以 Rational Unified Process 開發大型系統

Maria Ericsson

Rational Software 白皮書

TP 156

Rational®
the software development company

目錄

歷程

本白皮書描寫「交互連接系統的系統」，由 Ivar Jacobson、Karin Palmkvist 和 Susanne Dyrhage 於 1995 年 5~6 月發行於 ROAD [1]。本白皮書得益於數個大型系統開發專案的投入，並打算向 Rational Unified Process 5.1 版 [2] 和「統一建模語言」[3] 看齊。

交互連接系統的系統

開發大型系統時，複雜度大大提高。不只需要您瞭解一組更複雜的構件，而且由於您需要管理更大群的資源，所以還會引來額外負荷。本白皮書說明用來幫助控制所新增的複雜度額外負荷的架構型樣。架構模式和 [4] 所討論的其他地方稱爲**交互連接系統的系統**。

此建構在建置超大型或複雜系統時很有幫助，例如指令和控制系統或高度整合的 IT 解決方案。在大部分情況下，這些類型的「超系統」會分割成數個個別組件，每一個獨立開發成個別系統。超系統由一組交互連接的系統實作，它們彼此通訊來履行超系統的責任。其中一個系統代表整體功能，我們稱它爲**上級系統**。其他系統代表整體的一部分，我們稱之爲**子層系統**。上級系統與實作它的子層系統明確區分。不同類型的系統之間的關係也不同；從上級系統的觀點來看，子層系統是子系統，請參閱圖 1。

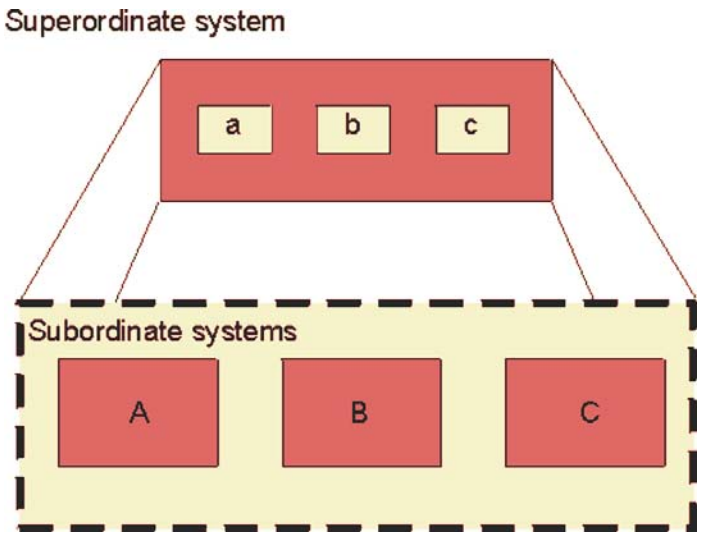

圖 1。上級系統的規格是由交互連接系統的系統實作，其中系統 A、B 和 C 分別是上級系統的子系統 a、b 和 c 的實作。

分隔上級系統與它的子層系統有幾個好處：

- 在所有生命週期活動期間（包括銷售和交貨），子層系統可個別加以管理。
- 將子層系統插入交互連接系統的其他系統中，就可以輕易地使用子層系統來實作其他上級系統。
- 當您開始建置系統時，您不一定知道它是否爲交互連接系統。您可以在開始時先使用「簡易」系統視圖，等到生命週期的後期，再決定您是否需要套用交互連接系統模式的系統。
- 如此一來，您就可以進行子層系統的內部變更，而不需要開發上級系統的新版本。只有因爲重大功能變更時，才需要上級系統的新版本。

每一個子層系統有一組相關聯的構件，彼此之間有明確的可追蹤性。從子層系統的構件集合到上級系統相對應的構件集合之間也保持可追蹤性。每一個子層系統可當作有自己的生命週期階段的個別開發專案加以管理：初始階段、詳述、建構和轉換。

如果您要建置的上級系統非常龐大，可能需要進一步分割子層系統，並將其視爲交互連接系統的系統。

軟體開發生命週期

在 Rational Unified Process 中，開發生命週期是從兩個視景來呈現和討論：管理視景和開發視景，請參閱圖 2。

從管理視景來看，您需要經過四個生命週期階段來開發系統或新一代系統。從開發視景來看，您反覆開發越來越完整的系統版本。您在反覆期間執行的活動，已在 Rational Unified Process 中組成規範集合。每一個規範的焦點在說明某些系統層面，而產生系統模型或文件集合。

