國立交通大學

管理學院(資訊管理學程)碩士班

碩　士　論　文

應用主題地圖建立物件導向程式庫樣式

Using Topic Maps to Establish Reuse Patterns of OO Programming Library

研 究 生：翁嘉宏

指導教授：劉敦仁教授

中華民國九十三年六月

應用主題地圖建立物件導向程式庫樣式

Using Topic Maps to Establish Reuse Patterns of OO Programming Library

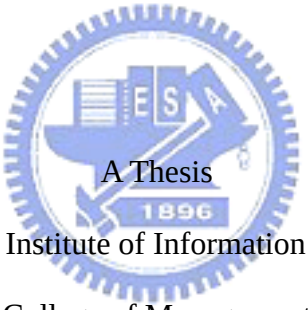
研究生：翁嘉宏　　Student: Chia-Hung Weng

指導教授：劉敦仁　　Advisor: Duen-Ren Liu

國立交通大學

資訊管理研究所

碩士專班論文



A Thesis

Submitted to Institute of Information Management

College of Management

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Business Administration

in

Information Management

June 2004

Hsinchu, Taiwan, the Republic of China

中華民國九十三年六月

應用主題地圖建立物件導向程式庫樣式

研究生：翁嘉宏　　指導教授：劉敦仁 老師

國立交通大學資訊管理研究所

摘要

隨著電腦技術進步，資訊化、數位化的速度也越來越快，越來越多人與企業相當依賴資訊系統。所以資訊系統的發展也越來越重要，如何能夠更快速、更正確的發展出對人與企業有用的系統呢？建立一套軟體的程式庫是相當重要的。但是如何建立、使用電腦軟體的程式庫不是一件簡單的事，程式庫給軟體工程師使用時，要讓他知道正確方式來使用，且使用這些元件時還要注意哪些細節，尤其建立一套應用程式時，通常是許多的程式庫交叉使用，如何釐清彼此關係而不至於誤用，且將複雜度降到最低，這是在發展應用程式時很重要的議題。

資料探勘的技術主要是想從大量的資料中，探勘出有用的資訊，將大量的資料中找出有價值和可利用的部份，應用在我們平常的活動或決策中，能夠更快速且正確地完成每件工作。

主題地圖可以提供、展現主題知識的方法，是一種使用數位化的方式來表現多個主題的資訊與關連，每個主題是一個題材名稱並關聯一些資訊與概念，有了主題地圖，人們可以更視覺化的來快速學習一些事情。

本研究是利用資料探勘的方式來挖掘程式庫與程式的關係，主要是從應用程式翠取分析出程式庫與程式之間的關係，以提供未來程式重整時的依據，並建立主題地圖供程式人員快速學習，來減少學習曲線。這個方式的優點是可以節省程式人員的時間，也減少公司所付出的成本。

在本研究的架構下，完成了使用自動化的機制，讓程式樣式能夠維持最新的版本。可輸入多個專案程式原始碼，得到這些程式碼的關係。我們可以觀察哪些程式影響哪些程式，哪些程式的品質對於專案的品質有較大的影響，這些明確的結果可以提供專案軟體工程師與程式庫軟體工程師體認當程式發生問題時該問題的嚴重程度，知道工作的優先順序與重要性。

Using Topic Maps to Establish Reuse Patterns of OO Programming Library

Student: Chia-Hung Weng Advisor: Duen-Ren Liu

Institute of Information Management
National Chiao-Tung University

Abstract

Following a rapid pace of Information Technology, the process of translating data into digital form and the utilization of computer systems have stronger momentum. A growing number of workers and companies closely depend on information systems, and naturally increase the importance of them. Although a software library can significantly contribute towards an efficient and effective development of software, to establish and apply a library is never a simple task. When a software engineer uses a library, he has to know proper procedures and details of each component. Besides, a library can frequently overlap various libraries in any software project. Obviously, it is a very important topic to clarify relations of libraries and avoid misapplication of them, as well as minimize their complexity.

Data Mining is a technique which is trying to dig out worthwhile information from massive data. If we can seek valuable and useful parts from abundant facts, and adjust our action or strategy based on them, then we can fulfill every task properly and precisely.

Topic Map is not only a method to provide and display thematic knowledge, but also a method to present information and relationship of multi-themes. Each theme is a name of a subject and a collection of information and concepts. With the aid of Topic Map, people can learn faster in a more visual way.

This study is to investigate the relationship between libraries and programs by the methodology of Data Mining. We extract the associations and analyze them so as to form a guideline for any future refactoring.

In addition, Topic Map can help software engineers to shorten the learning curve and save both programmers' time and companies' costs.

Within our framework, we use automatic mechanisms to maintain up-to-date versions of software. After inputting source codes, we obtain relationship of programs in a project. Therefore, we can observe how programs influence each other, and predict key programs when quality of the project is concerned. Thus project engineers and library engineers can acknowledge the possible damage when a specific program breaks down, and then they can know the severity and prioritize their tasks accordingly.

目錄

表目錄

圖目錄

第一章 緒論

第一節 研究背景及動機

我本身的工作經驗經歷過三家軟體公司，前後的工作時間大約有五、六年。我們可以了解一個成功的軟體公司必須要不斷寫符合客戶需求的軟體送交給客戶使用，也要能夠做好客戶的售後服務，所以開發軟體的速度與品質是一個專案成功的關鍵。

公司內人員的流動是很正常的事情，公司都會有人員離職，而該人員離開後，後來接手的員工對於該人員之前寫的原始碼需要花一些時間來了解，並熟析其中的關係。但是對於一些比較難以理解的程式，就會難以維護，常常很多邏輯、或判斷流程、相依性都很難從程式原始碼看出來。所以面對此類的問題，如何快速整理這些程式原始碼、理解後並知道如何修改是很重要的。

第二節 研究目的

物件導向軟體開發方式已經是軟體開發的主流，在軟體專案開始啟動時，我們也會透過系統分析、系統設計的步驟來實作此專案，因此有統一塑模語言(UML)[5]來統一、規範這些規則，其主要目的也是想用視覺化的方式來訂定、建構一套好的軟體系統。

雖然有統一塑模語言與CMMI 等工具來幫助軟體公司，能在接專案時先做先前的分析設計與文件撰寫，以避免專案的失敗，但是客戶(購買者)的心態都是想快點收到系統來使用，軟體公司為了賺錢，也是想快點將專案結束來收到錢，再繼續接下一個專案。所以為了趕上客戶的行程，工程師就沒有花太多時間來照著正規的步驟進行系統分析與系統設計。雖然工程師在建置專案時可以準時完成工作，但是事後工程師要維護自己的專案時，卻是要花上更多的時間來修正，如此公司為了趕此專案，後續的維護勢必會花費更大的成本。根本的原因在於開發時時程太趕，而很多事情、流程都沒有想清楚、定義明確，以致開發後的程式原始碼錯綜複雜難以維護，但是如何減少這種狀況發生呢？

有了以上實際遇到的問題，我們希望建立一套系統來分析程式原始碼，除了能夠了解程式與程式之間的關係外，進一步希望能將分析結果提供給維護的工程師，以及想要應用這個程式庫的工程師，來加快他們的學習曲線，增加公司的整體效率。

簡單來說，本研究目的在於利用資料探勘的技術來分析程式原始碼，使未來程式發展更物件化、更組織化，並使用主題地圖來表示程式庫樣式，利用圖形化介面與操作來清楚表達要說明的概念，快速提昇軟體工程師對於程式庫了解的效率。

第三節 研究流程

本研究目的在於發展一套程式庫樣式的主題地圖，主要是想提昇軟體工程師對於程式庫的吸收效率，若文字為基礎的方式來描述文件有時很難清楚表達要說明的概念，所以我們使用主題地圖來改進這樣的問題。

本研究的架構方式包括以下步驟：

A. 確認研究方向：針對本研究的動機與目的來確認本研究的方向。

B. 文獻探討：搜集及閱讀國內外相關的文獻與書籍，加以彙總整理。

C. 系統規劃：本研究的問題描述與規劃系統的架構。

D. 資料選取：說明如何選取研究資料

E. 建構系統

I. 資料前處理及資料轉換：分析程式原始碼的架構

本研究將提出自動化分析程式原始碼電腦輔助方式來分析物件導向的程式原始碼。

此程式可以將物件導向程式原始碼分析出類別實體化、方法呼叫，並將資料結果紀錄在資料庫中。

II. 使用關聯法則分析原始碼的架構

將步驟 I 的分析結果使用關聯法則分析，找出最常使用的程式樣式。

III. 將分析結果建立一個主題地圖

將處理的結果資料轉為主題地圖資料格式，主題地圖的資料格是採用 ISO 所定義的資料結構。

IV. 將主題地圖視覺化

利用一些工具將所分析出的主題地圖使用視覺化的方式展現出來。

F. 令人感到興趣的關聯法則評估：對於所挖掘出之關聯法則，評估令人感到興趣的關聯法則

G. 研究結論與建議：提出本研究之結論、限制及對未來研究之建議。

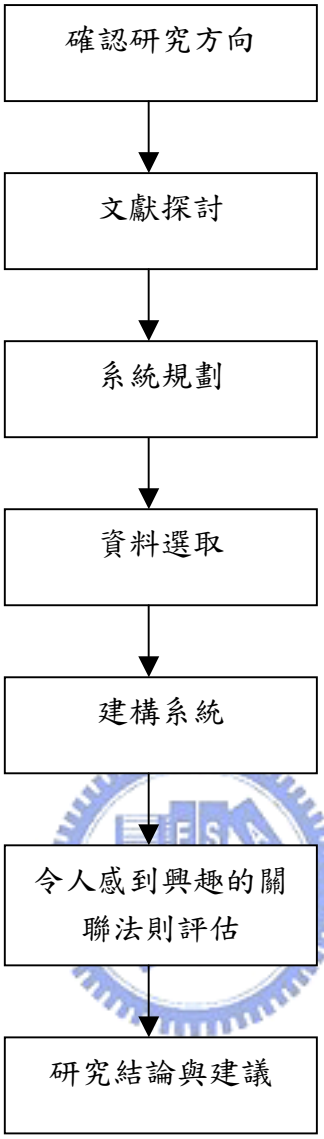


圖 1.3.1 本研究流程圖

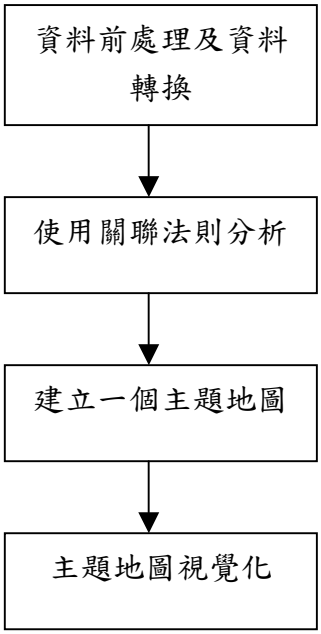


圖 1.3.2 建構系統流程圖

第四節 論文架構

本研究分為五章。第一章緒論，包含研究背景及動機、研究目的、研究流程及論文架構。第二章為文獻探討，包含知識管理、資料探勘、軟體工程、主題地圖。第三章為系統架構概論，包含了問題描述與規劃、系統架構。第四章為雛形系統建置，包含了資料簡述及資料選取、資料前處理、轉換。第五章為雛形系統結果，包含了資料探勘、建構主題地圖。第六章為結論與建議，包含了研究結論、研究限制、研究建議。

第二章 文獻探討

第一節 知識管理

什麼是知識？根據定義[21]我們可以說：知識是我們所知道某些事情。例如我們知曉、熟悉、經歷過、了解或理解一些事實、方法、理論、技術。知識是我們知道如何使一件事情發生，知道造成這樣結果的方法與道理。知識是我們累積一些方法、技術等，而這些資料可能是存在書籍、雜誌、電腦程式裡[21]。

顯性知識是這項知識已經被充分的了解，表現的方式可能以文字方式表示、表格方式表示、圖形方式表示等。例如：在數學式中，我們可以透過各種公式精準的計算出一個物體的面積、體積的數值。

隱性知識是這項知識我們還無法正確清楚的表達，換句話來說，我們知道的超過我們能夠表達的。例如：工程師在處理事情時，會依直覺的去處理工程上的問題，但是他並非毫無根據來處理這些事，所以這其中必定隱藏一些知識尚未能夠清楚表達出來的。還有一些隱性知識是我們常常運用的，但是我們本身卻都不知道我們在使用何種知識。例如：我們可以清楚的辨識每一個人的臉，但我們卻不知道透過什麼方法來辨識的。

隱性的知識可能存在於兩種格式。第一，知識藏在人的行為表現上或是社會網路上。第二，知識可能隱藏在人類創在出來的流程或是產品[13]。若能夠將隱性知識發掘出來，對於企業或是產業可能造成新的創新與改革。

我們也常常不自覺地使用一些隱性知識，但很少人會去發覺並整理出來。例如電腦程式的原始碼就是如此，其中藏有許多的隱性知識。而這些隱性知識代表著企業的核心價值、與企業流程，如果能夠將這些知識分析出來並將這些知識結構化會使公司企業得到更大的幫助，而開發的軟體公司也能清楚的發展更新、更好的軟體。

知識與知識間的轉換

A. 隱性知識轉成隱性知識

我們可以經過一連串的觀察、模仿與練習來獲得他人的知識。例如：以前還沒有九年國民教育時，大家的知識與工作技能，都不是在學校中學習的，幾乎都是師傅帶著徒弟，徒弟藉由觀察、模仿師傅的行為從中學習的，且學習後也沒有書籍或是文字紀錄、表達學習到的知識。徒弟靠著觀察師傅的所作所為，模仿學習，再自行練習後所得到心得而獲得知識，像這樣的知識傳遞就是屬於隱性知識變成隱性知識。

B. 顯性知識轉顯性知識

將不同的已知的知識組合後成為新的知識。例如：將市場上的統計資料用已知的資料探勘技術來分析，得到新的知識與結果供給CEO做市場策略使用。這些合併後的知識轉變出來的新知識是可以很清楚表達出更有價值的知識。

C. 隱性知識轉顯性知識

將隱藏的知識透過人的紀錄，修正成可用模型，定用文件記錄下來供其他人學習，或是依據這些規格來做事。透過文件可清楚表達，讓其他人容易獲得的知識，該文件即為一種顯性知識。

D. 顯性知識轉隱性知識

將顯性知識了解、吸收後，經過一段時間的經歷後，依據自己的看法、經驗而整理出屬於自己的獨特做法，這就是將顯性知識內化成隱性知識。

近來各公司企業、每個政府機構都越來越重視知識管理，甚至花大筆的經費建置知識管理的平台來達成組織學習與知識傳承，主要目的是將重要的知識留在組織中，不會因為人員的變遷異動而有所影響。

促使組織學習與知識傳承的六個步驟[23]：

A. 計畫

在我們計劃要做任何事情前，最好要將該做的事整理一下，什麼事該做，範圍界定清楚，而此事又該如何執行？而在計畫組織學習時也該如此，

我們也需要定義知識學習的範圍與界線，並且找尋企業中熟悉這些知識的專家與其他有意願參與的人員，依照他們專業，在計畫執行中定義角色來分工，並清楚定義他們的權利與義務，使組織學習能夠順利的進行。

B. 知識訪談

知識訪談簡單來說就是蒐集所有的知識，安排行程至每個組織中訪談人員，蒐集他們的知識與經驗，並且要確定他們的意見是正確無誤地記錄下來而沒有被誤解扭曲。而這些人所提的一些假設狀況與建議事項，可能會因為角色不同，立場不同，所提供的意見內容會包括不同的觀點，從不同的角度分析與評估事情，而這種現象正是我們期望的。

C. 萃取精華

這個階段的工作是將上一個階段所得到的知識訪談結果來做一個整理，由一群德高望重的人來主導整個整理的流程，這一群人也許人數不多，但是他們的專業與技術必須是得到他人的肯定。這一小群熟悉專業知識的人將這些未整理、雜亂的意見，加以刪減、歸類歸檔，並將所有的文件整理成有系統的、完整的知識樹，並將相關的文件放在相關主題的知識樹中。

D. 記錄

由專家整理的結果報告文件應該再回到該組織團體，由組織團體一起來完成使這份報告成為該組織的正式文件，首先每個部門將這些有系統的文件閱讀、思考過後，提供想法與意見，專家們將這些意見紀錄下來，經過幾次會議討論修改，將定案的版本交由文件收藏部門記錄成為發表前的版本。

E. 修正

將上一個階段的文件版本，需經過幾次的校稿，將錯字誤字，或是語意不通的句子，重新做整理編排，由編錄者修正與確認裡面的內容是否無誤，將編排後的結果交給專家審過以後，即可成為最後的版本。

F. 散佈知識

透過一些管理機制與制度，將這些文件放在網路上與閱讀室讓需要的人容易查閱與取得，其中需要注意的是權限問題，並不是所有的人都有閱讀、查詢、修改這些文件的權限。查詢文件時比較敏感的資料是不會被查詢出來

的，機密以上的資料不可讓人隨意閱讀，修改資料時也要特別注意，當原作者將資料上傳後，他是否還有權限可以修改？這些相關作業在我們散佈知識時，要特別注意，需要事先想清楚些。

另外利用資訊的技術我們可以提供電子報或是簡訊的機制，讓需要的人訂閱，當有相關最新訊息時，主動告知相關的訂閱人員。

第二節 資料探勘

一、資料探勘簡介

什麼是資料探勘？簡單來說，資料探勘是從大量資料中存取或發掘知識[12]。資料探勘可結合智慧化及各種統計方法分析此龐大的資料，來發掘出不同而有利的資訊與知識，並提供未來決策或預測之用[38]。資料探勘最終的目的就是希望將資料經由採礦之後可以發掘出新知識，而這些知識的價值在於可以提供前所未知的資訊，供其他人在決策支援或其他方面運用。

資料探勘由以下步驟所組成的[12]：

A. 資料清理(Data Cleaning)：將不需要、多餘的的資料清除，對錯誤的、遺漏的資料做修正處理，最主要的目的是將雜亂無章的資料整理成一致的資料，並消除干擾(noise)。

B. 資料整合(Data Integration)：將多種資料來源組合在一起。例如有些資料是在資料庫中，有些資料放在 Excel 中，將這些資料經過轉換程式存在統一的資料庫中。

C. 資料選擇(Data Selection)：從資料庫中搜尋分析與任務相關的資料。在資料中所有的資料欄位選擇我們所需的欄位來做分析。

D. 資料轉換(Data Transformation)：將資料轉換或統一成適合探勘的格式，例如通過匯總或聚集操作來達成。

E. 資料探勘(Data Mining)：運用資料探勘的演算法來挖掘並取得分析後的資料樣式。

F. 模式評估(Pattern Evaluation)：經評估或辨識資料樣式是否令人感到

興趣，也就是根據某種有趣度度量(Interestingness Measures)來辨識並表達知識的真正有趣模式。

G. 知識表達(Knowledge Presentation)：使用視覺化的表達技術，來讓使用者瞭解探勘出的知識。

資料探勘主要的功能有，關聯法則分析(Association Analysis)、分類和預測(Classification and Prediction)、群組分析(Cluster Analysis)、序列分析(Sequence Analysis)、廣義化(Generalization)[26]。除此之外應還有孤立點分析(Outlier Analysis)。

