國立暨南國際大學資訊管理研究所

碩士論文初稿

以Ontology為基礎之內容管理系統

Ontology-based Content Management System

指導教授：俞旭昇　博士

研究生：許洛豪

中華民國九十七年月日

論文名稱：以 Ontology 爲基礎之內容管理系統
校 院 系：國立暨南國際大學資訊管理學系　　頁　　數：
畢業時間：年／月　　學 位 別：
研 究 生：許洛豪　　指導教授：俞旭昇　博士

論文摘要

內容管理系統經常被應用在各種不同領域中，例如文件管理與知識分享等。然而現行內容管理系統，多僅針對如何提供使用者簡便的環境進行文件內容的編修及發佈設計，對文件後設資料的處理則甚少提及，造成現行內容管理系統難以支援計算與推論機制的缺陷。

在前人的研究中指出，內容管理系統無法進行推論與計算的主因，在於缺乏足夠的後設資料，其中尤以關聯與屬性兩者最爲重要。然而在研究過程中，我們發現在前人的研究中，由於在關聯此後設資料中缺乏概念的層次，造成系統無法判斷明顯的語意錯誤，導至後續的推論不正確。

本研究究則提出了將 Ontology 架構應用於內容管理系統的構想。由於在 Ontology 的架構下可同時表達出概念與事實，因此得以解決上述推論錯誤的問題，亦使內容管理系統不僅能處理人類可讀 (Human-readable) 資料，其餘後設資料亦可同時以人類可讀及機器可讀 (Machine-readable) 兩種型式呈現。

在實作上，我們以 Drupal 此套開放源碼的內容管理系統爲核心，實作了一個 Ontology 延伸模組，以驗證此架構的可行性。

關鍵詞：Ontology、內容管理系統、知識分享

目錄

圖目錄

表目錄

第一章　緒論

1.1　研究背景

1.1.1　內容管理系統簡介

內容管理系統並非指特定的軟體或是技術，相反的，它是針對大型網站如何以「下一代」的方式處理資料的總稱。而此處的「下一代」係指透過網頁應用程式動態地建立網站的內容，改變以往手動建立 HTML 文件的方式[1]。

在設計上，內容管理系統通常會由內容管理程式 (Content Management Application，簡稱 CMA) 與內容遞送程式 (Content Delivery Application，簡稱 CDS) 所組成。CMA 負責提供使用者在不需要懂 HTML 的情況下建立、修改、刪除內容，以及進行使用者權限控管等功能，而 CDS 則負責將這些內容發佈成網頁格式，做為網站的內容[2]。

在實際應用上，部落格、Wiki 以及網路相簿由於具有上述內容管理系統的特性，因此經常被視為是內容管理系統的一種。專門提供使用者開放原始碼內容管理系統評分的網站 OpenSourceCMS，就將入口網站軟體、部落格、電子商務、群組軟體、相簿甚至 E-learning 軟體都列入內容管理系統軟體的清單中[1]。

儘管內容管理系統的種類非常多，而每一種內容管理系統所擁有的功能也不盡相同，但各種內容管理系統提供的功能，大致不出圖 1.1 的範圍[1]。

[1] `http://tinyurl.com/4lm5pr`

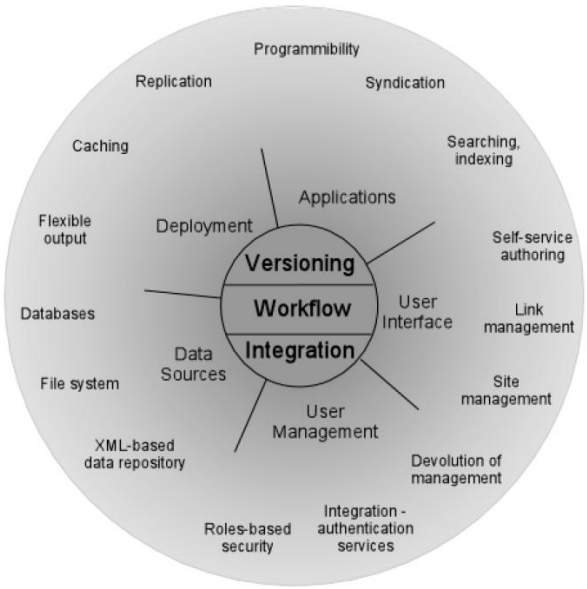

圖 1.1: 內容管理系統所具有的功能 [2]