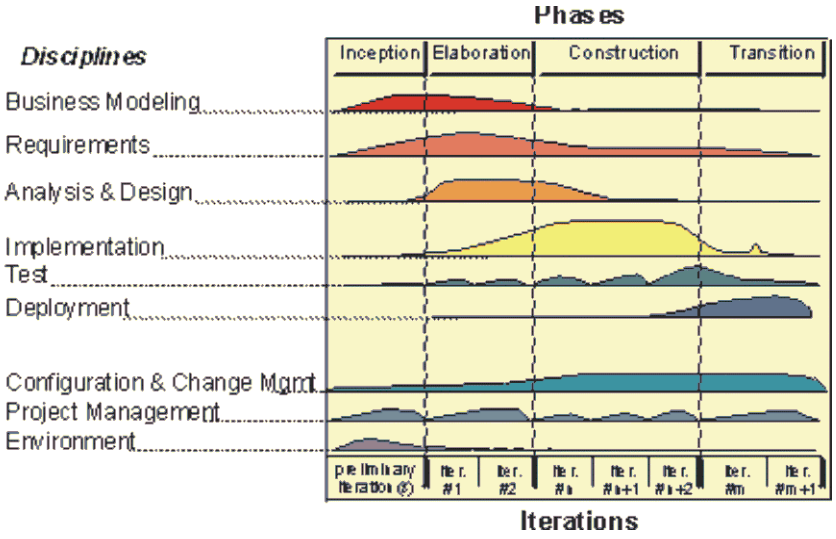

圖 2。反覆模型

將此套用到交互連接系統的系統中，上級系統及其每一個子層系統就會經歷自己的生命週期，且通常作當作個別專案來處理。

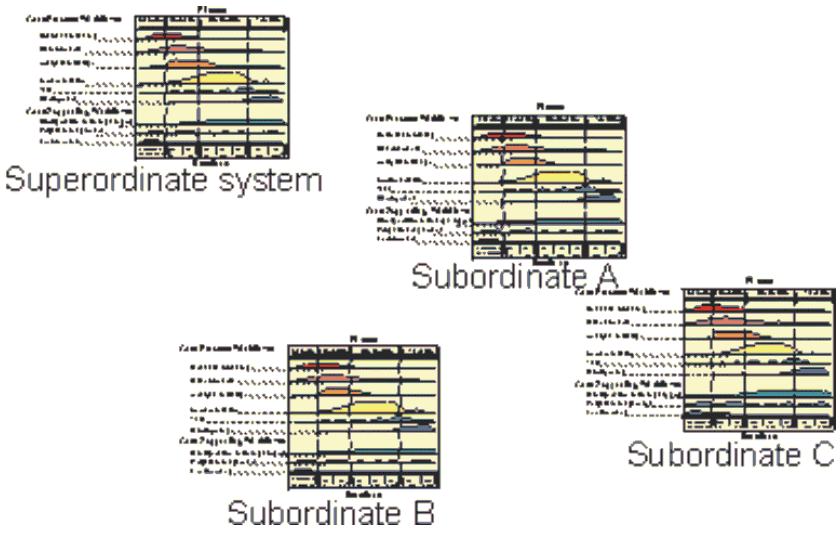

圖 3。不論上級或子層，每一個系統都經歷自己的生命週期。

當然，生命週期有相依關係。要正確管理那些相依關係，是開發交互連接系統的系統所面臨的難題之一。相依關係有下列幾種類型：

- 生命週期具有時間相依性。上級系統的生命週期先開始。當上級系統至少經歷一個反覆，且子層系統的界面非常穩定之後，子層系統的生命週期就可以開始。事實上，您甚至要等到經歷上級系統的一個反覆之後才會知道有哪些子層系統。
- 當子層系統的介面穩定之後，上級系統的生命週期會進入維護期。這表示除非有發生問題需要變更子層系統的介面，否則不會有任何作用中的開發活動。
- 子層系統的介面是由開發上級系統的那些人所擁有。如需介面的詳細資訊，請參閱 [3] 和 [5]。
- 實作子層系統的介面的類別是由開發子層系統的那些人所擁有。

系統開發的工作流程和構件

自然假設上級系統和子層系統可以相同的構件集合開發，及透過相同工作流程開發，與非複合式系統的一般方式相同。在可以展現其作法之前，必須先引進這些構件和工作流程。在 Rational Unified Process 中，我們引進了下列五個規範，請參閱圖 4。

- 商業建模 – 目的是要評估使用此系統的組織，以便更充分的瞭解需求和系統要解決的問題。其結果是一個商業使用案例模型和商業分析模型。此規範爲選用性。如果要運用此系統的組織很簡單，則可能沒有什麼附加價值。
- 需求 – 其目的是要擷取和評估需求，其焦點爲可用性。這產生一個使用案例模型，有代表與系統通訊的外部裝置的參與者，和代表交易序列的使用案例，而產生參與者值的可測量結果。
- 分析和設計 – 其目的爲探索預定的實作環境和它對系統建構的影響。這會產生一個物件模型（設計模型），包括使用案例實現化，以顯示物件如何溝通來執行使用案例的流程。這可能包括類別和子系統的介面定義，從提供的作業方面來指定其責任。這個物件模型也從實作語言、分送等方面做調整，以配合實作環境。有時候可以將分析結果視爲個別模型很有用，我們稱之爲分析模型。

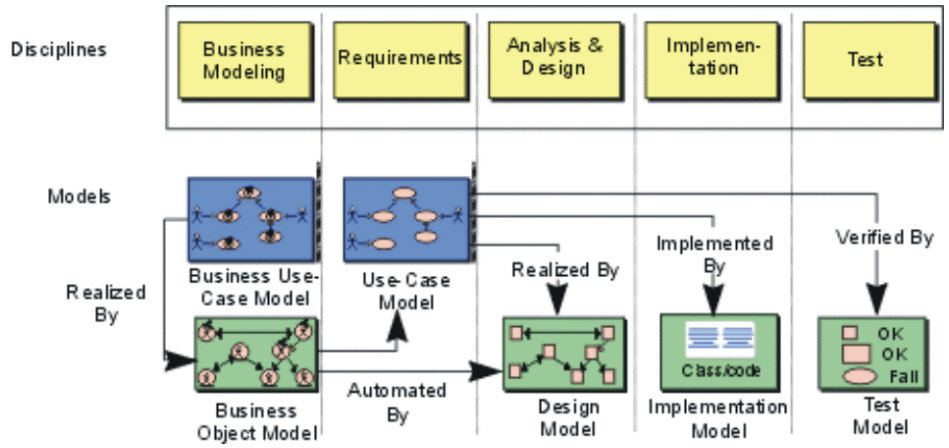

圖 4。每一個規範與一組特定模型相關聯。

- 實作 – 其目的爲在指定的實作環境中實作系統。這會產生程式碼、執行檔和檔案。
- 測試 – 其目的爲確保該系統爲所要的系統，且實作不會出錯。這會產生一個可隨時交付的已認證系統。

交互連接系統的系統之開發

我們的必要工作是定義如何在數個系統之間分配系統責任，每一個系統負責一個定義完善的責任子集。這表示主要目標是要在這些子層系統之間定義介面。完成之後，剩下的工作可根據「區分-解決」原則，針對每一個子層系統個別進行。因此，除了在完成實作之後加以測試之外，這是我們唯一需要對整個系統執行的動作。

分解的準則

因此，您如何決定要不要將系統拆解成一個交互連接系統的系統？以下是您要考量的一些性質：

- 對於複雜的大型系統，您可以將問題分割成一個個小問題，比較容易逐一瞭解。
- 您處理的是實際上分隔的系統嗎？使用舊版系統或舊版架構時，通常屬於這種情形。
- 拆解有助於定義系統組件之間自然而窄小的介面。
- 您可以決定使用一些主要商品現貨 (Commercial-Off-The-Shelf，COTS) 產品來實作系統的一個組件。拆解有助於說明您打算如何使用 COTS 產品。
- 分割可讓您從分散式開發組織中獲得最大利益，並在數個分散各地的團隊之間明確分割工作。

需要考量的風險如下：

- 分解過度會將所有細節的整體問題隱藏起來。
- 若使用實際上分隔的系統或實際上分隔的團隊，您可能會失去任何形式的重複使用，最後產生死板的系統。

組織

若要減輕上述風險，一定要指派一組人來監督整個開發工作。這組人通常稱爲架構團隊，他們的焦點應該放在下列各項主要考量：

- 有一個已定義的整體架構，由子層系統加以遵循。
- 對重複使用和子層系統之間經驗分享的合理重視。
- 明確瞭解要產生的構件以及子層和上級系統構件之間的關係。
- 由所有團隊定義有效的變更管理策略並加以遵循。

架構團隊可以但不一定擁有上級系統的開發權。如需有關組織的徹底討論，請參閱 [6]。

上級系統的生命週期

首先，您可以選擇要執行**商業建模**，以便更充分的瞭解系統的環境定義。在下列條件下，這會具有附加價值：

- 開發人員需要更充分的瞭解組織。
- 組織本身對於其執行業務的方式差異大，且專有名詞及流程需要統一，或者
- 軟體工程結合商業工程重造工作。

另請參閱 [6]。

此工作將產生一個商業使用案例模型和一個商業分析模型。另外，您可以選擇進行有限的商業工程，僅著眼於商業領域的主要概念，並在商業分析模型中記載它們。這通常稱爲領域建模。