A. 關聯法則分析(Association Analysis)

關聯法則分析是去發現關聯規則，而這些關聯規則展示了某些特殊的值頻繁地在給定資料集一起出現的規則。關聯法則運用在大量的資料中找出某一項目是否與另一個項目有關。例如：從超商顧客的購買記錄中，我們去找牛奶與麵包的關連，牛奶 => 麵包 [support=2%， confidence=60%]，表示在所有的交易筆數中同時包含牛奶與麵包的交易筆數有 2%，即支持度為 2%。在含有牛奶的交易筆數中包括麵包的交易筆數有 60%，即信賴度為 60%。

B. 分類和預測(Classification and Prediction)

分類的過程會找出描述並區分資料或概念的模型，以便能夠使用模型來預測標記未知的物件類別。

分類法是先定義一些類別，再將資料庫中的資料分到這些類別中。分類屬於監督式學習，最普遍的方法是決策樹(decision tree)，首先將資料分為兩部份，第一部分為訓練的資料，第二部分為測試資料，資料量大約是 2:1 的比例。先從訓練資料中找出歸類法則，再將測試資料放入這些法則中來評估歸類的正確性，若正確性還算不錯的話，則找出的規則可做為未來歸類使用。

C. 群組分析(Cluster Analysis)

將資料庫的資料分析其中的特性，將所有的資料區分為幾個群組，使每個群組內的資料都有很高的相似性，而不同的群組則顯示

不同的特性。與分類和預測不同的，群組分析在分析資料物件時，並不考慮已知的類別標記。此法可做為分類編製(Taxonomy Formation)，它會將觀察到的內容組織成類別階層結構，並把類似的事件組織在一起。

D. 序列分析(Sequence Analysis)

有些資料會隨著時間變化而有一定的規律或趨勢，利用此技術的分析可找到資料隨著時間變化的特定模式。序列分析可能包括時間相關資料的特徵化、區分、關聯、分類或叢集。其實只是用其它分析方式再加上時間因素而已。處理具有時間序列的資料，其使用的方式可能使用統計法或類神經網路技術。

E. 廣義化(Generalization)

資料廣義化是一個過程，它將任務相關的資料集從較低的概念層抽象到較高的概念層。當使用者只想從龐大的資料庫得到概略性結果時，就可以利用屬性關係做概略性的歸納，再用其他的技術，來發掘知識讓使用者得到滿足的需求。

F. 孤立點分析(Outlier Analysis)

資料庫中可能包含一些資料物件，它們與其他資料的一般行為或模型不一致，這些資料就是孤立點。大部分的探勘技術都會將孤立點視為雜亂或是異常而將它排除，然而在一些應用中，孤立點探勘比正常事件更加有趣，例如我們可應用在欺騙偵測等。

二、關聯法則分析

關聯法則分析的定義，關聯法則分析由 Agrawal 在 1993 年提出，主要是從大量資料項目集合之間發現有趣的關聯或相關的關係。最典型的例子是購物籃分析，透過顧客放在購物籃的不同商品之間的關聯，分析每筆交易紀錄，來了解顧客的購買行為，找出令人感興趣的關聯法則。而這些關聯法則的目的可用來制定行銷策略，例如相關產品的擺放位置、廣告策略、目錄設計之決策等。

關聯法則如下：

牛奶 => 麵包 [support=2%，confidence=60%]

表示在所有的交易筆數中同時包含牛奶與麵包的交易筆數有 2%，即支持度(support)為 2%。在含有牛奶的交易筆數中包括麵包的交易筆數有 60%，即信賴度(confidence)為 60%。

假設 I = {$i_1, i_2, \cdots, i_m$}是項目的集合。設任務相關的資料D是資料庫交易的集合，每個交易T是項目的集合，為I的子集合。每個交易有一個標識符號，稱作TID。設A 是一個項目集合，交易T包含A，A亦是I的子集合。

關聯法則的形式如 A=> B，A 為前因項目組(antecedent itemsets)，B 為後果項目組(consequent itemsets)。支持度是指在 A 和 B 兩個項目同時在交易紀錄 D 出現的次數與交易紀錄 D 的交易總筆數的比。支持度代表事件發生的機率，support(A=>B)代表 A 與 B 兩個交易同時發生的機率，其值介於 0%和 100%之間。信賴度為 A 和 B 兩個同時在交易紀錄 D 出現的次數與 A 項目在交易紀錄 D 出現次數的比。信賴度代表已發生某事件的情況下，另一事件發生的機率，confidence(A=>B)代表發生 A 交易項目下，又發生 B 交易項目的機率，其值介於 0%和 100%之間。

關聯法則是否可使用，需視同時滿足最小支持度門檻值(minimum support)和最小可信度的門檻值(minimum confidence)。同時滿足最小支持度門檻和最小可信度門檻的規則稱做強規則(strong)。支持度門檻和可信度門檻的設定是很重要的，當門檻設太低時，會將重要性較低之項目也包含進來，而設太高又怕因此而失去某些重要規則。

第三節 軟體工程

一、UML 簡介

在軟體工業界，近年來越來越重視軟體發展的程序，原因是希望專案能夠順利的結案，其中包括了能在預算期間內將專案完成，並保證軟體的品質等[36]。物件導向分析與設計在現代的軟體設計中已經應用地很頻繁了，但是到底怎樣的設計方法可以將外在的人事時地物使用物件導向的觀念來表示呢？

統一化模型語言(Unified Modeling Language，UML)為建構軟體發展程序模型。有四個優點：第一、標準化，因為使用普遍的符號，程序模型能夠容易理解。第二、模型化，UML 提供大量集合的圖（類別圖、物件圖、合作圖、狀態圖等)。第三、視覺化，採用了多種易於瞭解的圖，能夠看了以後，馬上瞭解其關係與架構。第四、可重複使用，圖中是以物件來架構的，所以重改架構時，物件可重複的再使用[35]。

在統一塑模語言中有九種圖形：類別圖(Class diagram)、物件圖(Object diagram)、使用案例圖(Use case diagram)、順序圖(Sequence diagram)、合作圖(Collaboration diagram)、狀態圖(State Chart diagram)、活動圖(Activity diagram)、元件圖(Component diagram)、部署圖(Deployment diagram)，其意義簡述如下[16]：

A. 類別圖(Class diagram)

類別圖是一種靜態圖形，在塑造物件導向系統時，類別圖是其中最重要的圖形，透過類別圖可以將一組類別、介面、合作以及上述元素之間的關係顯示出來。一般而言類別圖具有以下元素：

- 類別
- 介面
- 操作與常數
- 方法
- 屬性的型態資訊
- 可瀏覽性
- 相依性、一般化關係與結合關係

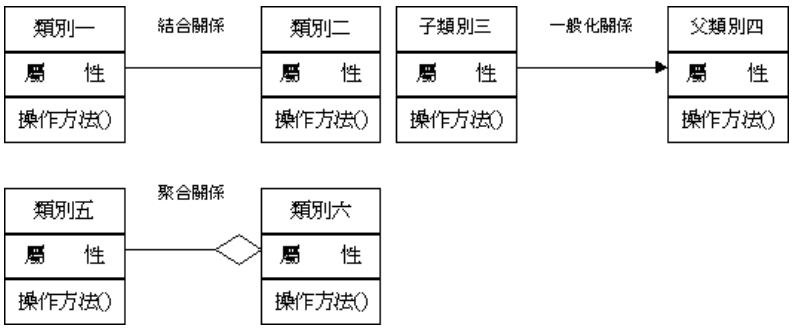


圖 2.3.1 類別圖

B. 物件圖(Object diagram)

物件圖可以顯示某一個時間點一組物件及這一組物件與其他物件的關係。我們可以利用物件圖來塑造系統的靜態設計觀點和靜態行程觀點。

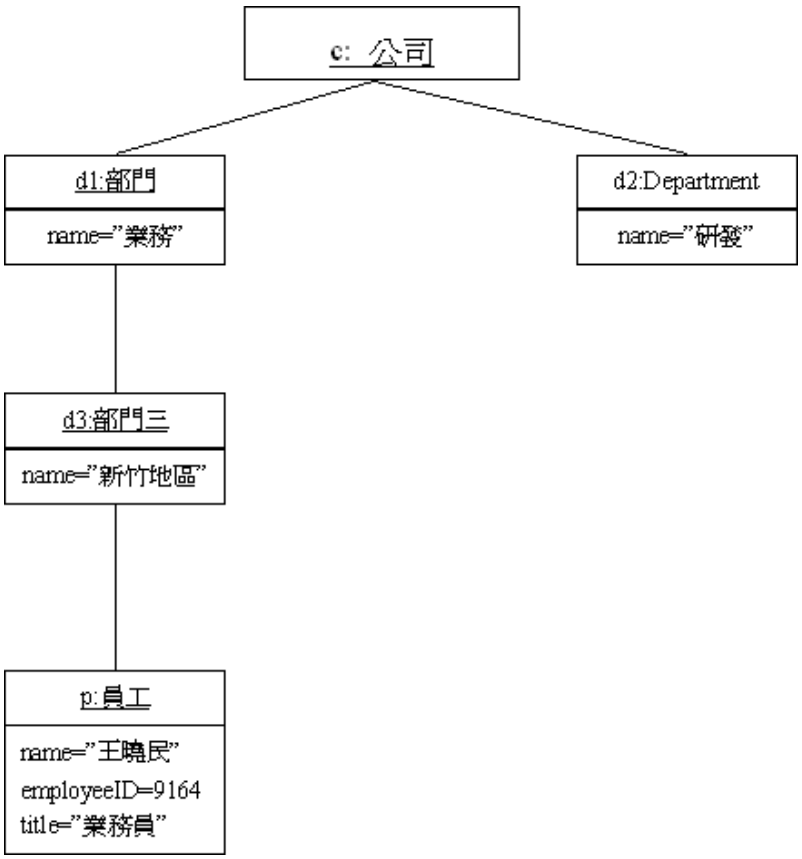


圖 2.3.2 物件圖

C. 使用案例圖(Use case diagram)

使用案例圖在 UML 種是用來塑造系統動態觀點的五大圖形之一，在 UML 中的使用案例不只是圖，它是文字型的文件，建立使用案例模型時的主要工作也是在寫文字而不是圖，然而，UML 中定義了使用案例圖，圖中可以展現使用案例與參與者的名稱的關係。

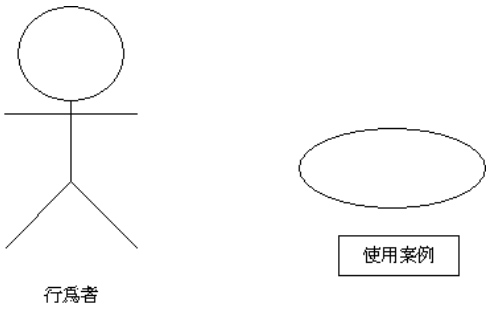


圖 2.3.3 使用案例圖

D. 順序圖(Sequence diagram)

互動圖包含兩種圖形：順序圖、合作圖。順序圖的特質：表現系統的動態角度、表現物件之間的互動、可塑造事件流、表現物件之間互動與時間或關係的差異。

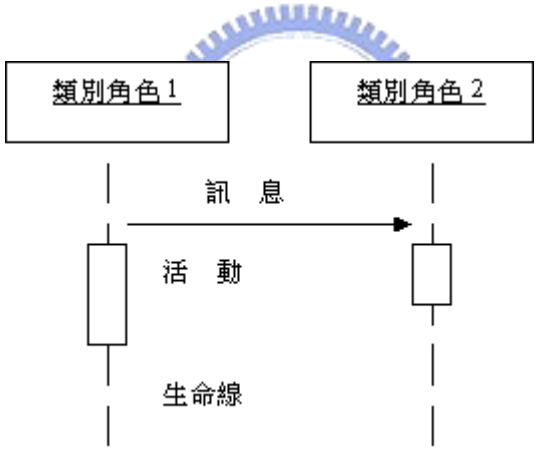


圖 2.3.4 順序圖

E. 合作圖(Collaboration diagram)

順序圖與合作圖都屬於互動圖所以擁有相同的特性，若依時間順序塑造控制流程就使用順序圖，若依物件組織塑造控制流程就使用合作圖。

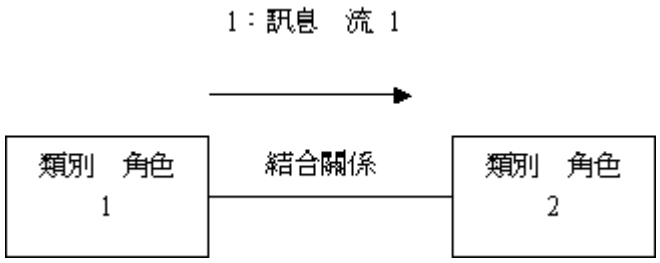


圖 2.3.5 合作圖

F. 狀態圖(State Chart diagram)

在一個系統中事件(event)是代表很顯著或是值得注意的事件，UML 使用狀態圖展現物件中我們感興趣的事件與狀態，以及物件回應事件的行為。

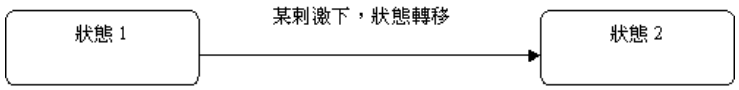


圖 2.3.6 狀態圖

G. 活動圖(Activity diagram)

活動圖提供我們很豐富的表示法可以秀出活動的先後順序，我們可以將它應用在企業的工作流程或企業流程中，也可以秀出使用案例中的步驟。基本上活動圖就是流程圖，可以顯示活動與活動之間的控制流程。

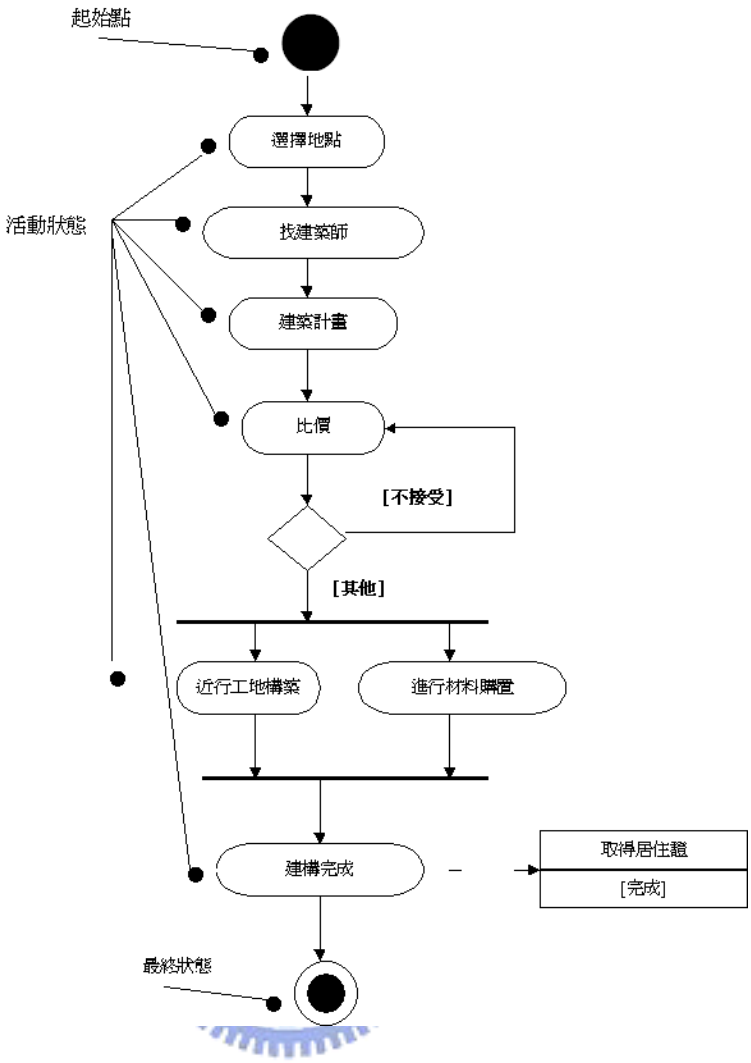


圖 2.3.7 活動圖

H. 元件圖(Component diagram)

元件代表系統中可模組化、可配置化、可取代的部份，裡面會把實作封裝起來並顯示一組介面出來。元件圖是塑模圖形裡面兩種(元件圖、部署圖)可以用來塑造物件導向系統實作角度的圖形之一。透過元件圖可以表現出一組元件之間的組織與相依關係。

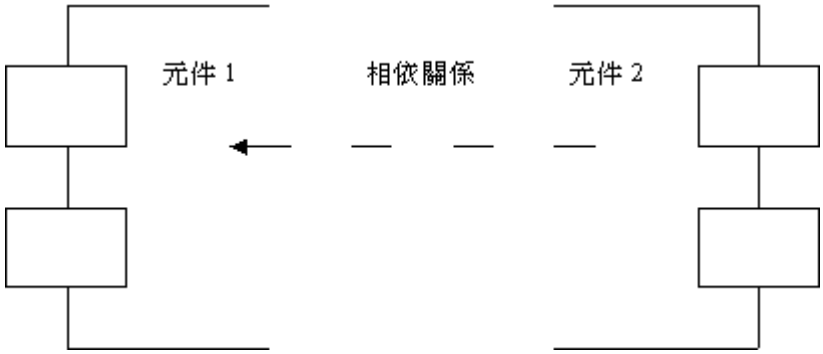


圖 2.3.8 元件圖

I. 部署圖(Deployment diagram)

透過部署圖可以表示執行時間進行處理的節點以及存在該節點上的元件。

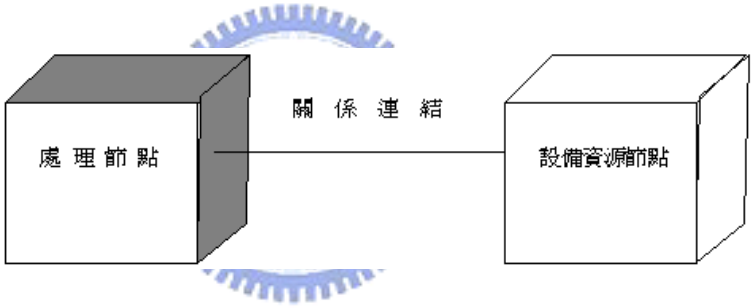


圖 2.3.9 部署圖

二、物件導向程式庫樣式(OO Programming Library Pattern)

為了提昇軟體開發的效率，我們會建立程式庫。建立程式庫的原因是將軟體工程師常用的類別程式放入程式庫，而達到物件重用(Reuse)的目的。

Will Tracz 曾經說過：If you have components to reuse, then you need to glue them together… After you glue pieces together long enough, you start seeing a pattern, then you can reuse the glue too[25,p41]

