從元件規格到實際元件實施之對映研究：一個基於非同步訊息呼叫的方法

Research for mappings from component specifications to component implementations：an approach based on asynchronous message call

林至中 陳建霖
Jyh-Jong Lin Chien-Lin Chen
銘傳大學資訊管理研究所
Institute of Information Management, Ming-Chuan University
Taoyuan, Taiwan, R.O.C

摘要

元件式軟體工程(Component-based Software Engineering)已逐漸取代傳統的軟體工程，成為發展企業軟體的新一代軟體工程典範。它有著組合(composition)及重用(reuse)的特性。而元件式軟體工程必須要有元件塑模方法論在背後支持，才能夠將企業的需求轉換成最終的元件，當使用元件塑模方法論最後產出元件規格後，下一步就是實際將所描述的元件規格用業界各種軟體元件模式標準(例如 COM+、EJB、CORBA 等)來實作(implementation)。而在這樣的實作上必須要將所描述的元件規格(component specification)圓滿地表現出來，其關鍵就是要有良好且完整的對映(mapping)。此外，由於網際網路的蓬勃發展，在軟體元件架構上也已邁向了分散式物件運算的架構，而在一個成功的分散式運算中，是經常需要非同步訊息(asynchronous message)傳遞的。所以本文主要就是對於從元件規格到實際元件實施的對映上，加入非同步訊息傳遞的考量，進而提出一個新的對映模式。並且於文章的後段會使用一個線上數位學習課程訂購系統來作為展示此對映研究構思的例子。至於在元件塑模的方法論上，本文選擇較優良的 UML Components 作為對映研究的方法論，但由於 UML Components 方法論仍然存在些許缺陷，所以在考量不影響元件於原 UML Components 方法論的系統架構(system architectures)之分層模式下，本文實際上採用的是經過擴充改善的版本。在實際元件實施上，則是採用 Sun 公司的 EJB(Enterprise Java Beans)軟體元件模式標準，主要是因為 EJB 是 J2EE(Java 2 Enterprise Edition)的核心，而 J2EE 支援了較為廣泛應用於現有企業系統的分散式運算服務(例如

RMI/IIOP、XML-RPC、CORBA、SOAP 等等)。透過此新對映模式，當業界在元件軟體發展上是使用 UML coponents 及 EJB，且有非同步訊息傳遞考量之需求時，就能夠良好且快速的完成實際實作(implementation)。

關鍵字:元件式軟體工程、對映、非同步訊息、UML Components、 EJB

Abstract

Component-based Software Engineering has gradually replaced the traditional software engineering. With the trait of composition and reuse, it has become the paradigm of new software engineering techniques for developing enterprise software. Component-based Software Engineering demands the support of a component modeling methodology that enables transferring the enterprise requirements into the final component. After using a component modeling methodology to produce component specification, the specification can be practically implemented in a variety of standards of software component model (e.g., COM+、EJB、CORBA) in the enterprise. In order to successfully implement the specification, excellent and complete mapping is a key factor. Furthermore, due to the thriving development of Internet, the architecture of software component has matched that of distributed object operation. Since a successful distributed operation frequently requires the transfer of asynchronous message, this study mainly focuses on mappings from component specifications to component implementations that augment the consideration about the transfer of asynchronous message to come up with a new model of mapping. To illustrate, an on-line e-Learning curriculum order system will be used for demonstrating the mapping idea. For completing the component modeling methodology, this paper adopts an amended edition of the original UML Components methodology. In the context of the practical implementation of component, this paper adopts EJB (Enterprise Java Beans) software component model standard. The reason is because EJB is the core of J2EE (Java 2 Enterprise Edition), and J2EE supports the distributed operation services (e.g., RMI/IIOP、XML-RPC、CORBA、SOAP) that are widely applied by many enterprise systems. With this new mapping model, when enterprises use UML Components and EJB on component software development, and any systems have requirements of transferring asynchronous message, the implementation can be more effective.

Keyword: Component-based Software Engineering, mapping, asynchronous message, UML Components, EJB(Enterprise Java Beans)

壹、緒論

一、研究背景與動機

過去以來，資訊科技在支援資訊硬體產業上突飛猛進，電子工程師可以透過組裝，測試良好的零件，達成十分可靠且穩定的產出，像過去，我國的資訊硬體產業的成功模式在於能以高品質的國際化水準專業，大量的生產以達到市場的規模經濟，而能做到如此，就是靠許多的中小企業各自生產部份元件，經由緊密的產業分工體系與高效率的管理整合。反觀傳統的資訊軟體產業公司，大多每一個新的軟體產品都需要程式設計師從無到有的產生，就算程式語言的撰寫從第一代一直到第五代，程式設計師仍然要撰寫大小不一的程式碼，在軟體工程中，我們一直在追求發展低成本、快速，且具有高品質軟體的目標，而傳統的開發方式要達到此目標會遇到瓶頸，所以近年來發展出元件式軟體發展(Component-based Software Development)技術，元件式軟體發展技術成功的將軟體產業從手工開發原始產業進化到利用工具來輔助開發之自動化產業。

元件式系統的開發，其發展過程要著重於元件的再利用與組裝彈性，雖然“再用”是元件式系統發展被重視的主要因素之一，但是要了解，採用元件式系統發展方式未必達到再用的效益，除了要建立一套強調效用的機制外，元件開發之前的分析與設計是確保元件能夠再度被用的關鍵，元件開發前的系統分析工作必須從領域分析著手，找出相同領域中不同應用間的相同部份，再進行程式的分析設計，才能確保所開發的元件能在後續的發展過程中被重複使用，而目前的元件式軟體工程(Component-based Software Engineering)已有許多的方法論，如：Catalysis[D’Souza，1999]、SCIPIO[Veryard，1998]、O2BC[Ganesan，2001]、UML Components[Chessman，2001]，能讓發展者依據標準的工作程序逐步建構出元件式軟體，可期待的，未來一定會有更成熟的方法論出現。

大多數在元件方法論之研究都著重於使用某方法論其產出的元件規格(component specification)所帶來的優勢上，很少研究會關心從元件規格到實際元件實施(component implementation)上的對映(mapping)問題，但是對映是個很重要的議題，因爲對映能決定實施(implementation)是否能圓滿的表現出規格(specification)。

而隨著元件式軟體發展(Component-based Software Development)技術的成熟，軟體元件架構也邁向了分散式物件運算的架構，而在一個成功的分散式運算中常需要非同步的訊息傳遞。

本文章主要就是對於從元件規格到實際元件實施的對映上，加入非同步訊息傳遞的考量，進而提出一個新的對映模式。

二、研究目的

對於從元件規格到實際元件實施的對映上，其元件規格產出必須要選擇一種元件塑模方法論來做爲研究的對象，相對的，在實際元件實施上，也必須選擇一種軟體元件架

構做爲研究對象。

(一) 對於經過評估後所選出的元件塑模方法論，必須了解在其塑模流程中，是否有非同步訊息傳遞考量的部分，若無，則要將此部分的考量適當的擴充到塑模流程中，以因應分散式物件運算之軟體元件架構的趨勢。而對於經過評估後所選出的軟體元件架構也必須要了解架構中是否有提供能達成非同步訊息傳遞考量的技術與實作方式，若無，則必須尋求在此架構下滿足此需求的方法。

(二) 針對經過評估而選出元件塑模方法論及軟體元件架構提出一個加入非同步訊息傳遞考量的新對映模式。此爲本研究的主要目的。

在將以上兩點加以整合後，即可以完整地從分析到實作上徹底因應分散式物件運算之軟體元件架構的趨勢。此對業界在元件軟體發展上有實際且立即的貢獻，即在業界是相當有實用性的。

三、研究流程

研究流程依序共分爲以下五個階段：

(一)了解目前元件式軟體工程的現況背景，整理現有的元件塑模方法論，分析各個方法論的優缺點。選出較爲優良且較不具複雜度的方法論做爲研究對象。另外，也要針對廣泛應用於現有企業系統的各種軟體元件模式標準加以比較優劣處，並選出其中較優良的軟體元件模式標準來當做本研究的另一個對象。

(二)將所選擇的元件塑模方法論完整的回顧，並找尋在其塑模流程中是否有納入非同步訊息傳遞的考量，若無，則分析在此方法論之塑模流程中，應該在哪步驟加入此考量之塑模流程擴充，並實際地將此考量適當的擴充到塑模流程中。另外，也要將所選出的軟體元件模式標準完整的了解，以清楚知道此標準所支援與提供的哪部分功能及技術能達到非同步訊息傳遞考量之需求，若無，則必須尋求在此架構下滿足此需求的方法。

(三)針對所選取的元件塑模方法論，思考應該從哪一面向切入以作此對映的研究，決定後再以此面向與所選取的軟體元件模式標準相應，實際提出一個加入非同步訊息傳遞考量的新對映模式。另外，爲了整個元件系統對映的完整性，也要找尋其它相關的對映研究之文章，來補充其它不是非同步訊息傳遞考量的對映部分。

(四)利用所選取且有非同步訊息傳遞考量的元件式塑模方法論，實際分析一個分散式元件運算的個案。當元件規格產出後，則根據所提出的新對映模式，與選擇的軟體元件模式標準做實際的對映。並探討採用此對映模式後，該案例所能提升的效益。

(五)提出未來可以繼續研究的方向。

貳、文獻探討

一、元件塑模方法論與軟體元件架構的選擇

在選擇元件塑模方法論上，UML Components 經評估後是現今方法論中較佳的[陳鴻明，2003]，故選擇 UML Components 方法論。但此方法論仍然有些許在“流程模

型”及“塑模指導準則”的缺陷[陳鴻明，2003]，故考量不影響元件於原 UML Components 方法論的系統架構(system architectures)之分層模式下，本文實際上採用的是經過擴充改善的版本[陳鴻明，2003]。

在實際元件實施上，則採用 Sun 公司的 EJB(Enterprise Java Beans)軟體元件架構。但選擇 Sun 的 EJB，而非 Microsoft 的 COM+的原因是因爲 EJB 是 J2EE(Java 2 Enterprise Edition)核心，選用 EJB 就是指選擇的是 J2EE 平台。相對的，選用 COM+就是指選擇的是.NET 平台，現在元件式軟體發展(Component-based Software Development)技術成熟，軟體元件架構不只邁向了分散式物件運算的架構，甚至更進一步的朝向所謂的 Web Services[David Booth et al.，2004]運算架構，而從對分散式技術及 Web Services 的支援角度來看，J2EE 支援了較爲廣泛應用於現有企業系統的分散式運算服務，雖然 J2EE 不支援 COM+，但 COM+只能在 Windows 平台上運行，所以選擇 J2EE 平台確實是比較明智的，表 1 和表 2 是它們的比較列表[CNET Networks, Inc.]。

