Chapter 5

系統塑模

本章內容

5.1 環境模型(Context models)

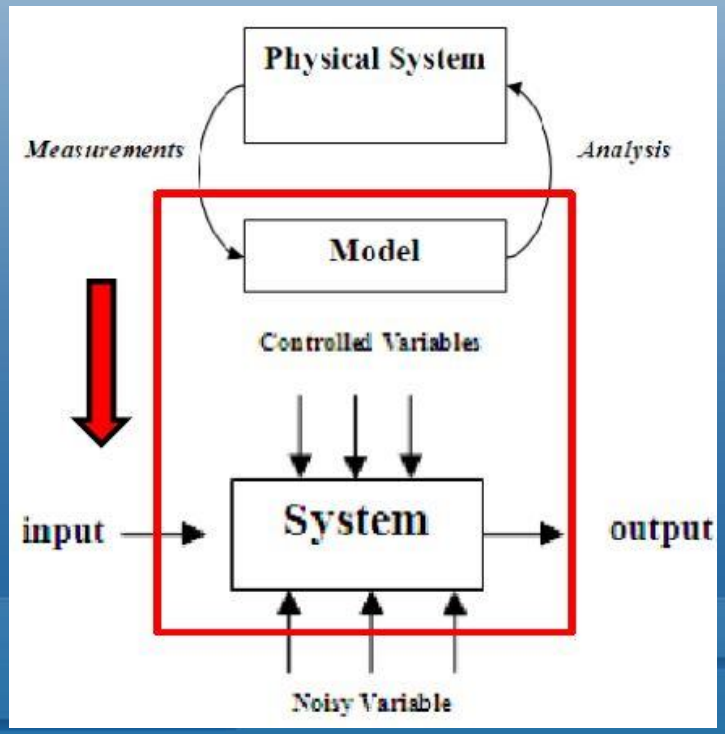
5.2 互動模型(Interaction models)

5.3 結構模型(Structural models)

5.4 行為模型(Behavioral models)

5.5 模型驅動式架構 (MDA, Model-driven Engineering)

- 系統塑模是針對某系統開發它的各種抽象模型的程序，每種模型代表系統不同的角度或觀點。
- 現在的系統塑模 (system modeling) 通常是指使用某種圖形化符號表示法，也就是統一塑模語言 (Unified Modeling Language, UML) 的幾種圖形來表示系統。



- 無論是對現有系統和準備開發的系統，你都可能會開發模型：
  1. 現有系統的模型是在需求工程期間使用，它可協助釐清現有系統會做什麼，而且可用來當作討論它優缺點的基礎。
  2. 新系統的模型是在需求工程期間使用，目的是協助對系統的其他利害關係人解釋提出的需求。
  - 工程師使用這些模型來討論設計提案，以及製作系統的實作文件。假如是採用模型驅動式工程的程序 (Brambilla, Cabot, and Wimmer 2012)，還可能從這些系統模型產生出完整或部分的系統實作。

- 從不同觀點來看系統，可能會開發出不同的模型。舉例來看：
  1. **外部觀點**：建立系統環境的模型。
  2. **互動觀點**：建立系統與它的環境或系統的元件彼此間互動情況的模型。
  3. **結構觀點**：建立系統的架構或此系統處理的資料結構的模型。
  4. **行為觀點**：建立系統動態行為以及對事件如何反應的模型。

- 圖形模型有3種常見的使用方式：

1. 用來促進針對現有或提案系統的討論。
   - 此時的模型可能不完整，而且可能以非正式的方式使用塑模符號表示法。
2. 用來記錄現有系統的說明文件。
   - 它們不需要完整。但是一定要正確。
3. 當作系統的細部描述，可用來產生系統實作。
   - 這時的系統模型必須完整而且正確。

- 5種UML圖形：

1. **活動圖 (activity diagram)**：
   - 描繪在程序或資料處理過程中的活動。
2. **使用案例圖 (use case diagram)**：
   - 描繪系統與它的環境之間的互動情形。
3. **序列圖 (sequence diagram)**：
   - 描繪行動者與系統之間以及系統元件彼此之間的互動情形。
4. **類別圖 (class diagram)**：
   - 描繪系統中的物件類別，以及這些類別之間的關聯性。
5. **狀態圖 (state diagram)**：
   - 描繪系統對內部和外部事件如何反應。

5.1 環境模型

- 應該先決定系統的邊界 (boundary) 在哪裡，也就是先確認系統包含哪些部分，哪些部分不包含。
  - 這牽涉到要與系統的利害關係人，一起決定想要系統包含哪些功能，以及在這個系統的運作環境中想進行什麼樣的處理和運算。
- 由於社會和組織的考量，使系統邊界的定位可能會取決於某些非技術性因素。
  - 比如說，某系統邊界也許會刻意的定位成讓分析程序全在某一地點完成，因為這樣可以避開某個特別難纏的經理人。
- 一旦系統邊界決定之後，接下來的分析活動部分便是定義系統環境，以及系統與外在環境的相依性。