我們這裏舉出五個物件導向中常用的關係：類別繼承(class

inheritance)、類別實體化(class instantiation)、方法呼叫(function invocation)、方法重製(function overriding)、隱性呼叫(implicit invocation) [19]。

A. 類別繼承

物件導向的繼承性可定義為將具有共同特性的類別導入到一個通用的類別底下。而此通用的類別就可視為這些類別的父類別，而底下較具特色的類別則可稱為子類別[11]。

B. 類別實體化

實體化是物件導向程式設計的專門用語，是指產生一個類別實體的動作[2]。換言之，實體化就是用一個類別產生一個物件。所以物件可能產生在 stack 或是 heap 中，若宣告一個非指標物件實體化就會放在 stack，若使用 new 或 malloc 物件就會放在 heap 中。

C. 方法呼叫

物件利用訊息的傳遞與其他物件溝通，這些訊息將會呼叫方法，而被呼叫的方式將會利用函式來處理傳來的物件並回傳其結果。其中呼叫一個方法指的是傳遞訊息者須等待接收者在執行某些方法後所回傳的結果，反之傳遞一個訊息所指的是傳遞不需要等待其回傳結果[11]。

D. 方法重製

重製性能就是子類別可以將父類別所提供的方法改寫，使用另一個方法取代之。所以就一個類別先繼承一個父類別後，再將父類別既有的方法重新改寫一次，未來將子類別實體化後，使用此方法呼叫都會用子類別改寫過的方法，而不會使用父類別的方法。但若子類別繼承父類別後，不將父類別的方法重製過，呼叫此方法時還是會呼叫父類別的方法。

E. 隱性呼叫

現在的應用程式大概都會有接收訊息後再來處理這些訊息的

機制，而發出訊息者就等於是呼叫接收者去做一些方法的呼叫，這就是隱性呼叫。

例如若使用 Microsoft Windows 的 SDK 寫一個 window 程式都需要寫一個訊息處理的程序來接收外來的訊息，當我們操作 window 的各種功能時就等於呼叫裡面的程序，window 與 window 程式間也可以互相傳送訊息，window 程式內部也可以送訊息給自己，這些都是屬於隱性呼叫。

第四節 主題地圖

一、何謂主題地圖

主題地圖是 ISO 所定的標準，提供定義主題以及主題與主題間彼此的關連，藉由這份標準來標明一組或多組之間彼此有關的主題。在主題地圖中存在一個資訊核心的結構匯集文件，用來集散地圖的主題資訊。

知識管理系統應該具有”容易找尋”的特性，當你想找某項知識時，知識管理系統應該提供一些方式讓你容易找到你要的東西。主題地圖是其中一個精確的知識表示方式，它提供了一些法則，讓人知道如何命名與如何定義一個主題的方式。這點是非常重要的，有了明確的規則讓人遵循，大家可用相同的方式建資料、查詢資料，如此可以減少大家在溝通上的成本，也可以提高做事的效率。

在資訊軟體中也有提供一些機制讓軟體工程師快速找到相關的資訊，例如微軟的 MSDN[29]。Sun Microsystems 也定義了一種 Document 的方式為 JavaDoc[28]，JavaDoc 最主要的目定是從軟體工程師寫在原始碼的註解擷取出來，將這些註解分析後產生出說明文件，轉出的格式依不同公司實做的產品不同，不過大部分的都會支援 HTML，提供放在網頁上直接瀏覽，Java Sun 的 J2SE 1.4 API 即是最好的說明網址[27]。

名稱通常是一個辨識主題的標籤，名稱不需要是唯一的，在許多狀況下也不可能是唯一。有時會有很多主題擁有同一個名稱，當有這樣的情況發生時，需要說明一個主題名稱是如何被應用的，以區分不同意思有同名的問題。例如： China 可能是代表中國的意思，但有些人聽到 China 也許會當成是陶瓷器，這兩者都對，只要他們知道自己正要描述什麼，而能夠讓他人辨識他正在說明的是屬於什麼意思即可。

對於一個主題而沒有其他資訊來補充他到底在說什麼，是讓人很困難辨識的，所以必須提供相關的知識與背景加強描述。而這些描述資訊也很難完全充足，所以無法百分之百將主題描述的清楚。

若我們想要使用電腦來識別主題，依目前技術我們可以使用模糊理論來辨識，但是也無法百分之百保證他的正確性。主題在什麼時候會是什麼意思實在很難明確定義。目前人在處理這些不清楚的知識還是比電腦來的精確。所以依目前的資訊技術，電腦還是很難將主題名稱正確地跟主題連在一起，若一定要使用電腦辨識，必須提供電腦精準、不會模稜兩可的資訊，才可以使電腦正確的識別主題的意義。

所以主題的識別是要提供一個途徑來精準地標示出唯一主題，主題的辨識必須是 one and only one 去識別。主題的識別資訊必須要可以自我描述且包含足夠的資訊來識別出一個主題，除此就不需要額外的資訊。這種識別方式是最符合電腦辨識的，使用 XML Topic Maps 正好符合這樣的需求，在 ISO 標準中有更有彈性且適合描述各種不同事物的機制[8]。

主題地圖提供一些方式來做關連，定義主題以及主題間彼此的關係，使用這樣的技術來標明一組或多組彼此有關聯的主題，在主題地圖中存在一個資訊核心的結構匯集文件，用來集合地圖的主題資訊[37]。

主題地圖含有很多主題，並且提供傳達定址式的群組資訊物件描述主題與主題彼此間的關連，主題地圖含有許多主題的識別與特徵，且主題地圖是一種電子式的表達來呈現各種主題的意思，主題的特徵是有許多名詞名稱、與許多的關聯資訊與關聯概念。基於這樣的結構，主題地圖裡面所提供的資訊足以進一步轉變成有價值的知識庫，且針對不同需求的使用者提供不同的主題地圖，而不同的主題地圖內提供不同的訊息、不同的專業知識供使用者運用。

要使主題地圖成為有用的工具必須首先輸入許多的識別資訊與特徵，因為在你建立主題地圖時，你必須要跟電腦說明這些主題的意思、與每個主題的特徵，有了主題與資訊的連結，未來才可以輸入要尋找的資訊找到相關主題。

主題地圖含有以下元素[8]：

A. 特性(Identities)

我們使用特性來識別不同的主題，重點是儘可能的將所有關於該主題的特性都能夠描述出來。主題地圖有三個特性：主題定址(subject address)，主題表示(subject indicators),資料來源位置(source locators)。

I. 主題定址：主題定址是一項資源，該資源可以明確指出主題是什麼。

II. 主題表示：主題表示是一項資源，該資源可以表示出主題是什麼，它是屬於間接的定址。當主題無法直接定址時，這樣的表示方式是很有用的，其實大部分的主題都是無法直接定址。主題定址與主題表示的不同在於，主題定址是直接的表明主題是什麼，而主題表示則是指出另一項東西，而由這項東西來說明、描述主題是什麼。

III. 資料來源位置：資料來源位置是很多資源，而這些資源會造成這項主題表現。換言之，資料來源位置是指主題地圖中地圖的來源、出處。

B. 領域範圍(scope)

當我們定義一個主題的特性時，我們會不經意地放入我們的想法與主張，但是這些想法與主張並非每一個人都相同，可能會依個人的區域或是依民族不同而有所不同，主題地圖的元素也會受到領域範圍不同而有所不同。

I. 名稱：有些名稱的應用只能在某些定義的背景下應用，但是在另一種情況下使用可能就會是一種錯誤。

II. 相關資訊(occurrences)：某些相關資訊在特性的條件下可能是適當的，但是當時空改變後，這些相關資訊就不是很適當。

III. 相關概念(associations)：某些人對於這個主題的意見有他個人的看法與意見，但是每個人的意見可能相同也可能差異很大，甚至矛盾、衝突。

定義主題的範圍的主要目的在於，主題會有許多不同面，並無

法一次全盤的表示出來。若真的沒有範圍的限制下，代表的意義是這樣主題的描述是到處可使用的，不會因為人事時地物而有所改變。所有的主題特徵都可以加以限制範圍、但也可以不限制。

C. 基本名稱(base names)

基本名稱就是主題的名稱，基本名稱就是用在表示題材的標籤。之前提過，同一個主題在不同的狀況下可能會有不同的名稱。所以同一個主題可能會有一個以上的名稱，而且名稱的數目沒有限制。

基本名稱會有多樣性的特徵，他們可能依照不同的型態而有不同的基本名稱。

例如：啤酒這樣的主題

啤酒(觀點：中文)
ㄆㄧˊ ㄐㄧㄡˇ（觀點：中文發音）

Beer(觀點：英文)
BIR（觀點：英文 KK 音標）

D. 關聯資訊(occurrences)

一般來說，一個主題通常都有許多資訊，資訊與主題一定有某些原因而有所關聯，這些資訊就是透過關聯資訊與主題連結的。藉由關聯資訊，你可以清楚知道資訊與主題的關係為何？為什麼會有所關聯的。但是如果你要將相關資訊與主題連結在一起的話，就要透過相關概念(associations)，因為關聯資訊是可以指出主題特徵，而這些特徵的跟主題有直接關聯。

關連資訊有兩種：來源不清楚的資源稱做內部相關資源(inline occurrences)，而有根據來源證明的資源稱做外部相關資源(external occurrences)

透過關聯資訊可以從由主題連到相關的資訊，且可再由相關的資訊在連回到主題。

再舉啤酒為例：

I. 用大麥和啤酒花為主要原料發酵製成的酒，有泡沫和特殊的香味，味道微苦，含酒精量較低(型態：描述 觀點：中文)

II. An alcoholic beverage brewed from malt and hops(型態：描述 觀點：英文)

III. http://www.realbeer.com/(型態：入口網站 觀點：無限制)

以上的關聯資訊，前兩個為內部相關資源，最後一個為外部資源。

E. 相關概念(associations)

主題與主題之間的關聯使用相關概念來串聯起來。而每個主題與主題中的關連都扮演著不同的角色，彼此之間的關係也用一種型態建構起來。

例如：台北與中華民國這兩個主題有一個關係，台北所扮演的角色是“首都”而中華民國扮演的角色是“國家”。而聯繫著彼此關係的型態是“國家的首都”。

相關概念的關聯是多維度的，再舉以上的例子，我們可以說台北是中華民國的首都，也可以說中華民國的首都是台北。這兩句話的意思是相同的，我們只需要描述一次，不過兩個的描述都對，也就是說我們可以從一個相關概念去創造出多個描述方式。

二、主題地圖的視覺化(Topic Maps Visualization)

視覺化是一項很有前景的技術，當使用者使用很大量的資料或資訊時，為了讓使用者瀏覽起來更容易一些、或是操作更方便。若我們有了一些視覺化的工具時，可以更清楚地觀察彼此之間的關係，且也能幫助我們讓資料處理起來更容易一些[1]。

主題地圖的視覺化最主要的目的在於，幫助使用者能夠更快容易地看清楚資料的關連性、更容易探索資料間彼此之間的架構。主題地圖的視覺化有

兩項需求：清楚地表示(representation)、良好的操作瀏覽(navigation)。

清楚地表示：可以讓使用者得到主題地圖的鳥瞰圖，並且他很快找到他有興趣的地方,可以由大範圍、大方向快速地縮小範圍找到其中詳細的資料。

良好的操作瀏覽：可以讓使用者快速的找到、並取得相關資料。因為資料太多，所以在瀏覽的過程中最好能夠建議使用者的瀏覽路徑，供他們參考以方面找尋資料，尤其新的使用者或是想找尋新的主題的時候，建議的瀏覽路徑就很有價值[9]。

使用視覺化來探索資料的箴言是：首先要全盤概觀，然後找到定點將目標推進並過濾條件，最後再依需求得到更詳細的資料[4]。

以下列出一些主題地圖視覺化的方式：

A. 圖形與樹狀結構(Graphs and Trees)

使用圖形或是樹狀結構的方式呈現對人們是很直覺的,因為這種表示方式所呈現的是將主題地圖使用圖形表示,並用圖形網路來表示主題與主題之間的關聯。

像圖 2.4.1 所示就是一種 3D 互動式主題地圖顯示方式[3]，這是使用一些點與邊所構成的,但是當有成千上百個主題時，這種簡單的呈現方式就會變得越來越雜亂,我們就越難透過此方式找到我們要的資訊，所以我們需要更好的展現工具來呈現主題地圖。

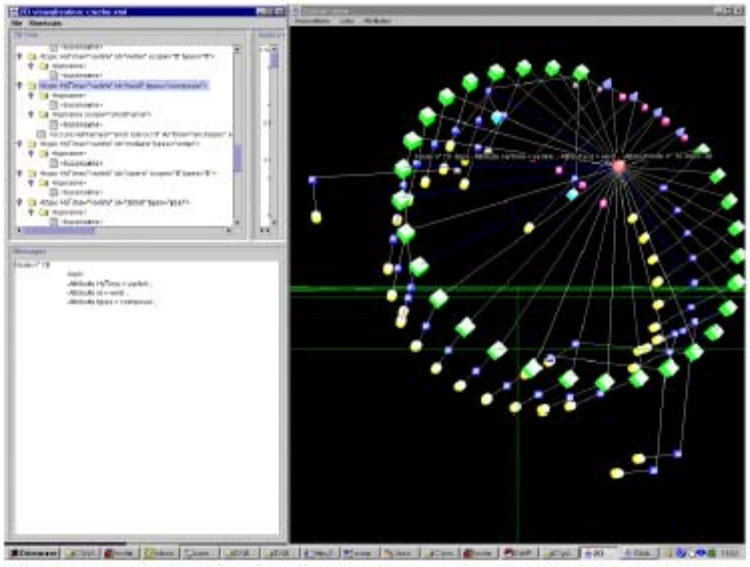


圖 2.4.1 UNIVIT Representation

圖形與樹狀結構是非常適合主題地圖的顯示方式，尤其樹狀結構更符合人們閱讀的方式，因為樹狀比圖形更有結構性，但是主題地圖不是等級制度的(hierarchical)，所以無法全部使用樹狀結構來顯示，不過可將小部分的主題地圖轉成樹狀結構。我們可以運用樹狀結構中的優勢將小部分的主題地圖明顯的表示出來。

圖形與樹狀結構符合我們第一項需求(清楚地表示)，因為它可以呈現整個主題地圖，然而我們需要看到我們有興趣的主題詳細的資料。為了看到詳細資料，我們必須縮小主題地圖的範圍，更集中地鎖定小範圍的詳細資料。

圖形化的鳥瞰方式可以明白的了解主題地圖的結構與相關性，但是其中一個主題被選擇後，應該要很容易看到這個 Topic 的詳細資料。所有就有一種主題地圖的顯現方式很像 Web 的展示方式，當使用者點下一個連結就會打開一個主題或一個關聯。一個範例如圖 2.4.2[22]。

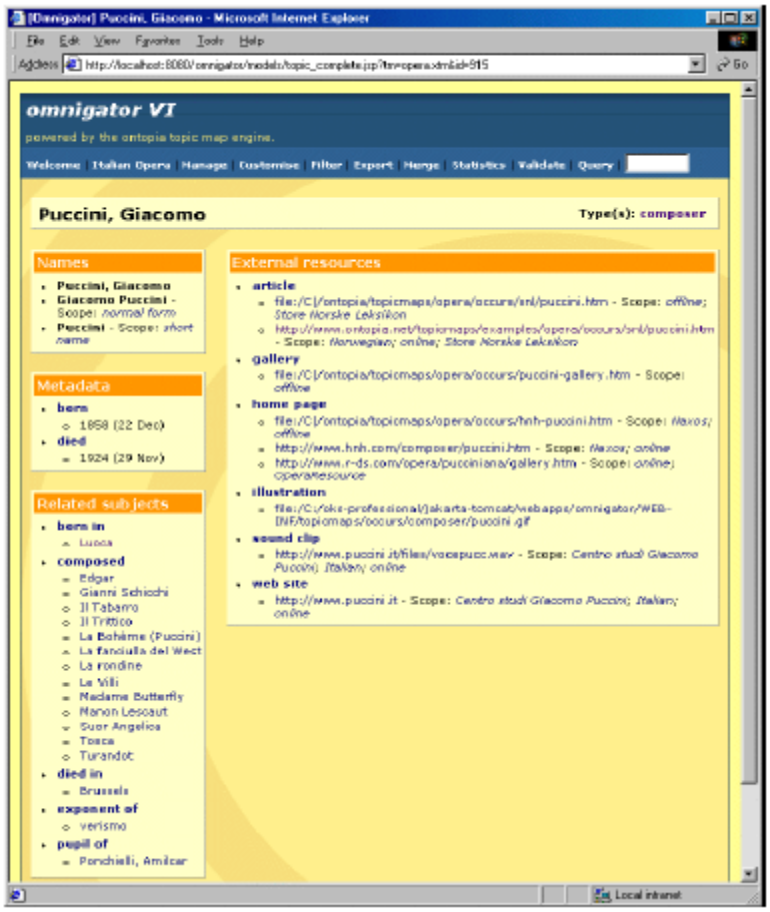


圖 2.4.2 Ontopia Navigator

另一個 View 可以在鳥瞰圖，且可以看到同一個階層的詳細資料，如圖 2.4.3[20]。

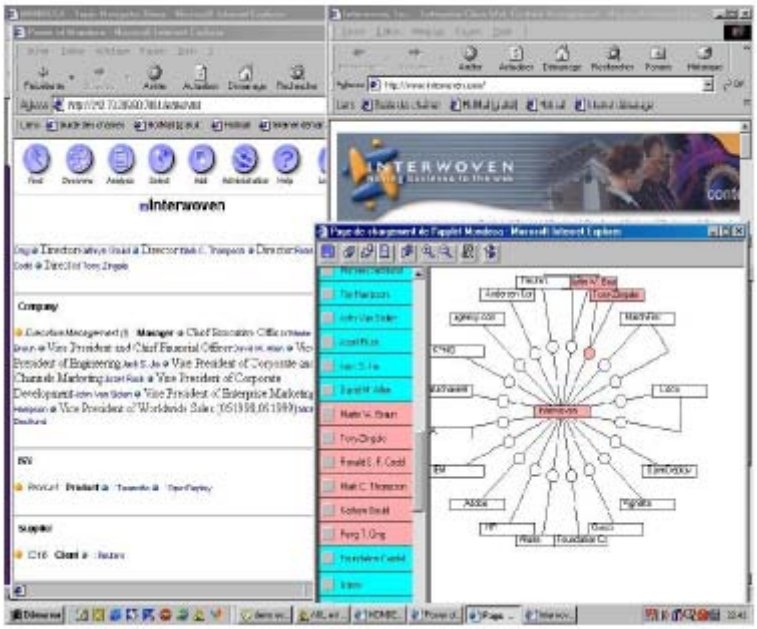


圖 2.4.3 Mondeca’s Topic Navigator

B. 地圖(Maps)

主題地圖就是屬於一種知識地圖，屬於地圖的型態，所以依現代的地理學上，有許多畫製地圖的技術，我們也可以使用這樣的方法達成這樣我們要的目的。例如圖 2.4.4 就是其中一種[17]。

