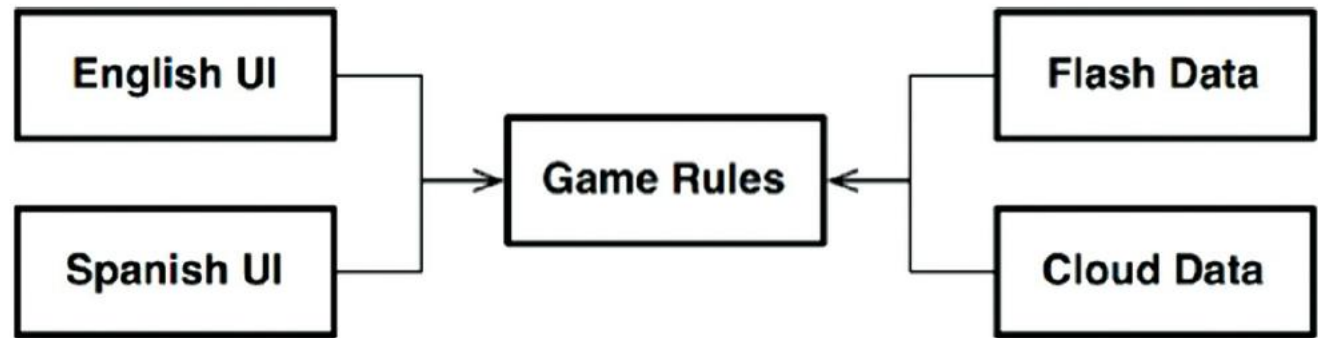


Figure 25.2 Following the Dependency Rule

整潔的架構?

這種情況，會使用簡單程式作為更大一點系統的代理。

但我們是否找到所有重要的架構邊界?