當您以一組商業模型來刺激商業發展時，需要開始在整個系統上誘導**需求**。就像對其他系統一樣，我們同樣需要對於交互連接系統的系統有需求建模。使用案例模型是表達這種結果非常直接的方式，請參閱 [7]。查看上級使用案例模型最直接明確的方式就是假設它完全擷取系統的行爲需求。不過，這種情況較爲罕見。由於我們需要實作此系統與其他系統，所以整體系統可能非常複雜。因此，最好不要在這個層次就那麼詳盡。上級使用案例模型通常會提供一張關於系統功能需求的完整簡圖。在此層次上不需要太過詳細，因爲在每一個實作的子層系統內會執行詳細建模。許多需求也不一定會出現在切割數個子系統的上級使用案例中。這類需求可稱爲子系統的「本端」需求。

分析和設計的目的是爲了達到系統的健全架構，這對於交互連接系統的系統當然很重要。上級系統的開發人員必須達到子層系統的健全結構，但完全不需要干擾其內部結構。因此我們將使用子系統，建立成小型組件的系統分區模型。爲了取得一組正確的子系統，以及對於如何在這些子系統之間分配上級系統的責任有初步瞭解，我們開發了分析模型。在執行高階使用案例時，分析類別應該代表系統中各項工作所扮演的角色。因此，與高階使用案例模型相比，分析模型提供的是完整物件結構的簡圖。

功能相關的分析類別會在子系統中形成群組。因此我們得到的子系統結構不切實際，因爲它只以功能準則爲基礎，例如，它並未將分送需求納入考量。舊版系統的存在常常爲具有影響力的因素。舊版系統或許可以履行分析模型所定義的一些或許多責任。這類系統的存在甚至造成分析中的責任重新分割，使您可以達到現有功能重複使用的最大效率。

此設計結果的子系統結構可能與我們在分析期間以功能準則爲基礎而定義的子系統結構差別很大。因此，結果我們得到的設計子系統結構，其中每一個子系統將由子層系統實作，請參閱圖 5。爲了使每一個子層系統能夠個別地繼續其開發工作，所以我們爲每一個子系統定義了介面。事實上，定義介面是在上層上執行的最重要活動，因爲介面爲子層系統的開發提供了規則。它並沒有定義設計類別，您唯一要做的是定義設計子系統的介面。

除了一些探索系統特定技術層面的原型化工作之外，在上級系統的生命週期中，沒有任何**實作**。

最後規範是**測試**，在此情況下，是指不同子層系統組合時的整合測試，它也測試協同作業中的交互連接系統是否有根據每一個上級使用案例的規格來執行每一個上級使用案例。

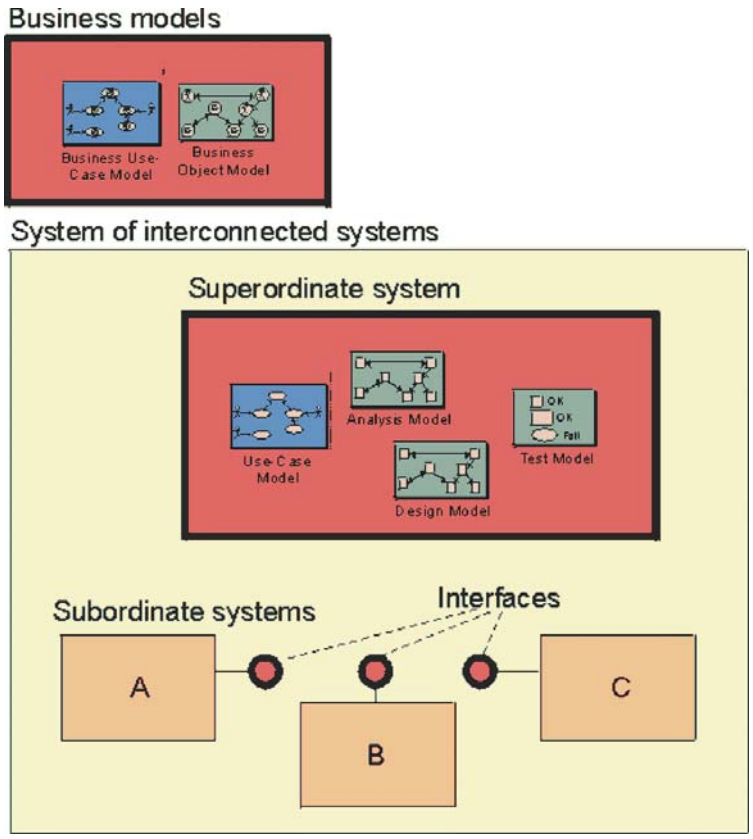

圖 5。上級系統是由一組模型來描述，其中定義在高階設計模型中的子系統將由子層系統實作。子層系統的介面是由上級系統所擁有。

爲了說明您要如何使用上級系統，這裡有數個反覆計劃範例；一個是針對上級系統生命週期初始階段的反覆，一個是針對詳述階段的反覆。我們使用活動圖來說明反覆計劃。這些圖表中的動作狀態對應於 Rational Unified Process 中所定義的活動。