1.1.2 內容管理系統應用

目前內容管理系統已被大量應用在各種場合中，而隨著 Web 2.0 「使用者創作內容」理念的興起，也有愈來愈多的廠商提供各式各樣的內容管理系統供使用者使用，例如無名小站的網路相簿，Blogger.com 提供的部落格等。

此外，也有許多組織使用內容管理系統建立官方網站，舉例而言，國立中央大學企業管理學系，即是使用開放源碼的 WordPress 此一部落格系統架設該系的網站[2]。

除了上述的應用外，內容管理系統也經常被應用於知識分享，如採用 Wiki 架構的維基百科[3]，以及本校採用 Moodle 此套開放源碼線上教學平台軟體架設的課程資訊網等[4]，均是以內容管理系統做為知識分享的實例。

在產業應用中，衛浴台灣這個以「台灣衛浴產業交流平台」為目標的網站[5]，則是以 Drupal 此套內容管理系統架設，提供了諸如衛浴產品的型號規格，台灣衛浴廠

[2] http://www.ba.ncu.edu.tw/
[3] http://zh.wikipedia.org
[4] http://moodle.ncnu.edu.tw
[5] http://www.tw-bathroom.info/

商列表，以及部落格和討論區等。由此可以看出，內容管理系統不僅被應用在個人網站及學術與非營利組織中，產業界內亦有使用。

1.2 研究動機

僅管內容管理系統已被大量應用在各種領域中，然而目前市面上的內容管理系統，大部份均將重點放在網頁的建立、呈現與管理上，對於網頁後設資料的支援則較為不足。

這是由於網頁所使用的 HTML 標記語言，主要是以排版為主，其目的在於呈現人類可閱讀的文章，而非提供利於程式自動化處理，可供計算與推論的機器可讀資料。

這樣的網站建設方式，固然適合使用在以閱讀為主體的個人網站與企業網站。但若我們期望能夠建立一產業資料庫網站，讓使用者在閱讀的同時，也能夠利用程式分析資料庫網站內的資料，計算諸如整體產業產出等，那麼現行的內容管理系統，就顯得尚有需要改進的不足之處。

因此我們提出了這一份研究，試圖改進現有的內容管理系統，使其能夠支援一定程度的計算與推論。

1.3 研究目的與範圍

本研究的目的，在於擴充現有的內容管理系統，使內容管理系統不僅只具有網站架設與維護的功能，更進一步讓內容管理系統能夠處理網頁資料呈現之外的後設資料。

我們期望透過這樣子的擴充，讓內容管理系統不只能應用於網站架設上，同時也能夠利用網站內的資料，進行一定程度的分析與自動化處理。

舉例而言，我們希望這套系統將來能應用在產業資料庫網站中。在這個網站中，產業內的相關廠商可以填寫生產的產能等參數，而透過這些參數與事前定義的

產業結構，將可以對最終商品的總產能進行估算。

由於時間上的限制，我們將本研究的範圍集中在如何於內容管理系統中加入適當的架構，達成加入後設資料處理功能此一目標。對於產業資料庫的實際建立及模擬，則將其列為未來研究方向。

1.4 論文架構

本論文架構，在第二章相關研究的部份會先針對內容管理系統進行介紹，並探討在目前現有的內容管理系統架構中，有哪些系統試圖解決後設資料的處理問題，以及其優缺點，最後則會對 Ontology 進行簡單的介紹。