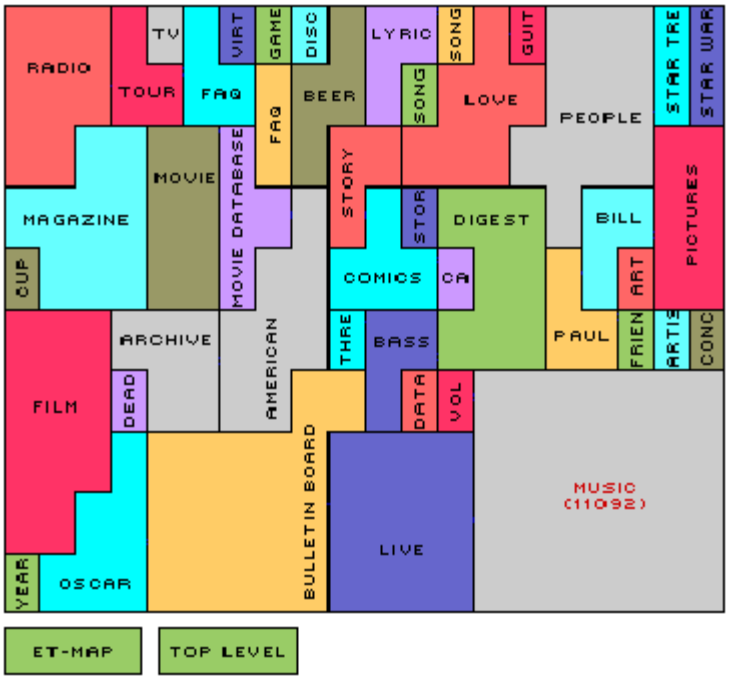


圖 2.4.4 Tree-Map

Self-Organizing Maps(SOM)[24]理論可以將主題地圖放在二維的座標中，如此做法可以使相關的主題之間座標會比較近，較無相關的主題之間的座標會比較遠。如圖 2.4.5 所例[18]。

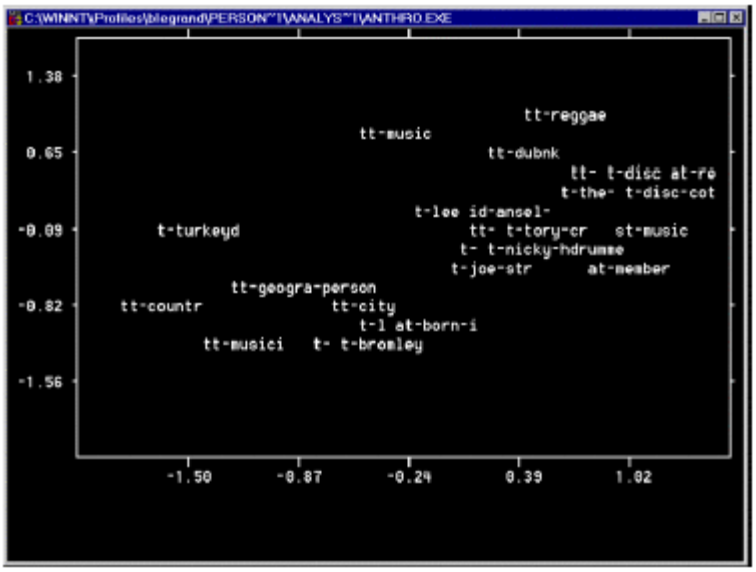


圖 2.4.5 MDS Map

另一種 ThemeScape 的顯示方式如圖 2.4.6[30]，這種表現方式就像現實中存在的地圖，有山有村莊，它表現的方式也是越相關的主題距離越近，而山越高代表這個主題相關的文件越多。

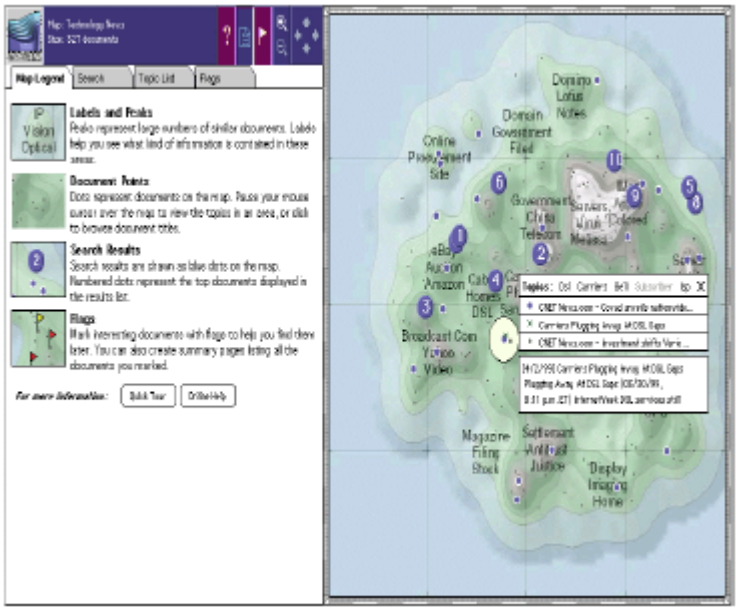


圖 2.4.6 Themescape Representation

另一種是使用虛擬城市的方式來表現的[10]。如圖 2.4.7，主

題是使用建築物來表現，使用者可以使用走的或是飛的來探索這個城市，而主視窗下有兩個小視窗可以輔助表現主題與主題的關連性或其他資訊。

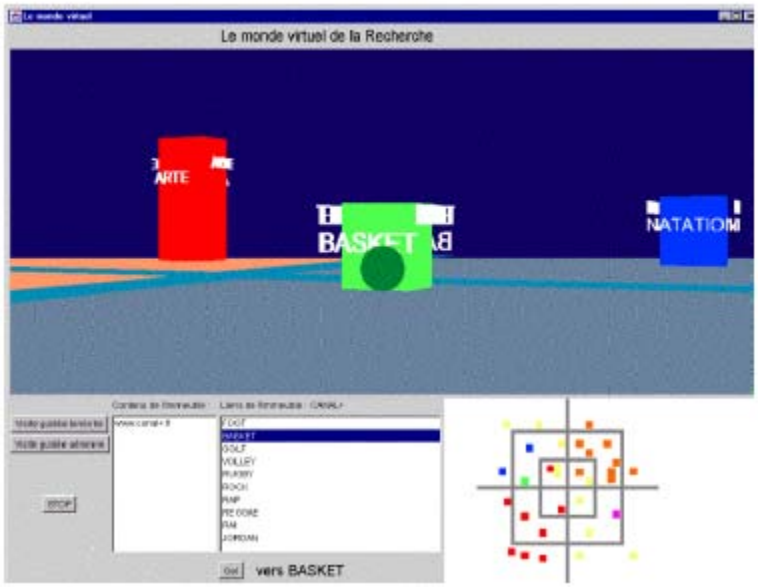


圖 2.4.7 Topic Map as a Virtual Word

第三章 系統架構概論

第一節 問題描述與規劃

管理企業知識對企業來說是掌握勝利、贏得未來的競爭優勢的一個重要關鍵，而許多重要而有價值的知識都隱藏在企業許多專案之中，這些知識其實是企業非常重要的資產，如何利用知識管理的理念將這些知識從隱性知識轉變成顯性知識供企業直接使用呢？因應現代知識的快速增長，管理這些知識對企業來說變得越來越重要，除了能夠讓企業更適應環境變化、組織改變，所以知識管理也是現代企業能夠永續存在的重要關鍵。

如何做好企業的知識管理呢？基本上一個知識管理系統要能做到良好的知識取得、保存與流通。一個好的知識管理系統也能夠讓企業創造新的知識，維護、保存企業的智慧。

一般來說，專案只是一個產品製造或是一個服務產生的中間過度過程，所以每個團隊成員的組成也是為了執行某個專案而組成的，在專案執行過程中成員溝通、整合知識時難免會遇到一些困難需要時間彼此瞭解，且因為新專案建立與專案結案時會造成團隊成員離開或是再聚合，因為這些因素增加了企業在知識累積時的困難[14]。

因此，知識管理系統成為解決這些問題的重要關鍵，許許多多執行過專案的資料就是這些知識管理系統的資料來源，再透過資料探勘的技術再找出許許多多隱藏在專案的知識，資料探勘的技術在這裡就是扮演發掘專案知識的重要角色。

對於一些軟體專案中，我們期望能夠透過資料探勘發掘出程式類別的樣式，而這些樣式實際上是一直被重覆使用的程式原始碼，我們稱為重用樣式(reuse patterns)。我們希望透過資料探勘的方式來找出這些資料，利用這些資料來建立一套知識管理系統，這套知識管理系統將對撰寫軟體的公司、部門有一定的幫助。圖 3.1.1 為本研究的研究流程圖。

我們期望本研究能帶來：

A. 利用資料探勘技術來揭露目前專案的程式架構。

B. 結合多個現有的資訊技術，實做出軟體程式分析系統來找出更符合現實的程式樣式。

C. 使用自動化的機制，讓程式樣式能夠維持最新的版本。

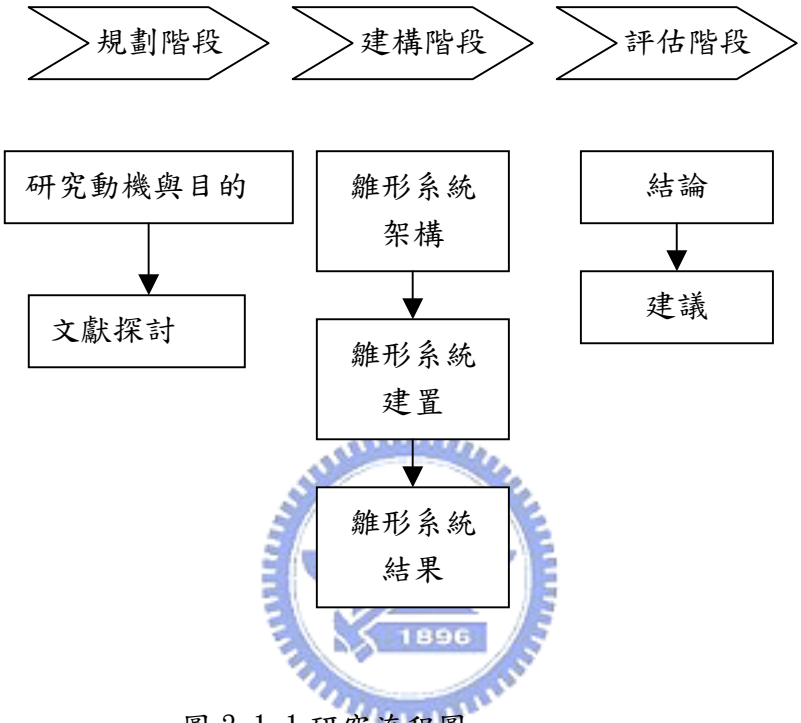


圖 3.1.1 研究流程圖

第二節 系統架構

本研究能夠將程式中物件導向的相關資料抽離出來，轉換成視覺化的點、線、面，將程式中抽象的流程經由主題地圖來呈現，具體建構出一個展示程式結構的主題地圖。因為藉由主題地圖實作呈現，可以有效組織管理大量資訊，建立最佳知識導覽介面與機制。本研究最終呈現是建構出一標準化、規格化之知識地圖技術架構，圖 3.2.6 為程式主題地圖示意圖。

本節說明如何建構知識管理主題地圖之系統架構，建構一個主題地圖有三個主要流程，第一、取得建構主題地圖的資料，本研究是將每個軟體專案中的程式碼擷取出來當成主題地圖的資料。第二、資料探勘找出隱藏在資料中的知識。第三、將處理過的資料轉成主題地圖所需要的資料格式，使用這

些資料來建構主題地圖。

圖 3.2.1 說明本研究之系統架構。我們執行一個軟體專案的過程中，會經過系統分析、系統設計、程式撰寫、程式除錯與測試的流程，這些流程所產生的產物是一些文件與許多程式的原始碼。這些程式的原始碼經由一些工具分析擷取出我們要的資訊後，我們將我們要的資訊存在資料庫中。透過資料探勘技術找出我們感興趣的資料，將這些資料重新整理後建構成主題地圖供使用者查詢、瀏覽。

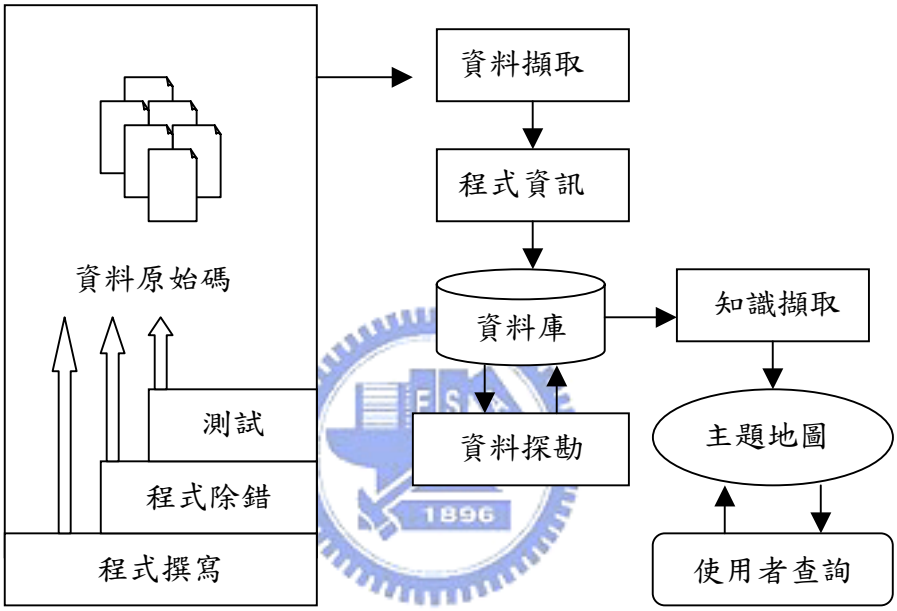


圖 3.2.1 系統架構圖

圖 3.2.2 是從資料流的角度來表示本研究之流程，原始資料是程式中的原始碼，我們將原始碼簡化成特定的文字格式，一般文字格式可以方便我們容易處理這些資料，為了未來存取方便性我們將這些資料存至資料庫中。將這些資料存至資料庫後，利用資料探勘技術將其中的知識分析擷取出來，在將結果再存回至資料庫中。為了將最後的結果展示成主題地圖，我們將資料轉換成 XML 的格式，使主題地圖的工具軟體可以讀取並顯示出主題地圖。

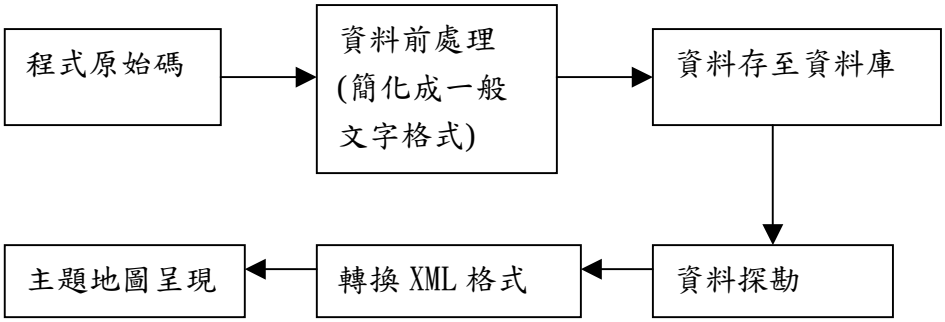


圖 3.2.2 系統資料流表示圖

主題地圖採用 XTM 做為實作標籤語言，所以也俱有 XML 的特性與優點，XML 為 W3C 所制定的標準，是一種開放式標準，所以可應用於許多相關的技術來使用之，例如 XSL、DOM、DTD…等等，未來因應不同用途需求，需要使用不同的主題地圖工具時，使用 XML 的資料格式可以方便我們不需再修改資料格式可再重複使用。

在 UML 九種圖形中塑造物件導向系統時，有一個很重要的圖形，那就是類別圖，類別在物件導向程式中是一個最基本的元素，透過類別圖可以將類別、介面、合作以及上述元素之間的關係顯示出來。物件導向中五個常用的關係為類別繼承(class inheritance)、類別實體化(class instantiation)、方法呼叫(function invocation)、方法重製(function overriding)、隱性呼叫(implicit invocation)。本研究著重在方法呼叫，透過自動化的系統來分析出物件導向程式中類別方法與其他類別方法之間的呼叫對應關係，發覺哪些關係是較常被使用到的，而這些重用樣式正是我們比較有興趣的。

在我們分析的程式原始碼中，有兩種資料格式：一種是定義檔，附檔名為.H。另一種是程式邏輯檔，副檔名為.C 或是.CPP。

原始的定義檔.H 中包含有下列幾項資料：

A. 註解：使用//或是/**/ 將文字說明包含進來，這部份的文字說明對瞭解程式結構並無太大意義，所以在資料選取過程時就會將這部份的資料去除。

B. 定義：使用#if、#endif、#define 等，這些識別文字只是提供軟體工程師能夠更容易維護程式而已。使用#define 的方法

其中一個目的是方便軟體工程師在修改程式時只要修正定義值，程式部分會自動帶入此定義值而不用找出欲修改值，一個一個修改。使用#if、#endif 的其中一個目的是，若有許多專案有使用到此程式，而每個專案在程式中都有小部分的差異時，只要在專案中定義一個定義值，#if、#endif 就能將此專案需要的程式碼包含進來，排除不需要的程式碼。定義這部份的資料對瞭解程式關聯的結構並無太大意義，所以在資料選取時就會將這部份的資料去除。

C. 包含：使用#include 來包含另一個定義檔。因為我們會讀取所有的定義檔，並解析之，所以定義檔包含哪些定義檔我們也不需要知道，所以這部份的資料在資料選取時會被去除。

D. 類別資料：由 class 這個關鍵值為開始，後一個文字為類別名稱，再兩個大括號{}包含類別的所有變數、方法等。所以我們可以由 class 後面的文字得到該類別的名稱，再由兩個大括號中所包含的資料分析出該類別有哪些變數與哪些方法。本研究要分析程式的類別方法與程式類別的方法之間的關係，所以會將類別中的變數去除，只抽離類別名稱與該類別中有哪些方法，這些資料我們會先轉成一般文字檔後，再存在資料庫中當成我們的分析資料。