5.1 環境模型

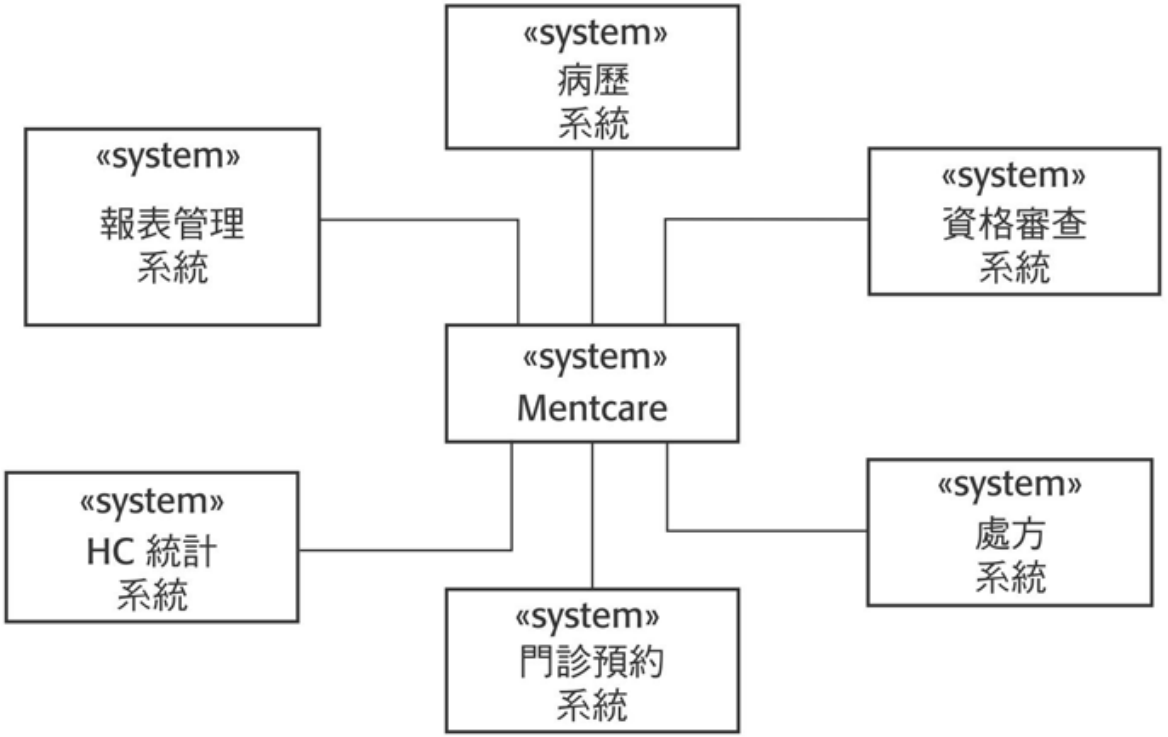


圖 5.1 Mentcare 系統的環境

- 環境模型雖然可以描述系統的環境（內含一些其他的自動化系統），但是無法顯示系統與環境中其他系統之間的關係。

5.1 環境模型

- UML活動圖可用來顯示系統所使用的企業程序。

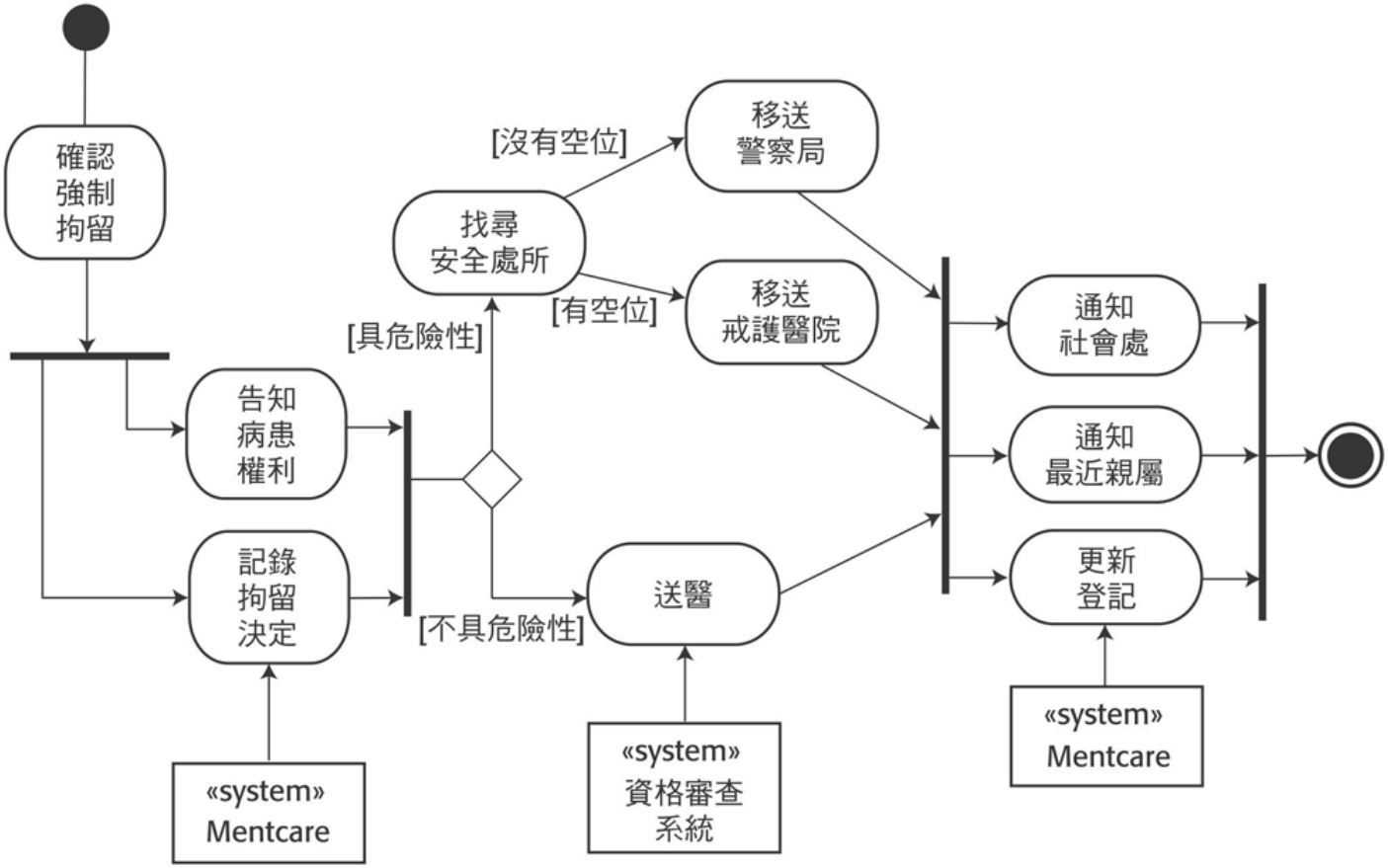


圖 5.2　強制拘留的程序模型

5.2 互動模型

- 建立使用者互動的模型是很重要的，因為有助於找出使用者需求。
  - 建立系統與系統之間互動的模型，能強調出可能發生的溝通問題。
  - 而建立元件間互動的模型，則有助於我們了解目前提出的系統結構，是否能達成要求的系統效能和可信賴度。
- 本節說明兩種相關的方法：
  1. **使用案例塑模(Use Case diagram)**
  2. **序列圖(Sequence diagram)**

5.2.1 使用案例塑模(Use case diagram)

- 使用案例塑模 (use case modeling) 一開始是由Ivar Jacobson在1990年代開發出來的 (Jacobsen et al. 1993)，後來被納入成為UML的一部分，而創建出一種專門支援使用案例塑模的UML圖形。
- 每個使用案例代表一個關於外界與系統之間互動的獨立任務。

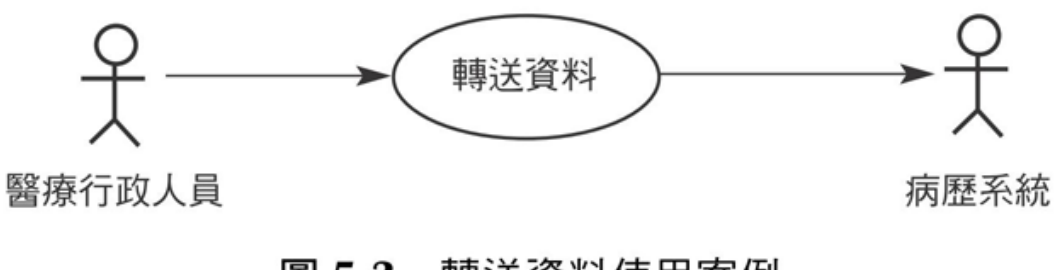


圖 5.3 轉送資料使用案例

- 人形符號原本只代表人員的互動，但現在也會用來表示其他外界的系統和硬體。

5.2.1 使用案例塑模

- 使用案例圖只能對互動做非常簡單的概述，因此要了解它實際上包含什麼，你需要提供更詳細的資訊。
  - 這裡的詳細資訊可以是一段簡單的文字描述、使用表格以結構化方式來說明，或者使用序列圖。

Mentcare 系統：轉送資料 (Transfer data)	
行動者	醫療行政人員、病歷系統 (PRS)
說明	醫療行政人員可能會從 Mentcare 系統傳輸資料到衛生機關的總病歷資料庫。被傳送的資訊可能是更正過的個人資料 (住址、電話號碼等)，或者是病患的診斷和治療過程摘要
資料	病患的個人資料、治療過程摘要
刺激	由醫療行政人員輸入命令
回應	確認 PRS 已經更新
註解	醫療行政人員必須有適當的權限才能存取病患資料和 PRS

