# 物件導向資料庫系統之版本模型設計*
# Design A Version Model for Object-oriented Database Systems

馮懷碧, 賈坤芳
Hwai-Bih Feng, Kuen-Fang Jea

陳世恭, 呂瑞旭, 高國峰, 張國森, 陳麗芬, 林淑美, 戴建誠
Shih-Kung Chen, Jui-Hsu Lu, Kuo-Fong Kao, Kuo-Sen Chang, Li-Fen Chen, Shwu-Meei Lin, Jian-Cheng Dai

中興大學資訊科學研究所
Institute of Computer Science
National CHUNG-HSING University
jea@dblab.cs.nchu.edu.tw

資訊工業策進會 軟體工程實驗室
Software Engineering Laboratory
Institute For Information Industry
xkchen@iiidns.iii.org.tw

## 摘要

*本論文介紹一個新的版本模型(version model)，此版本模型對大型物件(large object)提出一個新的表示方法來避免造成浪費儲存空間，同時也可有效的處理一般物件(regular object)及大型物件(large object)。*

關鍵詞:物件導向資料庫，版本模型，大型物件。

## Abstract

*This paper introduces a new version model, with a new representation scheme to avoid the waste of storage space for large objects. It can also process regular objects and large objects efficiently.*

*Keywords: object-oriented database, version model, large object.*

## 1.簡介

在一個沒有版本概念的資料庫系統中，當執行某一交易(transaction) 而修改資料時，原先的資料值將因被新的資料值所取代而消失，未來若欲參考或使用原先的資料值將無從得知。對某些應用如 CAD/CAM 或軟體開發等而言，將某些舊的資料版本妥善管理以供未來有效應用是非常必要的。例如對 CAD/CAM[4，6] 應用上已開發引擎產品的不同版本，加以適當儲存與管理，可以縮短未來開發新引擎所需的時間及加速產品的產率。

此外資料庫的版本控制 (version control)[3]尚有下列之目的及重要性：(1)在同步控制方面 (concurrency control)：如果同一資料有許多不同版本，我們可以利用多版同步控制方法(multiversion concurrency control scheme)，來排定數個交易(transaction)的執行次序，以增加整個資料庫系統的效能，減少資料使用時彼此牽制 (locking) 的機會。(2)在系統回復方面 (recovery)：如果系統 crash，我們可以利用已儲存於系統之舊資料版本，將系統迅速回復到完整一致的狀態 (consistent state)，讓系統重新開始。(3)增強分散式系統之效能：我們可利用資料有不同版本的特性，將版本複製到不同的 site，以加速資料存取及交易執行的時間。(4)可供製作 "update-free" 資料庫：我們可利用版本的觀念，來製作"update-free"資料庫，當有資料要更新 (update) 時，我們對此資料就產生一新的版本，而不把原先的版本摧毀。

本論文所提出的版本模型是以 generic object 的方式來製作。Generic object 是一種資料結構，用來描述同一個物件的所有版本之間的關係及其特性，而且每一個物件都有一個 generic object。每一個版本可以有四種狀態，這些狀態分別爲，暫時型版本(transient version)、正運作型版本(working version)、永久型版本(released version)以及刪除型版本(deleted version)，版本之間的關係是用指標來指示。

本文結構如下:第二節回顧與本文相關之研究，第三節介紹我們的版本模型及此版本模型所提供的運算，第四節將介紹 ORION 的版本模型，然後再跟我們的版本模型做比較，第五節探討製作此版本模型之細節，第六節總結本論文並敘述未來將繼續進行之工作。

## 2.相關研究

由於版本控制可增強資料庫系統的功能或效能，因此過去有部份研究者探討資料庫系統之版本

*本研究由資策會贊助,本論文並不代表資策會的觀點

控制問題。

## 2.1 關連式資料庫上的版本模型

在傳統的關連式資料庫系統上大都利用時間性資料庫(temporal database)來增添版本的觀念，例如陳世恭[12]和鄭旭峰[13]所使用的方法是加入有效時間(valid time)和交易時間(transaction time)兩個屬性，以區分不同的資料版本。

而在實際關連式資料庫系統產品中 Postgres 系統提供建立版本模型之功能。Postgres 所提供的版本模型是將每一個 relation 視爲一個版本，而每個版本修改時，修改部份被記錄在三個 relations 中，分別爲 base、v_add、和 v_del。base relation 是用來記錄原始版本的資料內容，v_add relation 是用來記錄此版本裏所加入的資料列(tuple) ，v_del relation 是用來記錄此版本裏所刪除掉的資料列(tuple) 。此種作法只是將所有修改、刪除或新增的資料列(tuple)記錄下來、因此當要找出某一個版本時還必須花一些時間將這三個 relations 做整合。