在討論了相關的系統優缺點後，第三章系統設計的部份，我們提出了我們的系統的設計理念與架構，同時針對系統中的相關名詞進行定義與說明。

第四章系統實作的部份，我們會針對如何達成第三章所提到的功能做說明，而最後一章的結論，則說明了本研究的貢獻，以及未來研究方向。

第二章　相關研究

在此章中，我們將分別針對兩種不同型態的內容管理系統以及其使用上的限制進行討論，第一種是功能較為簡單，開放性較强的 Wiki，另一種則是入口網站型式的內容管理系統。

討論完這些內容管理系統的限制後，我們亦會提出針對這些限制所思考出的解決方案。

2.1　Wiki

2.1.1　Wiki 概述

Wiki 是內容管理系統的一種，最初由 Ward Cunningham 於 1995 年所提出，其目的在創造一種透過網路而達到多人協同寫作的超文本系統，語源則來自於夏威夷語中 WikiWiki 一詞，意為「快點快點」[3]。

根據 Ward Cunningham 於 2001 年出版的書籍中，Wiki 的定義為「一種允許一群用戶通過簡單的標記語言來創建和連接一組網頁的社會計算系統」[4]。

要特別注意的是，在上述 Ward Cunningham 的定義中，並未詳細說明 Wiki 系統的實作細節與規格，Wiki 僅是一種概念性的總稱；而每個 Wiki 系統軟體的實作，稱為「Wiki 引擎」，各 Wiki 引擎所採用的使用者管理模型、標記語言等也均有不同；使用特定 Wiki 引擎所建立的網站則稱為「Wiki 網站」。為求表達上的精確，在本研究中將會詳細區分這三者的不同，表 2.1 為一簡單的整理。

表 2.1: Wiki 相關名詞定義

名詞	定義	說明
Wiki / Wiki 系統	一種允許一群用戶通過簡單的標記語言來創建和連接一組網頁的社會計算系統。	概念上的總稱，沒有特定的規格與標準。
Wiki 引擎	實作 Wiki 概念的軟體。	如 MediaWiki 與 PHP-Wiki，每一套 Wiki 引擎均有不同的特色與限制。
Wiki 網站	利用過 Wiki 引擎建置的網站。	例如 Wikipedia 即為一 Wiki 網站。

在 Wiki 系統的設計初期，Ward Cunningham 便將開放性視為 Wiki 系統的原則之一，當有人看到頁面中內容不完整或架構不良時，可以立即對其進行編修，而不需要通知原作者進行修改。而隨著各式 Wiki 引擎及網站的發展，假設人性本善以及對人的信任也成為了 Wiki 的衍生原則之一[5]。

雖然 Ward Cunningham 並未詳細定義 Wiki 所應具有的功能與技術規格，但現今的 Wiki 引擎通常具有下列的特色：

- 可以開放讓匿名使用者直接編輯頁面。
- 編輯時採用直接輸入 WikiText 標記語言。
- 採用的文件標記語言 WikiText 比 HTML 更簡潔。
- 具有版本控制機制，使用者可以比較或恢復到某個特定的版本。
- 系統內的文章可以利用超鏈結相互參照。
- 可以參照到尚未建立的頁面，當使用者點入該鏈結時，則可創建該頁面。

在操作上，使用者在 Wiki 系統中透過編修簡單的 WikiText 標記語言原始碼，即可產生相對應的網頁頁面，然而要特別注意的是，在不同 Wiki 引擎中，其所採用的 WikiText 標記語言格式也有所不同。

由於 Wiki 提供了一個簡單的文件協同編輯環境，多數不特定的使用者可以不斷修改與潤飾已經存在的文件，或是新增一份文件，因此台灣的使用者亦有人將 Wiki 翻譯成「共筆」－－共同筆記，其餘使用的譯名尚有維客、圍紀、快紀等。