圖 5.4 「轉送資料」使用案例的表格描述

5.2.1 使用案例塑模

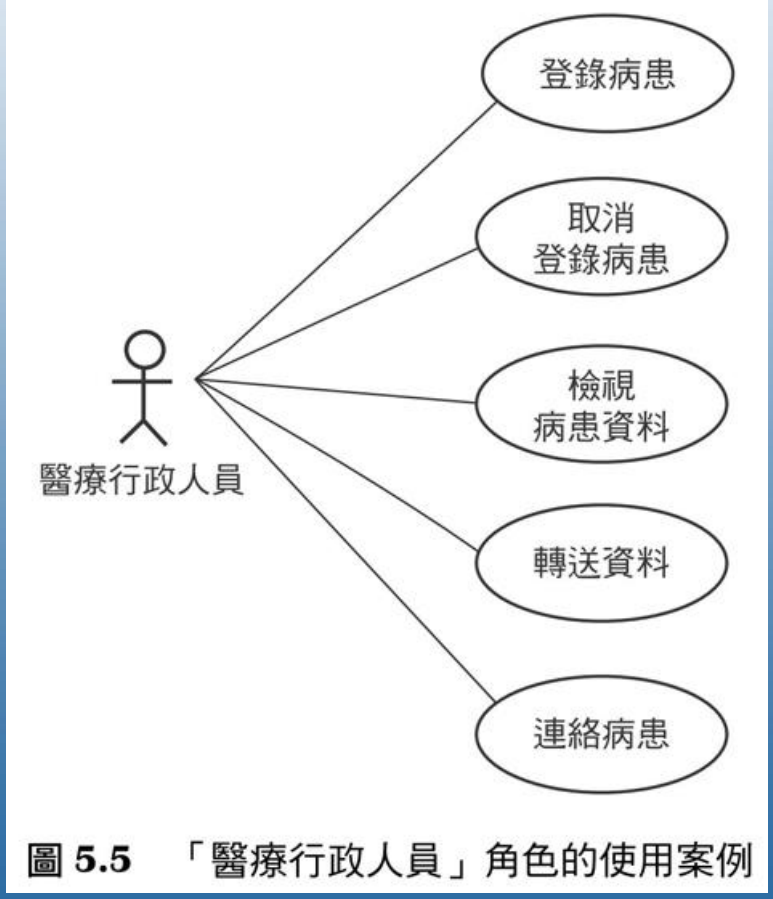


圖 5.5 「醫療行政人員」角色的使用案例

5.2.2 序列圖(Sequence diagram)

- UML的序列圖 (sequence diagram) 主要是用來描述行動者與系統的物件之間，以及物件彼此之間的互動情形。
- 序列圖的目的是展示在某個特定的使用案例期間，所發生的各種互動的順序。
- 在序列圖中，相關的物件和行動者都是對齊圖的最頂端，從它們畫出垂直的虛線。物件之間的互動是用有註解的箭頭來顯示。