## 2.2 物件導向資料庫上的版本模型

目前物件導向資料庫已日益受重視，因此亦有論文探討如何在物件導向資料庫添加版本的觀念及版本控制機制 (mechanism)。如 Kamita *et al.*[5]所提出的是每個物件都有私有和共享的版本，利用 check in/check out 的方式來達成版本控制，該模型可應用於分散式系統的環境下。

而 David Beech 及 Brom Mahbod[1]所提出的版本模型是以 generic instance 方式來做，一個版本集合只能有一個 generic instance，generic instance 內包含有版本的相關資料及版本之間的關係，而版本間的關係是用指標來指示。每一個版本可以有兩種狀態，這些狀態分別爲，正在運作型版本(working version)及永久型版本(frozen version)。以上這些版本模型對版本的產生都是將原本的物件拷貝一份，對大型物件並無有效的處理，這樣的做法會使大型物件造成浪費很大的空間。Simon Monk 及 Ian Sommerville[10]提出一種版本模型，用類別版本(class version)的方式來建立版本，也就是每一個版本就是一個類別(class)。而版本的產生是因爲對類別的屬性(attribute)做新增或刪除。

在實際物件導向資料庫系統產品，約有數個較有名的系統提供版本( version )的觀念，如 Objectstore，ORION[2][7][8] ， IRIS[1] ，O2[11]及 Ontos 等。

ORION 與 IRIS 所提供版本的方法與語法很相近，利用 generic object 去整合相關的 version object，提供語法去產生版本、刪除版本、搜尋版本。O2 對複合物件(complex object)可以產生版本，而複合物件裏的每個物件也會有各自的版本。當產生一個新的版本時，新的版本只有儲存此版本和其先前版本的不同部份，這樣的作法可以節省一些空間，而且新版本可以由多個不同的版本組合而成。

## 3.我們所提出的版本模型(Proposed Model)

本論文所提出的版本模型是將 ORION 的版本模型修改而成，ORION 版本模型的新版本產生是拷貝一份原有的版本，這樣的做法對大型物件的資料庫系統會造成浪費很大的儲存空間。所以我們的版本模型是針對 ORION 的缺點來做修改，使加上版本模型的資料庫系統效能不會太差。我們所設計的版本模型是以下列這些條件爲目標:①版本不能浪費太大的儲存空間②系統對版本的搜尋速度要快③某一物件所衍生的版本數不能太多。以下將一一的介紹我們所設計的版本模型的概念。

## 3.1 物件的類型(Types of Object)

版本可以由簡單物件(simple object)和複合物件(complex object)兩種物件來產生。對簡單物件做修改時可以產生一個新的版本物件或者是直接對此物件做修改，一個物件是否爲可版本性物件(versionable object)由系統來決定，對複合物件或者是其內的任何一個單元物件做修改時對複合物件都會產生新的版本物件，而其內的單元物件只要做修改也會對這單元物件產生另一新的版本物件。

## 3.2 版本的分類(Taxonomy of Version)

目前設計版本時，一般都分爲兩類:

- 類別版本(class version) : 對類別的性質和運作方式可以做修改或者是刪除，而且也可以對類別的性質做新增。
- 物件版本(Instance or object version) : 可以對物件的性質做修改。

我們的模型中是採用物件版本的方式。

## 3.3 版本的狀態(States of Versions)

每一個版本可以有四種狀態，分別爲:

(1) 暫時型版本(transient version) : 每一個新產生的版本其狀態爲暫時型版本。對此狀態的版本只能做修改和刪除，同時只能由產生此版本的使用者來做。

(2) 正運作型版本(working version) : 當某一個暫時型版本產生新版本時，此版本就會升級爲正運作型版本。對此狀態的版本只能做刪除，不能做修改，同時只能由產生此版本的使用者來做。

(3) 永久型版本(released version) : 此狀態的版本爲最終穩定的狀態，不能夠對此狀態的版本做修改及刪除，但可以將此狀態的版本重新產生一個新的 generic object。