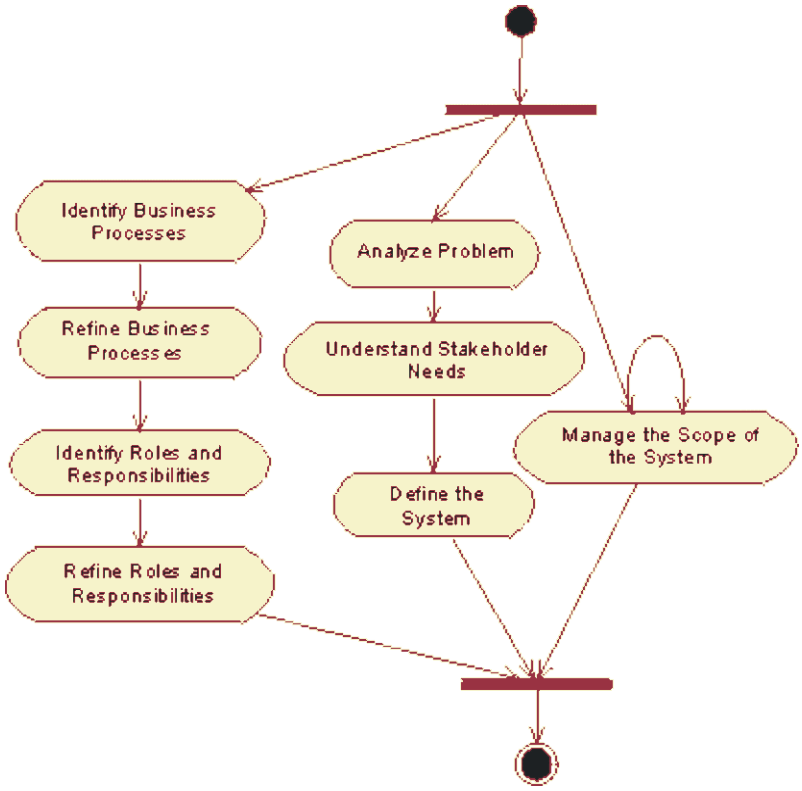

圖 6。說明上級系統初始階段範例反覆計劃範例的活動圖。

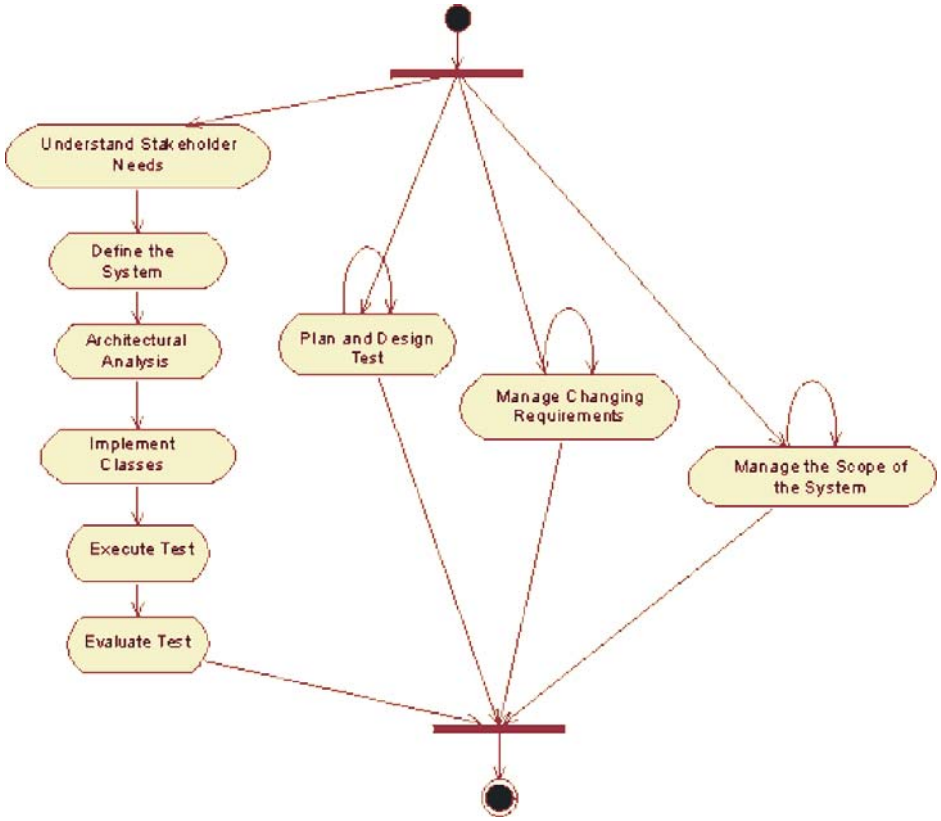

圖 7。說明上級系統詳述階段反覆計劃範例的活動圖。這裡的動作狀態是「實作類別」，因爲您可以執行一些有限的原型實作來探索系統的技術層面。

子層系統的生命週期

每一個子層系統通常是以黑箱作業發展，將它所溝通的其他系統視爲參與者。如上所述，您可以爲每一個這種系統執行一般活動和開發一般模型。如果在上層的模型有詳細定義，您就可以在不同等級的模型之間取得完整遞回，但如上所述，這種情形較爲罕見。