除了上述譯名外，由於「維基百科」是最為人知的 Wiki 網站之一，隨著「維基」一詞漸漸被普遍使用，「維基」這個詞也經常被誤認為 Wiki 的譯名[6]。但實際上，「維基」一詞出自於 Wikipedia 網站的譯名「維基百科」，同時也是維基媒體基金會的註冊商標，用在由該基金會透過 MediaWiki 此套 Wiki 引擎所建構的所有中文網站中[1]。

2.1.2 Wikipedia 與 MediaWiki

維基百科 (Wikipedia) 是最為人知的 Wiki 網站之一，其目標在於透過使用者自主參與編寫條目，完成一部自由的百科全書[7]。在全球各地使用者的參與之下，目前維基百科共有約 250 種語言的版本，其中英文版的維基百科條目數已達二千多萬條，而中文版維基百科亦超過十四萬條條目。

在 2.1.1 小節中，我們曾提到假設人性本善已成了 Wiki 系統的原則，當使用者群體成長到一定程度時，善意使用者的人數會超越破壞性的使用者人數，因此即使有惡意使用者將文件破壞時，多數的善意使用者將會利用版本控制的功能將文件回復至正確的版本。

以維基百科中文版為例，截至 2008 年 1 月底為止，共有 161,906 篇條目，而其中僅有大約 500 個條目因為被頻繁地破壞或其他理由而被管理員設為保護頁面，只有註冊的使用者或具有管理權限的管理員能夠進行編修，僅佔全部文件數目的百分之 0.3[2]。

在實作上，維基百科及維基媒體基金會的其餘子計劃，均採用了 MediaWiki 此套 Wiki 引擎架設。MediaWiki 原本是專為維基百科而撰寫的自由軟體，採用 GPL 授權，目前也被廣泛運用於其他 Wiki 網站的架設。

[1]維基媒體基金會是負維基百科計劃的非營利性組織，其餘計劃尚有維基詞典、維基教科書與維基文庫...等。

[2]根據中文版維基百科首頁總條目統計數據及特殊頁面中「被保護的頁面」列表計算。

程式碼 2.1: Wiki 條目原始碼（節錄）

```
'''London''' is the [[capital]] and largest urban area
of [[England]] and the [[United Kingdom]].

London has an official population of 7,512,400...
```

MediaWiki 採用 PHP 做為程式語言，除了支援協同編輯、版本控制等基本功能外，亦加入了諸如檔案上傳、模版與延伸套件等進階的功能。

由於 MediaWiki 提供了許多進階的功能，使得 Wiki 系統更具實用性。然而 MediaWiki 對於後設資料的處理依然有其限制，在下一小節中，我們將針對現行 MediaWiki 的對於後設資料處理的問題進行討論。此外，我們也會在往後的章節中介紹其他研究者針對這些問題所提出的解決方法。

2.1.3 MediaWiki 的特色及限制

Wiki 的主體為供人直接閱讀的文章，並且强調文章與文章間的連結，因此通常在 Wiki 系統中可以輕易地對不同的文章進行超連結，只需要在 WikiText 中標示出需要建立超鏈結的部份，系統即會自動將其鏈結到同名的文章[8]。

以維基百科所使用的 MediaWiki 系統為例，使用者只需在編寫文件時，將特定字詞以中括號括住，即可建立超鏈結。當括號內的字詞已存在於文件庫時，系統會自行建立超鏈結至該篇文章，若該篇文章不存在時，則會鏈結至新增該篇文章的操作介面。

舉例而言，英文版維基百科的 London 條目中，其中一段的原始碼內容如程式碼 2.1 所示[3]。當使用者瀏覽此條目時，其中的 capital, England 以及 United Kingdom 字樣，都會成為超鏈結，指向不同的條目，供使用者交互閱讀參考。