原始的邏輯檔.CPP 中包含有下列幾項資料：

A. 註解：使用//或是/**/ 將文字說明包含進來，這部份的文字說明對瞭解程式結構並無太大意義，所以在資料選取過程時就會將這部份的資料去除。

B. 定義：使用#if、#endif、#define 等，這些識別文字與.H 中的定義意義相同，只是提供軟體工程師能夠更容易維護程式而已。定義這部份的資料對瞭解程式關聯的結構並無太大意義，所以在資料選取時就會將這部份的資料去除。

C. 包含：使用#include 來包含定義檔。因為我們會讀取所有的定義檔，並解析之，所以程式邏輯檔包含哪些定義檔我們也不需要知道，所以這部份的資料在資料選取時會被去除。

D. 類別資料：以類別名稱加上”::”的符號為起頭，後一個文字

為方法名稱，再由兩個大括號{}包含進來類別方法的所有邏輯程式，我們可以由類別名稱與”::”符號後面的文字得到該類別方法的名稱，再由兩個大括號中所包含的資料可分析出該類別方法有哪些變數與呼叫了哪些方法或是呼叫其他類別哪些方法。本研究要分析程式中的類別方法與類別方法呼叫哪些類別方法之間的關係，所以會將類別方法中的變數去除，只抽離類別方法呼叫哪些類別方法，這些資料我們會先轉成一般文字檔後，再存在資料庫中當成我們的分析資料。

我們將類別中的方法資料存成一般的文字檔，文字格式裏包含檔案名稱、類別名稱、該類別中的所有方法。其中檔案名稱是指我們在哪一個檔案分析出該類別資訊。一般的文字檔的文字格式如圖 3.2.3。

```
檔案名稱
  Classes
    類別名稱
  Functions
    類別中的所有方法，文字格式 類別::方法
```

圖 3.2.3 輸出程式中類別與方法資訊的文字格式

我們將類別方法呼叫哪些類別方法的資料存成一般的文字檔，文字格式裏包含呼叫類別方法與所有被呼叫類別方法。一般的文字檔的文字格式如圖 3.2.4。

```
呼叫類別方法，文字格式 類別::方法
| 被呼叫類別方法一，文字格式 類別::方法
| 被呼叫類別方法二，文字格式 類別::方法
| …
```

圖 3.2.4 程式中類別方法呼叫類別方法的資料文字格式

我們會將解析出來的一般文字檔的資料存在資料庫，第一個資料表應該紀錄類別名稱與該類別的所有方法，所以此資料表的欄位至少要有類別名稱、類別方法兩個欄位供我們記錄。第二個資料表應該記錄呼叫的類別、呼

叫的方法與被呼叫的類別，被呼叫的方法，所以此資料表應有四個欄位，呼叫的類別、呼叫的方法、被呼叫的類別，被呼叫的方法。

透過資料探勘技術我們可以分析出呼叫類別方法與被呼叫類別方法的支持度與信賴度。此結果直接存在資料庫中，故此資料表應該紀錄呼叫類別、呼叫類別的方法、被呼叫類別、被呼叫類別的方法、支持度、信賴度，所以此資料表要有呼叫類別、呼叫類別的方法、被呼叫類別、被呼叫類別的方法、支持度、信賴度六個欄位供我們記錄。

為了方便展示主題地圖，我們會將資料庫分析出的資料轉成 XML 後，再提供給主題地圖的程式運用。我們將 XML 的欄位定義成圖 3.2.5。

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<topicmap xmlns:xlink="http://www.w3.org/1999/xlink/namespace">
    <topic id="class-類別名稱">
        <topname>
            <basename>
                類別名稱
            </basename>
        </topname>
    </topic>
    <topic id="method-類別名稱::類別方法">
        <topname>
            <basename>
                類別方法
            </basename>
        </topname>
    </topic>
    <topic id="classmethod-類別名稱::類別方法">
        <topname>
            <basename>
                classmethod
            </basename>
        </topname>
    </topic>
    <assoc type="classmethod-類別名稱::類別方法">
        <assocrl anchrole="class">
                class-類別名稱
```

```
        </assocrl>
        <assocrl anchrole="method">
                method-類別名稱::類別方法
        </assocrl>
    </assoc>
</topicmap>
```

圖 3.2.5 資料庫轉成主題地圖的 XML 格式

本研究是將物件導向程式原始碼解析出來程式的類別與程式類別的方法，並找出其中的關係。所以我們將這些資料區分成類別層與方法層。類別層是從所有程式碼中分析出的所有類別，每個類別在主題地圖中是一個獨立的主題。方法層是類別中的所有方法，每個類別方法在主題地圖也是獨立的主題。類別與方法彼此之間的相互關係我們使用相關概念來連接，圖 3.2.6 表示此程式主圖地圖的概念。

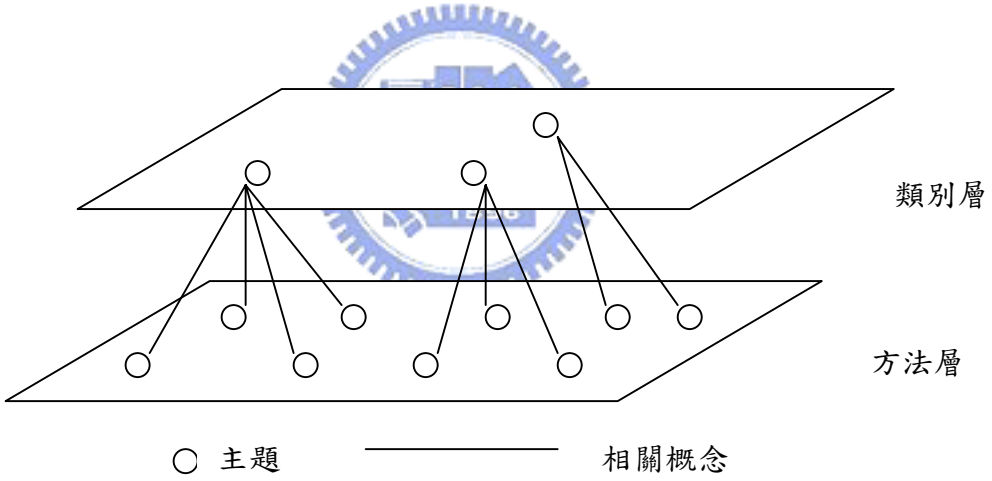


圖 3.2.6 主題地圖示意圖

本研究軟硬體環境如下：

CPU：Intel Pentium III 1066MHz
RAM：384MB
HDD：20GB
作業平台：Windows XP
資料庫：MS SQL Server 2000

開發工具：Microsoft Visual Studio .NET 2003

我們將原始碼擷取出的資料使用資料探勘技術探勘後，將探勘的結果存在資料庫中。從資料庫中經由查詢的方式把我們所要的知識取出來，使用主題地圖的方式展示，最後提供主題地圖給使用者查詢、使用這些知識。

圖 3.2.7 說明我們將原始資料(程式原始碼)從一個一個檔案中讀出，分析出其中的類別與方法，這些物件導向中的類別、類別中的方法、方法與方法的關係就是程式中的知識，再將這些知識使用主題地圖的方式展示出來。

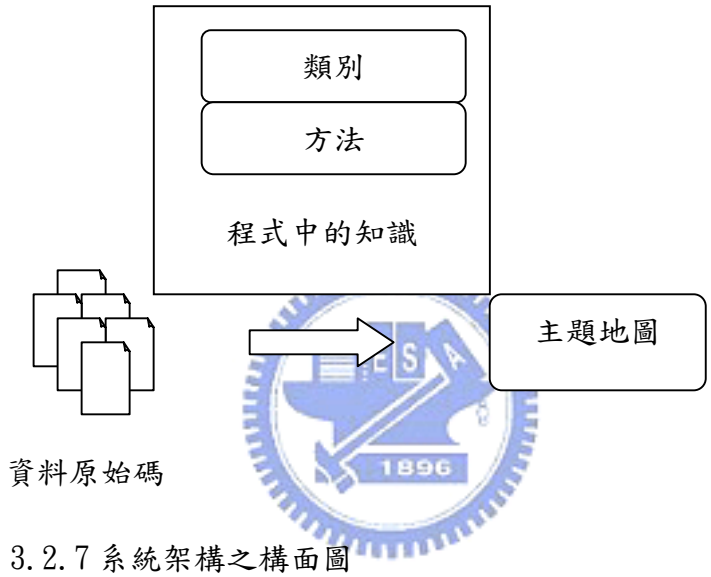


圖 3.2.7 系統架構之構面圖

本研究採用多層架構來建構系統，分別敘述如下：

A. 資料庫：資料庫可為本機資料庫或是透過網路存取的資料庫，儲存相關資訊，例如：程式類別與方法、呼叫類別方法與被呼叫類別方法之間的關聯等。

B. 應用層：透過資料探勘與軟體工程的結合，分析每個軟體專案中的知識與流程，未來將提供給使用者快速查詢、瀏覽這些資訊。

C. 顯示層：利用主題地圖的技術，建立使用介面供使用者操作查詢，並使用主題地圖視覺化技術來呈現結果。

第四章 雛形系統建置

第一節 資料簡述及資料選取

本研究的資料是將 73 個專案原始碼加以分析，雖然這 73 個專案都是 Microsoft Visual C++的程式，但是這 73 個專案的屬性、功能性質都不太相同，有些專案是應用程式、有些專案是 ActiveX 元件、COM 元件，有些則是程式庫。這 73 個專案所有的資料包括了 1465 個 CPP 檔、1644 個 H 檔，760 個類別，7748 個方法。原始資料格式如下：原始的 H 檔格式如圖 4.1.1、原始的 CPP 檔格式為圖 4.1.2。

我們依這 73 個專案的屬性不同，將這些專案的個數統計如表 4.1.1，並將這些屬性的意義說明如下：應用程式是 Microsoft Windows 程式，附檔名為 EXE，主要提供給使用者操作使用，使用者操作的結果可能是呈現在畫面上或是輸出到檔案、印表機等裝置。程式庫是 Microsoft 平台下的 DLL 檔案，提供其他應用程式呼叫、使用，程式設計師可以將常用的功能寫在程式庫中，已提供給每個應用程式共同使用，特別注意 DLL 版本問題即可。ActiveX 元件是將常用的使用者介面放在 DLL 中，ActiveX 元件的程式可以包含使用者的操作使用介面或顯示使用介面。COM 元件是 Microsoft 為了達成程式分散成多層式架構的一種程式寫法，COM 元件附檔名也是 DLL。雖然 ActiveX 元件與 COM 元件附檔名都是 DLL 但是跟一般的程式庫 DLL 程式寫法不同，且必須註冊後才可使用。

表 4.1.1　73 個專案屬性列表

功能性	**專案個數**
應用程式	51
程式庫	5
ActiveX 元件	6
COM 元件	11
合計	**73**

```
// DRMPIDoc.h : interface of the CDRMPIDoc class
//////////////////////////////////////////////////////////////////////////
#if !defined(AFX_DRMPIDOC_H__40AD8943_B1DB_43FF_907B_BB25D129315E__INCLUDED_)
#define AFX_DRMPIDOC_H__40AD8943_B1DB_43FF_907B_BB25D129315E__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
class CDRMPISrvrItem;
class CDRMPIDoc : public COleServerDoc
{
protected: // create from serialization only
	CDRMPIDoc();
	DECLARE_DYNCREATE(CDRMPIDoc)
// Attributes
public:
	CDRMPISrvrItem* GetEmbeddedItem()
		{ return (CDRMPISrvrItem*)COleServerDoc::GetEmbeddedItem(); }
// Operations
public:
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CDRMPIDoc)
	protected:
	virtual COleServerItem* OnGetEmbeddedItem();
	public:
	virtual BOOL OnNewDocument();
	virtual void Serialize(CArchive& ar);
	//}}AFX_VIRTUAL
// Implementation
public:
	virtual ~CDRMPIDoc();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
protected:
	virtual CDocObjectServer* GetDocObjectServer(LPOLEDOCUMENTSITE pDocSite);
// Generated message map functions
```

```
protected:
      //{{AFX_MSG(CDRMPIDoc)
            // NOTE - the ClassWizard will add and remove member functions here.
            //    DO NOT EDIT what you see in these blocks of generated code !
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous
line.
#endif // !defined(AFX_DRMPIDOC_H__40AD8943_B1DB_43FF_907B_BB25D129315E__INCLUDED_)
```

圖 4.1.1 原始資料 H 檔格式

```
// DRMPIDoc.cpp : implementation of the CDRMPIDoc class
#include "stdafx.h"
#include "DRMPI.h"
#include "DRMPIDoc.h"
#include "SrvrItem.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CDRMPIDoc
IMPLEMENT_DYNCREATE(CDRMPIDoc, COleServerDoc)
BEGIN_MESSAGE_MAP(CDRMPIDoc, COleServerDoc)
      //{{AFX_MSG_MAP(CDRMPIDoc)
            // NOTE - the ClassWizard will add and remove mapping macros here.
            //    DO NOT EDIT what you see in these blocks of generated code!
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////////
// CDRMPIDoc construction/destruction
CDRMPIDoc::CDRMPIDoc()
```

```
{
	// Use OLE compound files
	EnableCompoundFile(false);
	// TODO: add one-time construction code here
}
CDRMPIDoc::~CDRMPIDoc()
{
}
BOOL CDRMPIDoc::OnNewDocument()
{
	if (!COleServerDoc::OnNewDocument())
		return FALSE;
	// TODO: add reinitialization code here
	// (SDI documents will reuse this document)
	return TRUE;
}
/////////////////////////////////////////////////////////////////////////////
// CDRMPIDoc server implementation
COleServerItem* CDRMPIDoc::OnGetEmbeddedItem()
{
	// OnGetEmbeddedItem is called by the framework to get the COleServerItem
	//  that is associated with the document.  It is only called when necessary.
	CDRMPISrvrItem* pItem = new CDRMPISrvrItem(this);
	ASSERT_VALID(pItem);
	return pItem;
}
/////////////////////////////////////////////////////////////////////////////
// CDRMPIDoc Active Document server implementation
CDocObjectServer *CDRMPIDoc::GetDocObjectServer(LPOLEDOCUMENTSITE pDocSite)
{
	return new CDocObjectServer(this, pDocSite);
}
/////////////////////////////////////////////////////////////////////////////
// CDRMPIDoc serialization
void CDRMPIDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
```

```
            // TODO: add storing code here
      }
      else
      {
            // TODO: add loading code here
      }
}
/////////////////////////////////////////////////////////////////////////////
// CDRMPIDoc diagnostics
#ifdef _DEBUG
void CDRMPIDoc::AssertValid() const
{
      COleServerDoc::AssertValid();
}
void CDRMPIDoc::Dump(CDumpContext& dc) const
{
      COleServerDoc::Dump(dc);
}
#endif //_DEBUG
/////////////////////////////////////////////////////////////////////////////
// CDRMPIDoc commands
```

圖 4.1.2 原始資料 CPP 檔格式

我們利用第二章所提到的資料探勘的技術來發掘重用樣式，首先我們假設在我們執行過的軟體專案中有許多類別方法彼此之間是有關係的，而這些類別方法與類別方法之間的關係是常常在許多專案中被重用，所以我們利用關聯法則分析來挖掘程式類別方法與程式類別方法間的關聯。例如：

```
BOOL CBookMarkOper::AddBookMark(CString sKey,CString sText)
{
            CCryptClass crypt;
            CString sOut;
            crypt.EncryptString(sKey,sText,sOut);
            ……
            return TRUE;
}
```

圖 4.1.3 程式呼叫片段

圖 4.1.3 表示了 CbookMarkOper 類別中的 AddBookMark 方法呼叫了 CcryptClass 類別中的 EncrytString 方法。這樣的一個程式片段就代表一筆交易資料。透過資料探勘的技術，我們可以分析這些交易資料的支持度與可信度來了解這樣的關係是否可以當成程式庫樣式。

我們資料庫的設計基碼(schema)如圖 4.1.4，欄位 id 為自動編號，新增每筆資料時會自動加一，欄位 InvocClass 為呼叫的類別，欄位 InvocMethod 為呼叫的方法，欄位 Class 為被呼叫的類別，欄位 Method 為被呼叫的方法。我們將每個程式片段分析後，轉變成每筆每筆交易資料存放在資料庫中，再透過資料探勘的技術找出最常使用的呼叫類別方法與被呼叫類別方法，最後使用主題地圖來顯示結果供軟體工程師參考。

```
If exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[tblInvocation]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[tblInvocation]
GO
CREATE TABLE [dbo].[tblInvocation] (
      [id] [int] IDENTITY (1, 1) NOT FOR REPLICATION  NOT NULL ,
      [InvocClass] [varchar] (50) COLLATE Chinese_Taiwan_Stroke_CI_AS NOT NULL ,
      [InvocMethod] [varchar] (50) COLLATE Chinese_Taiwan_Stroke_CI_AS NOT NULL ,
      [Class] [varchar] (50) COLLATE Chinese_Taiwan_Stroke_CI_AS NOT NULL ,
      [Method] [varchar] (50) COLLATE Chinese_Taiwan_Stroke_CI_AS NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[tblInvocation] WITH NOCHECK ADD
```

```
    CONSTRAINT [PK_tblInvocation] PRIMARY KEY  CLUSTERED
    (
        [id]
    )  ON [PRIMARY]
GO
```

圖 4.1.4 建立類別方法呼叫類別方法資料庫的 script

第二節 資料前處理、轉換

本研究的原始資料是 Microsoft Visual C++的程式，為了避免有不同平台會有一些問題，所以我們的作業平台採用 Microsoft Windows，資料庫則使用 Microsoft SQL Server 2000。

原始資料的格式如圖 4.1.1 與圖 4.1.2，總共 73 個專案，1465 個 CPP 檔、1644 個 H 檔。首先我們透過解析程式將 73 個專案，1465 個 CPP 檔，1644 個 H 檔分析出有哪些類別與方法，再找出這些類別中的方法呼叫哪些類別的方法，以便知道這些程式彼此之間的關係。

我們將原始碼轉換成一般文字格式檔的過程中，將所有的定義檔去除了註解、定義、包含等資料，分析類別中的資料，去除了變數的資訊與方法中的參數，得到類別中所有方法的資訊。本研究使用 yacc 與 lex 的相關技術[32][15][6]分析結果如圖 4.2.1，網路上也有相關 yacc 與 lex 原始檔，本研究列出 James A. Roskind 的 lex 原始檔[31]放置在附錄 A 中供未來研究參考，也讓本研究的資料能夠更完整，其他相關的原始碼可從 http://custom.lab.unb.br/pub/plan/c/iecc/file/c++grammar/取得。

分析結果的文字格式中列出檔案名稱，及該檔案中所包含的類別與方法。例如圖 4.2.1 說明 DRMAP.H 中含有一個類別 CDRMAP，此類別有六個方法分別是，CDRMAP、CheckInstance、CheckVersion、getMAC、InitProfile，~CDRMAP。我們知道這些方法有其中兩個方法是比較特別的，CDRMAP 是此類別的建構方法，~CDRMAP 是此類別的解建構的方法。