若爲子層系統，您可以執行**需求**相關活動。上級系統的介面和使用案例將成爲您瞭解子層系統的界限及其參與者是誰的主要來源。

在執行子層系統的**分析和設計**時，上級系統中所定義的介面及高階使用案例將成爲您的界限條件。

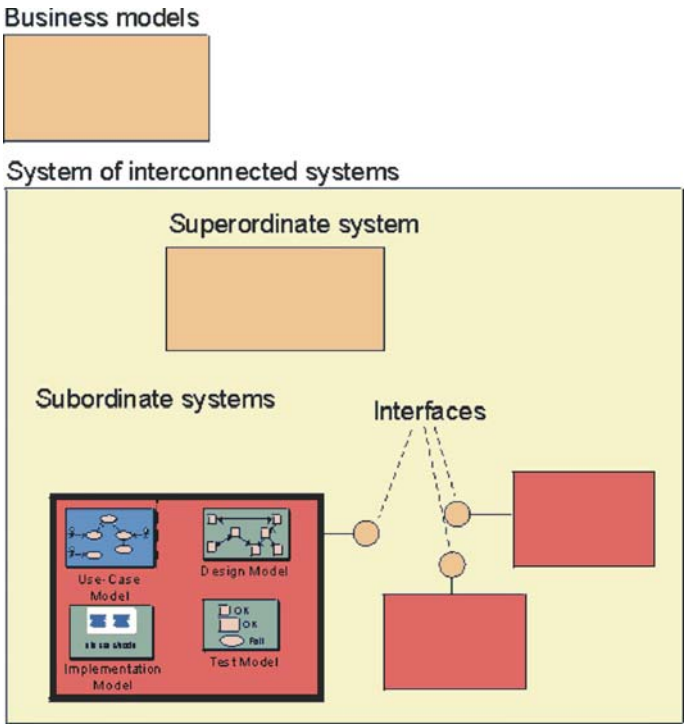

圖 8。子層系統是由它們自己的模型集合來描述。

爲了說明您要如何使用子層系統，這裡有來自其生命週期的兩個反覆計劃範例。

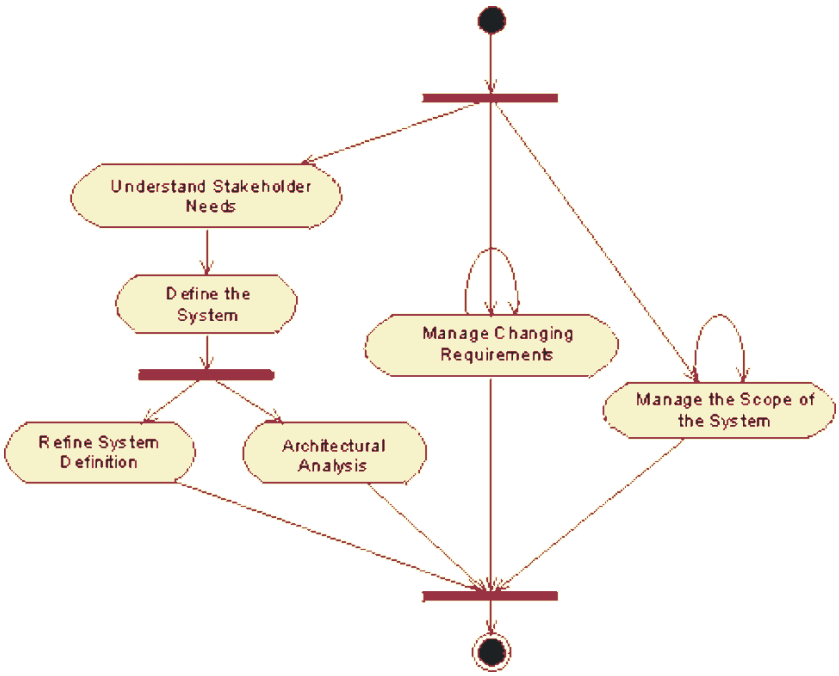

圖 9。此子層系統的初始階段反覆計劃範例。這是不完整反覆，因爲沒有產生任何執行檔。

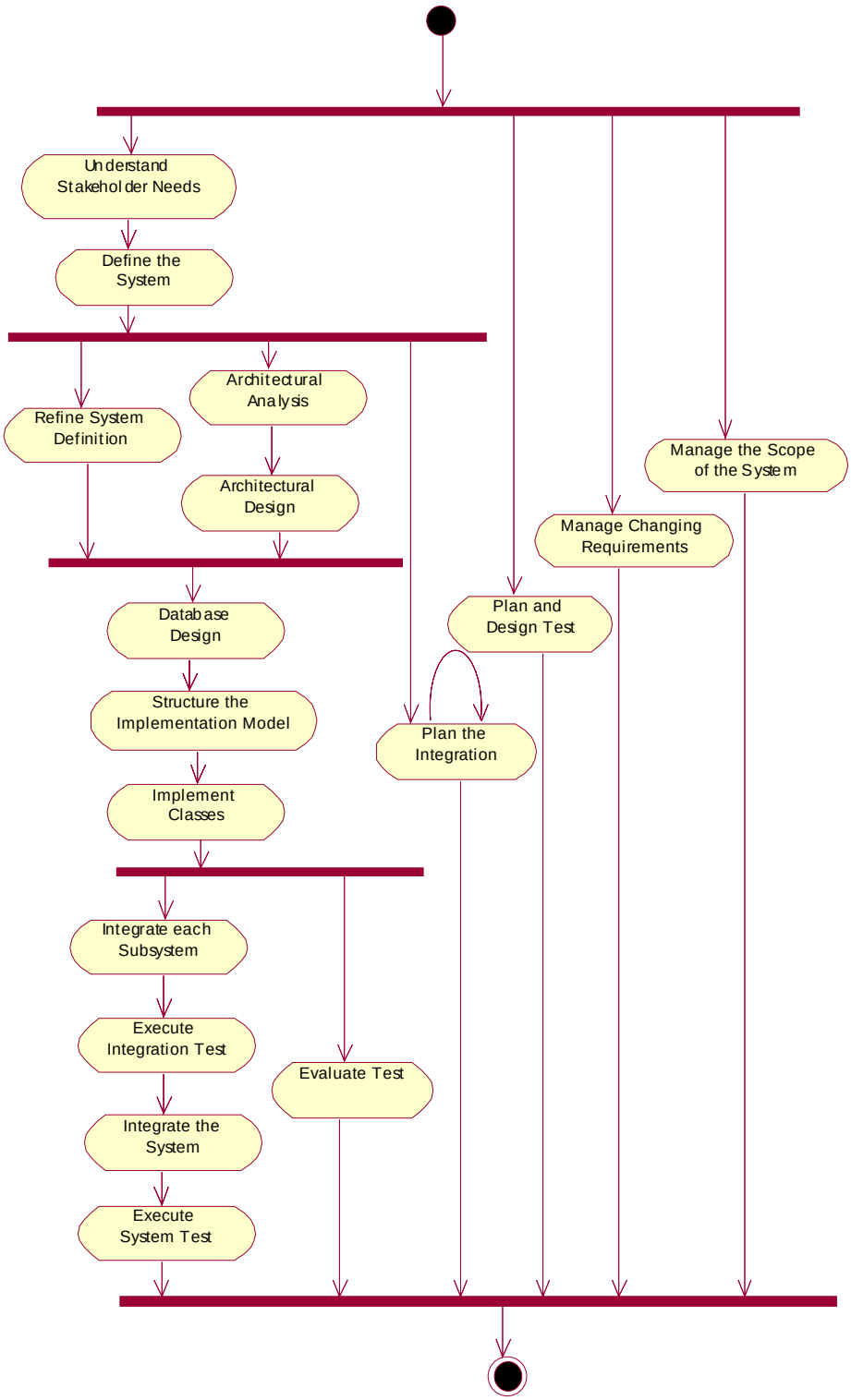

圖 10。子層系統的詳述反覆計劃範例。詳述的焦點在於完成已修正的系統定義和架構。

交互連接系統的系統中的使用案例

在交互連接系統的系統中，您應該爲每一個系統（包括上級和子層）建置一個使用案例模型。它們的相依方式如下（另請參閱圖 11）：

- 上級系統中的高階使用案例（不一定會，但通常會）分割成子系統。每一個「分割」使用案例變成其子層系統模型中的使用案例，請參閱圖 11。
- 從子層系統的觀點來看，其他子層系統是其使用案例模型中的參與者，請參閱圖 12。

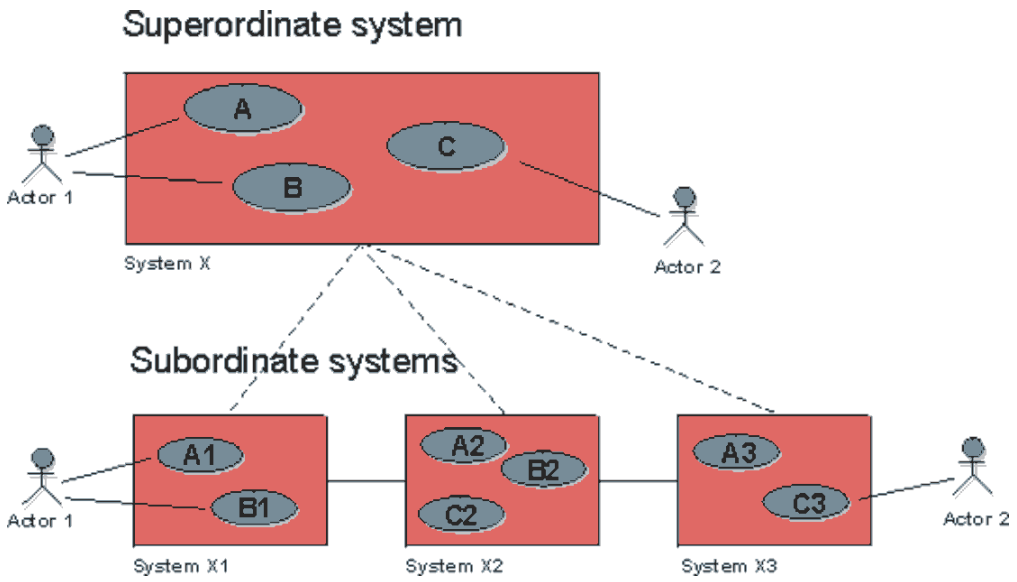

圖 11。上級系統中的高階使用案例之間的關係，和子層系統中詳細的使用案例。

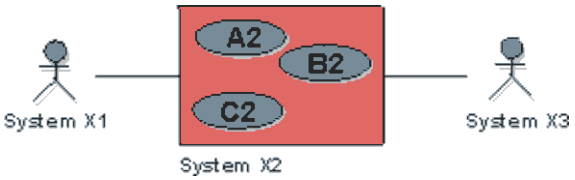

圖 12。在子層系統 X2 的使用案例模型中，其他子層系統 X1 和 X3 被視爲參與者。

對於描述上級系統的使用案例，有一些特殊考量。就某種意義來說，由於您會重新描述每一個子層系統的所有需求，因此，太過深入去探究這些使用案例是沒有意義的。在正常情況下，通常只需要寫下高階使用案例事件流程的逐步大綱，而不需要加以詳述。