例如: 改變文字通訊機制

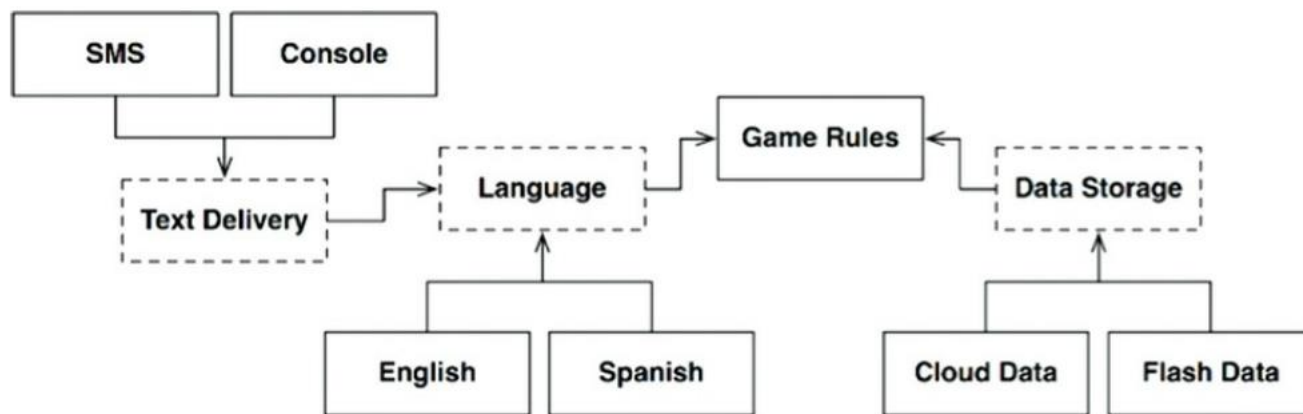


Figure 25.3 The revised diagram

整潔的架構？

此時虛線框表示定義了一個API的抽象元件，
由其上方或下方的元件實作。

由Game Rules定義及Language實作來通訊

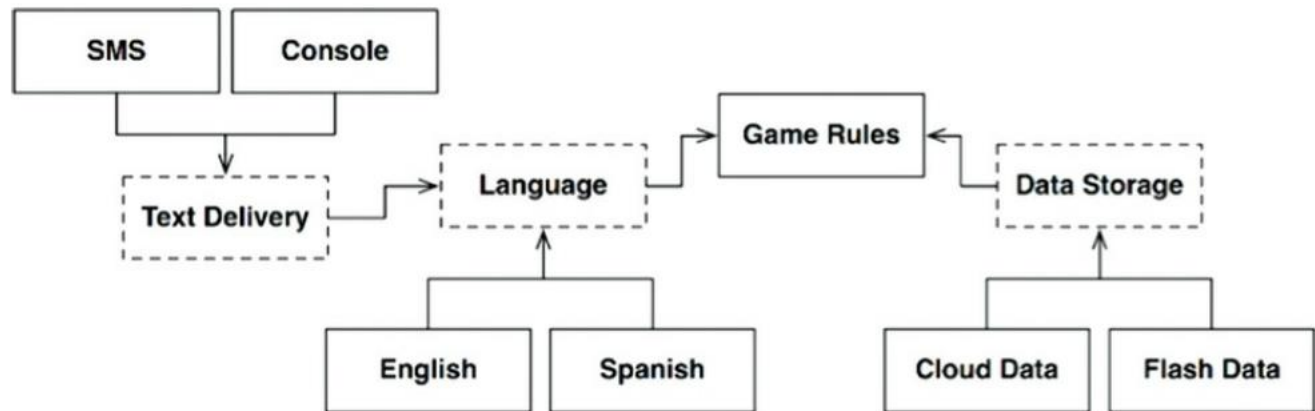


Figure 25.3 The revised diagram

整潔的架構？

API由使用者定義及擁有，
而不是由實作者來定義及擁有。

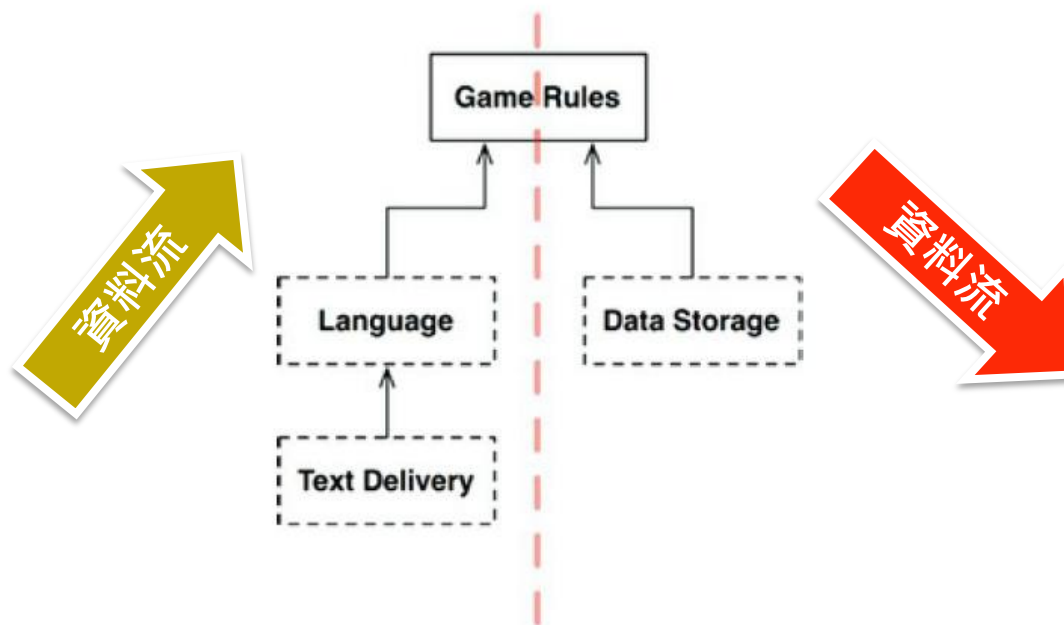