二、改進版的 UML Components 方法論

Chessman所提出的UML Components

表 1:對舊有分散式技術的支援

	J2EE	.NET
CORBA	支援	不支援
RMI/IIOP	支援	不支援
COM+	不支援	支援

表 2:對新一代 Web Services 的支援

	J2EE	.NET
XML-RPC	支援	不支援
SOAP	支援	支援

[Chessman，2001]方法論主要的精神是把元件式塑模方法論保持在規格的層次上，也就是說整個方法論不會牽涉到實作設計的部份，如此系統分析師就可以將焦點著重在元件所能提供的服務上，而不用擔心實作細節的困擾。此方法論所使用的概念及表示法是運用以下七個來源所產生的，可以說這七個部分是生成UML Components方法論的根基，其分別是Catalysis、Advisor[Advisor]、Syntropy[Cook，1994]、Rational Unified Process(RUP)[Jacobson，1999]、Open Information Model(OIM)[OIM]、UML1.3版[OMG，1999]及Object Constraint Language(OCL)[Warmer1999]。UML Components是使用統一塑模語言(UML)做爲其整個元件塑模的描述用工具，並透過適當的將UML擴充，使其變成更適合元件分析。

在系統架構(System Architecture)上，UML Components定義了四個層次，所謂系統架構是指最終系統的結構全貌，它使我們能夠思考軟體單元(software unit)放入在整個應用系統某位置的目的與用途，所以是很有用的。這裡的軟體單元指的就是元件，這樣的分層，每一層都有個別的邏輯與工作意含，所以每一層各有適合的元件類型，各層的元件類型其性質與功能必須要能滿足各

層次的責任範圍，而不同層次間也會有所相互連結互動的行爲，所以各層的元件也要能有相對應的功能與動作。這四層分別是使用者介面(User Interface)、使用者對話(User Dialog)、系統服務(System Services)、企業服務(Business Services)，圖1是其架構層圖示。

當中使用者介面與使用者對話是屬於client的部分，系統服務與企業服務是屬於server的部分，系統服務所屬的元件稱系統元件，企業服務所屬的元件則稱企業元件，而使用UML Components方法論元件塑模就是要產出這兩個層次所屬的元件。以下特別說明這兩個層次的含意:

1．系統服務(System Services):

系統的外部表現層，提供系統的服務存取，這一層扮演下面企業服務層的外觀之角色，它提供企業服務層一個更一般化的呈現背景，這層強調特殊化，代表著領域導向(Domain-Oriented)，此層可以使用來滿足特定系統的需求，正由於領域導向的關係，所以系統服務層的元件是不具備著再利用(Reuse)的特性。

2．企業服務(Business Services):

這層是核心企業(core business)資訊、規則和轉換的實現，此層強調一般化，所以此層的元件具備有再利用的特點，經常可以再利用於其它的系統。

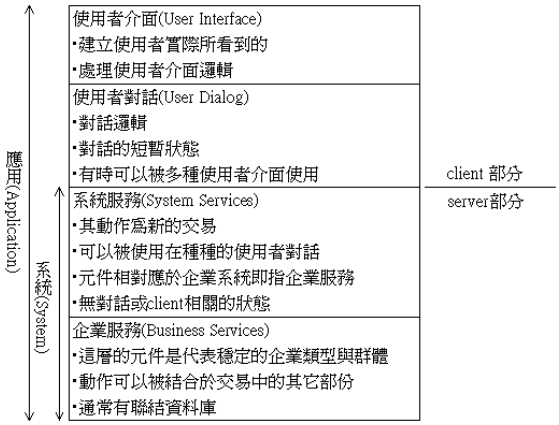

圖1：架構圖(Architecture layers)

原來及改良版的UML Components主要之流程都有相同的四大步驟。依執行順序分別爲需求定義(Requirement Definition)、元件識別(Component Identification)、元件互動(Component Interaction)及元件規格(Component Specification)。圖 2 是原 UML Components 方法論的流程模型，而圖 3 則是改良版的流程模型。

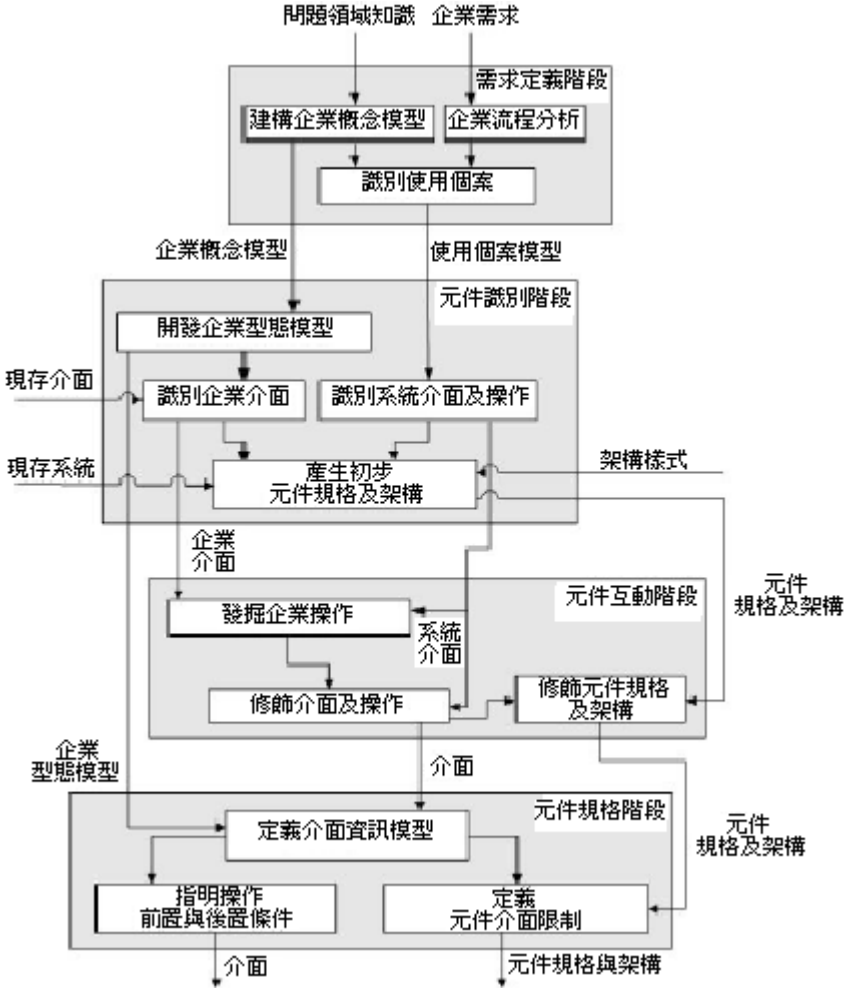

圖 2：原 UML Components 流程模型

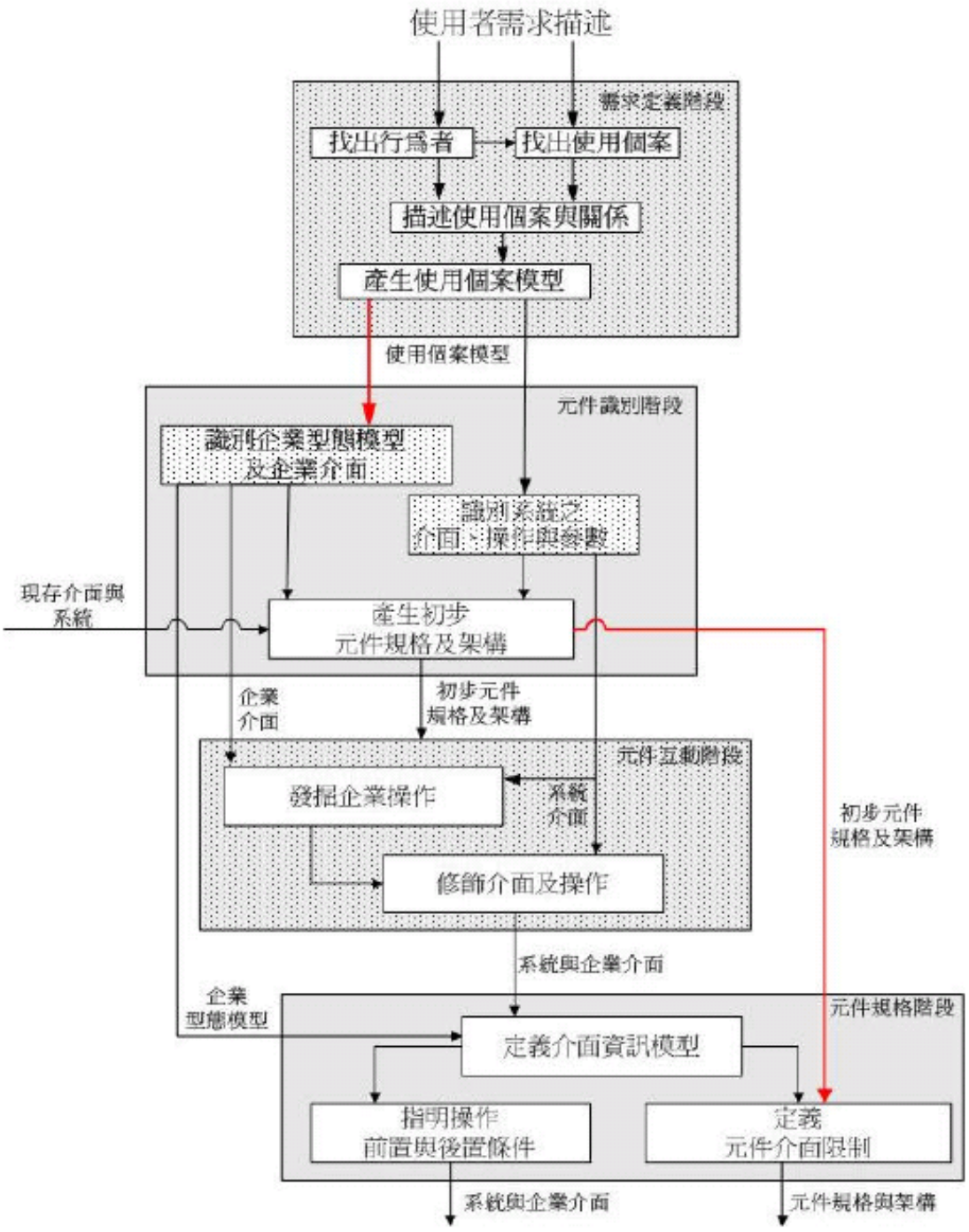

圖 3：改進版的 UML Components 流程模型

經過這四個步驟，就可以得到元件的規格架構及介面，而程式設計師則可以根據這些資訊選擇元件的取得來源及組裝方式。

原版與改良版最大的不同是在第一個步驟需求定義階段，在第二個步驟元件識別階段及第三個步驟元件互動階段中只有些許的不同，而最後第四個步驟元件規格階段則是完全一樣。想當然其中的一些操作意含之細節一定會有所不同，但是改良版之系統架構(System Architecture)仍是依循原版方法論所定義的四個層次，其所追求的最後元件規格與架構產出依舊是系統與企業元件，所以使用此改良版的UML Components方法論並不會喪失原版的主要意旨，反而能因此在“流程模型”及“塑模指導準則”上得到更好的支持。