在這個使用案例模型中，您不應該使用任何使用案例關係（一般化、延伸、併入）。一般而言，因爲下列原因，它沒有附加價值：

- 您不會詳細描述高階使用案例，因此您不需要擔心文字會出現在數個地方。
- 當您將高階使用案例「分割」成子層系統時，您還是會建立資訊結構。將此結構與其他結構化機制混在一起，會令人混淆不清。

不過，如果您的目標是要在交互連接系統的系統中尋找可重複使用的元件，則爲重大例外。將上級使用案例模型結構化以尋找通用使用案例，是尋找可重複使用的元件的有效方法。若要取得本主題的詳細資料，請參閱 [6]。

交互連接系統的系統中的設計模型

在交互連接系統的系統中，包括上級和子層在內的每一個系統都應該有它自己的設計模型。設計模型在下列方面是相關的：

- 上級系統設計模型中的子系統定義子層系統的界限。
- 您在上級系統的子系統上定義的作業是您定義子層系統介面的主要來源。

上級系統設計模型的描述比子層設計模型更爲簡略。您會產生下列各項：

- 子系統，簡略描述。
- 使用案例實現化，就子系統如何分工合作而論。記載這些高階使用案例實現化的一般方式是繪製序列圖。它的作法是產生這些圖表，讓您定義高階使用案例如何分割成子層系統，請參閱「圖 13」。
- 子系統的作業。
- 子系統的介面定義。

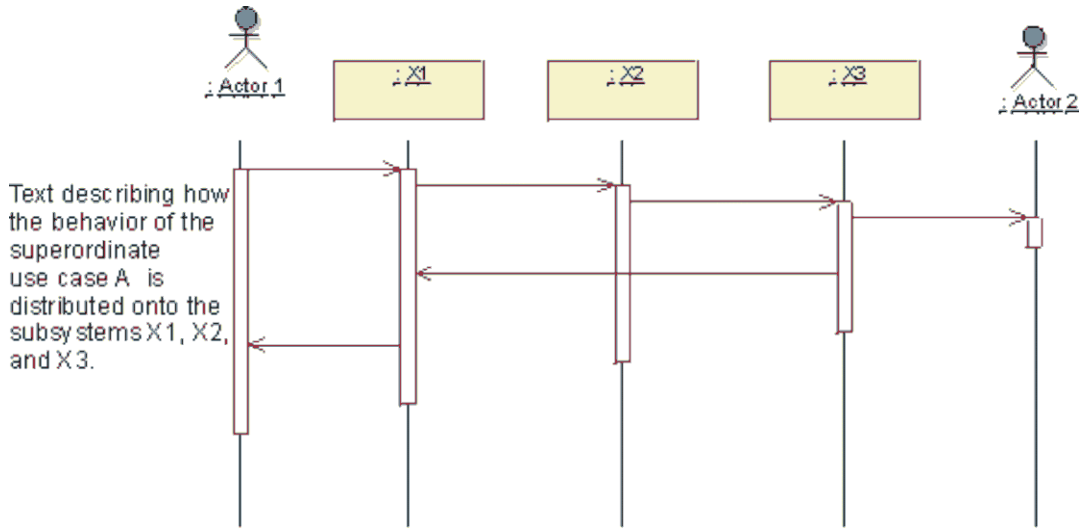

圖 13。上級使用案例 A 的實現化的序列圖。

交互連接系統的系統中的資訊集合

大部分組織花了很多心思在瞭解如何管理其構件及正確瞭解其相依關係。在上一節，我們特別討論到上級和子層使用案例模型和設計模型之間的相依關係。其實還需要考量一般相依關係的問題。

當系統經歷一次生命週期時，您會產生構件並將它們組織成資訊集合 [8]，請參閱圖 14。這些集合是依據「共同」形成的構件加以組織。

- 您可以根據要建置的應用程式類型，自訂每一個集合的確切內容，但集合維持不變。
- 您需要瞭解集合之間的相依關係，才能以有效的方式維護構件之間的可追蹤性。

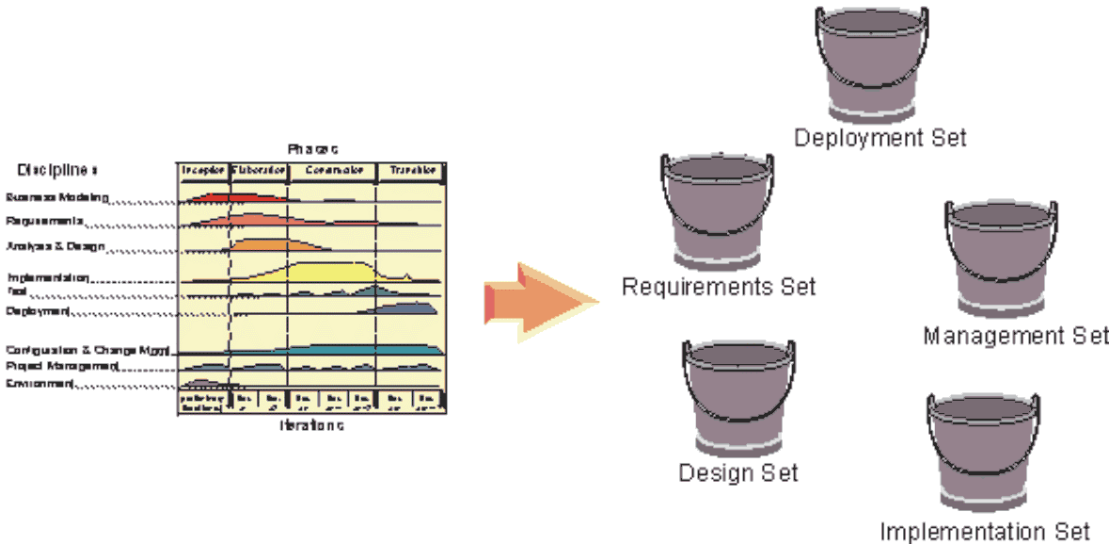

圖 14。系統的生命週期將產生資訊集合。

在交互連接系統的系統中，上級和每一個子層系統將產生它自己的資訊集合，請參閱圖 15。

- 子層資訊集合與其相對應的上級資訊集合之間有相依關係。
- 在子層系統之間，相對應的資訊集合的內容類型可能不同，這是因爲應用程式類型可能不同。
- 除了履行上級系統中定義的相同子系統介面之外，相對應的子層資訊集合應該是獨立的。