Figure 25.4 Simplified diagram

跨越流

永遠都是兩個資料流嗎？

想像一個網路版的Hunt the Wumpus

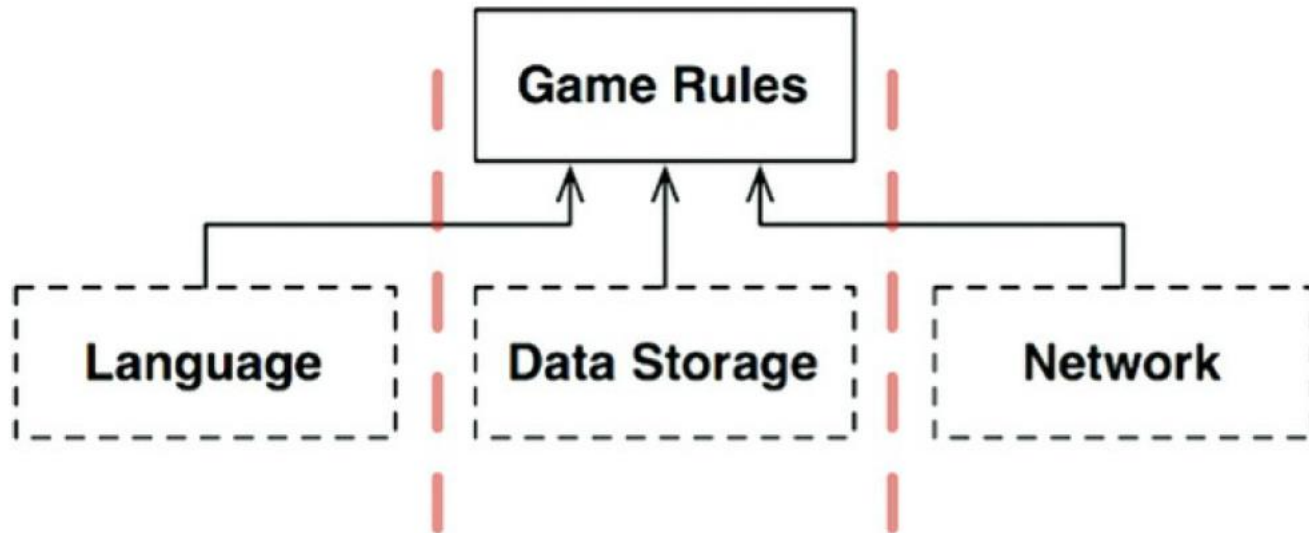


Figure 25.5 Adding a network component

分割流

所有的流最終都會在單個頂部元件中相遇嗎？

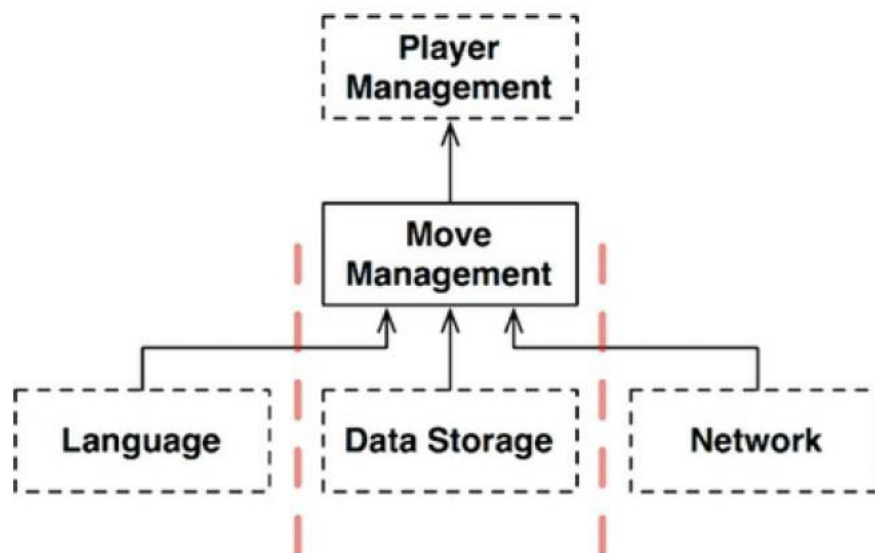


Figure 25.6 The higher-level policy manages the player

分割流

想像一個 Hunt the Wumpus 的多人遊戲版本

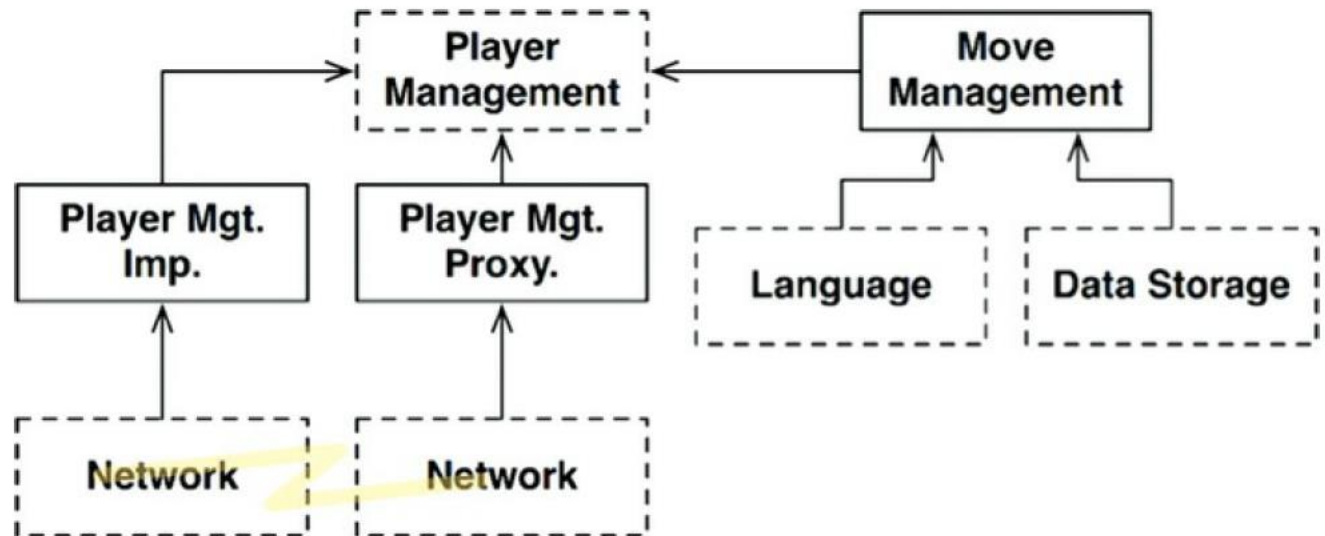