三、JMS 與 Message-driven Bean

(一)JMS(Java Message Service):

JMS 是實作自 JMS API，用來提供訊息傳遞的功能，所有應用程式或是應用程式的元件都可以透過訊息傳遞的方式相互溝通，並使用非同步的運作模式。訊息的發送者與接收者不一定要同時間存在，就如同信件的傳送一樣。client 端的 JMS 應用程式透過 JMS API 與 JMS Server 溝通、取得連結、傳送訊息與接收訊息等等。基本上，JMS 的程式模型是以 JMS API 爲基礎，以供用戶端的程式產生、傳送、接收與讀取訊息用。因此，爲了使一個利用 JMS 技術的產品能作用，下列幾個部份需要包含在內，圖 4[周政宏，2002]是其關係圖:

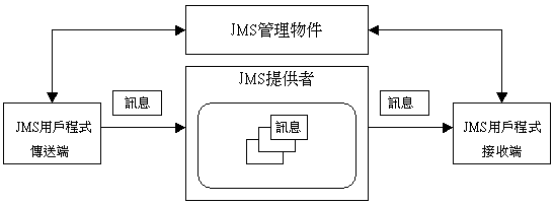

圖4：JMS的程式模型關係圖

1.用戶端程式:

這可爲訊息的傳送者(又稱爲產生者，producer)，或接收者(又稱爲消耗者，consumer)。用戶端同時具有產生者及消耗者之功能。

2.JMS提供者(provider):

此爲符合JMS標準的訊息系統，主要是用來控制與管理訊息。

3.訊息:

用戶間通訊的資料是以JMS訊息的形式存在。

4.管理物件(administered object):

在管理工具事先定義好的JMS物件，這是用戶端和JMS提供者連接所需的物件。

JMS提供兩種訊息傳送的通訊模型:

1.點對點(point-to-point，簡稱PTP):

其目的地爲queue，以下是使用queue的順序:

(1)訊息產生者將訊息傳送到一個特定的queue。

(2)傳送訊息給已登入伺服器的訊息消耗者。

queue內所儲存的訊息會維持到訊息被消耗或是過期爲止，其特點是每個訊息只有

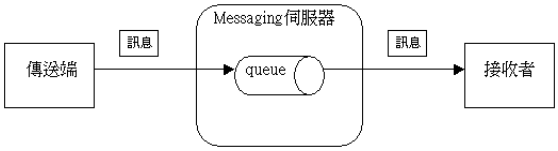

圖5：point-to-point模型的訊息傳送與接收模式

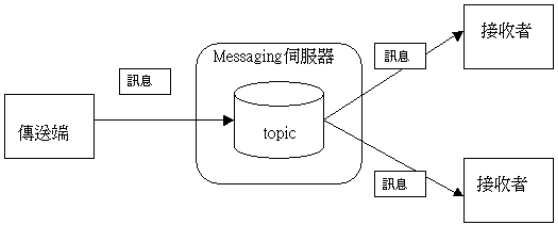

圖6：publish/subscribe模型的訊息傳送與接收模式

一個消耗者，而傳送者和接收者之間並沒有時間上的相依性，也就是傳送者和接收者並沒有前後的關係，傳送者和接收者不需同時執行。就單純表現來看，點對點是一對一(one-to-one)傳輸的模型，圖5[沈建男，2003]顯示其間的關係。

2. 發行與訂閱(publish/subscribe，簡稱Pub/Sub):

其目的地爲topic，以下是使用topic的順序:

(1)訊息產生者製作訊息，將訊息傳送給topic(發行/ publish)。

(2)訊息消耗者會自己登錄到topic(訂閱/subscribe)。

(3)Messaging伺服器會將訊息傳送給所有登錄在topic的訊息消耗者。

不同的是，這爲一對多的關係，亦即一個訊息可有多個消耗者，而傳送者和接收者之間有時間上的相依性，也就是只有活動中的接收者才會接收到訊息。上述的接收者稱爲非持久的訂閱者(non-durable)。爲了收到所有出版到topic的訊息，則需要使用持久的訂閱者(durable subscriber)，它不需要維持在活動的狀態，訊息會維持到被知會該訊息已接收或是過期，圖6[沈建男，2003]顯示其間的關係。

(二)Message-driven Bean

爲了發揮JMS非同步訊息處理的特性，EJB2.0新增了一種enterprise bean型態，即message-driven bean。它可以實現EJB的非同步(asynchronous)之呼叫。Message-driven Bean裡面的所有方法都只會被EJB Container所呼叫，它不提供client端存取介面。client端的應用程式將訊息傳送至Messaging Server之Queue或Topic，而實際向message-driven bean傳送訊息的是Massaging Server，當訊息被送至Message-driven Bean時，bean實體的MessageListener函數(onMessage( ))會自動被執行，並且會傳入接收到的訊息，因此MessageListener函數可以針對訊息的內容做進一步的處理，即執行業務邏輯部份。由上述可以知道Message-driven Bean是訊息消耗者，它具有等待非同步訊息到來之功能。圖7是Message-driven Bean的架構概觀圖。

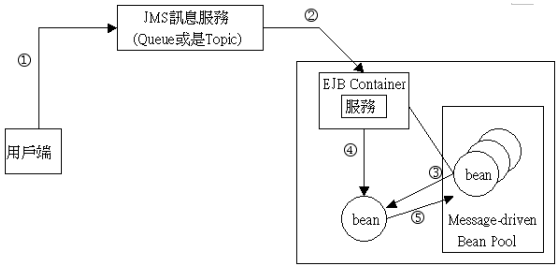

圖7：Message-driven Bean的架構概觀圖

[說明]:

①用戶端(生產者)傳送一個訊息到JMS訊息服務(即Messaging服務)。

②訊息被傳送到Container。用戶端可以另外連接到某個伺服器繼續作其他的事情。

③Container會向JMS訊息服務確認訊息已收取，這時JMS訊息服務就可以將已收取的訊息從Queue或是Topic中移除了。而接下來，Container會從pool當中找一個Message-driven bean(指bean的instance)出來處理這個訊息。

④Container將訊息傳給這個bean(Container會將訊息做爲參數，呼叫這個bean的MessageListener介面方法onMessage( ))。

⑤當bean的交易確認之後，接著Container就會將bean送回pool之中。但是如果在bean的 onMessage() 裡面已經將交易退回(rollback)，那麼Container就會告訴訊息服務將訊息放回到Queue或是Topic當中。

參、對映(mapping)模式

在原版及改良版的UML Components方法論之第二階段元件識別(Component Identification)中有一個必須的步驟，即識別系統介面及操作。這個步驟主要的任務是根據每個使用個案，產生對應使用者和系統互動的系統介面、操作以及參數。而至於產生操作與參數的依據則爲檢視使用個案每一個步驟，判斷其是否有必要被塑模成系統操作。

在原版及改良版的UML Components方法論之第三階段元件互動(Component Interaction)中有另一個必須的步驟，即發掘企業操作。由於上一階段元件識別(Component Identification)所產生的企業介面不具有任何的操作，所以這個步驟的目的是要爲每個企業介面找出適當的企業操作。而由於企業介面的目的是要支援上層的系統介面，因此，要找出企業操作與參數則必須透過分析系統操作而得。分析的方式是依據系統操作的複雜程度，將系統操作一一分配或拆解後各別分配到對應的企業介面(或外部已有的元件之介面)，如此一來，每個企業介面即可獲得其應有的企業操作與參數。

在上述元件互動(Component Interaction)階段的發掘企業操作步驟中，主要是採用UML中的合作圖(Collaboration Diagram)作爲分析工具。

在分析時可能會有複雜操作傳遞的情形，例如圖8的“送出訂單( )”系統操作所示。由於送出訂單時必須將書籍資訊與付款的信用卡資訊一併處理。因此學生書籍訂購系統必須將此操作分解爲“拋轉訂單( )”與“書

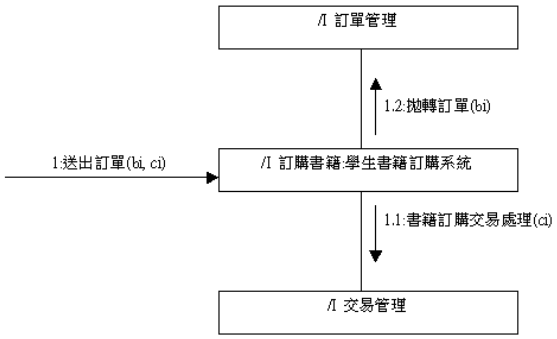

圖8：複雜操作傳遞

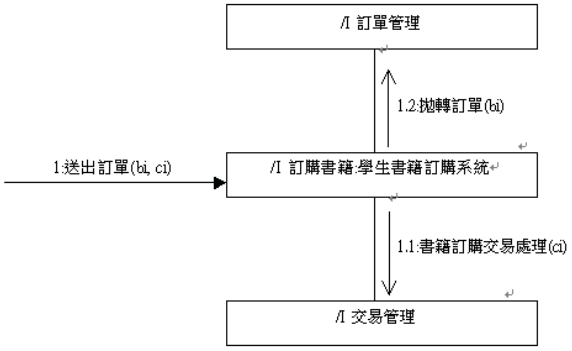

圖9：採用非同步訊息之複雜操作傳遞

籍訂購交易處理( )”二個子操作。並分別由“I 訂單管理” 企業介面與“I 交易管理” 外部已有的元件之介面來負責處理。

訊息是物件之間互動時所溝通的內容，訊息又分爲同步訊息及非同步訊息兩種。同步訊息爲送出訊息後，當動作還沒有執行完畢時，不進行下一個動作。而非同步訊息爲送出訊息後，不等待動作執行完畢，就進入下一個動作。