應該花最少的工夫去維護上級系統和子層系統之構件之間的可追蹤性。應該優先維護系統內的可追蹤性。

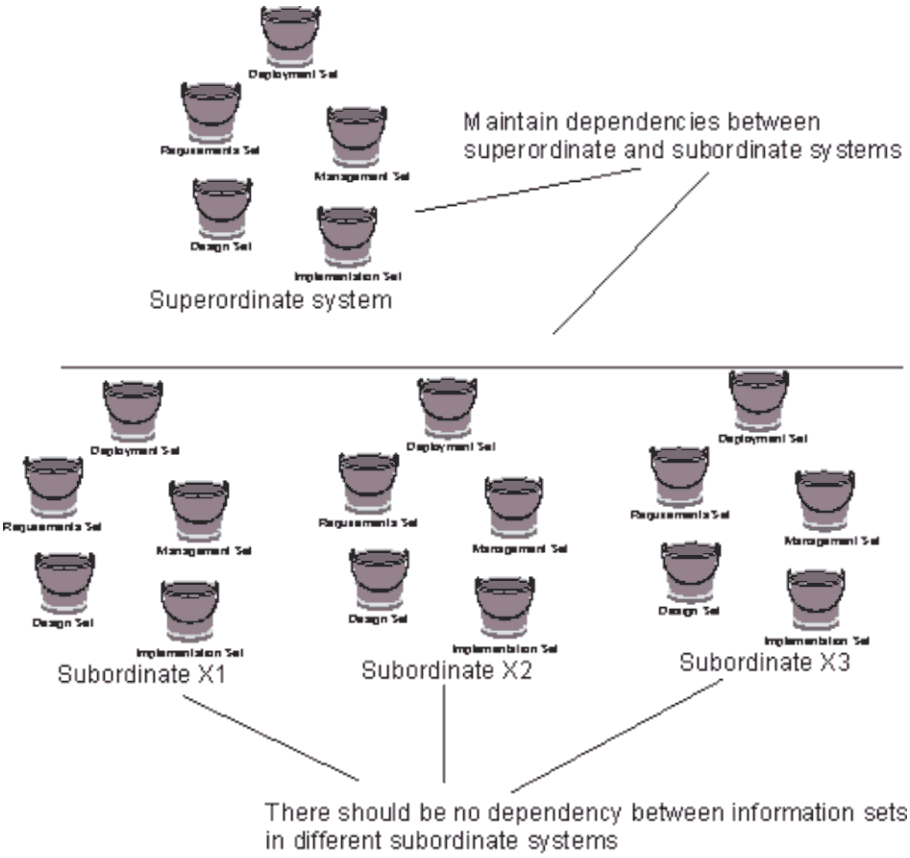

圖 15。在交互連接系統的系統中，每一個系統將產生它自己的資訊集合。

交互連接系統的系統中的架構

在交互連接系統的系統中，包括上級和子層在內的每一個系統都應該已定義它自己的架構。

對於上級系統而言，架構文件應該討論下列各項：

- 上級系統的主要使用案例或實務。
- 交互連接系統的系統的分層。
- 如何處理子層系統之間的重複使用，以及要重複使用的內容。
- 主要機制及其實作，其通用性足夠讓所有子層系統使用。例如，所有子層系統應該使用一般機制來進行溝通、錯誤報告和錯誤管理，否則上級系統將無法以同質系統來操作。

對於子層系統而言，架構文件應該釐清下列各項：

- 在交互連接系統的系統內，子層系統扮演的角色。
- 子層系統的主要使用案例或實務。
- 子層系統如何使用爲交互連接系統的系統定義的分層結構。另一種說法是，您需要定義子層系統如何扮演在上級系統的分層架構中已定義給它的角色。
- 將使用的一般主要機制及使用的方式，以及新增哪些特定應用程式專用的主要機制。
- 如何套用重複使用。尤其，在兩個以上的子層系統之間哪些是一般子系統，以及建置哪些機制讓子層系統進行溝通。

系統之間的關係

您知道一般系統開發活動也可以套用到交互連接系統的系統所實作的系統中。這是有好處的，因爲它表示您不需要以非常不同於其他系統使用的方式來處理這類系統。您也可以利用其他子層系統的形式，將上級系統與它的實作妥善分隔。在交互連接系統的系統中，每一個系統有它自己的生命週期。由於每一個系統有不同的性質，您可以利用開發流程的變化性來產生它們。就 Rational Unified Process [2] 而論，對於每一個系統您應該有不同的開發案例。

最後請注意，在交互連接系統的系統中，相關系統之間存在著獨立性：

首先，請看子層系統。每一個子層系統在上級系統的設計模型中實作一個子系統。子系統是依賴彼此的介面而非明確地依賴彼此，請參閱圖 12。因此，您可以交換一個子系統與它的新版本，而不會影響其他子系統，只要新的子系統符合相同介面即可。您就可以讓子層系統之間有完全相同的關係。每一個子層系統將它的周圍環境視爲一組介面。這表示您可以交換系統，只要新系統對其他系統扮演相同角色即可，也就是說，只要它可以用相同一組介面來表示即可。系統參考彼此介面的方式，是由子系統與上級模型的介面之間相對應的關係指定。

在子層系統的使用案例模型中，與它互動的其他子層系統的介面，是以參與者來表示。您可以說子層系統是把其他系統的介面當作是由相對應的參與者所提供，因此，不必直接參考其他系統，請參閱圖 16。請注意，介面 B 出現在圖 12 的數個地方，這表示它實際上是上級系統的子系統與相對應的子層系統所參考的相同介面。

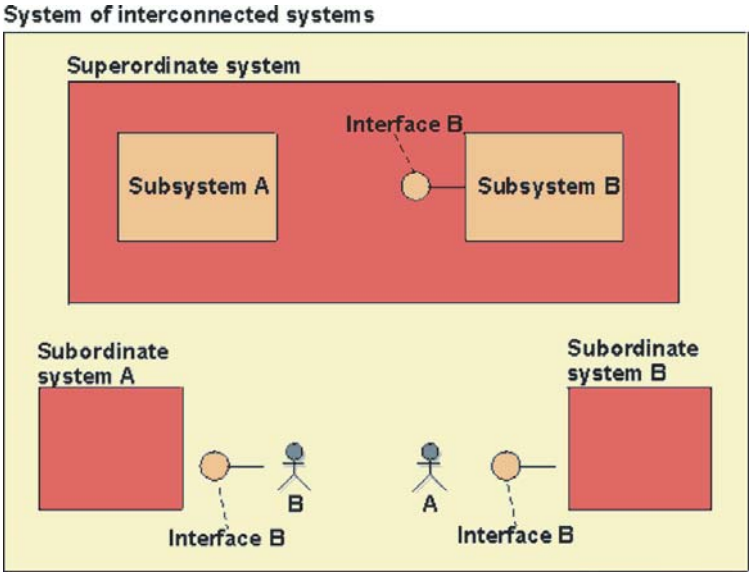

圖 16。上級系統的子系統只有透過其介面才彼此依賴。因此，實作的子層系統會得到相同的獨立性。在上級系統的模型中，子系統 B 提供介面 B 給其他系統。因此，相對應的子層系統 B 需要提供相同介面 B 給其他子層系統。

上級系統又如何呢？它與其子層系統之間的關係如何？在下列方面，它是獨立於它的實作系統之外：每一個實作系統只是我們在上級系統模型中指定的實作而已，而不是其規格的一部分。爲了實用起見，您必須在不同層次定義系統之間的可追蹤性，以便追蹤需求，而最「井然有序」的作法就是只在介面之間定義這類鏈結，請參閱圖 11。事實上，您甚至可以說，子層系統只不過是提供上級模型中已定義之介面的實作而已。