Figure 25.7 Adding a micro-service API

總結

- 處處存在架構的邊界
- 我們不應該預設抽象的必要性
- 也必須意識到，事後增加邊界的成本也很高

不要在專案開始階段，草率地決定要實作的邊界
你必須觀察，隨著系統發展，哪裡需要邊界

注意第一個摩擦，因為目前邊界不存在
目標在實作成本低於忽略成本的拐點



Chapter26. 主元件

用來建立、協調和監督其他元件的元件



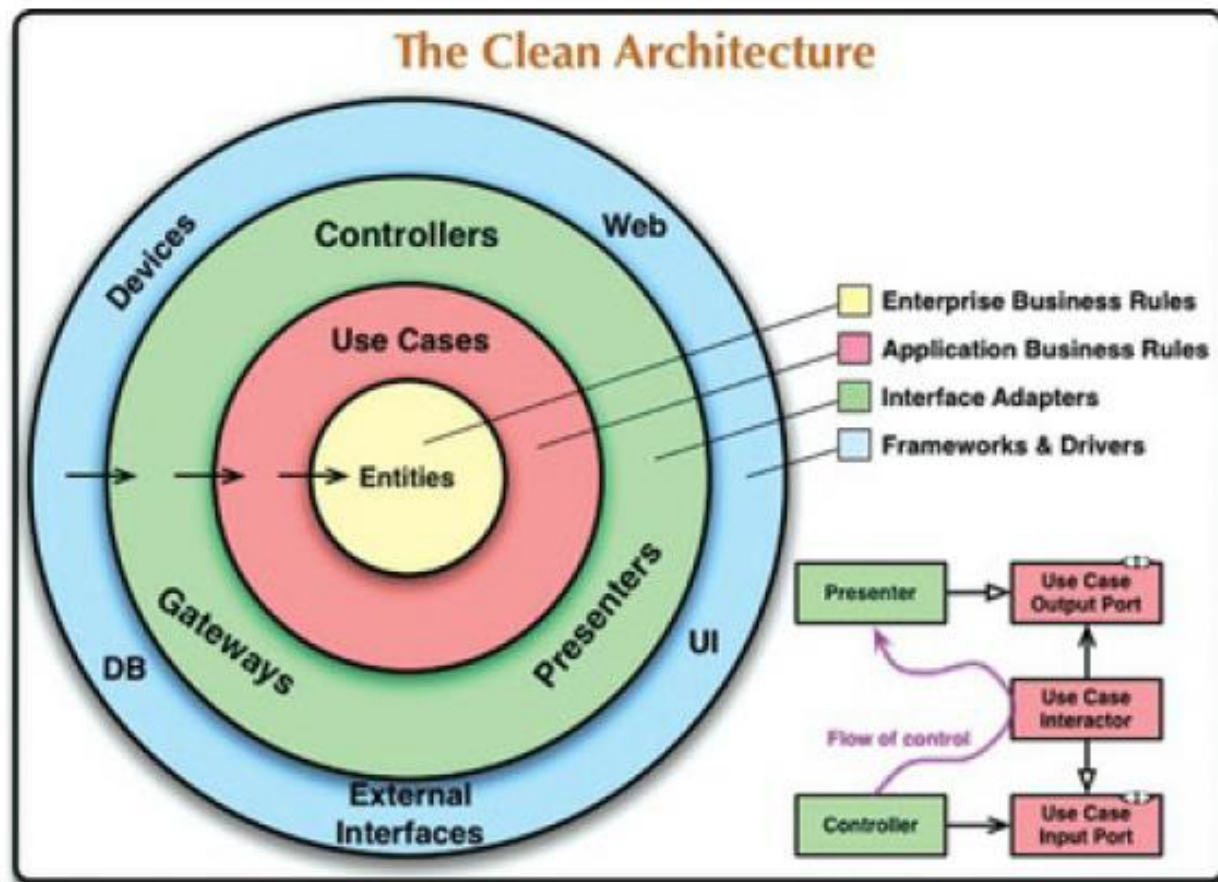
最終的細節

- Main元件就是最終的元件
- 也是最低層次的策略
- 除了作業系統，沒有人依賴它