5.2.2 序列圖

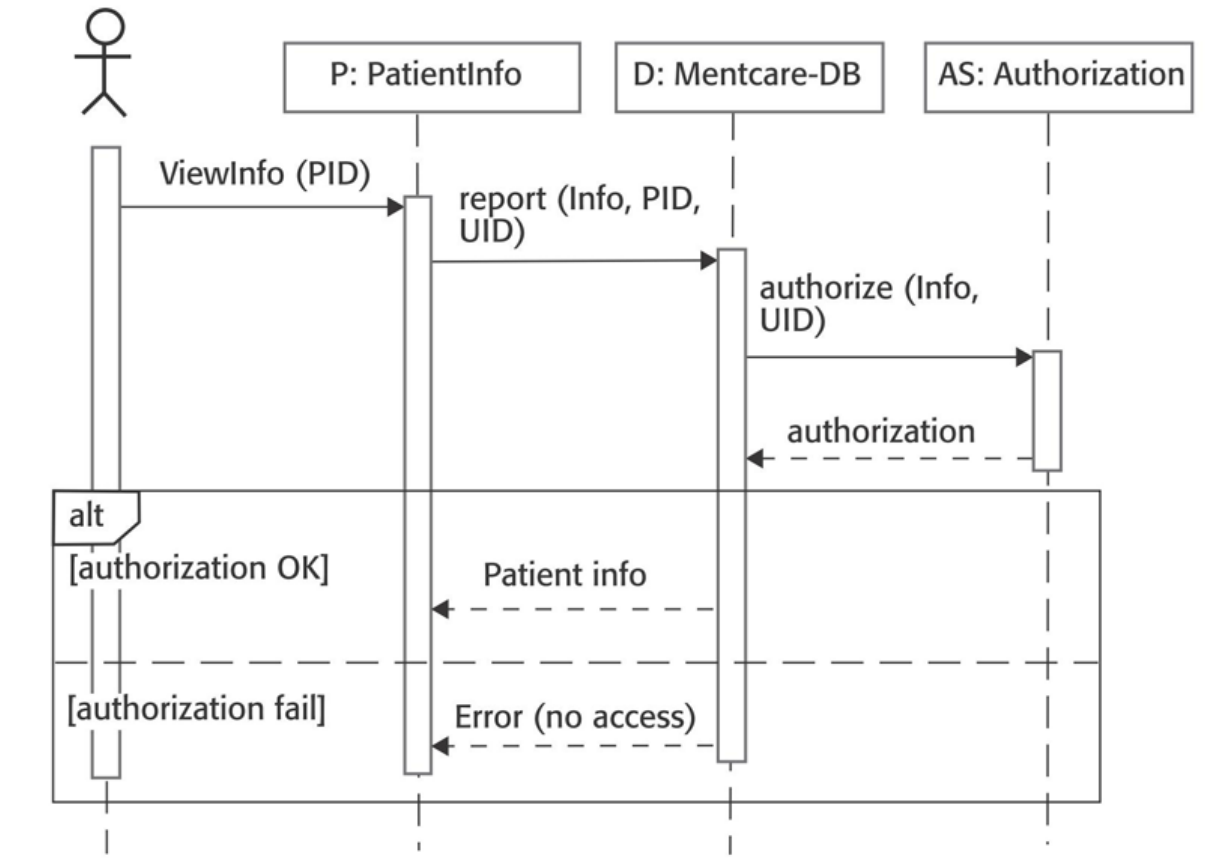


圖 5.6 「檢視病患資料」的序列圖

5.2.2 序列圖

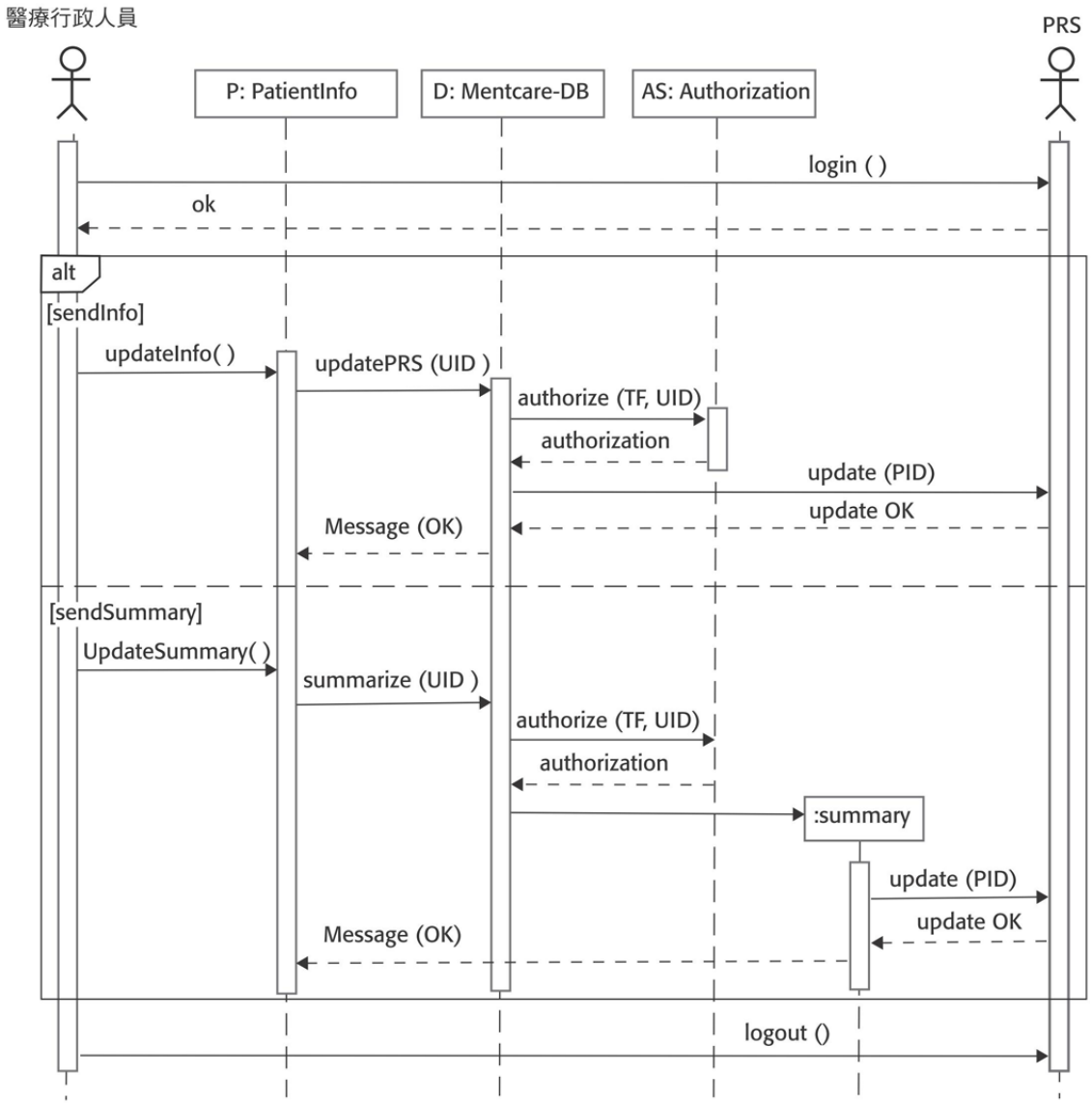


圖 5.7 「轉送資料」的序列圖

5.3 結構模型

- 軟體的結構模型 (structural model) 是以組成系統的元件和它們之間的關係 (relationship) 來顯示系統的架構。
- 結構模型可以是顯示系統設計結構的靜態模型，或者是顯示系統執行中架構的動態模型。
- 當你在討論和設計系統架構時，會建立系統的結構模型。

5.3.1 類別圖

- 類別圖 (class diagram) 是在開發物件導向系統模型時，用來顯示系統中的類別，以及這些類別之間的關聯 (association)。
- 以比較不嚴格的定義而言，物件類別可視為某種系統物件的一般定義；而關聯則是這些類別之間的連結，顯示在這些類別之間有某種關係存在。
- 當你在軟體工程程序的早期階段開發模型時，物件是代表在真實世界的某項事物，例如一位病患、一份處方、一位醫師等。

5.3.1 類別圖

圖 5.8 UML 類別和關聯

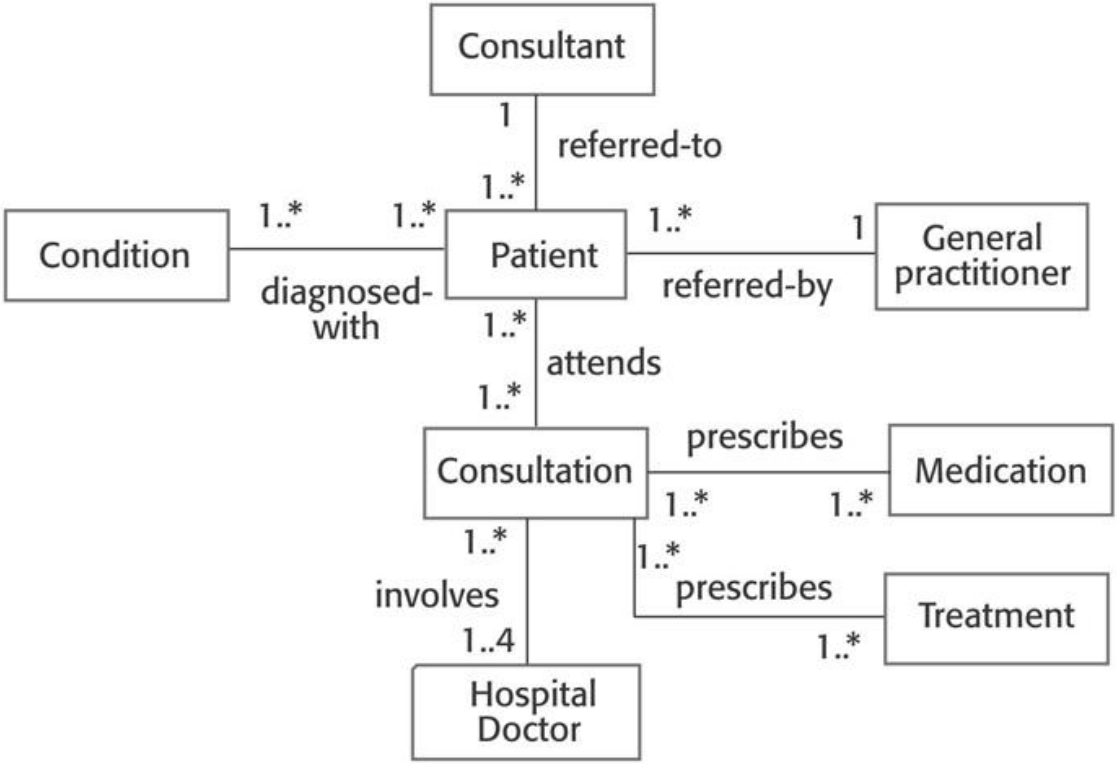


圖 5.9 Mentcare 系統的類別和關聯

5.3.1 類別圖

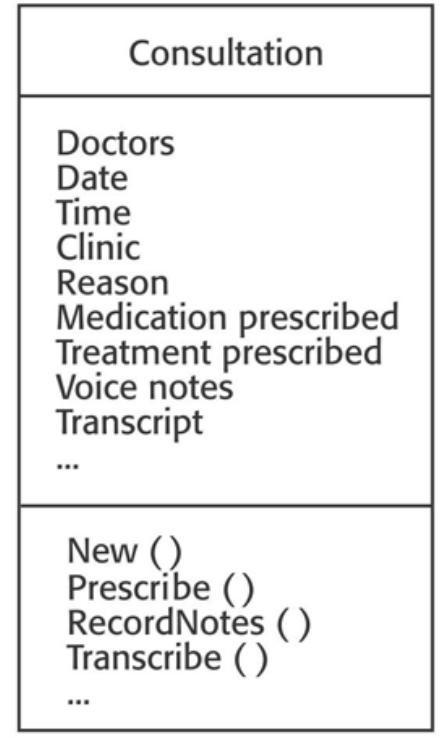


圖 5.10 Consultation 類別

5.3.2 一般化關係

- 一般化 (generalization) 是一種我們每天都會用來減輕複雜度的技巧。
- 對於每天會碰到的各種事物，不會去一樣樣學習它們的細節，而是把這些事物歸類 (如動物、汽車、房屋等)，並且直接學習這些分類的特性。
- 這讓我們能推理出同一類別的不同成員有哪些共同的特性 (例如松鼠和老鼠都是齧齒目動物)。

5.3.2 一般化關係

- 在為系統建立模型時，研究系統中有哪些類別可以進行一般化是很有用的。
- 如果將來有人提出變更要求，你就不必把系統裡全部的類別都查一遍，確認是否有被變更影響到。
- 在物件導向語言如Java中，它們使用語言中內建的類別繼承(class inheritance)機制來實作一般化。