思考圖8之複雜操作傳遞，其在UML合作圖中只使用“——►”來表示訊息傳遞，這是屬於同步訊息傳遞的表示法。可是這樣的同步訊息傳遞是很浪費CPU效能的，因爲CPU會卡在等待回應之動作處而無法往下繼續處理其他動作，若考量這一連串的動作必須依序執行，則犧牲CPU效能是可接受的，但如果動作間不需要依序執行，則此就非常不明智了，畢竟能充份利用CPU效能，就可服務更多的需求及更快獲得執行結果。故此可採用非同步訊息傳遞的方式。但此方法論於此並無非同步訊息傳遞的考量及表示法。

如經考量過後，決定採用非同步訊息傳遞，則圖8改成圖9。在圖9中，本研究所採用的非同步訊息傳遞之表示法爲UML1.5版[OMG，2003]的 “——>” 。

此時當訊息“送出訂單(bi, ci)”傳送到“/I 訂購書籍:學生書籍訂購系統”後，就能個別呼叫“1.1:書籍訂購交易處理(ci)”及“1.2:拋轉訂單(bi)”而不用依序處理了。由於在此，系統元件傳遞訊息到企業元件是採用非同步訊息傳遞的方式，所以CPU效能就能充份利用了。

在EJB中，非同步訊息傳遞是須要加入某些機制來達成的，這會增加層次的複雜度，所以是否要加入非同步訊息傳遞機制就必須要考量實際的系統操作情形，如操作呼叫是對位在本端伺服器的企業元件，即非分散式的運算，則就比較不會有延遲處理的情形，此時便不需要使用非同步訊息傳遞，以上的拋轉訂單(bi)就是如此。但是如果操作呼叫必須透過網路呼叫其它伺服端中的元件方法而有網路傳輸延遲的顧慮，或它端系

統之伺服器又常會負載過大而有延遲處理的考量，那這時就應該使用非同步訊息傳遞，如上述的書籍訂購交易處理(ci)。

要實現非同步訊息處理，則可以採用JMS(Java Message Service)與Message-driven Bean來完成[1]。此時系統元件是負責判斷傳過來的操作訊息應該拆解成哪些不同的子操作，並將其中要傳送的非同步操作之訊息，以JMS訊息的形式傳給JMS訊息服務，當Message-driven Bean從JMS訊息服務中取得非同步操作之JMS訊息後，再實際傳送要傳送的操作訊息到相對應的企業元件(或外部已有的元件)去做處理。以上例來說，圖10是其過程的示意圖。

此處的Message-driven Bean是用來間接執行系統元件所有介面之各個操作中需要非同步訊息傳遞考量的子方法呼叫，而呼叫是指對其他的元件。即某一個系統元件對其他元件方法的呼叫會由Message-driven Bean實際正確地去執行，而Message-driven Bean就是對其他元件方法呼叫的實際代理者。

Message-driven Bean所作的方法呼叫是對應目前系統元件所作之呼叫的，所以當系統元件換成另一個時(即指換了另一個應用

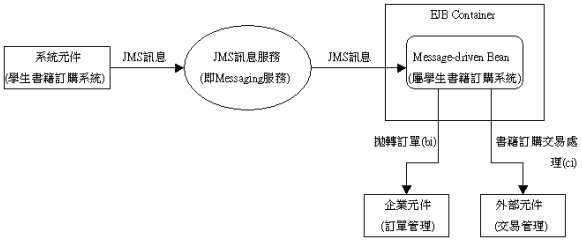

圖10：圖9例子之過程示意圖

系統)，則此和前對應呼叫內容的Message-driven Bean就不能使用了。所以我們可以說Message-driven Bean跟系統服務層的系統元件性質一樣，都是不具備著再利用(Reuse)的特性。另外，對Message-driven Bean的產生來說，Message-driven Bean是因應UML Components方法論之第三階段元件互動(Component Interaction)中的發掘企業操作步驟而產生的，所以此Message-driven Bean元件確實是UML Components方法論的產物。

由上述可知，Message-driven Bean元件是可納入UML Components方法論所提出之四層次系統架構(System Architecture)中的系統服務層或企業服務層的，且它跟系統元件一樣是屬於系統服務層。

由於UML Components方法論元件塑模所產出的是系統元件及企業元件。所以我們對於元件規格到實際元件實施的對映上並不會將四個層次中的使用者介面層及使用者對話層包含在內。爲了方便，我們將這兩層的相關應用程式(例如各種瀏覽器及Web container軟體)稱爲對話軟體(dialog software)，在UML Components方法論與EJB

[1]如果要保證讓Enterprise Bean可以佈署到任何一個EJB2.0相容的伺服器，則不能使用Java API來實作執行緒[Kathy Sierra et al.，2004]，而Message-driven Bean是唯一的方法。故不能在系統元件之session bean中，以Java API實作執行緒(Thread)的方式來滿足非同步訊息傳遞之考量。

元件的新對映模式上，此對話軟體層並不會參與對映。另外會將系統服務層的系統元件稱爲系統軟體，而將企業服務層的企業元件稱爲企業軟體，圖11是所提出的新對映模式之圖示。

由UML Components方法論的四層系統架構可以知道系統元件是系統的外部表現邏輯層，提供系統的服務存取，所以是使用session bean來實作，故如“圖11:新對映模式”所示。

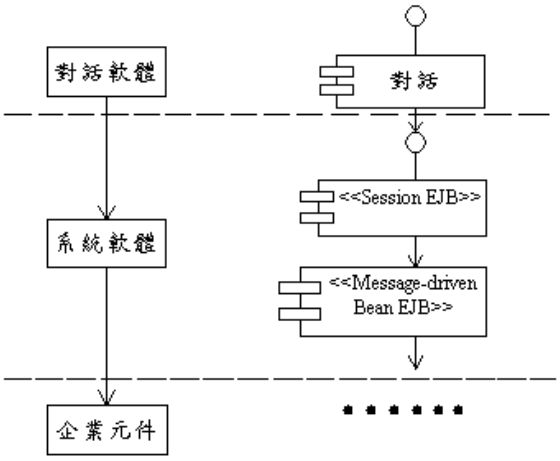

圖11：新對映模式

圖11中的Message-driven Bean元件往下的對象是不一定到企業元件的，也可能是到外部已有的元件上，像圖6中的交易管理元件就是外部已有的元件，所以對映模式圖中的箭頭並沒有超過系統軟體與企業元件之分線。

在Message-driven Bean的責任範圍與命名之規範上，應該要考量到以後維護的效率及系統資源善用這兩點。在這裡，我們直觀地以將來可能會維護到的對象來考慮三種Message-driven Bean的責任範圍與命名之方式，經考量後決定採用第三種方式，即以“使用個案中之各系統操作”爲責任範圍與命名。其各方式的意含與採用第三種方式的原因如下所述:

1．以“使用個案”爲責任範圍與命名:

若一個使用個案中，其系統操作有分解成需非同步訊息傳遞之子操作情況時，即爲此使用個案建立一個Message-driven Bean，並將此使用個案中所有需非同步訊息傳遞之子操作納入此Message-driven Bean實際呼叫之，此時這個Message-driven Bean即命名爲與該使用個案名稱相關之名稱。

但是這個以使用個案爲責任範圍與命名的方式是不理想的，因爲在一個大而複雜的系統中，一個使用個案裡可能有眾多的系統操作，且當中的每一個系統操作也可能會分解出許多需非同步訊息傳遞之子操作，所以當要維護某一個不知在何處的子操作時，並無法由Message-driven Bean的名稱資訊快速找到待維護的子操作，因爲這時我們只能先去查看在元件互動階段之發掘企業操作步驟的每一個系統操作分解圖，看哪一個系統操作分解圖有此子操作，知道是哪一個系統操作分解圖有此子操作之時，再以此系統操作名稱於系統介面彙總圖中得知是屬哪一個使用個案，如此才能依其使用個案名稱找到相符的Message-driven Bean，由於此在子操作的維護上造成了兩層次的查詢，所以維護的效率並不佳，故不採用此法。

2．以“系統操作分解之各子操作”爲責任範圍與命名:

爲每一個需非同步訊息傳遞之子操作單獨建一個相應的Message-driven Bean，此時這個Message-driven Bean即命名爲與該需非同步訊息傳遞之子操作名稱相關之名稱。

這樣雖然當要維護某需非同步訊息傳遞之子操作時，可以直接參考Message-driven Bean名稱，使維護便利而效率高，但是在整個系統元件中可能要維護的不止是Message-driven Bean，還有更高階的系統介面及操作，當需維護的是系統操作時，則必須查看各個系統操作分解圖來得知此Message-driven Bean是屬於哪一個系統操作的分解後產物，如此的系統操作維護是一層式的查詢，所以還可接受。但同理，若需維護的是系統介面時，由於意義即同於維護使用個案，所以情況又更加複雜，此時必須先查看各個系統操作分解圖來得知某Message-driven Bean是屬於哪一個系統操作的分解後產物，再經由查看系統介面彙總圖來得知此所屬之系統操作是否屬於該需維護的系統介面，若是，則將此Message-driven Bean作連帶維護；若否，則再考慮下一個Message-driven Bean，直到需維護之系統介面的所有Message-driven Bean均找出來，以作完整的連帶維護，這是兩層次的查證，且這樣的查證會有所查的Message-driven Bean並非責任歸屬於所需維護之系統介面的缺點，所以其維護效率頗差。

另外，針對每一個需要非同步訊息傳遞之子操作單獨建立一個相應的Message-driven Bean，是相當浪費系統資源的。這是因爲EJB Container會預先建立一定數量的Message-driven Bean實體(instance)，並將之放置於instance pool裡，而所謂pool指的是記憶體區塊，故對於在此建一個實體只是爲了單獨一個實際呼叫動作而言，是浪費系統資源的。

由於以上兩缺陷，所以我們並不採用此法。

3．以“使用個案中之各系統操作”爲責任範圍與命名:

爲每一個有分解出需非同步訊息傳遞之子操作的系統操作建立一個Message-driven Bean，並將系統操作所分解出的所有需非同步訊息傳遞之子操作納入此Message-driven Bean作實際的呼叫，而Message-driven Bean即命名爲與該系統操作名稱相關之名稱。

此爲三種方式中最理想的。因爲若要找所須維護的子操作，則只需要查看在元件互動階段之發掘企業操作步驟的每一個系統操作分解圖即可，是爲一層式的查詢，而不用像以”使用個案”爲責任範圍與命名一樣，需多查看一層系統介面彙總圖。另外，此法也不會跟以”系統操作分解之各子操作”爲責任範圍與命名方法一樣有相同的缺點，即太過於浪費系統資源及當維護使用個案或系統操作時效率不佳，在此第三種方式中，當維護系統介面時只須查看系統介面彙總圖，則可直接對此法之Message-driven

Bean作完全連帶的維護，這只爲一層式的查詢。由上所述而因此建議使用此法。

由於在此新對映模式中加入的是非同步訊息傳遞的考量，而此考量是屬於系統服務層次的，所以在此僅畫出系統服務層次的對映。至於UML Components方法論的另一個重點，即企業服務層之企業元件，因爲不是此對映研究所關注的部份，所以其對映並不加以討論。但是爲了整套對映的完整性，本研究即直接套用其他學者在此企業服務層的對映研究結果。其分別以管理者bean(Manager Bean)、階層的(Hierarchical)、獨身(Singleton)這三種對映方式來看待企業服務層的對映[Yi Lin，2004]，以下便對這三種企業服務層與EJB的對映模式加以說明:

1．管理者bean(Manager Bean)

此模式的企業服務層之每一個企業元件都是以一個session bean來實作的。爲了使session bean容易實作，通常使用helper classes(標準的Java classes)來實作企業型態(Business Type)，並且讓每一個企業元件管理這些型態實體的一個集合。例如有一個企業元件CourseMgr，它有兩個相關的企業型態，分別爲Course和Section，首先建立兩個helper classes，一個是爲Course所建立的，另一個是爲Section所建立的，而這兩個helper classes提供方法以處理存取資料庫的動作。再來，使用一個session bean CourseMgr來包裝這兩個classes，並建立這個course元件和一個提供給系統元件使用的介面。此時就有一個管理者bean CourseMgr來協調在這些型態上的動作了。故在這種對映模式下，一個企業元件要建立一個管理者角色的session bean、一個或多個與其企業型態相對應的helper classes，以及一個供系統元件存取使用的介面。

圖12是其原文中所畫的對映模式圖(粗框除外)[Yi Lin，2004]，其粗框部份即可以直接套用在本文提出之新對映模式的企業服務層部份。

2．階層的(Hierarchical)

這個模式的構想是將一個企業元件分解成一個管理子元件和其他子元件。在企業服務層中，前一個管理者bean的對映模式是使用session beans去實作最低層的型態，這會有永續性(persistence)[2] 的問題，爲了避免這個缺點，此階層的(Hierarchical)之對映模式使用entity beans去實作那些型態[3]，以直接存取資料庫，由於entity beans是永續化的(persistent)，而且在多重client端中可以被共

[2]各個構成要素的實體，保存於資料庫等永久性的記憶領域，bean的狀態及資料的實體必須經常維持一體性。所謂維持一體性是指，bean的內容更動時，其內容必須同時更改成一覽表。這個處理過程稱爲永續化，其特性稱爲永續性(persistence)。

[3]session bean與entity bean的最大不同，在於永續性的有無。entity bean有容器管理的永續性(Container Managed Persistence)，開發者可不必描述專爲維持永續性的程式碼，而可以專注於純粹的業務邏輯上。

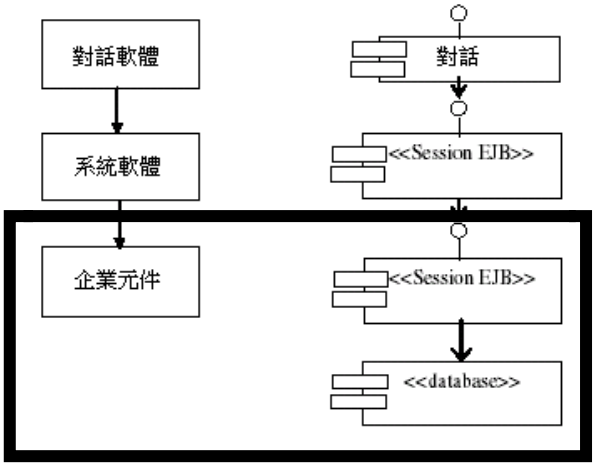

圖12：管理者bean(Manager Bean)的對映模式

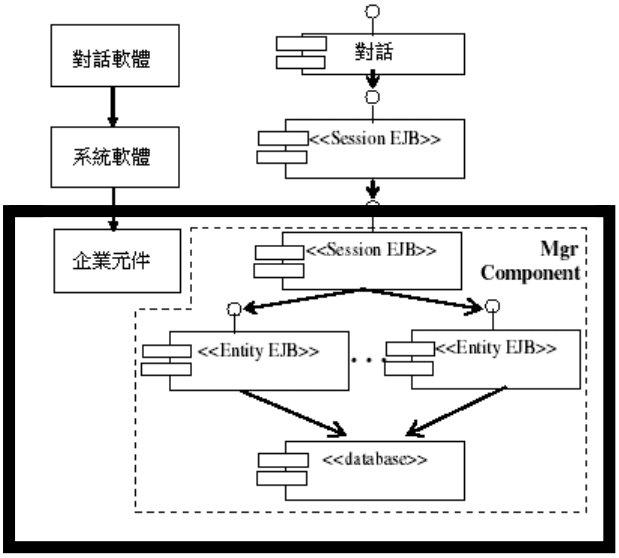

圖13：階層的(Hierarchical)之對映模式

享，因此使用entity beans去封裝資料庫表格是很理想的。同樣以一個企業元件CourseMgr來說明，它有兩個相關的企業型態，分別爲Course和Section，因應這兩個企業型態會有兩個資料庫表格被建立，因此這些型態是緊密依賴在資料庫上的，而課程登記系統(course registration system)會對此企業元件的每一個型態使用一個entity bean來實作。再使用一個CourseMgr的session bean來管理這些型態，並協調在CourseMgr企業元件上的動作，這裡的動作指的是存取以entity beans實作的類型實例。

此階層的(Hierarchical)之對映模式可以增加資料庫存取和資料共享的可靠性(reliability)。

圖13是其原文中所畫的對映模式圖(粗框除外)[Yi Lin，2004]，其粗框部份即可以直接套用在本文提出之新對映模式的企業服務層部份。

3．獨身(Singleton)

在前兩個對映模式中，使用純粹的Enterprise JavaBeans去實作元件，而在此獨身(Singleton)EJB，則是使用EJB及標準的Java classes來表現實作。跟正規的Java程式發展來比較，EJB的發展是較複雜的，EJBs在滿足分散式系統的需求上是非常有用的，對於一些大系統來說，企業元件(business components)可能是被不同的團隊所發展並分散到不同的伺服器(servers)上，所以前兩個對映模式對於這種情況是相當合適的，跟正規的Java程式比較，enterprise beans的發展是更複雜的，並且因爲enterprise beans的通訊成本的關係，所以其執行的效率是較低的。

思考相對的小系統，像是課程登記系統(course registration system)，它只將會被使用

在單一的大學，這些企業元件典型是存在相同的伺服器上，而其資料庫也可能在那相同的伺服器上，因爲一個分散式的高層次是不必要的，所以我們對一些元件可以使用標準的 Java classes，此所設計的獨身(Singleton)EJB對映模式就是針對發展一個這樣的小系統。

此小系統的企業元件不是分散在世界各地的，這個對映模式可以被描述成使用標準的Java packages (或classes)去實作企業元件，並且在系統服務層使用一個enterprise bean去封裝(wrap)這些企業元件。每一個企業元件都是一個標準的Java package，而每一個企業元件會提供一個介面給系統元件。在每一個Java package裡面，如果必須的話，我們也可以設計階段的(hierarchical)之結構，例如有一個複雜的企業元件，它含有六個企業型態，那我們就可以爲每個企業型態定義一個class，而此企業元件再將這些classes封裝起來，並且實作相應介面所要求的方法。但是對於一個簡單的元件來說，階段的(hierarchical)之結構就可不必了。在這些企業元件之Java classes中要實際定義存取資料庫並建立資料庫系統能力，此能力是指針對任何所必須的交易(transactions)與一致性(concurrency)控制要點。

這個獨身(Singleton)EJB之對映模式是三個對映模式中最簡單的，因爲寫標準的Java classes比寫enterprise beans更爲容易，這個對映模式簡化了發展程序，並且更親近初學的程式設計師。

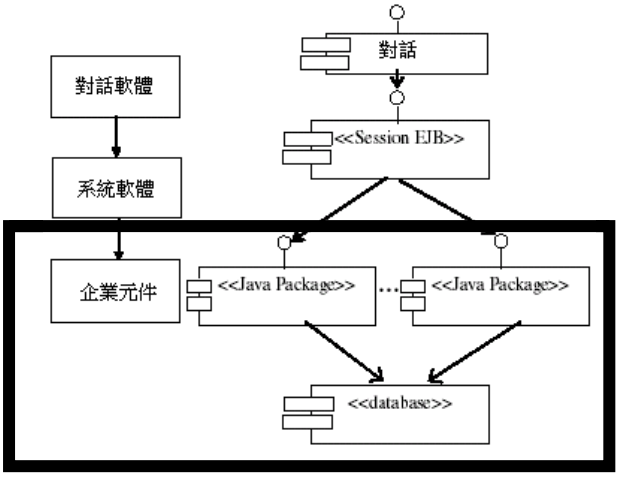

圖14：獨身(Singleton)的對映模式

圖14是其原文中所畫的對映模式圖(粗框除外)[Yi Lin，2004]，其粗框部份即可以直接套用在本文提出之新對映模式的企業服務層部份。

這三種企業服務層次的對映模式都有其優缺點，以下是他們各別的優缺點描述:

1．管理者bean(Manager Bean):

優點:

(1) 在enterprise beans的三種類型當中(即session beans、entity beans、message-driven beans)，session bean是最簡單的，所以此管理者bean的方法之優點就是它的簡易性(simplicity)。

缺點:

(1)session beans無法處理永續性(persistence)，所以我們必須爲每個管理者(manager)角色之session bean加入維持永續性(persistence)的程式碼。

2．階層的(Hierarchical):

優點:

(1) 跟第一個方法比較，階層的(Hierarchical)之方法的優點就是它增加了資料庫存取及資料共享的可靠性(reliability)。

缺點:

(1)階層的(hierarchical)層次(layer)可能帶來無效率及複雜的實作(implementation)。

3．獨身(Singleton):

優點:

(1)獨身(Singleton)EJB對於那些熟悉Java而不熟悉EJBs的開發者來說是一個好方法。

缺點:

(1)因爲這個方法在企業元件(business components)中不使用enterprise beans，即無法利用EJB container來處理交易(transactions)與同步性(concurrency)，故開發者必須在Java類別(classes)實作企業類型(business types)時，自己處理交易(transactions)及同步性(concurrency)。

而以下是這三種企業服務層次之對映模式的使用時機概述:

1．管理者bean(Manager Bean):

系統爲分散式的；且不注重永續性(persistence)的維持。

2．階層的(Hierarchical):

系統爲分散式的；且注重永續性(persistence)的維持。

3．獨身(Singleton):

系統爲集中式的，即所有元件都在同一個伺服器(server)上，資料庫也可能在此伺服器上。

肆、案例：e-learning數位學習網站

一、UML Components方法論元件塑模

[使用者描述]:

①案例背景:

宏昇集團是一個跨國性的多角化經營之企業，該企業的業務範圍跨及金融、營造等眾多行業，集團本身就已有建構良好的MIS系統。最近該集團想跨及資訊教育訓練這個領域，經評估後收購了國內的恆泰資訊股份有限公司。這間公司本來的業務範圍是自行及接受政府委託辦理電腦專業人才職業訓練，公司原本在臺灣的六個都會區都有設置學習場所，所使用的是一般傳統教學的方式。但是宏昇集團爲了因應網路時代能拓展服務的範圍，決定除了原本的傳統教學方式外，另建置線上數位學習網站，期望以更多元的課程種類及能隨時隨地高效率學習來招攬更多的顧客群。宏昇集團將學習管理系統(learning management system)交給外部廠商外包開發，而教材爲自行製作，現在都已建置完成了。目前所要建置的是一個線上課程採購站。

②系統目標:

分析整理出四點，在此不加以贅述。

③使用者與企業(作業與功能)需求描述:

分析整理出二十點，在此不加以贅述。

(一)、需求定義階段

1．找出行爲者:

(1)透過使用者與企業需求描述中的名詞、代名詞與名詞片語等，找出合乎行爲者定義的人或相關系統[Booch，1994]。

(2)由使用者與企業需求描述整理出關係。

(3)從關係之描述，可以看出行爲者的互動。

=>可以發現線上課程採購網站的行爲者爲客戶、課程訂購系統、現有的帳務系統。

2．找出使用個案:

可以從案例的初始行爲者所引發或參與的事件及其所完成的功能或目的找出所有的使用個案[Booch，1994]。此案例有二個初始行爲者，分別爲客戶以及課程訂購系統，以下即爲由初始行爲者所推導出來的使用個案。

=>註冊、新增訂購商品、修改訂購商品、處理訂單。

3．描述使用個案與關係:

使用個案描述，原則上以事件條列式之描述爲主，應表達事件的行爲者、前提、目標、主要事件列、例外事件列以及變動情況等。其是利用使用者與企業需求的描述，產生各個使用個案的描述。

=>經逐一檢視每一個使用個案之描述，發現「新增訂購商品」個案描述中，當客戶並不是系統會員時，會要求其進行註冊。反之，如果已是會員，則可以直接購物。因此，新增訂購商品使用個案與註冊使用個案有Extend[Jacobson et al.，1996]關係。

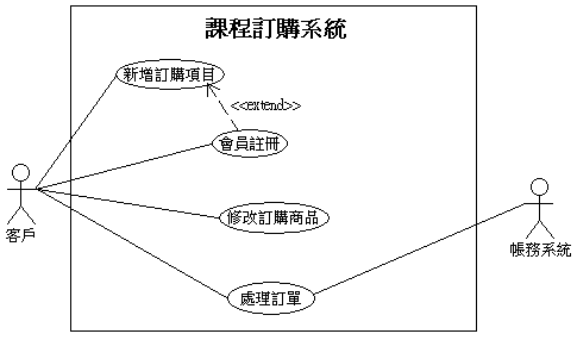

圖15：恆泰數位學習課程訂購系統使用個案模型

完成使用個案描述後，接下來可以透過藍圖的方式找出資料詞彙，以滿足使用個案所需要的表單資料。

4．產生使用個案模型:

最後，根據上個步驟產生的使用個案描述，繪出所有行爲者與使用個案。此處的行爲者有“客戶”、“帳務系統”。而使用個案則有“新增訂購項目”、“會員註冊”、“修改訂購商品”與“送出訂單”。

再來確定這兩個行爲者與四個使用個案之間的互動關係，並繪製於使用個案圖上，最後就完成了數位學習課程訂購系統的使用個案模型，如圖15。

(二)、元件識別階段

1． 識別企業型態模型及企業介面:

(1) 產生個別企業型態圖 (Business Type Diagram):

a.確認企業型態及屬性。

b. 繪製各個使用個案之企業型態圖 (Business Type Diagram)。

(2)產生企業型態模型(Business Type Model):

將上一步驟中所產生的個別企業型態圖(Business Type Diagram)統整後，即可產生初步的企業型態模型(Business Type Model)。

(3)修飾企業型態模型(Business Type Model):

a. 標示出企業型態模型(Business Type Model)間的企業規則及限制:

可以將限制透過UML的限制式標明。而在此的限制是利用物件控制語言(OCL)[Warmer，1999]來撰寫，透過物件控制語言，可以清楚的定義企業型態間的限制。

b. 識別出核心企業型態(Core Business Type):

(a)繪出企業型態模型中型態間的相依關係。

(b)決定核心企業型態:

找出企業型態相依關係後，接下來可以引用扇入(Fan-In)的概念[Marquis，2002]，找出核心企業型態候選集合，接著再根據核心企業型態的二個特性來評估其是否可成爲核心企業型態。

=>在核心企業型態模型圖(此圖是由初步的企業型態模型圖所演變)中可以看出購物車、課程、課程用參考書和客戶都有扇入(Fan-In)，所以它們被歸類於核心型態候選集合中。接著針對候選集合中的型態一一檢視。首先根據核心型態的第一個特性，判斷候選集合中的型態是否具有獨立的識別性。接下來，再根據第二個特性，判斷每個型態有無指派關係的存在。最後留在核心型態候選集合中的課程、購物車及客戶三個型態即是符合核心型態特性的企業型態。

c.由核心企業型態(Core Business Type)產生出企業介面(Business Interface)以及標示其責任歸屬(Responsibilities)。

經過上述三個步驟後，即可產生一個具有企業規則與限制、核心型態、企業介面及責任歸屬的企業型態模型，如圖16。

2．識別系統介面及操作:

這個階段主要的任務爲根據每個使用個案，產生對應使用者和系統互動的系統介面、操作以及參數。而至於產生操作與參數的依據則爲檢視使用個案每一個步驟，判斷其是否有必要被塑模成系統操作。

=>其中，從處理訂單使用個案主要事件列中可以看出當客戶填寫完信用卡及交貨資料後，即可將給課程訂購系統，所以系統可以提供”送出訂單( )”來滿足客戶需求。另而從處理訂單使用個案主要事件列也可以看出送出訂單後，系統會與現有的帳務系統元件提供的”I 交易管理”進行訂單的後續處理動作。由於這個現有元件的操作依賴於”送出訂單()”操作的分解結果，所以必須等到下一個階段對”送出訂單()”操作進行分解後，再行推導出來。”送出訂單()”系統操作的參數表達方式爲：”I 處理訂單：送出訂單(in creditinfo: CreditDetail, in deliveryinfo: DeliveryDetail

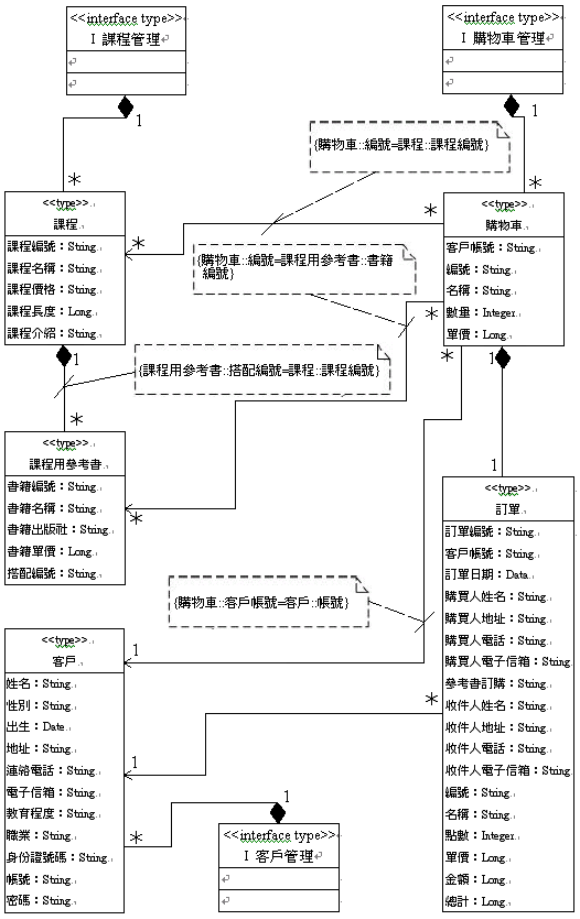

圖16：修飾後的企業型態模型

, in goods: CourseGoods[], in Provider: String[])：Integer”。

最後，則會將這個階段所發掘的系統介面、操作以及操作，整理成系統介面彙總圖。

3．產生初步的元件規格架構:

(1)產生系統元件規格。

(2)產生企業元件規格。

(3)將系統元件規格和企業元件規格整合，並指明相依關係，即可產生初步的元件規格

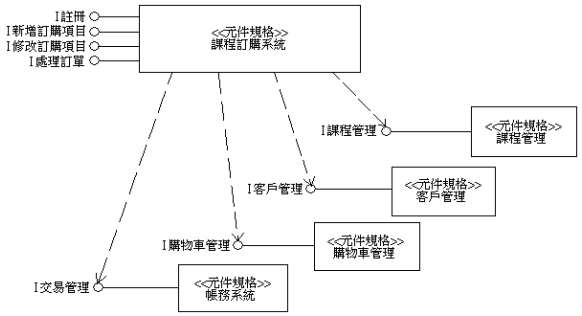

圖17：初步元件規格架構

架構。圖17是初步元件規格架構。

(三)、元件互動階段

1．發掘企業操作:

其中，對於處理訂單使用個案的“送出訂單 (in creditinfo: CreditDetail, in deliveryinfo: DeliveryDetail, in goods: CourseGoods[], in Provider: String[])”系統操作，由於這個操作牽涉到兩個部份，即訂單轉換與線上課程訂購交易處理，所以是屬於複雜的操作轉換。在此要將此系統操作一一拆解，把各項子操作轉送到各個應負責的元件介面，圖18是拆解後的操作分解圖。

這兩個子操作在此圖是有執行之先後順序的，必須先執行1.1:線上課程訂購交易處理(ci,di,p)，之後再執行1.2:拋轉訂單(di,g,p)，這裡的1.2:拋轉訂單(di,g,p)是爲了將完成交易的訂單資料存入資料庫，以供顧客及系統管理者事後的查詢。

2．修飾介面及操作:

(1).維護參照完整性

a.利用資料關係矩陣找出企業型態間的參

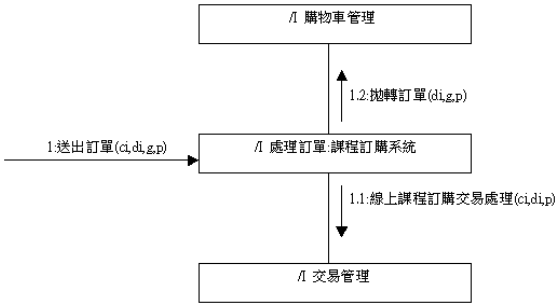

圖18：送出訂單( )操作分解圖

照關係。

b.利用操作參照矩陣找出有問題的企業操作。

=>根據以上二個步驟得知，所有的企業操作都不會產生參照問題。因此可以確保元件間參照完整性被良好的維護。

(2).整理企業介面

這個階段的任務是要將前面幾個步驟中，修飾完成的介面操作及參數做最後一步的檢查。如果發現任何問題，可以馬上回溯到對應的步驟，這樣可避免到後面階段再做修改所造成的額外成本。當檢查過後沒有發現任何的問題，就將可以具有完整操作參數的企業介面。

=>由於企業介面擁有的操作及參數皆源自於各個系統介面。所以當系統介面具有的操作沒有任何問題時，則可推論企業介面的操作也無問題。所以將具有完整操作的"I 課程管理"、"I 客戶管理"及" I 購物車管理"這三個企業介面整理成企業介面彙總圖。

因爲完成了第三階段，即元件互動階段後，對映的所有資訊都已經具備，所以接下來的第四階段，即元件規格階段，由於無關本研究的新對映模式，所以便不作案例說明，也就是說UML Components方法論元件塑模到此元件互動階段完，即案例說明完畢，接下來將會邁向下一小節：實際對映(mapping)。

二、實際對映(mapping)

思考"圖18：送出訂單( )操作分解圖"，可知帳務系統是問題所在處，因爲宏昇集團本來就是一個跨國性的多角化經營之企業，該企業的業務範圍跨及眾多行業，集團本身就已有建構良好的MIS系統。像帳務系統及供應商管理系統這樣的系統早已設置於集團中心以供各個業務單位使用，由於帳務系統的設置點與課程訂購系統不同，所以都是以網路作爲通訊的媒介，所以綜上所述可知，由於像帳務系統這樣的系統是提供給各個業務單位使用的，所以負荷量大，且課程訂購系統又是透過網路來作同步訊息傳遞，所以課程訂購系統的系統元件對帳務系統的"I 交易管理：線上課程訂購交易處理(in creditinfo: CreditDetail, in deliveryinfo: DeliveryDetail, in provider: String)：Integer"之動作要求，可能會有等待處理的情形，因此可以使用Message-driven Bean來做非同步的訊息傳遞，這樣就能不用等待回覆而繼續作其它的動作。也就是說圖中的"1.1: 線上課程訂購交易處理(ci,di,p)"未完成，即可作下一步的"1.2: 拋轉訂單(di,g,p)"，如此就能充份利用CPU的

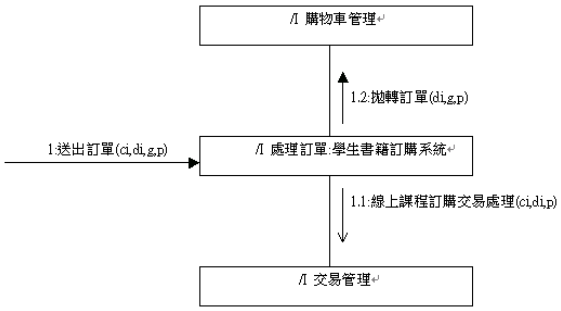

圖19：非同步訊息傳遞

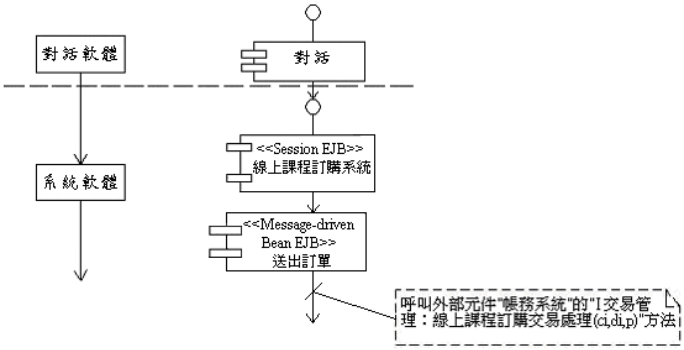

圖20：對映圖

效能且可以儘快完成“1:送出訂單(ci,di,g)”的整套動作。所以改成非同步訊息傳遞的作法，圖18會改成圖19。

在這個案例中需要採非同步訊息傳遞機制的是對外部原有之元件，而跟系統元件一樣設於本機且沒有動作延遲處理顧慮之企業元件是不需要採非同步訊息傳遞機制的，所以在非同步訊息傳遞的新對映模式中，企業元件是不參與的，故對映圖會如圖20所示。其圖之型式即爲本研究在系統服務層所提的非同步訊息傳遞之新對映模式。

三、效益探討

[前提]：

由於內文切換(Context Switch)爲作業系統的層次，且影響時間極小，故於此並不加以考慮。

以案例中之圖18及圖19兩分解動作爲例：

(一)、一般同步訊息傳遞：

1．單個client進行單個系統操作之情況，如圖21(a)。

2．同理可推知，以多個client同時進行多個系統操作，例如以三個client同時進行多個系統操作，如圖21(b)。

(二)、使用非同步訊息傳遞：

1．單個client進行單個系統操作之情況，如圖21(c)。

2．同理可推知，以多個client同時進行多個系統操作，例如以三個client同時進行多個系統操作，如圖21(d)。

可知當有n個client同時進行n個相同系統操作時：

(一)、若使用一般同步訊息傳遞：

則無論$t_{wait} > n * t_{CPU}$，如圖21(a)及(b)，或$t_{wait} < n * t_{CPU}$，或$t_{wait} = n * t_{CPU}$

$=> Time_{total} = t_{wait} + n * t_{CPU}$

(二)、若使用非同步訊息傳遞：

1．$t_{wait} > n * t_{CPU}$，如圖21(c)及(d)

$=> Time_{total} = t_{wait}$

2．同理，當$t_{wait} = n * t_{CPU}$

$=> Time_{total} = t_{wait} = n * t_{CPU}$

3．同理，當$t_{wait} < n * t_{CPU}$

$=> Time_{total} = n * t_{CPU}$

例如：假設有5個client同時進行案例中之圖

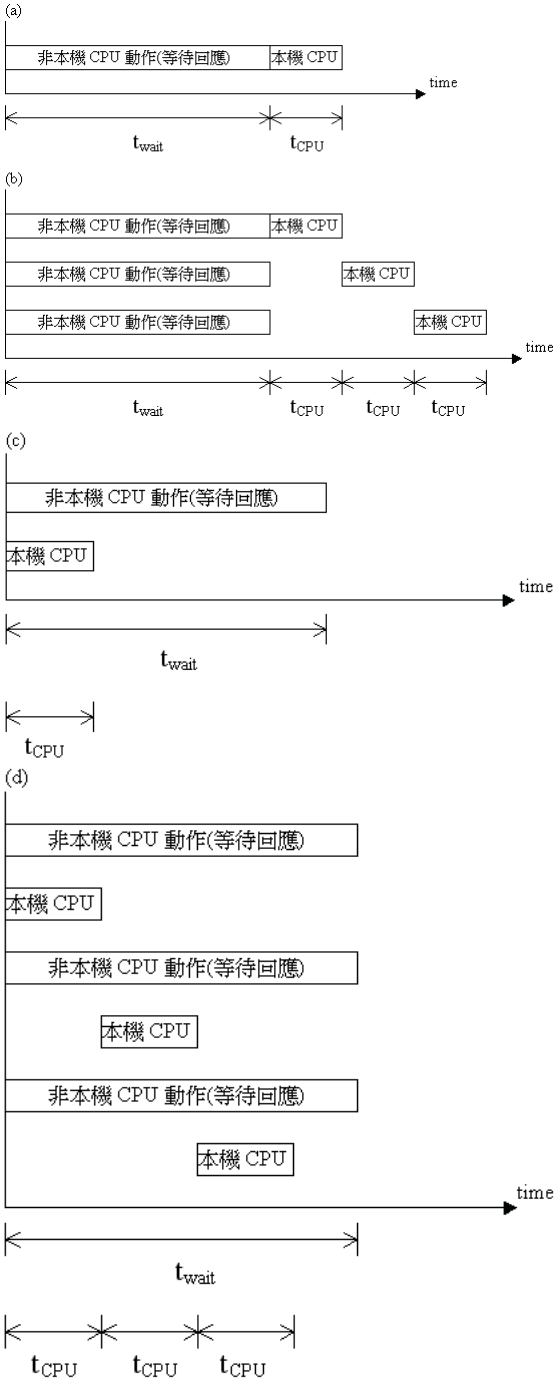

圖21：效益探討圖

18及圖19的系統操作“送出訂單( )”，且設分解之“線上課程訂購交易處理(ci,di,p)”的t_{wait} = 10000 毫秒，而 “拋轉訂單(di,g,p)” 需t_{CPU} = 1000毫秒，探討將圖18改成圖19所帶來的效益。