- 任務是建立所有的工廠、策略和其他全域設施
- 然後將控制權交給系統的高層抽象部分

你可以把Main想成最髒的元件



主元件



Main是位於整潔架構最外圈的一個髒且低層級的模組

總結

將Main看作應用程式的一個plugin：

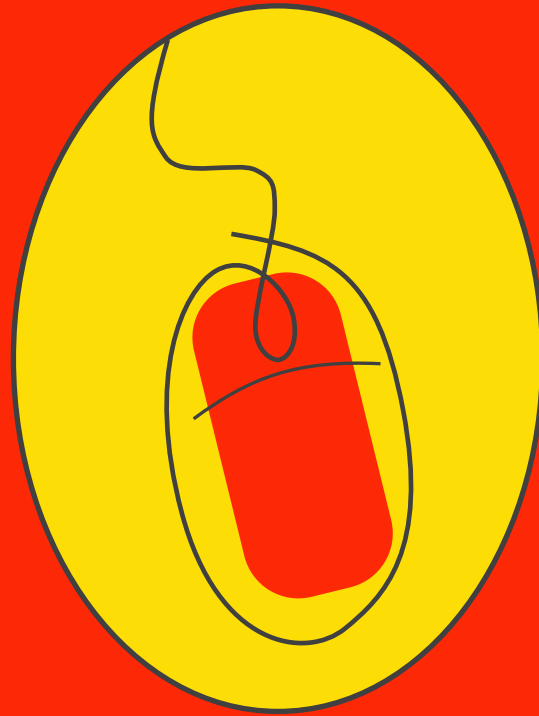
- 用來設定初始條件和設置
- 收集所有外部資源
- 然後將其控制權轉移至應用程式的高級策略

Dev有一個Main plugin
Test有另一個Main plugin

不同國家/地區有各自的Main plugin
不同客戶有各自的Main plugin

將Main看作躲在架構邊界後面的plugin時
設置問題就變得更容易了





感謝您的聆聽