5.3.2 一般化關係

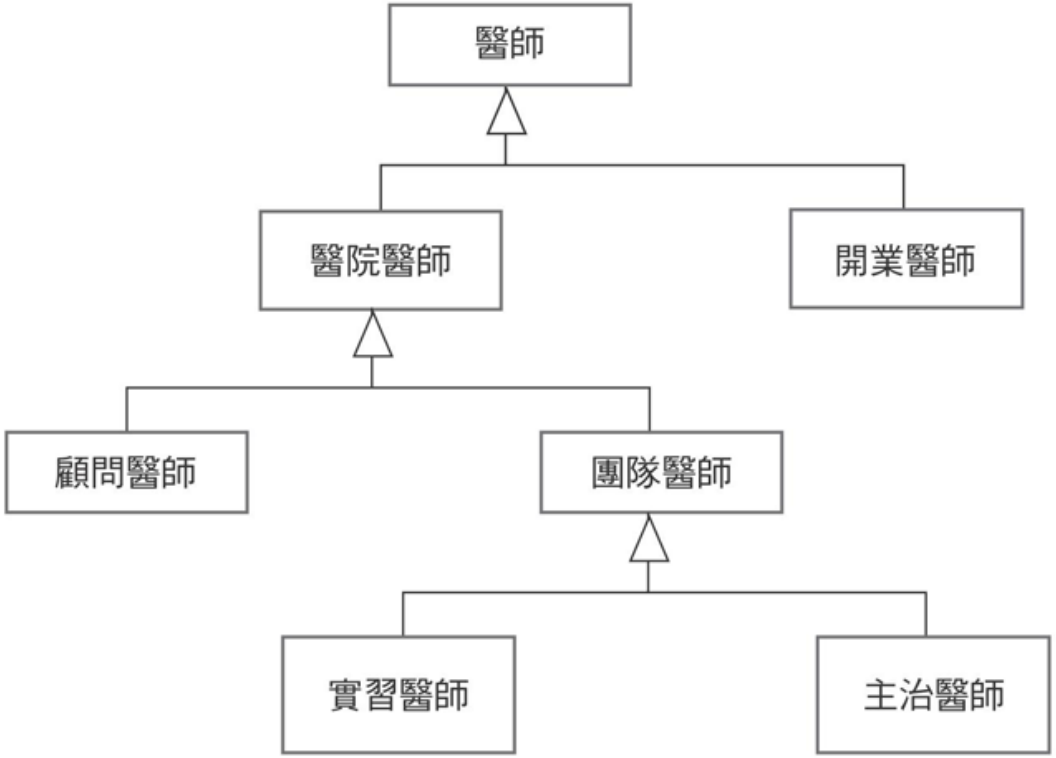


圖 5.11　一般化階層架構

5.3.2 一般化關係

- 在一般化架構中，高階類別相關聯的屬性和運算動作，同時也會是低階類別的屬性和運算動作。
- 低階類別是子類別 (subclass)，繼承上層超類別 (superclass) 的屬性和運算。之後這些低階類別可以加上自己獨特的屬性和運算。

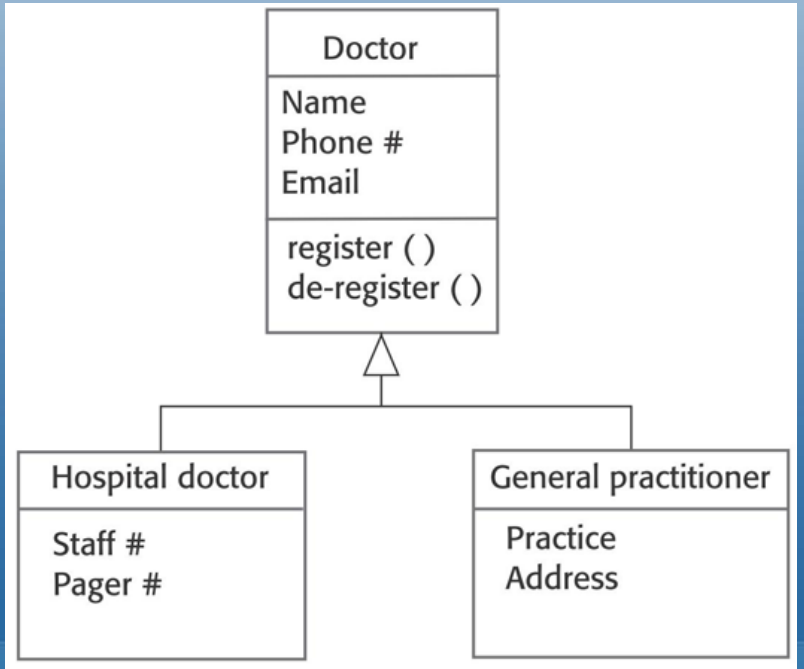


圖 5.12　加上詳細資訊的一般化階層架構

5.3.3 聚合關係

- 聚合關係 (aggregation)，意思是某個物件 (整體) 是由其他一些物件 (部分) 所組成。表示符號是在代表整體的類別旁的連結線加上一個菱形。

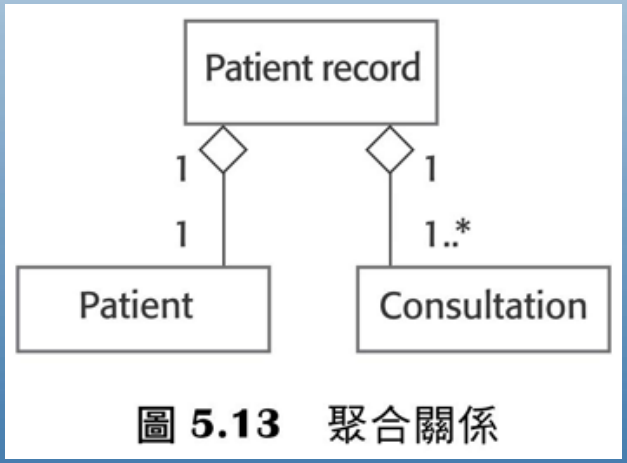


圖 5.13　聚合關係

5.4 行為模型

- 行為模型 (behavior model) 是描述系統執行中動態行為的模型。
- 它展示當系統要回應來自環境的刺激時，會發生或應該發生什麼事。

1. **資料 (Data)：**
   - 有些必須由系統處理的資料剛抵達。資料的可用性會觸發處理動作。
2. **事件 (Event)：**
   - 發生某個會觸發系統處理的事件。事件可能會有相關聯的資料，不過不一定有。

5.4 行為模型

- 大部分的商業系統是資料處理系統，主要都是由資料所驅動的。它們大多是受輸入到系統內的資料所控制，很少去處理外部事件。它們的處理動作包括對資料的一連串動作，然後產生輸出。
- 相反的，即時系統通常由事件所驅動，資料處理的部分很少。

5.4.1 資料驅動塑模

- 資料驅動模型 (data-driven model) 顯示參與處理輸入資料和產生相關輸出的一連串行動。
- 它們在需求分析期間特別有用，因為可顯示出系統從這端到那端之間的處理過程。