```
DRMAP.h
  Classes
    CDRMAP
  Functions
    CDRMAP::CDRMAP
    CDRMAP::CheckInstance
    CDRMAP::CheckVersion
    CDRMAP::getMAC
    CDRMAP::InitProfile
    CDRMAP::~CDRMAP
```

圖 4.2.1 程式中類別與方法資訊的資料結果

第二種文字檔案格式是哪些類別中的方法呼叫哪些類別的方法，我們將邏輯檔去除了註解、定義、包含，分析類別中每個方法的資料，只保留了呼叫類別的方法與被呼叫類別的方法，去除了其他的資訊，結果如圖 4.2.2。此結果的格式中列出呼叫類別方法與此類別方法呼叫哪些類別方法，例如圖 4.2.2 說明 CDRMAP 的類別中的 InitProfile 方法呼叫了 CpersonProfile 的類別中的 IsFileExit 方法、CregUserDialog 的類別中的 CregUserDialog 方法、CpersonProfile 類別中的 GetMAC 方法…等。

```
CDRMAP::InitProfile
| CPersonProfile::IsFileExit
| CRegUserDialog::CRegUserDialog
| CPersonProfile::GetMAC
| CPersonProfile::CreateProfile
| CPersonProfile::CheckProfileDate
| CPersonProfile::OpenProfile
| CPersonProfile::DeleteProfile
| CPersonProfile::IsFillPersonData
| CFillPersonDataDialog::CFillPersonDataDialog
| CPersonProfile::GetEMail
| CPersonProfile::GetPassword
| CPersonProfile::WriteFillPsersonData
```

```
| CGetSetting::GetFillDataURL
```

圖 4.2.2 程式中類別方法呼叫類別方法的資料結果

我們將原始的定義檔與邏輯檔轉成比較好處理的文字格式後，再將這些文字檔案解析，將每個類別中所有的方法存至資料庫，將呼叫類別方法與被呼叫類別方法當成一筆一筆的交易資料轉到資料庫。

A. 我們將所有的類別中的方法寫到資料庫，資料庫的基碼格式如圖 4.2.3。

B. 我們將所有類別的方法呼叫類別方法資料寫到資料庫，資料庫的基碼格式如圖 4.1.4。

在圖 4.2.3 中的欄位 id 為自動編號，新增每筆資料時會自動加一，欄位 ClassName 為類別名稱，欄位 ClassMethod 為該類別其中的一個方法。

```
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[tblClass]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[tblClass]
GO
CREATE TABLE [dbo].[tblClass] (
      [id] [int] IDENTITY (1, 1) NOT FOR REPLICATION  NOT NULL ,
      [ClassName] [varchar] (50) COLLATE Chinese_Taiwan_Stroke_CI_AS NOT NULL ,
      [ClassMethod] [varchar] (50) COLLATE Chinese_Taiwan_Stroke_CI_AS NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[tblClass] WITH NOCHECK ADD
      CONSTRAINT [PK_tblClass] PRIMARY KEY  CLUSTERED
      (
            [id]
      )  ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[tblClass] WITH NOCHECK ADD
      CONSTRAINT [IX_tblClass] UNIQUE  NONCLUSTERED
      (
            [ClassName],
            [ClassMethod]
      )  ON [PRIMARY]
GO
```

圖 4.2.3 建立類別資料庫的 script

第五章 雛形系統結果

第一節 資料探勘

我們將程式庫樣式分析後的支持度與信賴度的資料存入資料庫，資料庫的設計表格如圖 5.1.1。在圖 5.1.1 中的 ID 為自動編號，新增每筆資料時會自動加一，欄位 InvocClass 為呼叫類別，欄位 InvocMethod 為呼叫類別的方法，欄位 Class 為被呼叫類別，欄位 Method 為被呼叫類別的方法，欄位 Support 為支持度，欄位 Confidence 為信賴度，欄位 DataCount 為資料筆數。

```
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[tblMining]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[tblMining]
GO
CREATE TABLE [dbo].[tblMining] (
      [id] [int] IDENTITY (1, 1) NOT FOR REPLICATION  NOT NULL ,
      [InvocClass] [varchar] (50) COLLATE Chinese_Taiwan_Stroke_CI_AS NOT NULL ,
      [InvocMethod] [varchar] (50) COLLATE Chinese_Taiwan_Stroke_CI_AS NOT NULL ,
      [Class] [varchar] (50) COLLATE Chinese_Taiwan_Stroke_CI_AS NOT NULL ,
      [Method] [varchar] (50) COLLATE Chinese_Taiwan_Stroke_CI_AS NOT NULL ,
      [Support] [float] NOT NULL ,
      [Confidence] [float] NOT NULL ,
      [DataCount] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[tblMining] WITH NOCHECK ADD
      CONSTRAINT [PK_tblMining] PRIMARY KEY  CLUSTERED
      (
            [id]
      )  ON [PRIMARY]
```

```
GO
```

圖 5.1.1 支持度與信賴度資料庫的 script

我們定義資料庫所存的 7657 筆資料，760 個類別，7748 個方法為所有的資料集合。本研究將這些資料都存在 Microsoft SQL 的資料庫中，Microsoft SQL 的工具有提供資料探勘的功能,所以我們可以透過 Microsoft SQL 的工具做資料探勘來達到我們的需求，並使用自己寫的程式來計算出支持度與可信賴度。

資料庫所存的 7657 筆資料，760 個類別，7748 個方法為所有的資料集合，呼叫類別方法為前因項目組，被呼叫類別方法為後果項目組，支持度是指在呼叫類別方法和被呼叫類別方法兩個項目同時在所有的資料集合出現的次數與所有的資料集合的比。信賴度為呼叫類別方法和被呼叫類別方法兩個同時在所有的資料集合出現的次數與呼叫類別方法在所有的資料集合出現次數的比。得到的結果如表 5.1.1 支持度與信賴度結果。

表 5.1.1 支持度與信賴度結果一

呼叫類別方法	被呼叫類別方法	支持度	信賴度
BCMenu:: AddBitmapToImageList	BCMenu:: LoadSysColorBitmap	6.83386865304449E-05	0.142857142857143
BCMenu:: AddFromToolBar	BCMenu:: FindMenuOption	6.83386865304449E-05	0.333333333333333
BCMenu:: AddToGlobalImageList	BCMenu:: IsLunaMenuStyle	6.83386865304449E-05	0.2
BCMenu:: AppendMenuA	BCMenu:: AppendMenuW	2.05016059591335E-04	1
BCMenu:: AppendODMenuA	BCMenu:: AppendODMenuW	1.3667737306089E-04	1
BCMenu:: AppendODMenuW	BCMenuData:: BCMenuData	1.3667737306089E-04	0.2
BCMenu:: AppendODMenuW	BCMenu:: LoadFromToolBar	6.83386865304449E-05	0.1
BCMenu::	BCMenu::	6.83386865304449E-05	1

AppendODPopupMenuA	AppendODPopupMenuW		
BCMenu:: AppendODPopupMenuW	BCMenu:: BCMenu	6.83386865304449E-05	0.5
BCMenu:: DeleteMenu	BCMenu:: GetSubMenu	6.83386865304449E-05	0.142857142857143

其次定義資料庫所存 7657 筆資料，760 個類別，7748 個方法為所有的資料集合，被呼叫類別方法為前因項目組，呼叫類別方法為後果項目組，支持度是指在被呼叫類別方法和呼叫類別方法兩個項目同時在所有的資料集合出現的次數與所有的資料集合的比。信賴度為被呼叫類別方法和呼叫類別方法兩個同時在所有的資料集合出現的次數與被呼叫類別方法在所有的資料集合出現次數的比。得到的結果如表 5.1.2 支持度與信賴度結果。我們將表 5.1.1 與 5.1.2 兩個表格相互比較整理於表 5.1.3

表 5.1.2 支持度與信賴度結果二

呼叫類別方法	被呼叫類別方法	支持度	信賴度
BCMenu:: AddBitmapToImageList	BCMenu:: LoadSysColorBitmap	6.83386865304449E-05	1
BCMenu:: AddFromToolBar	BCMenu:: FindMenuOption	6.83386865304449E-05	7.14285714285714E-02
BCMenu:: AddToGlobalImageList	BCMenu:: IsLunaMenuStyle	6.83386865304449E-05	0.166666666666667
BCMenu:: AppendMenuA	BCMenu:: AppendMenuW	2.05016059591335E-04	1
BCMenu:: AppendODMenuA	BCMenu:: AppendODMenuW	1.3667737306089E-04	0.333333333333333
BCMenu:: AppendODMenuW	BCMenuData:: BCMenuData	1.3667737306089E-04	0.25
BCMenu:: AppendODMenuW	BCMenu:: LoadFromToolBar	6.83386865304449E-05	0.333333333333333
BCMenu:: AppendODPopupMenuA	BCMenu:: AppendODPopupMenuW	6.83386865304449E-05	1
BCMenu:: AppendODPopupMenuW	BCMenu:: BCMenu	6.83386865304449E-05	0.25

BCMenu:: DeleteMenu	BCMenu:: GetSubMenu	6.83386865304449E-05	0.1

表 5.1.3 支持度與信賴度結果比較

呼叫類別方法	被呼叫類別方法	支持度	信賴度一	信賴度二
BCMenu:: AddBitmapToImageList	BCMenu:: LoadSysColorBitmap	6.83386865304449E-05	0.142857142857143	1
BCMenu:: AddFromToolBar	BCMenu:: FindMenuOption	6.83386865304449E-05	0.333333333333333	7.14285714285714E-02
BCMenu:: AddToGlobalImageList	BCMenu:: IsLunaMenuStyle	6.83386865304449E-05	0.2	0.166666666666667
BCMenu:: AppendMenuA	BCMenu:: AppendMenuW	2.05016059591335E-04	1	1
BCMenu:: AppendODMenuA	BCMenu:: AppendODMenuW	1.36677373060895E-04	1	0.333333333333333
BCMenu:: AppendODMenuW	BCMenuData:: BCMenuData	1.36677373060895E-04	0.2	0.25
BCMenu:: AppendODMenuW	BCMenu:: LoadFromToolBar	6.83386865304449E-05	0.1	0.333333333333333
BCMenu:: AppendODPopupMenuA	BCMenu:: AppendODPopupMenuW	6.83386865304449E-05	1	1
BCMenu:: AppendODPopupMenuW	BCMenu:: BCMenu	6.83386865304449E-05	0.5	0.25
BCMenu:: DeleteMenu	BCMenu:: GetSubMenu	6.83386865304449E-05	0.142857142857143	0.1

我們藉由比較信賴一與信賴二來了解這兩組結果的意義，信賴度一是呼叫類別方法為前因項目組、被呼叫類別方法為後果項目組，信賴度二是被呼叫類別方法為前因項目組、呼叫類別方法為後果項目組。

信賴度一達到可信度門檻代表呼叫類別方法相當依賴被呼叫類別方法，若被呼叫類別方法程式邏輯有問題、或是執行效能沒有很有效率，對於被呼叫類別有相當大的影響。我們將信賴度一排序來表示出被呼叫類別方法影響呼叫類別方法的影響程度。若軟體工程師的工作是負責整合、應用程式的話，可以透過此類的報表來了解要特別注意哪些程式庫，哪些程式庫對他的影響較大，哪些程式庫更新時要重新測試自己的程式，以避免程式庫不適當地改寫影響了我們的程式。

信賴度二代表了被呼叫類別方法影響哪些呼叫類別方法的程度，當程式有修改的，需要針對哪些元件或是專案重新測試。若軟體工程師負責該程式程式庫的話，可看出他所負責的程式庫影響了哪些元件或是專案，更新時需要通知誰或是測試哪部分的程式。

我們取最小支持度的值是 0.0005，取最小可信度為 0.5，將這兩個值當成門檻值，過濾資料探勘的資料，取出超過門檻值的資料，最後過濾後的結果列在表 5.1.4 與表 5.1.5。

呼叫類別	呼叫方法	被呼叫類別	被呼叫方法	支持度	信賴度
CrdsOrder	CRdsOrder	COrderUnit	COrderUnit	5.47E-04	0.533333
CwebGetPage	ThreadGetPage	CWebGetPage	CFunctionParam	6.83E-04	0.666667
CxImageJPG	CxFileJpg	CxImageJPG	CxFileJpg	5.47E-04	0.8
CmainFrame	CreateObject	CMainFrame	CMainFrame	8.88E-04	0.8125
CADORecordset	SetFieldValue	CADORecordset	PutFieldValue	0.00123	0.9
CSocketReceiver	UserLogin	CSocketReceiver	Connect	6.83E-04	0.909091
CStringOper	CharStr2HexStr	CStringOper	Char2Hex	8.20E-04	1
CSocketClient	OnReceive	CClientBZ	WriteLog	6.15E-04	1
CSocketClient	OnReceive	CClientBZ	DealData	6.15E-04	1
CStringOper	HexStr2CharStr	CStringOper	Hex2Char	8.20E-04	1
CSystemTray	CSystemTray	CSystemTray	Initialise	9.57E-04	1
CTabWindow	AddTab	CTabItem	CTabItem	6.15E-04	1
CTabWindow	CreateView	CTabWindow	AddTab	0.001845	1
CTabWindow	CTabWindow	CTabWindow	CreateFontA	6.15E-04	1
CTabWindow	EnableTab	CTabItem	Enable	6.15E-04	1
CTabWindow	GetTabLength	CTabItem	GetLength	6.15E-04	1
CTabWindow	OnDestroy	CTabItem	~CTabItem	6.15E-04	1

CTabWindow	OnPaint	CTabWindow	DrawTab	6.15E-04	1
CTabWindow	OnPaint	CTabWindow	DrawSelTab	6.15E-04	1
CTabWindow	SetTabLabel	CTabItem	SetText	6.15E-04	1
CTabWindow	ShowTab	CTabItem	Show	6.15E-04	1
CXmlMetacommThread	GetStockData	CXmlMetacommThread	GetStockBaseData	6.83E-04	1
CWebBrowser2	Create	CWebBrowser2	GetClsid	6.83E-04	1
CTabWindow	SwitchView	CTabWindow	CanSwitchView	6.15E-04	1
CTabWindow	SwitchView	CTabWindow	OnSwitchView	6.15E-04	1
CColorListCtrl	AddColorType	CColorListCtrl	CColorListType	6.83E-04	1
CColorListCtrl	CColorListCtrl	CAutoKillFocusEdit	CAutoKillFocusEdit	6.83E-04	1
CColorListCtrl	CColorListCtrl	CSortHeadCtrl	CSortHeadCtrl	7.52E-04	1
CColorListCtrl	OnItemclick	CSortHeadCtrl	SetSortImage	7.52E-04	1
CCryptClass	CryptString	CRijndael	CRijndael	7.52E-04	1
CCryptClass	CryptString	CRijndael	MakeKey	7.52E-04	1
CDynDialog	AddDlgControl	CDynDialog	SetDlgRectangle	5.47E-04	1
COOBmpToolBar	InitImageList	COOBmpToolBar	BuildImageList	6.83E-04	1
CHyWebHTMLView	Create	CHyWebHTMLView	GetClsid	5.47E-04	1
CINet	PostData	CINet	UseHttpSendReqEx	7.52E-04	1
CDynDialog	AddDlgControl	CDynDialogItem	CDynDialogItem	5.47E-04	1
CDynDialog	AddDlgControl	CDynDialogItem	InitDialogItem	5.47E-04	1
CDynDialogItem	InitDialogItem	CDynDialogItem	GetNewUniqueID	5.47E-04	1
CHyWebGetPage	Create	CHyWebGetPage	GetClsid	5.47E-04	1
CRijndael	DecryptBlock	CRijndael	DefDecryptBlock	7.52E-04	1
COOExToolBar	SizeToolBar	COOExToolBar	WrapToolBar	6.83E-04	1
COOIconToolBar	SetIcon	COOIconToolBar	ConvertIconToGrayScale	6.83E-04	1
CRijndael	EncryptBlock	CRijndael	DefEncryptBlock	7.52E-04	1
CSocketClient	CSocketClient	CClientBZ	CClientBZ	6.15E-04	1
CSocketClient	Init	CClientBZ	InitClient	6.15E-04	1
CSocketClient	OnClose	CClientBZ	OnClose	6.15E-04	1
COOExToolBar	AssignImageList	COOExToolBar	SetHotImageList	6.83E-04	1
COOExToolBar	AssignImageList	COOExToolBar	SetStandardImageList	6.83E-04	1
COOExToolBar	AssignImageList	COOExToolBar	SetDisableImageList	6.83E-04	1
COOExToolBar	CalcDynamicLayout	COOExToolBar	CalcFixedLayout	6.83E-04	1
COOExToolBar	CalcLayout	COOExToolBar	GetButton	6.83E-04	1
COOExToolBar	CalcLayout	COOExToolBar	SizeToolBar	6.83E-04	1
COOExToolBar	CalcLayout	COOExToolBar	SetButton	6.83E-04	1

COOExToolBar	OnNcPaint	COOExToolBar	DrawGripper	6.83E-04	1

表 5.1.4 大於最小支持度與信賴度結果一

呼叫類別	呼叫方法	被呼叫類別	被呼叫方法	支持度	信賴度
CColorListCtrl	CColorListCtrl	CSortHeadCtrl	CSortHeadCtrl	7.52E-04	0.52381
CStringOper	CharStr2HexStr	CStringOper	Char2Hex	8.20E-04	0.666667
CStringOper	HexStr2CharStr	CStringOper	Hex2Char	8.20E-04	0.666667
CSystemTray	CSystemTray	CSystemTray	Initialise	9.57E-04	0.666667
CWebGetPage	ThreadGetPage	CWebGetPage	CFunctionParam	6.83E-04	0.666667
CTabWindow	CreateView	CTabWindow	AddTab	0.001845	0.75
CSocketClient	OnClose	CClientBZ	OnClose	6.15E-04	0.9
CSystemTray	SetStandardIcon	CSystemTray	SetIcon	8.88E-04	0.928571
CTabWindow	AddTab	CTabItem	CTabItem	6.15E-04	1
CTabWindow	CTabWindow	CTabWindow	CreateFontA	6.15E-04	1
CTabWindow	EnableTab	CTabItem	Enable	6.15E-04	1
CTabWindow	GetTabLabel	CTabItem	GetText	6.15E-04	1
CTabWindow	GetTabLength	CTabItem	GetLength	6.15E-04	1
CTabWindow	HitTest	CTabWindow	HitTest	6.15E-04	1
CSocketReceiver	UserLogin	CSocketReceiver	Connect	6.83E-04	1
CSocketClient	CSocketClient	CClientBZ	CClientBZ	6.15E-04	1
CSocketClient	Init	CClientBZ	InitClient	6.15E-04	1
CSystemTray	SetIcon	CSystemTray	SetIcon	9.57E-04	1
CxImageJPG	CxFileJpg	CxImageJPG	CxFileJpg	5.47E-04	1
CTabWindow	OnDestroy	CTabItem	~CTabItem	6.15E-04	1
CTabWindow	OnHScroll	CTabWindow	GetTabLength	6.15E-04	1
CTabWindow	OnSize	CTabWindow	ResizeTab	6.15E-04	1
CTabWindow	OnSizeParent	CTabWindow	ResizeTab	6.15E-04	1
CTabWindow	ResizeTab	CTabWindow	GetTabLength	6.15E-04	1
CTabWindow	SetTabLabel	CTabItem	SetText	6.15E-04	1
CTabWindow	ShowTab	CTabItem	Show	6.15E-04	1
CWebBrowser2	Create	CWebBrowser2	GetClsid	6.83E-04	1
CColorListCtrl	OnKillFocus	CColorListCtrl	RepaintSelectedItems	8.88E-04	1
CColorListCtrl	OnSetFocus	CColorListCtrl	RepaintSelectedItems	8.88E-04	1
CColorListCtrl	SortNumericItems	CColorListCtrl	SortNumericItems	7.52E-04	1

CAdo	Execute	CAdo	Execute	6.83E-04	1
CADORecordset	GetFieldValue	CADORecordset	dump_com_error	0.00123	1
CADORecordset	SetFieldValue	CADORecordset	PutFieldValue	0.00123	1
CColorListCtrl	AddColorType	CColorListCtrl	CColorListType	6.83E-04	1
CColorListCtrl	SortTextItems	CColorListCtrl	SortTextItems	7.52E-04	1
CColorListCtrl	~CColorListCtrl	CColorListCtrl	RemoveAllColorType	6.83E-04	1
CCountTimeRdsOrder	QueryMoney	CCountTimeRdsOrder	SendMessageTime	6.83E-04	1
CCryptClass	DecryptString	CCryptClass	CryptString	7.52E-04	1
CCryptClass	EncryptString	CCryptClass	CryptString	7.52E-04	1
CHyWebGetPage	Create	CHyWebGetPage	GetClsid	5.47E-04	1
CHyWebHTMLView	Create	CHyWebHTMLView	GetClsid	5.47E-04	1
CINet	PostData	CINet	UseHttpSendReqEx	7.52E-04	1
CMainFrame	CreateObject	CMainFrame	CMainFrame	8.88E-04	1
COOBmpToolBar	COOBmpToolBar	COOExToolBar	COOExToolBar	6.83E-04	1
COOExToolBar	CalcFixedLayout	COOExToolBar	CalcLayout	6.83E-04	1
COOExToolBar	CalcSize	COOExToolBar	GetButtonSize	6.83E-04	1
COOExToolBar	InitImageList	COOExToolBar	AssignImageList	6.83E-04	1
COOExToolBar	LoadToolBar	COOExToolBar	ResizeToolBar	6.83E-04	1
COOExToolBar	OnNcPaint	COOExToolBar	DrawGripper	6.83E-04	1
COOExToolBar	ResizeToolBar	COOExToolBar	CalcButtonSize	6.83E-04	1
COOExToolBar	SetButtonDropDown	COOExToolBar	CalcButtonSize	6.83E-04	1
COOExToolBar	SetTextMode	COOExToolBar	ResizeToolBar	6.83E-04	1
COOExToolBar	WrapToolBar	COOExToolBar	GetButtonSize	6.83E-04	1
COOIconToolBar	COOIconToolBar	COOExToolBar	COOExToolBar	6.83E-04	1
COOIconToolBar	SetIcon	COOIconToolBar	ConvertIconToGrayScale	6.83E-04	1
CRdsOrder	CRdsOrder	COrderUnit	COrderUnit	5.47E-04	1
CRijndael	DecryptBlock	CRijndael	DefDecryptBlock	7.52E-04	1
CRijndael	EncryptBlock	CRijndael	DefEncryptBlock	7.52E-04	1

表 5.1.5 大於最小支持度與信賴度結果二

我們依據表 5.1.4 與表 5.1.5 的結果討論出其中的意義，我們將這些結果區分成四種狀況。狀況一、該筆資料在表 5.1.4 的信賴度等於一，在表 5.1.5 的信賴度也等於一。狀況二、該筆資料在表 5.1.4 的信賴度小於一，在表 5.1.5 的信賴度等於一。狀況三、該筆資料在表 5.1.4 的信賴度小於一，在表 5.1.5 的信賴度也小於一。狀況四、該筆資料再表 5.1.4 的信賴度等於

一，在表 5.1.5 的信賴度小於一。

狀況一：若資料在表 5.1.4 的信賴度等於一，在表 5.1.5 的信賴度也等於一時，表示此兩種方法的關係是非常緊密的，一個專案的程式有這樣的關係是不好的，通常在良好的多層式架構中是不會有這種情況發生，所以必須將此兩個方法加以改進，將此兩個方法依狀況分割或是合併。

狀況二：若資料在表 5.1.4 的信賴度小於一，在表 5.1.5 的信賴度等於一時，表示被呼叫的方法只被該呼叫方法呼叫，沒有被其他方法呼叫，所以可將此被呼叫的方法依狀況分割或是合併，再調整呼叫方法的程式。

狀況三：若資料在表 5.1.4 的信賴度小於一，在表 5.1.5 的信賴度也小於一時，表示該方法的程式並沒有同步，該方法的程式有修改過但是並沒有同步到其他專案的同一個類別方法，只要將所有專案的程式，把同一個類別中的同一個方法同步成相同即可。

狀況四：若資料在表 5.1.4 的信賴度等於一，在表 5.1.5 的信賴度小於一時，我們認為這樣的狀況是合理、正常，不需要再做調整。不過，我們可根據表 5.1.5 信賴度值的大小來判斷這兩個方法的關係緊密性，越大表示許多專案中都有使用到這樣的關係，此種關係可當成是一種樣式，可進一步整理出這種樣式的意義。

第二節 建構主題地圖

我們將程式碼的關係呈現在主題地圖上，所以我們將之前分析的資訊轉成 XML，再透過主題地圖的程式來展示，本研究使用免費軟體的主題地圖程式(Topic Map Designer)來展示本研究的結果[33]。

我們在 xml 中使用 topic 這個標籤來定義主題如圖 5.2.1，assoc 這個標籤來定義相關概念。在 topic 標籤下的子標籤 basename 定義為基本名稱，topic 屬性 id 則為識別 id，此 id 是屬於唯一編號，不可重複。assoc 下的子標籤 assocrl 來定義相關概念的角色，其中的屬性 anchrole 為在此相關概念下的角色名稱。我們在圖 5.2.2、圖 5.2.3、圖 5.2.4 中表示相關結果。

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<topicmap xmlns:xlink="http://www.w3.org/1999/xlink/namespace">
    <topic id="class-CxImage">
        <topname>
            <basename>
                CxImage
            </basename>
        </topname>
    </topic>
    <topic id="method-CxImage::~CxImage">
        <topname>
            <basename>
                ~CxImage
            </basename>
        </topname>
    </topic>
    <topic id="classmethod-CxImage::~CxImage">
        <topname>
            <basename>
                classmethod
            </basename>
        </topname>
    </topic>
    <assoc type="classmethod-CxImage::~CxImage">
        <assocrl anchrole="class">
                class-CxImage
        </assocrl>
        <assocrl anchrole="method">
                method-CxImage::~CxImage
        </assocrl>
    </assoc>
    <topic id="method-CxImage::AlphaClear">
        <topname>
            <basename>
                AlphaClear
            </basename>
        </topname>
    </topic>
```

```
</topicmap>
```