然而，對於非如此簡單範例的系統而言，這是不夠的。介面只指定在特定互動時間點進行的活動。子層系統可以有數百個介面，每一個介面有數十個作業。在介面說明中，使一個介面的輸入與另一個介面的一或多個輸出相關是不切實際的。這就是爲什麼您需要使用案例來解說子層系統的語意。

您可以推斷，當交互連接系統的系統實作一個系統時，每一個相關的系統是獨立於其他系統之外的，但他們非常依賴彼此的介面。這提供您一個非常好的平台，來並列開發子層系統。

應用程式區域

交互連接系統的系統的架構和建模技術可用於不同類型的系統，例如：

- 分散式系統
- 大型或複雜的系統
- 結合數個商業領域的系統
- 重複使用其他系統的系統
- 系統的分散式開發

也可以顛倒此情況：從一組現有的系統中，我們可以組合系統來定義交互連接系統的系統。事實上，在某些情況下，這就是大型系統在其發展初期的發展方式。您知道自己有可以交互連接的系統，這麼做會建立一個「大型系統」，比兩個個別系統有更多附加價值。

事實上，對於可以將系統的不同部分視爲自己一部分的任何系統，建議將它定義爲交互連接系統的系統。即使它今天是單一系統，譬如因爲分散式開發、重複使用原因或客戶只需要購買其中一部分，以後也需要將系統分割成數個個別產品。

總而言之，我們將進一步瞭解一些案例，這些案例可使用交互連接系統的系統架構。我們將在每一個範例中表示，有問題的系統會*同時*視爲單一系統*和*一組個別系統，表示它應該當作交互連接系統的系統所實作的一個上級系統來處理。

大型系統

電話網路可能是全球最大的交互連接系統的系統。這是需要兩個以上的系統層次來處理複雜性的一個絕佳範例。它也是最上層上級系統由標準化主體所擁有，故不同競爭公司開發的一或數個子層系統必須符合此標準的一個案例。這裡，我們將討論行動電話網路 GSM (Global System of Mobile Telephony)，說明以交互連接系統的系統來實作大型系統的好處。

大型系統的功能常常結合數個商業領域。例如，GSM 標準涵蓋整個系統，從呼叫訂閱者到被呼叫訂閱者都有。換句話說，它同時包含行動電話和網路節點的性能。因爲系統的不同組件甚至爲不同類型之客戶個別購買的產品，所以也應該視爲他們自己的系統。例如，開發全套 GSM 系統的公司將行動電話賣給訂閱者，將網路節點賣給電話操作員。這是把 GSM 系統的不同組件當作不同子層系統的原因之一。另一個原因是，要開發像 GSM 這麼大型而複雜的系統成爲單一系統需要花費的時間太長；不同組件必須由數個開發團隊並列開發。

在另一方面，因爲 GSM 標準涵蓋整個系統，所以也需要將整個系統視爲上級系統。這可幫助開發人員瞭解問題領域，及不同組件彼此如何相關。

分散式系統

對於分散在數個電腦系統之間的系統，交互連接系統的系統架構非常適合。從定義上來看，分散式系統一定包含至少兩個組件。因爲定義完善的介面在分散式系統中是必要的，所以這些系統也非常適合以分散方式*開發*，也就是說，由數個自主開發團隊並列工作。分散式系統的子層系統甚至可以當作自己的產品來銷售。因此，將分散式系統視爲一組個別系統是合乎常理的。

分散式系統的需求通常涵蓋整個系統的功能，有時候並不預先定義不同組件之間的介面。此外，如果開發人員不熟悉該問題領域，不論其分散方式如何，他們首先必須考量整個系統的功能。將它視爲單一系統，有兩個很重要的原因。

舊版系統的重複使用

大型系統幾乎經常重複使用舊版系統。舊版系統可描述成子層系統。您可以「重新設計」使用案例模型及舊版系統的分析模型，以瞭解它在上級系統的更大型環境定義中如何運作。這些重新設計的模型不一定是完整的，但至少它們需要涵蓋舊版系統中對交互連接系統的剩餘系統的功能有直接影響的功能，或可能需要修改的功能。

組合套件的使用

系統可以是兩個以上組合套件的整合和自訂。企業資源規劃（Enterprise Resource Planning，ERP）系統就是一個很好的例子。許多 ERP 系統都是子層系統的組合，例如 MRP (Material Resource Planning)、Inventory Management、Supply Chain Management 等等。其他領域也有類似組合，例如人力資源或薪資申請。它們類似組合系統，您必須列舉，並與其他標準套件交互連接，才能獲得完整的系統。若要瞭解套件集合共同執行些什麼，您需要上級系統。這是金融團體的許多客戶今天所面臨的情況。

摘要

本白皮書介紹交互連接系統的系統的架構型樣。此建構允許不止在一個模型內遞回，它還把每一個子系統視爲具備本身條件的一個系統，可在每一個系統的所有構件集合之間遞回。所介紹的架構已使用於數個通訊系統所實作的系統。每一個相關系統由它自己的模型集合加以描述，與其他系統的模型分開。

使用此技術的優點顯而易見：您可以著手處理更複雜的問題，並使用區分 - 解決 (divide and conquer) 技術來瞭解問題。然而，缺點是您會有更多經常性不同步排程的風險。您也看到這樣的例子：組織發現自己很難對上級系統運用反覆生命週期，因而冒著將風險推向上級系統生命週期末端的危險。您也需要注意是否有遵循合理有效的重複使用策略，避免開發一組「死板」的系統。

給定的範例說明交互連接系統的建模系統的架構在許多不同的應用程式區域中很有幫助。事實上，您可以對於可將不同組件視爲本身系統的任何系統使用建議的架構。

參照

[1] Jacobson, I.; Palmkvist, K.; and Dyrhage, S., *Systems of Interconnected Systems*, ROAD, 2(1), 1995.

[2] *Rational Unified Process* 5.1 版。

[3] Rumbaugh, J.; Booch, G.; Jacobson,I., *UML Reference Manual*, Addison Wesley Longman, 1999.

[4] Herbert A. Simon, *The Sciences of the Artificial*, MIT Press, 1981.

[5] Jacobson, I.; Bylund, S.; Jonsson, P., *Using Contracts and Use Cases to Build Plugable Architectures*, Journal of Object-Oriented Programming, May/June, 1995.

[6] Jacobson, J.; Griss, M.; Jonsson, P., *Software Reuse – Architecture, Process and Organization for Business Success*, Addison Wesley Longman, 1997.

[7] Jacobson, I., *Use Cases in Large-Scale Systems*, ROAD, 1(6), 1995.

Rational
the software development company

兩個總公司：

Rational Software
18880 Homestead Road
Cupertino, CA 95014
電話：(408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
電話：(781) 676-2400

免付費專線：(800) 728-1212
電子郵件：info@rational.com
網址：www.rational.com
國際辦事處：www.rational.com/worldwide