5.4.1 資料驅動塑模

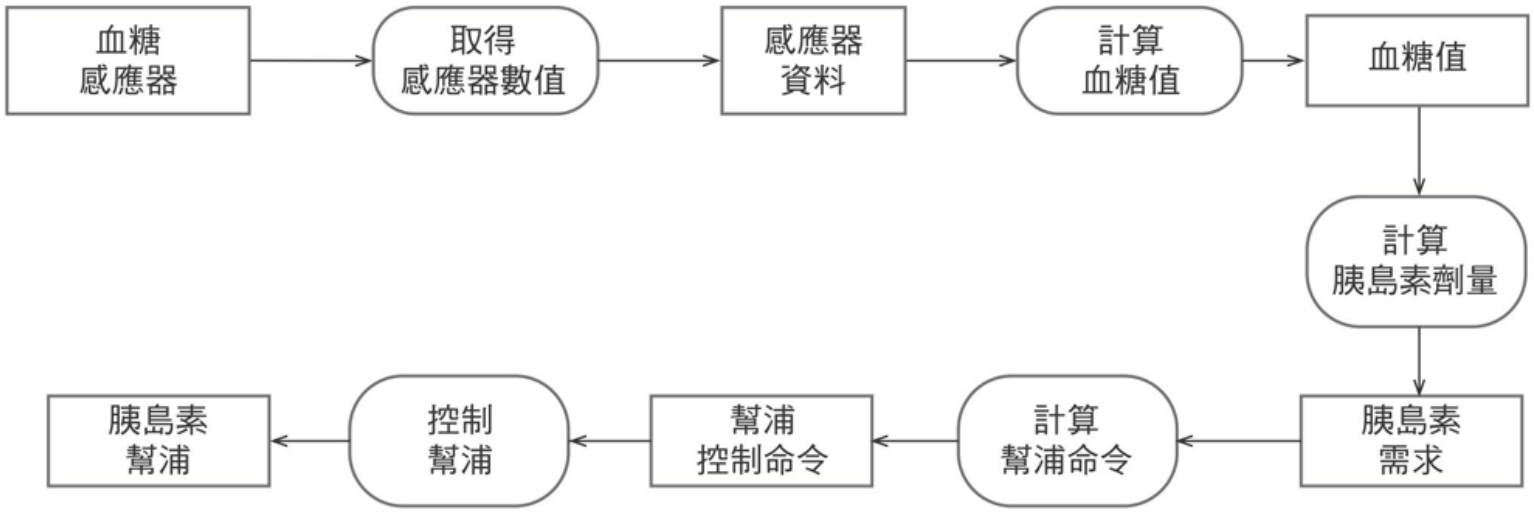


圖 5.14　胰島素給藥系統的活動模型

5.4.1 資料驅動塑模

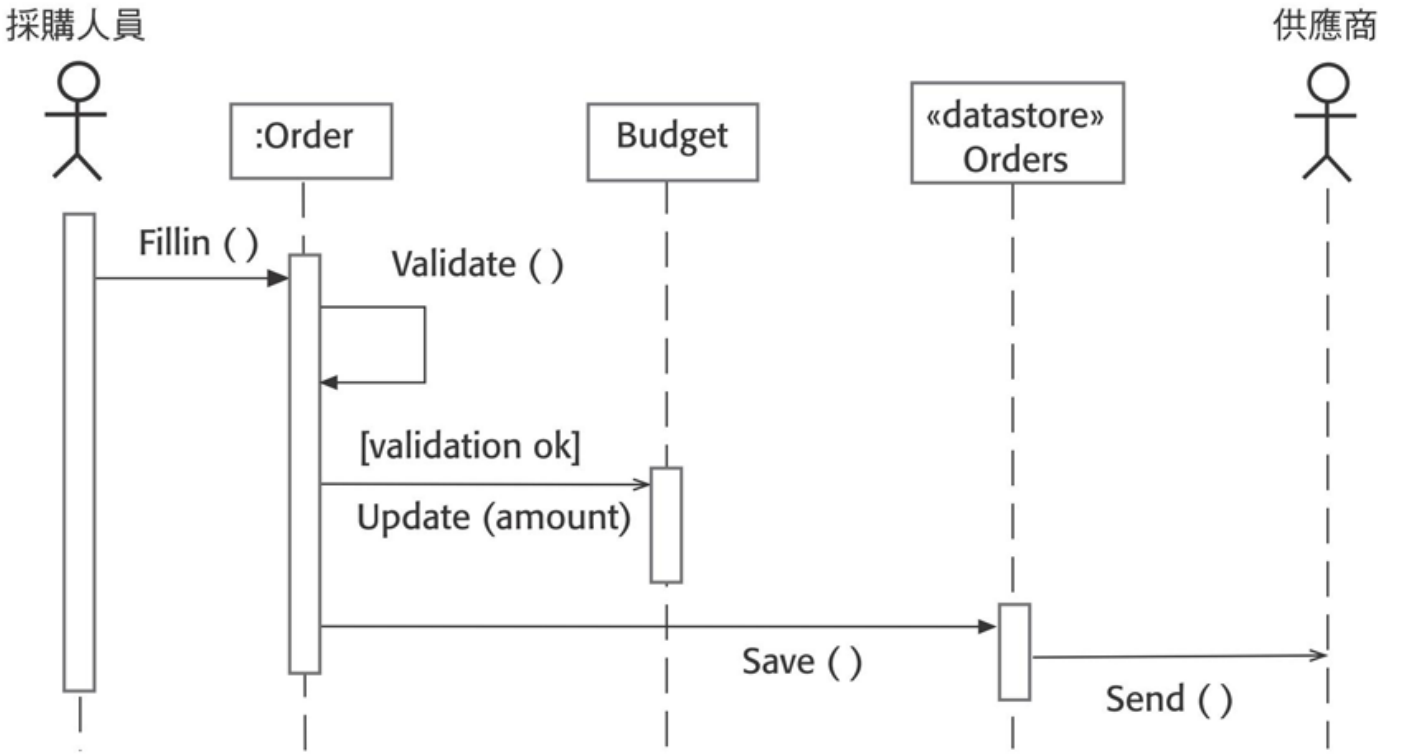


圖 5.15 訂單處理

5.4.2 事件驅動塑模

- 事件驅動塑模 (event-driven modeling) 描述系統如何回應外在和內部事件。它所根據的假設是此系統的狀態個數是有限的，而且事件 (刺激) 可能會觸發狀態轉換到另一個狀態。

- UML使用狀態圖(State diagram)來支援事件驅動塑模，它的狀態圖是根據Statecharts (Harel 1987)。狀態圖會顯示系統狀態以及造成狀態發生轉換的事件，但是不會顯示資料在系統內的流動方式，不過它可能包含在每個狀態要進行計算的額外資訊。