圖 5.2.1 資料庫轉主題地圖的 XML

在圖 5.2.2 中左邊的樹狀結構是顯示相關概念關係樹狀結構，其中 A 代表每個相關概念的節點，T 為相關概念中兩個相關主題， R 為主題地圖與主題地圖在相關概念關連扮演的角色，。

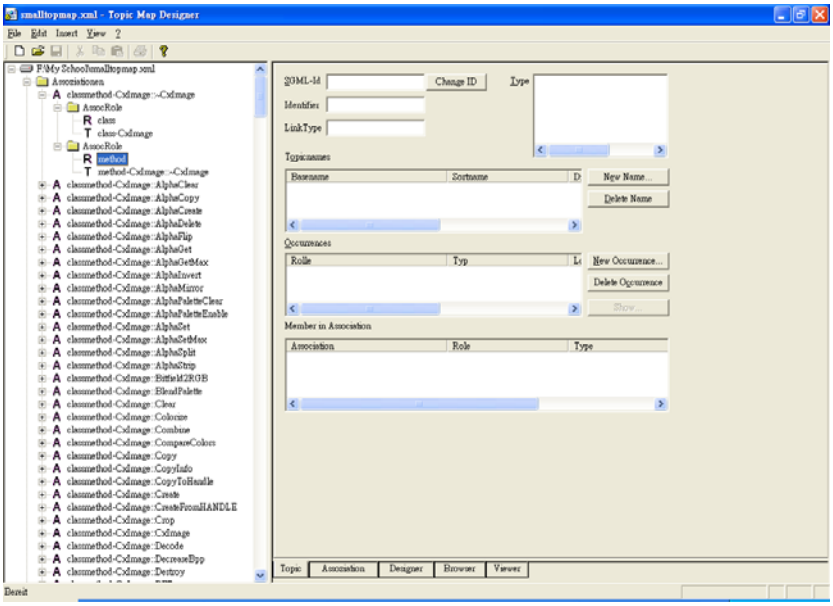


圖 5.2.2 相關概念關係樹狀表示

在圖 5.2.3 中左邊的樹狀結構是顯示主題地圖的樹狀結構，其中 T 代表每個主題地圖，B 為每個主題地圖的 basename。

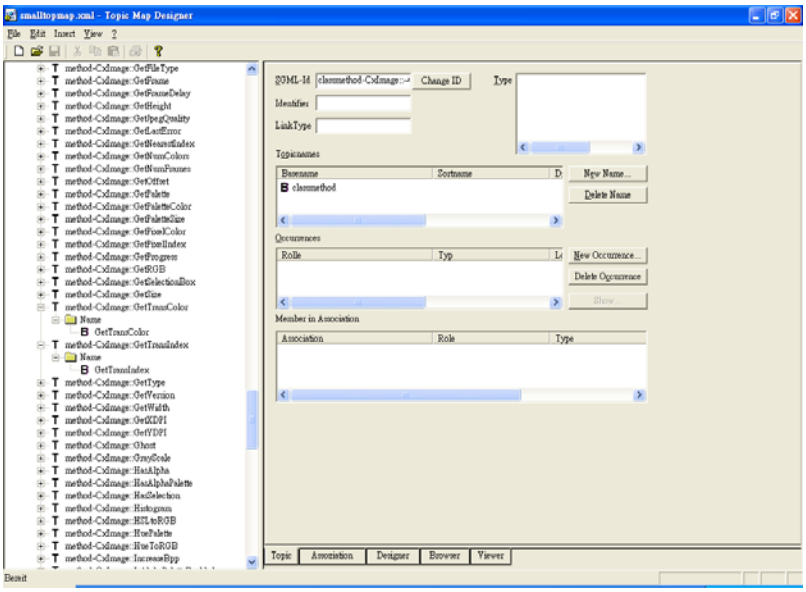


圖 5.2.3 主題地圖樹狀表示

在圖 5.2.4 中顯示整個主題地圖，藍色的方塊表示每個主題地圖，藍色線為主題地圖與開始點的連結，紫色的線表示主題與主題的相關概念。黃色的方塊為使用者正在看的主題地圖。

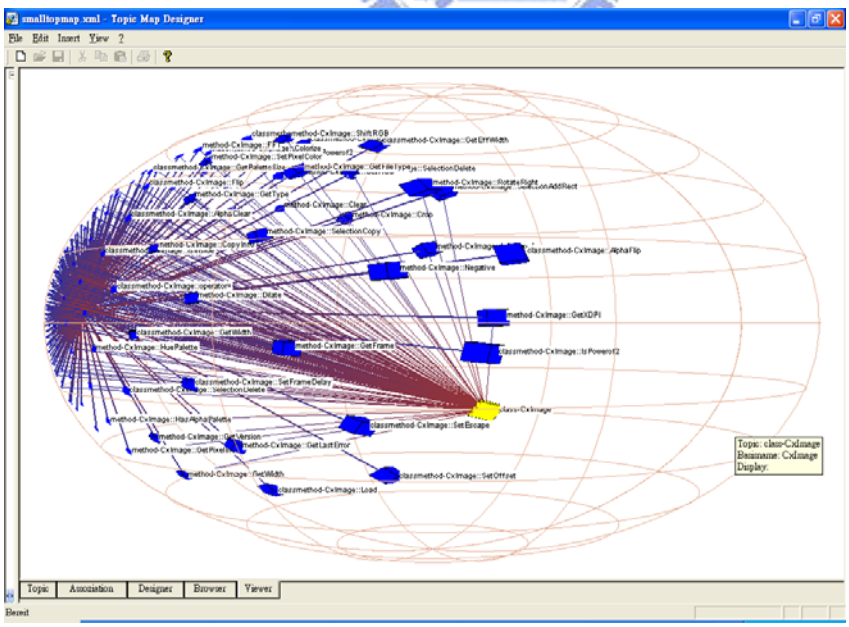


圖 5.2.4 主題地圖圖形狀表示

最後我們可以將主題地圖輸出成標準格式 XTM[34]，輸出成標準文件的優點是，不論未來我們所使用顯示主題地圖的軟體為何，我們都不需要再修改我們資料格式，主題地圖的軟體都會支援我們的檔案格式，並依每個主題地圖軟體的特色展示成主題地圖。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<topicMap xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns="http://www.topicmaps.org/xtm/1.0/">
      <topic id="class-CxImage">
            <baseName>
                  <baseNameString>
                    CxImage
                </baseNameString>
            </baseName>
      </topic>
      <topic id="method-CxImage::~CxImage">
            <baseName>
                  <baseNameString>
                    ~CxImage
                </baseNameString>
            </baseName>
      </topic>
      <topic id="classmethod-CxImage::~CxImage">
            <baseName>
                  <baseNameString>
                    classmethod
                </baseNameString>
            </baseName>
      </topic>
      <association id="IDAUAOUF">
            <instanceOf><subjectIndicatorRef
xlink:href="classmethod-CxImage::~CxImage"/></instanceOf>
            <member>
                  <roleSpec><topicRef xlink:href="#class"/></roleSpec>
                  <topicRef xlink:href="#class-CxImage"/></member>
            <member>
                  <roleSpec><topicRef xlink:href="#method"/></roleSpec>
                  <topicRef xlink:href="#method-CxImage::~CxImage"/></member>
```

```
    </association>
</topicMap>
```

圖 5.2.5 主題地圖 XTM 格式

第六章 結論與建議

第一節 研究結論

本研究提出建立程式樣式系統的架構，此架構適用於物件導向的程式語言。我們分析了程式語言的架構，以便瞭解程式與程式間的關係，再利用資料探勘技術找出程式樣式，哪些程式是常重複使用的，哪些程式與程式關係是很緊密的。

在本研究的架構下，完成了使用自動化的機制，讓程式樣式能夠維持最新的版本。可輸入多個專案程式原始碼，得到這些程式碼的關係。我們可以觀察哪些程式影響哪些程式，哪些程式的品質對於專案的品質有較大的影響，這些明確的結果可以提供專案軟體工程師與程式庫軟體工程師體認當程式發生問題時該問題的嚴重程度，知道工作的優先順序與重要性。

第二節 研究限制

在我們程式原始碼資料中，包括 73 個專案，雖然每個專案的原始碼都非常完整，但是這些專案的只侷限在某些領域應用上，涵蓋面並不夠廣，所以本研究所用到的類別涵蓋的範圍也不夠廣。且目前也只分析類別方法與被呼叫類別方法的關係上，所以建議未來研究可在其他物件導向的關係可再作加強。

本研究所使用的原始碼資料的 73 個專案都是 Microsoft Visual C++的程式，資料產生的部分都是使用 Microsoft 平台可使用的工具、程式。對於非物件導向或是非 Microsoft Visual C++的程式語言分析方式將要做適當的調整。

第三節 研究建議

A. 可以找出物件導向中更多的關係：

本研究只找出物件導向中，方法呼叫方法的關係，若要考慮物件導向的完整性，需要再分析類別繼承，方法重製等關係。

B. 建立瀏覽程式碼的主題地圖

本研究的主題地圖使用樹狀與網狀結構來展示。對於程式庫而言，樹狀結構無法清楚表示程式庫關係。網狀可以較清楚表示，但是若程式碼太多、太大彼此關係的連線就會太複雜，不易看清楚，在未來的研究可再思考主題視覺化的部份，怎樣的圖形適合程式庫的展示。

參考文獻

[1] Khalil Ahmed, 2001, Developing a Topic Map Programming Model, In Knowledge Technologies 2001, http://www.idealliance.org/papers/xml2001/papers/html/05-04-03.html

[2] Tom Archer, 陳力維譯, 2001, 完全剖析 C#, 華彩軟體股份有限公司

[3] Le Grand, B., Soto, M., 2000, Information management - Topic Maps visualization, XML Europe 2000, Paris, France, http://www.gca.org/papers/xmleurope2000/papers/s29-03.html

[4] Shneiderman, B., 1996, The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations, In Proceedings IEEE Visual Languages, Pages 336-343

[5] Grady Booch, James Rumbaugh, Ivar Jacobson, 張裕益譯, 2001, UML 使用手冊, 博碩文化股份有限公司

[6] Lovanto, M. E., Kleyn M. F, 1995, Parser visualizations for developing grammars with yacc. In Proceedings of the 26th Technical Symposium on Computer Science Education, Pages 345-349

[7] Martin Fowler, 2000, Refactoring, Addison-Wesley

[8] Geir Ove Genmo, 2002, Automagic Topic Maps, http://www.idealliance.org/papers/xmle02/dx_xmle02/index/organisation/d0e31428.html

[9] Le GRAND, B., SoTo M., 2002, Visualisation of the Semantic Web: Topic Maps Visualisation, International Conference on Information Visualisation, London, UK, IEEE Computer Society, Pages 344-351

[10] Le Grand, B., Soto, M., 2001, XML Topic Maps and Semantic Web Mining, Semantic Web Mining Workshop, ECML/PKDD 2001 conference, Freiburg, Germany

[11] Andrew Haigh, 鄭雯妮譯, 2002, 物件導向程式設計與分析, 美商麥格羅.希爾國際股份有限公司 臺灣分公司

[12] Jiawei Han, Micheline Kamber, 2001, Data Mining Concepts and Techniques, Academic Press

[13] Joseph A. Horvath, 2000, Working with Tacit Knowledge, The Knowledge Management Yearbook 2000-2001, Butterworth Heinemann,

Pages 45-49
[14]Chouyin Hsu, Duen-Ren Liu, 2002, Towards a Framework for Discovering Project-based Knowledge Maps, In Proceedings of The Second International Conference on Electronic Business, Taipei, Taiwan
[15]Alan Kaplan and Denise Shoup, 2000, CUPV - A Visualization Tool for Generated Parsers，Proceedings of the thirty-first SIGCSE technical symposium on Computer science ducation 2000, Austin, Texas, United States
[16]GRAIG LARMAN，趙光正譯, 2002, UML 與樣式徹底研究-物件導向分析與設計以及統一流程入門, 臺灣培生教育出版股份有限公司
[17]Bruls, M., Huizing, K., van Wijk, J. J., 2000, Squarified Threemaps, http://friendshipvillage2.homestead.com/ThreeMaps.html
[18]Davison, M. L., 1992, Multidimensional scaling, Krieger Publishing Company
[19]Amir Michail, 2000, Data Mining Library Reuse Patterns using generalized Association Rules, In Proceedings of the 22nd International Conference on Software Engineering
[20]Mondeca, 2001, Topic Navigator http://www.mondeca.com/site/products/products.html
[21]Fred Nickols, 2000, The Knowledge in Knowledge Management, http://home.att.net/~nickols/Knowledge_in_KM.htm
[22]Ontopia, 2001, Ontopia Topic Map Navigator, http://www.ontopia.net/solutions/navigator.html
[23]Roth, G. and Kleiner, A., 1998, Developing organizational memory through learning histories, Organizational Dynamics, Autumn, 1998, Pages 43-59
[24]Kaski, S., Honkela, T., Lagus, K., Kohonen, T., 1997, WEBSOM - self_organizing maps of document collections, In Proc. of Workshop on Self-Organizing Maps 1997 (WSOM'97), Espoo, Finland, June 1997
[25]Will Tracz, 1995, Confessions of a Used Program Saleman: Institutionalizing Software Reuse, Addison-Wesley
[26]Fu, Y., 1997, Data mining Tasks, techniques and applications, IEEE Potentials, Vol. 16, No. 4, Pages 18-20
[27]http://java.sun.com/j2se/javadoc/ Core Java Javadoc Tool

[28]http://java.sun.com/j2se/1.4/docs/api/index.html
Java™ 2 Platform, Standard Edition, v 1.4.2 API Specification
[29]http://msdn.microsoft.com
Microsoft MSDN Home Page
[30]http://www.cartia.com/products/index.html
ThemeScape Product Suite
[31]http://custom.lab.unb.br/pub/plan/c/iecc/file/c++grammar/
James A. Roskind 的 yacc 與 lex 相關資料
[32]http://www.mtl2600.org/articles/article.php3?id_article=6
Introduction Lex and Yacc
[33]http://www.topicmap-design.com/
topicmap-design.com (a site with tools and information about topic maps)
[34]http://www.topicmaps.org/xtm/1.0/
XML Topic Maps (XTM) 1.0 TopicMaps.Org Specification
[35]王冠生，2000，以 UML 為基礎的軟體程序模擬架構，元智大學電機暨資訊工程研究所
[36]張裕益，2002，UML 理論與實作 - 個案討論與經驗分享，博碩文化股份有限公司
[37]曾吾洲，2001，資訊地圖：以軟體工程標準為例，元智大學資訊工程研究所
[38]劉宜妝，2002，資料採礦之應用研究 台灣地區漁市場行情資料庫之關聯法則分析，國立中興大學行銷學系

附錄 A