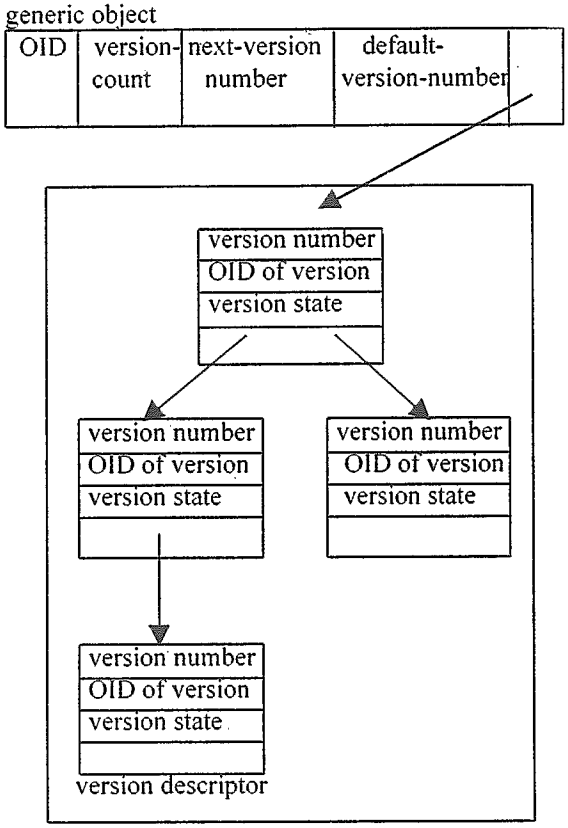
(4) 刪除型版本(deleted version) : 當一個正運作型的

版本被刪除時，此版本會在緩衝器裏被標誌爲已刪除的版本。但還不能夠將此版本刪除，因爲此版本還有其它版本正在繼承它。

以上這些狀態可以由系統自動產生或者是使用者要求系統產生。

### 3.4 版本的演化(Version Evolution)

我們是採用 generic object 的方式來製作版本模型，generic object 是一種資料結構，用來描述同一個可版本性物件的所有版本之間的關係及其特性，generic object 的架構如圖一所示。




Version Derivation Tree

圖一:Generic Object 的架構

每個可版本性的物件都會有一個 generic object，generic object 是由系統自動產生及維護的，每一個 generic object 都會有一個物件辨識碼(OID)。一個 generic object 必須包含以下這些資料:

(1) generic object 的物件辨識碼(OID of generic object)：用來區分不同物件的 generic object。

(2) 版本記數(version count)：用來記錄目前的版本物件總共有多少的版本。

(3) 下一版本號碼(next-version number)：這版本號碼是用來指定給下一個新產生的版本，產生一個新版本後此版本號碼會自動增加。

(4) (default-version-number)：主要是用來表示目前最新的版本，此版本號碼是由系統產生。

(5) (derivation-tree)：derivation-tree 是由一些版本描述符號(version descriptor)組成的樹，每一個版本描述符號代表一個版本物件，包含下列資料:

(a) 版本號碼(version number)

(b) 版本的 OID(OID of version)

(c) 版本的狀態(version state)

(d) 指向衍生版本的指標(pointer to children version)

當產生一個物件後，新的版本物件就可以從此物件衍生而來。每當產生一個新的版本時，系統就會指定一個版本號碼給這版本。不同的版本有不同的號碼，版本號碼不會有重覆，愈後面產生的版本其版本號碼就愈大。對某一個物件可以產生的版本個數由系統來定，例如爲 N 個，在這條件下 generic object 的版本計數值最大只能夠爲 N，而 generic object 裏的下一個版本號碼有可能比版本計數值大，因爲有些版本可能已被刪除。系統要產生下一版本號碼時先檢查版本計數值，如果等於 N 就不允許再產生新版本。

在產生一般物件及大型物件的版本時用不同的方式來表示。一般物件產生新版本時，將原本的物件拷貝一份，然後再做修改，對大型物件則只儲存差異(version difference)的部份。這種作法對大型物件的版本可以省去儲存重複資料的空間。

### 3.5 對版本所提供的指令(Operation Support for Version)

使用者可以對系統提出以下有關版本的要求:

(1) 產生新版本(create version)

使用者可以要求系統對某一物件產生新的版本。當使用者下達一個 query，要求對某一版本物件產生一個新的版本時，系統會先檢查此版本物件的版本號碼，看此版本物件爲一般物件或大型物件，同時也檢查其版本狀態，當條件都滿足時就產生一個新版本，而其狀態爲暫時型版本。

(2) 刪除版本(remove version)

使用者可以要求系統刪除某一個他所產生過的版本，但是此版本的狀態必須爲暫時型版本或正運作型版本。當使用者所要刪除的版本狀態爲暫時型版本時，系統會將此版本直接刪除，如果是正運作型版本時系統會先檢查此版本爲一般物件或大型物件，如果爲一般物件可以將此版本直接刪除，若是大型物件時就將此版本的狀態改爲刪除型版本，版本的資料卻仍保留住，因爲此版本仍有其它版本正在繼承它。