=>由上述知10000＞5 * 1000，即t_{wait}＞n * t_{CPU}

圖18屬一般同步訊息傳遞

=>$Time_{total} = t_{wait} + n * t_{CPU} = 10000 + 5 * 1000$
$= 15000$(毫秒)

圖19屬非同步訊息傳遞

=>$Time_{total} = t_{wait} = n * t_{CPU} = 10000$(毫秒)

➜以此研究所舉之案例可知改進了15000 – 10000 = 5000(毫秒)，故所提的對映模式能顯著地提升效益。

以上是爲預估值，而本研究實作案例之模擬系統，對照以證實上述：

此模擬系統採用如上例之假設，“線上課程訂購交易處理(ci,di,p)”的t_{wait} = 10000 毫秒及“拋轉訂單(di,g,p)” 需t_{CPU} = 1000毫秒爲系統程式本身內定，而client個數可由測試者輸入，如上例所述，在此會輸入5。爲求精確，測試者會重覆執行此系統三次，取其平均值，圖22是其中一次的執行過程及結果畫面。

=>取平均後知採一般同步訊息傳遞需 15844 毫秒，而採非同步訊息傳遞則只需 10307 毫秒，改進了 15844 – 10307 = 5537(毫秒)，可知所有數據近似上例所述之預估值，而所多出的數百多毫秒是內文切換(Context Switch)所需的時間，爲合理的消

圖 22：案例之模擬系統執行過程及結果

耗。所以本研究所提之對映模式確實有顯著的效益。

伍、結論

一、研究貢獻

(一)在UML Components方法論的第三個階段，即元件互動階段中，會透過分析系統操作來得出企業操作與參數，而UML中的合作圖(Collaboration Diagram)是其表達之代表性工具。另一方面，在不同情況下，如果使用循序圖(Sequence Diagram)能夠帶來更佳的效果或便利性，也可以使用循序圖做爲塑模工具。而UML Components中所使用的是UML 1.3版，UML 1.3版的合作圖及循序圖已經納入了非同步訊息傳遞的觀念及表達法了。此外，如文章前面所述，在分解系統操作時，可能會遇到複雜的操作傳遞之模式。綜上所述，UML Components的作者應該會將非同步訊息傳遞加入考量才是。但是作者疏忽了這一點而未將非同步訊息傳遞加入考量，本文章其一的貢獻就是實際地將此考量適當的擴充到塑模流程中，使UML Components方法論更爲完善。

(二)在UML Components方法論中加入非同步訊息傳遞的考量後，進而在從UML Components方法論之元件規格到EJB實際元件實施的對映上，研究提出了一個新的對映模式。這是貢獻之二，也是本文章主要貢獻所在。[4]

(三)這篇文章的出發點是因爲現今軟體元件架構上已邁向了分散式物件運算，所以經常需要非同步訊息(asynchronous message)傳遞的考量。而目前的軟體元件架構也更進一步演進到了所謂Web Services架構方向，在Web Services架構中服務仲介者(Service Broker)、服務需求者(Service Requester)、服務提供者(Service Provider)相互都是分散在各地的，這指出了未來在軟體元件架構上，非同步訊息(asynchronous message)傳遞的考量仍是必須的。另一方面，在分散式運算的領域中，出現了全球虛擬格網(Grid)運算[Global Grid Forum]的概念，也就是將閒置、分散的電腦或個別CPU藉由網路整合其

[4]在UML Components方法論書中8.4節也有提到UML Components與EJB的對映，相應於此方法論並無任何非同步訊息傳遞之考量，其提出之對映模式也沒有這樣的思維。

運算能力，成爲一個獨立處理資訊的系統。格網將打破個人電腦運算的界線，未來的運用將潛力無窮，其勢必衝擊固有的社會各個面向，對人類生活產生莫大影響。可以確定的是，「格網」研究將是一個新興的知識領域，其重要性不可漠視。而格網(Grid)運算也是屬分散式的運算，所以當然也需要有非同步訊息(asynchronous message)傳遞的考量。由Web Services及格網(Grid)運算的趨勢可知，在未來因分散式運算而需要納入非同步訊息(asynchronous message)傳遞考量是必然的，所以將來其他這類的對映(mapping)研究也勢必要有非同步訊息(asynchronous message)傳遞的考量，例如Catalysis[D'Souza，1999]對COM+。而提出這個不可忽視的非同步訊息(asynchronous message)傳遞之考量點也是本篇的貢獻之一。

由以上(一)、(二)兩點所陳述的貢獻可以知道這篇文章在業界是相當有實用性的。業界當是使用UML Components這個優秀的方法論來作爲元件式塑模方法時，即可以在方法論流程中實際加入因元件分散而使用非同步呼叫的考量。而當產出最終的元件規格後，若業界是使用前述較具優勢的Enterprise Java Beans元件軟體架構來做爲實作之元件模式時，那本文章最主要的貢獻，即實際的對映就大有幫助了，有了這樣完整且良好的對映模式，就可以將塑模出來的元件規格以最適合種類之Enterprise Java Beans或最佳方式來快速地實作。故這篇文章對業界在元件軟體發展上有實際且立即的貢獻。

二、未來研究方向

未來可以針對不同的元件式塑模方法論(例如Catalysis[D'Souza，1999]、SCIPIO[Veryard，1998]、O2BC[Ganesan，2001]、UML Components[Chessman，2001]等等)與各種主流的元件架構標準(例如EJB、COM+、CORBA等等)作所對映的研究，並且將來待此類對映研究均成熟後，可以再作回顧彙整的研究。如此的話，有了從元件規格到實際元件實施的完整良好對映，那軟體元件的實際開發就能夠更加順暢，而開發出來的產品品質也會更高。當然，元件式軟體發展(Component-based Software Development)勢必能更臻成熟。

參考文獻

1. [沈建男，2003] 沈建男，Enterprise Java Bean程式設計實務：整合網路資料庫存取的EJB元件程式設計，碁峰資訊股份有限公司，台北，2003。
2. [周政宏，2002] 周政宏，Java訊息傳遞，文魁資訊股份有限公司，台北，2002。
3. [陳鴻明，2003] 陳鴻明，元件塑模方法論：一個植基於UML的方法，國立中山大學資訊管理研究所碩士論文，http://etd.lib.nsysu.edu.tw/ETD-db/ETD-search-c/view_etd?URN=etd-0612103-122452，2003。
4. [Advisor] Sterling Software Component-

Based Development Method (available with COOL:Spex and COOL:Gen products), http://www.sterling.com/cool and http://www.ca.com/products

5. [Booch，1994] Booch, G., Object-Oriented Analysis and Design-With Applications, Addison-Wesley, USA, 1994
6. [Chessman，2001] Chessman, J., Daniels, J., UML Components, Addison-Wesley, Boston, 2001
7. [CNET Networks, Inc.] CNET Networks, Inc., http://cgi.taiwan.cnet.com/jpc /sp3-6.htm
8. [Cook，1994] Cook, S., and J. Daniels. Designing Object Systems, Prentice Hall, 1994
9. [David Booth et al.，2004] David Booth et al.,“Web Services Architecture”, http:// www.w3.org/TR/2004/NOTE-ws-arch-20040211/, World Wide Web Consortium (W3C), February 2004
10. [D’Souza，1999] D'Souza, D., Wills, A. C., Catalysis: Objects, Frameworks, and Components in UML, Addison-Wesley, Boston, 1999
11. [Ganesan，2001] Ganesan, R., Sengupta, S., “O2BC: a Technique for the Design of Component-Based Applications”, Proc. 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems, Santa Barbara, USA, 2001
12. [Global Grid Forum] Global Grid Forum (GGF), http://www.gridforum.org/ggf_ grid_understand.htm
13. [Jacobson et al.，1996] Jacobson et al., Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1996
14. [Jacobson，1999] Jacobson I. et. al., The Unified Software Development Process, Addison-Wesley, 1999
15. [Kathy Sierra et al.，2004] Kathy Sierra, Bert Bates, Head First EJB: Passing the Sun Certified Business Component Developer Exam, O'Reilly, 2003
16. [Marquis ，2002] Marquis, G., P., “Application of traditional system design techniques to web site design”, Information and Software Technology, Vol. 44, No. 9, pp.507–512, 2002
17. [Microsoft Corporation] Microsoft Corporation, http://www.microsoft.com/ com/default.mspx
18. [OIM] Metadata Corporation Open Information Model, http://msdn.microsoft. com/repository/oim/ and http://www. mdcinfo.com/OIM/documents.html
19. [OMG ，1999] OMG, OMG Unified Modeling Language Specification Version 1.3, http://www.omg.org/docs/ad/99-06-08. pdf, June 1999

20. [OMG，2001] OMG, CORBA Component Model Specification, WWW site, http://www.omg.org/techprocess/meetings/schedule/CORBA-Component-Model-RFP.html, Last access July 2th. 2001
21. [OMG，2003] OMG, OMG Unified Modeling Language Specification Version 1.5, http://www.omg.org/docs/formal/03-03-01.pdf, March 2003
22. [Sun Microsystems, inc.，2001] Linda G. DeMichiel, L. Umit Yalcinalp, Sanjeev Krishnan, Enterprise JavaBeans™ Specification Version 2.0, Sun Microsystems, inc., USA, August 2001
23. [Veryard，1998] Veryard, R., “SCIPIO: Aims, Principles and Structure”, SCIPIO Consortium, April 1998
24. [Warmer，1999] Warmer, J., Kleppe, A., The Object Constraint Language-Precise Modeling with UML, Addison-Wesley, 1999
25. [Yi Lin，2004] Yi Lin, Mapping Component Specifications to Enterprise JavaBeans Implementations, 2004

作者簡介

林至中

1995 年 12 月獲得美國德州大學阿靈頓分校電腦科學博士學位，並於 1996 年 8 月起服務於朝陽科技大學資訊管理系專任副教授，2003 年 8 月起轉赴銘傳大學資訊管理系專任副教授至今。學術研究領域爲軟體工程、資訊系統架構、系統分析設計、物件導向技術。

陳建霖

私立銘傳大學資訊管理研究所碩士班研究生。研究興趣爲元件式軟體發展技術、物件導向技術、資料庫系統原理。