```
%{

    /* Copyright (C) 1989-1991 James A. Roskind, All rights reserved.
    This lexer description was written by James A. Roskind. Copying
    of this file, as a whole, is permitted providing this notice is
    intact and applicable in all complete copies. Direct
    translations as a whole to other lexer generator input languages
    (or lexical description languages) is permitted provided that
    this notice is intact and applicable in all such copies, along
    with a disclaimer that the contents are a translation. The
    reproduction of derived files or text, such as modified versions
    of this file, or the output of scanner generators, is permitted,
    provided the resulting work includes the copyright notice
    "Portions Copyright (c) 1989, 1990 James A. Roskind". Derived
    products must also provide the notice "Portions Copyright (c)
    1989, 1990 James A. Roskind" in a manner appropriate to the
    utility, and in keeping with copyright law (e.g.: EITHER
    displayed when first invoked/executed; OR displayed continuously
    on display terminal; OR via placement in the object code in form
    readable in a printout, with or near the title of the work, or at
    the end of the file). No royalties, licenses or commissions of
    any kind are required to copy this file, its translations, or
    derivative products, when the copies are made in compliance with
    this notice. Persons or corporations that do make copies in
    compliance with this notice may charge whatever price is
    agreeable to a buyer, for such copies or derivative works. THIS
    FILE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED
    WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES
    OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

    James A. Roskind
    Independent Consultant
    516 Latania Palm Drive
    Indialantic FL, 32903
    (407)729-4348
    jar@hq.ileaf.com
    or ...!uunet!leafusa!jar

    ---end of copyright notice---


COMMENTS-

My goal is to see software developers adopt my C++ grammar as a
standard until such time as a better standard is accessible. The
only way to get it to become a standard, is to be sure that people
know that derivations are based on a specific work. The intent of
releasing this Flex input file is to facilitate experimentation with
my C++ grammar. The intent of the copyright notice is to allow
arbitrary commercial and non-commercial use of this file, as long as
reference is given to my standardization effort. Without reference
to a specific standard, many alternative grammars would develop. By
referring to the standard, the C++ grammar is given publicity, which
should lead to further use in compatible products and systems. The
benefits of such a standard to commercial products (browsers,
beautifiers, translators, compilers, ...) should be obvious to the
developers, in that other compatible products will emerge, and the
value of all conforming products will rise. Most developers are
```

aware of the value of acquiring a fairly complete grammar for a language, and the copyright notice (and the resulting affiliation with my work) should not be too high a price to pay. By copyrighting my work, I have some minor control over what this standard is, and I can (hopefully) keep it from degrading without my approval. I will consistently attempt to provide upgraded grammars that are compliant with the current art, and the ANSI C++ Committee recommendation in particular. A developer is never prevented from modifying the grammar or this file to improve it in whatever way is seen fit. There is also no restriction on the sale of copies, or derivative works, providing the request in the copyright notice are satisfied.

If you are not "copying" my work, but are rather only abstracting some of my work, an acknowledgment with references to such a standard would be appreciated. Specifically, agreements with my grammar and its resolution of otherwise ambiguous constructs, should be noted.

Simply put: "make whatever use you would like of the grammar and this file, but include the ``portions Copyright ...'' as a reference to this standard."

*/

/* Last modified 7/4/91, Version 2.0 */

/* File cpp5.l, becomes lex.yy.c after processing by FLEX */

/* This file is a dramatically cut down version of the FLEX input file used in my ANSI C Preprocessor. The executable version of my preprocessor is available on many platforms (shareware), but this is the only source extract that is currently being distributed. If you need a full ANSI C preprocessor, with extensive diagnostic capabilities and customization facilities, please contact me at the addresses given above. Current platforms include IBMPC (DOS/OS2), Sun (SPARC and Motorola), and IBM R/6000. ... end of commercial announcement.

This file is being distributed to facilitate experimentation and use of my C and C++ grammar.

Comment removal must be done during the lexing, as context (such as enclosure in string literals) must be observed. For this cut-down lexer, we will assume that comments have been removed (don't assume this if you are writing a compiler or browser!).

/* For each IDENTIFIER like string that is found, there are several distinct interpretations that can be applied:

1) The preprocessor may interpret the string as a "keyword" in a directive (eg: "pragma" or "include", "defined").

2) The parser may interpret the string as a keyword. (eg: "int").

3) Both parser and preprocessor may interpret the string as a keyword (eg: "if").

Since this file is based on source that actually lexically analyses text for both preprocessing and parsing, macro definitions were used throughout. The macro definitions supplied here have been customized to a C++ parse only, and all preprocessor keywords are passed as IDENTIFIER or TYPEDEFname. Also, since there is no symbol table to interrogate to decide whether a string is a TYPEDEFname, I simply assume that any identifier beginning with an upper case letter is a TYPEDEFname. This hack should allow you to check out how code

```
segments are parsed using my grammar. Unfortunately, if you really
want to parse major league code, you have to write a symbol table, and
maintain appropriate scoping information.



*/


/* Included code before lex code */
/************** Includes and Defines ****************************/


#include "y.tab.h" /* YACC generated definitions based on C++ grammar */

typedef char * YYSTYPE; /* interface with lexer: should be in header
                        file*/

char * yylval; /* We will always point at the text of the lexeme.
          This makes it easy to print out nice trees when YYDEBUG is
          enabled. (see C++ grammar file and its definition of
          YYDEBUG_LEXER_TEXT to be "yylval" */



#define WHITE_RETURN(x) /* do nothing */

#define NEW_LINE_RETURN() WHITE_RETURN('\n')

#define PA_KEYWORD_RETURN(x)   RETURN_VAL(x)  /* standard C PArser Keyword */
#define CPP_KEYWORD_RETURN(x)  PA_KEYWORD_RETURN(x)  /* C++ keyword */
#define PPPA_KEYWORD_RETURN(x) RETURN_VAL(x)  /* both PreProcessor and PArser keyword */
#define PP_KEYWORD_RETURN(x)   IDENTIFIER_RETURN()

#define IDENTIFIER_RETURN() RETURN_VAL(isaTYPE(yytext)?TYPEDEFname:IDENTIFIER)

#define PPOP_RETURN(x)       RETURN_VAL((int)*yytext) /* PreProcess and Parser operator */
#define NAMED_PPOP_RETURN(x) /* error: PreProcessor ONLY operator; Do nothing */
#define ASCIIOP_RETURN(x)    RETURN_VAL((int)*yytext) /* a single character operator */
#define NAMEDOP_RETURN(x)    RETURN_VAL(x)            /* a multichar operator, with a name */

#define NUMERICAL_RETURN(x) RETURN_VAL(x)            /* some sort of constant */
#define LITERAL_RETURN(x)   RETURN_VAL(x)            /* a string literal */

#define RETURN_VAL(x) yylval = yytext; return(x);


%}

identifier [a-zA-Z_][0-9a-zA-Z_]*

exponent_part [eE][-+]?[0-9]+
fractional_constant ([0-9]*"."[0-9]+)|([0-9]+".")
floating_constant (({fractional_constant}{exponent_part}?)|([0-9]+{exponent_part}))[FfLl]?

integer_suffix_opt ([uU]?[lL]?)|([lL][uU])
decimal_constant [1-9][0-9]*{integer_suffix_opt}
octal_constant "0"[0-7]*{integer_suffix_opt}
hex_constant "0"[xX][0-9a-fA-F]+{integer_suffix_opt}

simple_escape [abfnrtv'"?\\]
octal_escape  [0-7]{1,3}
hex_escape "x"[0-9a-fA-F]+

escape_sequence [\\]({simple_escape}|{octal_escape}|{hex_escape})
c_char [^'\\\n]|{escape_sequence}
s_char [^"\\\n]|{escape_sequence}
```

```
h_tab [\011]
form_feed [\014]
v_tab [\013]
c_return [\015]

horizontal_white [ ]|{h_tab}



%%


{horizontal_white}+     {
                  WHITE_RETURN(' ');
                  }

({v_tab}|{c_return}|{form_feed})+   {
                  WHITE_RETURN(' ');
                  }


({horizontal_white}|{v_tab}|{c_return}|{form_feed})*"\n"   {
                  NEW_LINE_RETURN();
                  }

auto                {PA_KEYWORD_RETURN(AUTO);}
break               {PA_KEYWORD_RETURN(BREAK);}
case                {PA_KEYWORD_RETURN(CASE);}
char                {PA_KEYWORD_RETURN(CHAR);}
const               {PA_KEYWORD_RETURN(CONST);}
continue            {PA_KEYWORD_RETURN(CONTINUE);}
default             {PA_KEYWORD_RETURN(DEFAULT);}
define             {PP_KEYWORD_RETURN(DEFINE);}
defined            {PP_KEYWORD_RETURN(OPDEFINED);}
do                  {PA_KEYWORD_RETURN(DO);}
double              {PA_KEYWORD_RETURN(DOUBLE);}
elif                {PP_KEYWORD_RETURN(ELIF);}
else               {PPPA_KEYWORD_RETURN(ELSE);}
endif              {PP_KEYWORD_RETURN(ENDIF);}
enum                {PA_KEYWORD_RETURN(ENUM);}
error              {PP_KEYWORD_RETURN(ERROR);}
extern              {PA_KEYWORD_RETURN(EXTERN);}
float               {PA_KEYWORD_RETURN(FLOAT);}
for                 {PA_KEYWORD_RETURN(FOR);}
goto                {PA_KEYWORD_RETURN(GOTO);}
if                 {PPPA_KEYWORD_RETURN(IF);}
ifdef              {PP_KEYWORD_RETURN(IFDEF);}
ifndef             {PP_KEYWORD_RETURN(IFNDEF);}
include            {PP_KEYWORD_RETURN(INCLUDE); }
int                 {PA_KEYWORD_RETURN(INT);}
line               {PP_KEYWORD_RETURN(LINE);}
long                {PA_KEYWORD_RETURN(LONG);}
pragma             {PP_KEYWORD_RETURN(PRAGMA);}
register            {PA_KEYWORD_RETURN(REGISTER);}
return              {PA_KEYWORD_RETURN(RETURN);}
short               {PA_KEYWORD_RETURN(SHORT);}
signed              {PA_KEYWORD_RETURN(SIGNED);}
sizeof              {PA_KEYWORD_RETURN(SIZEOF);}
static              {PA_KEYWORD_RETURN(STATIC);}
struct              {PA_KEYWORD_RETURN(STRUCT);}
switch              {PA_KEYWORD_RETURN(SWITCH);}
typedef             {PA_KEYWORD_RETURN(TYPEDEF);}
undef              {PP_KEYWORD_RETURN(UNDEF);}
union               {PA_KEYWORD_RETURN(UNION);}
unsigned            {PA_KEYWORD_RETURN(UNSIGNED);}
```

```
void                {PA_KEYWORD_RETURN(VOID);}
volatile            {PA_KEYWORD_RETURN(VOLATILE);}
while               {PA_KEYWORD_RETURN(WHILE);}


class               {CPP_KEYWORD_RETURN(CLASS);}
delete              {CPP_KEYWORD_RETURN(DELETE);}
friend              {CPP_KEYWORD_RETURN(FRIEND);}
inline              {CPP_KEYWORD_RETURN(INLINE);}
new                 {CPP_KEYWORD_RETURN(NEW);}
operator            {CPP_KEYWORD_RETURN(OPERATOR);}
overload            {CPP_KEYWORD_RETURN(OVERLOAD);}
protected           {CPP_KEYWORD_RETURN(PROTECTED);}
private             {CPP_KEYWORD_RETURN(PRIVATE);}
public              {CPP_KEYWORD_RETURN(PUBLIC);}
this                {CPP_KEYWORD_RETURN(THIS);}
virtual             {CPP_KEYWORD_RETURN(VIRTUAL);}

{identifier}          {IDENTIFIER_RETURN();}

{decimal_constant}  {NUMERICAL_RETURN(INTEGERconstant);}
{octal_constant}    {NUMERICAL_RETURN(OCTALconstant);}
{hex_constant}      {NUMERICAL_RETURN(HEXconstant);}
{floating_constant} {NUMERICAL_RETURN(FLOATINGconstant);}


"L"?[']{c_char}+[']     {
                  NUMERICAL_RETURN(CHARACTERconstant);
                  }


"L"?["]{s_char}*["]     {
                  LITERAL_RETURN(STRINGliteral);}



"("                   {PPOP_RETURN(LP);}
")"                   {PPOP_RETURN(RP);}
","                   {PPOP_RETURN(COMMA);}
"#"                   {NAMED_PPOP_RETURN('#') ;}
"##"                  {NAMED_PPOP_RETURN(POUNDPOUND);}

"{"                   {ASCIIOP_RETURN(LC);}
"}"                   {ASCIIOP_RETURN(RC);}
"["                   {ASCIIOP_RETURN(LB);}
"]"                   {ASCIIOP_RETURN(RB);}
"."                   {ASCIIOP_RETURN(DOT);}
"&"                   {ASCIIOP_RETURN(AND);}
"*"                   {ASCIIOP_RETURN(STAR);}
"+"                   {ASCIIOP_RETURN(PLUS);}
"-"                   {ASCIIOP_RETURN(MINUS);}
"~"                   {ASCIIOP_RETURN(NEGATE);}
"!"                   {ASCIIOP_RETURN(NOT);}
"/"                   {ASCIIOP_RETURN(DIV);}
"%"                   {ASCIIOP_RETURN(MOD);}
"<"                   {ASCIIOP_RETURN(LT);}
">"                   {ASCIIOP_RETURN(GT);}
"^"                   {ASCIIOP_RETURN(XOR);}
"|"                   {ASCIIOP_RETURN(PIPE);}
"?"                   {ASCIIOP_RETURN(QUESTION);}
":"                   {ASCIIOP_RETURN(COLON);}
";"                   {ASCIIOP_RETURN(SEMICOLON);}
"="                   {ASCIIOP_RETURN(ASSIGN);}

".*"                  {NAMEDOP_RETURN(DOTstar);}
"::"                  {NAMEDOP_RETURN(CLCL);}
```

```
"->"                {NAMEDOP_RETURN(ARROW);}
"->*"               {NAMEDOP_RETURN(ARROWstar);}
"++"                {NAMEDOP_RETURN(ICR);}
"--"                {NAMEDOP_RETURN(DECR);}
"<<"                {NAMEDOP_RETURN(LS);}
">>"                {NAMEDOP_RETURN(RS);}
"<="                {NAMEDOP_RETURN(LE);}
">="                {NAMEDOP_RETURN(GE);}
"=="                {NAMEDOP_RETURN(EQ);}
"!="                {NAMEDOP_RETURN(NE);}
"&&"                {NAMEDOP_RETURN(ANDAND);}
"||"                {NAMEDOP_RETURN(OROR);}
"*="                {NAMEDOP_RETURN(MULTassign);}
"/="                {NAMEDOP_RETURN(DIVassign);}
"%="                {NAMEDOP_RETURN(MODassign);}
"+="                {NAMEDOP_RETURN(PLUSassign);}
"-="                {NAMEDOP_RETURN(MINUSassign);}
"<<="               {NAMEDOP_RETURN(LSassign);}
">>="               {NAMEDOP_RETURN(RSassign);}
"&="                {NAMEDOP_RETURN(ANDassign);}
"^="                {NAMEDOP_RETURN(ERassign);}
"|="                {NAMEDOP_RETURN(ORassign);}
"..."               {NAMEDOP_RETURN(ELLIPSIS);}
"/*"            { comment(); }


%%

/* I won't bother to provide any error recovery. I won't even handle
unknown characters */

/*****************************************************************/
int isaTYPE(string)
char * string;
{
    /* We should really be maintaining a symbol table, and be
    carefully keeping track of what the current scope is (or in the
    case of "rescoped" stuff, what scope to look in). Since the
    grammar is not annotated with actions to track transitions to
    various scopes, and there is no symbol table, we will supply a
    hack to allow folks to test the grammar out. THIS IS NOT A
    COMPLETE IMPLEMENTATION!!!! */

    return ('A' <= string[0] && 'Z' >= string[0]);
}

comment()
{
      char c, c1;

loop:
      while ((c = input()) != '*' && c != 0)
            putchar(c);

      if ((c1 = input()) != '/' && c != 0)
      {
            unput(c1);
            goto loop;
      }

      if (c != 0)
            putchar(c1);
}
```