(3) 存取版本(retrieve version)

使用者可以利用 OID 或版本號碼要求系統對某一版本做存取。系統會根據使用者所提供的條件來將版本找出。

(4) 更改版本的狀態(change version state)

使用者可以要求系統對某一個他所產生過的版本狀態做修改。系統會根據使用者所提供的條件來檢查是否符合系統規定的條件，然後再來做修

改。

## 4. 製作版本的考量(Implementation Issue)

在我們的版本模型中，我們將物件辨識碼(OID)中的一個位元(bit)用來區分該物件是可版本性物件或不可版本性物件。每個可版本性物件包含以下這些資料，而不可版本性物件則只包含(1)項資料:

(1) 物件辨識碼(OID)
(2) 版本號碼(version number)
(3) 版本狀態(state of version)
(4) 此物件的 generic object 的 OID(OID of its generic object)

使用者可以用物件辨識碼或版本號碼來找出所需要的物件。每個版本號碼中包含的資料如下:

version number

| R/L object | number |
|---|---|

R/L object : 此欄位用來區分版本物件爲一般物件或大型物件。

number : 此欄位爲版本號碼。

在我們的模型中，對大型物件只儲存差異的部份，而其儲存格式如下:

version difference

| OID | field address | field difference | field address | field difference |
|---|---|---|---|---|

Version difference 裏的位址欄位爲所修改的欄位的位址，其後的資料爲此版本所修改成的資料。當使用者要存取某一個大型版本物件時系統會對相關的 version difference 做合併。

產生一個新的不可版本性物件和可版本性物件的物件格式不一樣，這部份在前面已提過。當使用者下達一個 query 要找出某一版本物件時，系統會依據使用者所提供的條件來將版本物件找出，所提供的條件例如有:

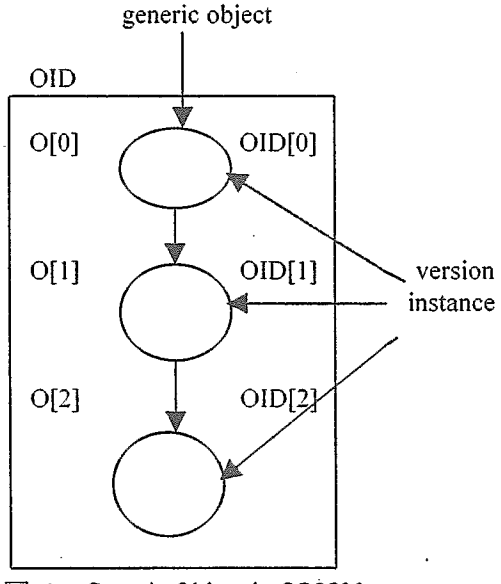
(1) generic object 的 OID 和版本號碼: 系統先由 generic object 的 OID 找出 derivation tree，將使用者所提供的版本號碼和 derivation tree 中的版本號碼做比較，如果不相等則從物件中的版本號碼來判斷此物件爲一般物件或大型物件，如果爲一般物件就會繼續往下找，如果爲大型物件，則會依據此版本物件裏的 OID 將資料找出，再放到記憶體裏去，然後再繼續往下找，找到的資料再跟先前放在記憶體裏的資料做合併，直到找到要找的物件辨識碼，再將此 version difference 跟記憶體裏的版本物件做合併。系統將所有相關的版本物件做合併後，再將物件的資料傳給使用者。
(2) 物件的辨識碼: 系統會依據物件辨識碼將物件找出，首先檢查版本號碼，看此版本爲一般物件或大形物件，如果爲一般物件將物件傳回給使用者，如果爲大形物件則依據物件中記錄的 generic object 的 OID 和版本號碼將物件找出，找物件的過程如(1)。

一個沒有提供版本模型的物件導向資料庫系統，如果要將此模型製作到系統時，必須建立版本物件的格式、generic object 的格式及 version difference 的格式。這些格式就如前面所提及的。而物件辨識碼也必須要做一些修改，例如將物件辨識碼中的一個位元用來區分可版本性或不可版本性物件。其餘可能修改到的部份爲 schema manager、object manager、storage manager 及 query manager。因爲在原本的物件導向資料庫系統中沒有提供版本的模型，所以在下達 query 時必定沒有將對版本下達 query 的觀念考慮進去，因此對 query language 的格式做修改，也是一件必須要做的事，因爲必須要提供使用者在下達 query 時也可以對版本下達 query。以上這些考量是在將版本模型製作到系統時必須要考慮到的地方。