5.4.2 事件驅動塑模

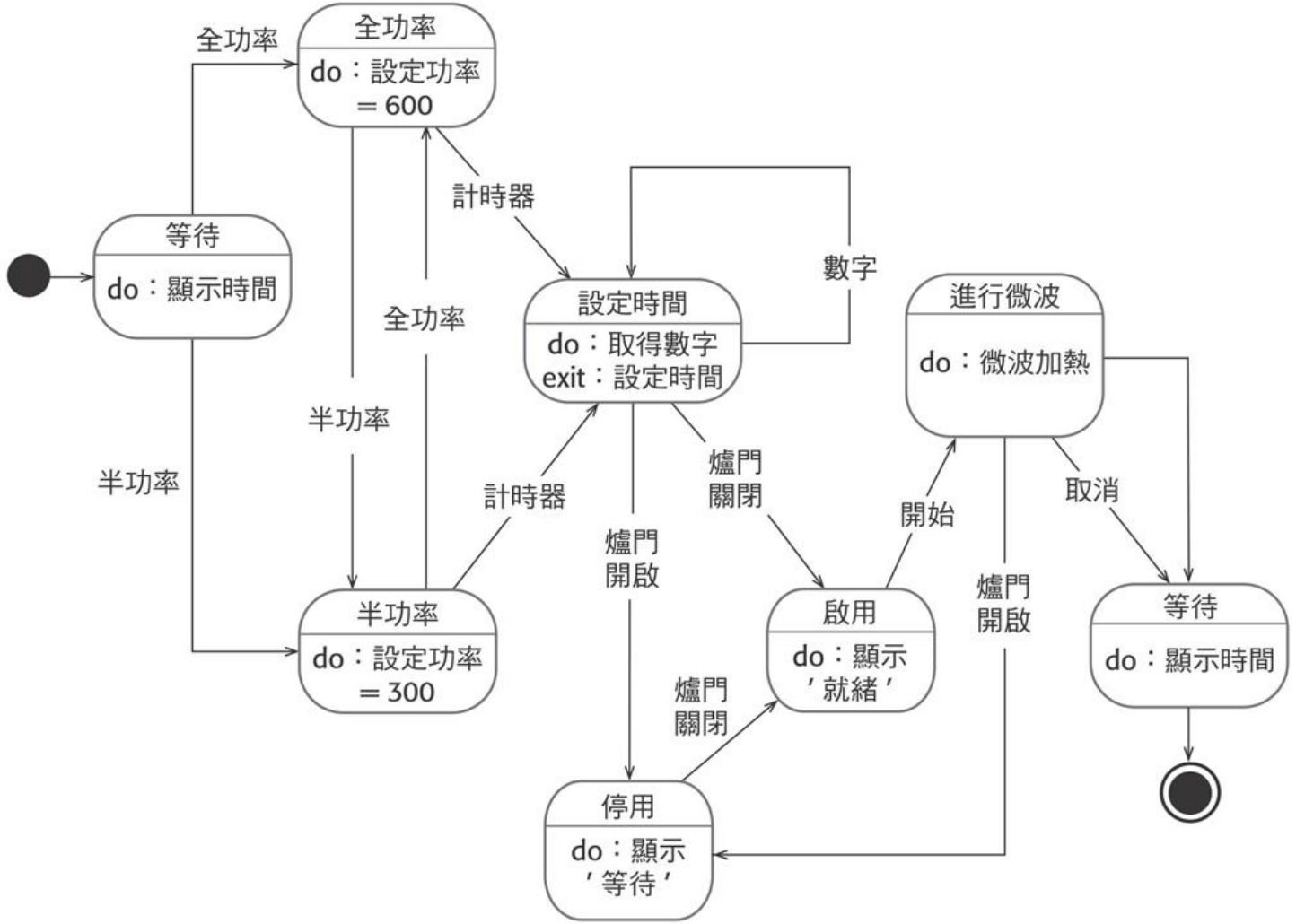


圖 5.16　微波爐的狀態圖

5.4.2 事件驅動塑模

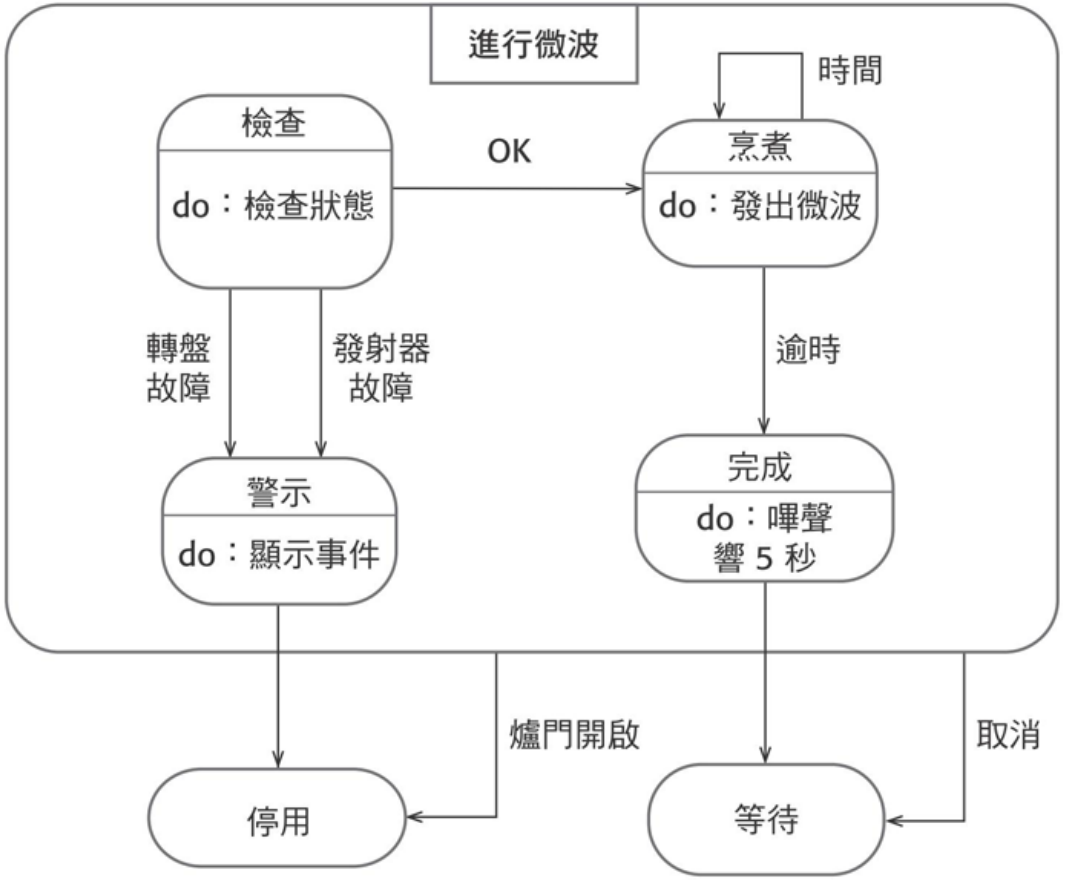


圖 5.17　進行微波狀態的狀態模型

5.4.2 事件驅動塑模

狀態	說明
等待	微波爐等待輸入，螢幕顯示目前時間
半功率	微波爐的功率設為 300 瓦，螢幕顯示「半功率」
全功率	微波爐的功率設為 600 瓦，螢幕顯示「全功率」
設定時間	烹煮時間設為使用者輸入值。螢幕顯示所選的烹煮時間，並且在時間設定後會更新
停用	基於安全理由微波爐被停用，微波爐內部的燈會亮，螢幕顯示「準備中」
啟用	微波爐被啟用，內部的燈關閉，螢幕顯示「準備烹煮」
進行微波	微波爐運作中，內部的燈會亮，螢幕顯示倒數時間。烹煮完成後，發出 5 秒鐘嗶聲，微波爐燈號亮起，發出聲響時螢幕顯示「烹煮完成」訊息
刺激	**說明**
半功率	使用者按下半功率按鈕
全功率	使用者按下全功率按鈕
計時器	使用者按下計時器的某個按鈕
數字	使用者按下某個數字鍵
爐門開啟	微波爐的門開關未關上
爐門關閉	微波爐的門開關已關上
開始	使用者按下開始按鈕
取消	使用者按下取消按鈕

圖 5.18　微波爐的狀態與刺激類型

5.4.3 模型驅動式工程

- 模型驅動式工程 (model-driven engineering, MDE) 是一種軟體開發方法，這種開發程序的主要輸出是模型而非程式 (Brambilla, Cabot, and Wimmer 2012)。
- 接著再從模型自動產生出能在硬體/軟體平台上執行的程式。
- MDE的擁護者認為它能提升軟體工程的抽象層次 (abstraction)，這樣工程師就不需要擔心程式語言或執行平台的細節。

5.5 模型驅動式架構 (MDA)

- 模型驅動式架構 (Mellor, Scott, and Weise 2004; Stahl and Voelter 2006) 是一種針對模型的軟體設計和實作方法，它使用UML模型的子集來描述系統。
- 這裡會建立不同抽象層次的模型。原則上，從高階的與平台無關的模型，有可能在無需人為介入下產生一個可運作的程式。

5.5 模型驅動式架構 (MDA)