## 5. ORION 的版本模型與我們的版本模型做比較

首先介紹 ORION 的版本模型。ORION 是用 generic object 的方式來描述版本之間的關係，版本的狀態分爲三種，分別爲暫時型版本，正運作型版本及永久型版本。ORION 的 generic object 如圖二所示，generic object 包含以下這些資料:

(1) 版本記數(version count) : 用來記錄目前物件的版本數目。
(2) 下一版本號碼(next-version number) : 這版本號碼是用來指定給下一個新產生的版本，產生一個新版本後此版本號碼會自動增加。




圖二 : Generic Object in ORION.

(3) (default-version-number) : 主要是用來表示目前最新的版本，此版本號碼可以由系統產生或者是使用者自己設定。
(4) (default-switch) : 用來表示 default-version-number 是由系統產生或者是使用者自己設定。
(5) (derivation-tree) : derivation-tree 是由一些版本描

述符號(version descriptor)組成的樹，每一個版本描述符號代表一個版本物件，包含下列資料:

(a) 版本號碼(version number)

(b) 版本的 OID(OID of version)

(c)指向衍生版本的指標(pointer to children version)

每個版本物件會由系統自動加入三個欄位，分別爲版本號碼(version number) ，版本狀態(state of version)及 generic object 的 OID。

在 ORION 的系統中，當要產生一個版本物件時都是將原本的物件拷貝一份後再去做修改。這樣的作法會造成浪費儲存空間，而我們的作法則只儲存差異的部份，所以大型物件的儲存空間就比 ORION 的作法節省。

## 6. 結論

本論文介紹了一些其他論文中的版本模型，同時也介紹了我們的版本模型。從我們的模型中可以很簡單而且明確的知道版本的所有狀態及功能，同時可以有效的處理簡單物件，複合物件，一般物件及大型物件。在我們的版本模型中，對大型物件的版本只儲存差異的部份，所以對大型物件的版本可以節省儲存空間，但是系統在找尋某一大型版本物件的資料時卻需要花一些時間來對版本做合併，我們的作法是用時間來換取空間。

由於物件導向資料庫已日益的受到重視，使用者對它的要求也越來越高，提供版本功能在物件導向資料庫系統中也變成一個不可缺少的功能。未來我們將實作本論文中所提出的版本模型，並對其效能做評估。

## 參考文獻


[1] D.Beech and B. Mahbod, "Generalized Version Control in an Object-Oriented Database," *Proc. 4th IEEE International Con. on Data Engineering,* 1988, pp.14-22.

[2] E. Bertino and L. Martino, Object-Oriented Database Systems, Addison-Wesley, 1993.

[3] K. Dittrich and R. Lorie, " Version Support for Engineering Database Systems," *IEEE Transactions on Software Engineering,* Vol. 14, No. 4, April 1988, pp. 429-437.

[4] R. Katz, E. Chang and R. Bhatejz, "Version Modelling Concepts for Computer-Aided Design Databases," *Proc. SIGMOD'86,* Washington D.C. USA, 1986.

[5] T. Kamita, S.Ichimura, K. Okada and Y, Matsushita, "A Database Architecture and Version Control for Group Work", *IEEE Proc.27th Annual Hawaii International Conf. on System Sciences,* 1994, pp.439-447.

[6] M. Ketabachi and V. Berzins, "Modeling and Managing CAD Databases," *IEEE Computer Magazine,* Vol. 20, No. 2, 1987, pp. 93-102.

[7] W. Kim, Object-Oriented Databases, Concepts, Databases, and Applications, Addison-Wesley, 1989

[8] W. Kim, Introduction to Object-Oriented Databases, MIT Press, 1990.

[9] L. Mckenzie, Jr. and R. Snodgrass, "Evaluation of Relational Algebras Incorporating the Time Dimension in Database," *ACM Computing Surveys,* Vol. 23, No. 4, December 1991, pp. 501-543.

[10]Simon Monk and Ian Sommerville, "Schema Evolution in OODBs Using Class Versioning," *SIGMOD RECORD,* Vol.22, No.3, September 1993, pp.16-22.

[11]Technical Overview of the O2 System,O2 Technology, July 1997.

[12]陳世恭, The Design of a Temporal Object-Oriented Database System, *國立中興大學應用數學研究所碩士論文, 1993.*

[13]鄭旭峰, Version Management in Temporal Object-Oriented Database, *國立中興大學應用數學研究所碩士論文, 1994.*