- 下列3種抽象的系統模型：

1. 計算獨立模型 (computation independent model, CIM)：

- 描述系統中重要的領域抽象資訊的模型，所以有時CIM也被稱作領域模型 (domain model)。

2. 平台獨立模型 (platform independent model, PIM)：

- 描述系統的運作但不會參考到它的實作。PIM通常是使用UML模型來描述，用來記錄靜態系統結構，還有對外在和內部事件的回應。

3. 平台特定模型 (platform specific model, PSM)：

- 針對每一種應用程式平台，將平台獨立模型轉換成不同的PSM。原則上可能有分好幾種層級的PSM，每一層加上該平台特定的更多細節。

5.5 模型驅動式架構 (MDA)

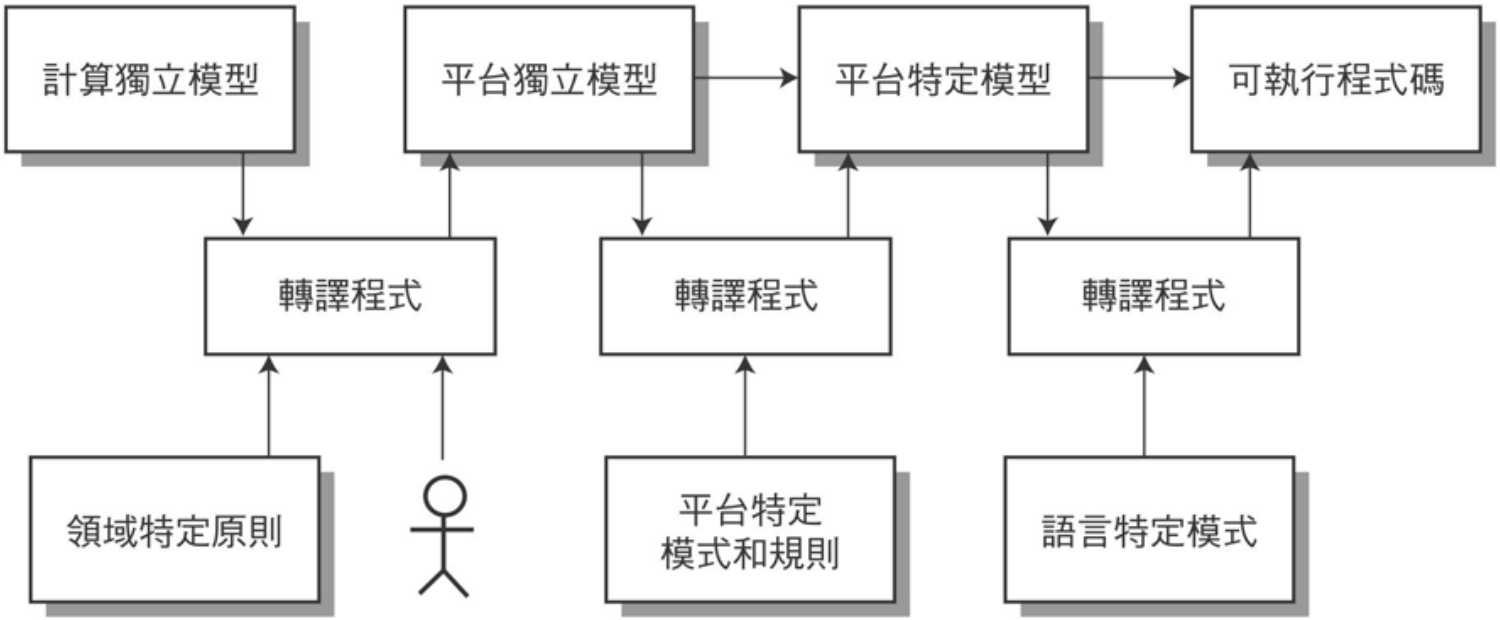


圖 5.19 MDA 轉換動作

- 在實務上，要將模型完全自動化的轉譯 (translation) 為程式碼還不太可能。

5.5 模型驅動式架構 (MDA)

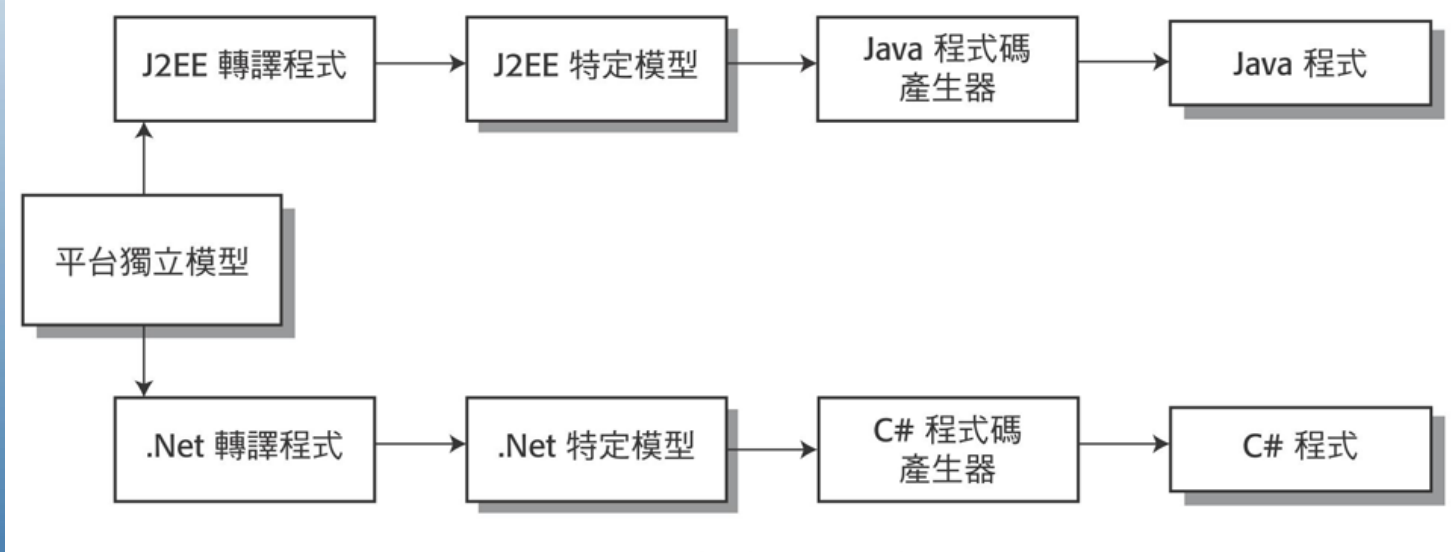


圖 5.20　多重平台的模型

5.5 模型驅動式架構 (MDA)

- 因為不同公司彼此的差異很大，所以沒有市面上的工具可用。因此當公司引進MDA，可能必須製作能運用在地環境提供的設備的特定用途轉譯程式。

- 他們不想開發或維護自有的工具，也不想在工具開發方面依賴一些小公司，因為他們可能會退出市場。

5.5 模型驅動式架構 (MDA)

- 還有一些其他原因，導致MDA一直沒成為軟體開發的主流方法：
  1. 模型是一種能促進討論軟體設計的好方式。不過對討論有幫助的抽象結果，並不一定就是適合實作的抽象結果。
  2. 對於大多數複雜系統而言，實作不是主要的問題。反而是需求工程、保全性和可信賴度、與舊系統的整合，還有測試這些都來得更重要。
  3. 贊成平台獨立 (無關) 的論點只對大型而壽命長的系統有效。對於那些為標準平台 (如Windows和Linux) 開發的軟體產品而言，使用MDA得到的好處，很可能不如引進它和準備工具所花的成本。
  4. MDA發展的期間正好也是敏捷式方法大受歡迎的期間，而後者把大家的注意力從模型驅動式方法轉移開來。