由上述的例子中，我們可以觀察到在 Wiki 系統中，條目間並無先後以及從屬的關係，條目內可以連結至尚未被建立的條目，等待其他使用者撰寫被鏈結的條目，

[3] `http://en.wikipedia.org/w/index.php?title=London\&action=edit`

當條目被建立後，該鏈結會自動更正，指向新建立的條目。

而由上述的原始碼中，我們可以發現 Wiki 是一種以供人閱讀的文件為導向的平台，在原始檔中，我們只撰寫了一段描述性的文字，並且將關鍵字連結至相關的文章。

在這樣的撰寫方式中，我們無法明確地以機器可讀的方式來說明「倫敦是英國的首都」，以及倫敦的人口數為 7,512,400 等事實，同時也無法利用這些數據進行自動化的處理，諸如依照城市人口數目排序並輸出列表等。

2.2 Semantic Wikipedia

2.2.1 Semantic Wikipedia 簡介

為了解決 2.1.3 小節中提到的問題，Max Volkel 等人在 2006 年提出了 Semantic Wikipedia 的概念，以補足目前 MediaWiki 引擎中對於後設資料處理不足的缺陷[8]。

Semantic Wikipedia 的目的在於將 Wiki 內供人閱讀為主的內容，加入機器可讀的註解 (Annotation)，期望可以做到諸如「依照國土大小將國家列出」等目前 MediaWiki 引擎無法做到的動態查詢功能，以及替條目與條目之間進行語意上的連結。

值得注意的是，雖然該研究的標題為 Semantic Wikipedia，但如同我們在 2.1.2 中所提及，維基百科僅是由 MediaWiki 此套 Wiki 引擎所建立的網站，因此該文中所實作的的延伸套件，事實上是針對 MediaWiki 設計，任何採用 MediaWiki 此套 Wiki 引擎所建立的 Wiki 網站，都可以使用這套延伸套件。而下文中所有 Semantic Wikipedia 字樣，均指該文中所提及為 MediaWiki 所設計的 Semantic 延伸套件，為免混淆，特此說明。

為了實現將現有 MediaWiki 引擎加上 Semantic Web 技術的這個目標，Semantic Wikiepdia 引入了兩個新的概念：屬性與關聯。

屬性這個概念的引入加强了 MediaWiki 條目中關於結構性資料的機器可讀性，程式得以更容易地處理結構化的資料，例如國土面積、人口總數、人均 GDP 等描述一個國家時必定會有的資料。而透過機器可讀的結構化資料，使用者則可以進行諸如列出人均 GDP 超過 20,000 美元國家等結構化的動態條件查詢。

關聯的部份，則加强了 MediaWiki 系統中，無法充份反應條目與條目間語意上的關聯此問題。透過語法的延申，使用者能夠表達出條目與條目之間的語意關聯，比如「『倫敦』是『英國』的首都」這樣的語意。

接下來，我們將分別針對該系統中屬性及關聯的設計分別進行簡單的介紹，以及探討在這樣的設計中，有什麼樣的限制與缺陷，以致於我們採用其他的方式來處理我們系統中的後設資料。

2.2.2 屬性

為了表達結構化的資料，並使其具有機器可讀性，Semantic Wikipedia 採用了讓使用者在編輯文章的文本時，直接將屬性的名稱及值內嵌在文章原始碼中的作法。使用者可以在文章中提及屬性的值，並且例用 MediaWiki 的特殊語法，替該屬性的值加註屬性名稱及資料型態。

舉例而言，若在一篇描述倫敦的文章中提及了如下的段落：

> As of 2005, the total resident population of London was estimated 7,421,328.
> Greater London covers an area of 609 square miles.

則其中 7,421,328 為 population 這個屬性的值，型態為整數；609 為 area 這個屬性的值，其型態為整數，單位為平方英里。這樣的描述，轉換為 Semantic Wikipedia 的文章原始碼，則會如程式碼 2.2 所示[8]。

Semantic Wikipedia 的設計架構中，上述原始碼 2.2 中的 area 與 population 等屬性的名稱是由使用者在編輯文本時自由選擇的。但需要注意的是，由於架構上的限制，Semantic Wikipedia 中某篇文章所具有的屬性，都必須在文章內文中出現。

程式碼 2.2: 內籤屬性的 Semantic Wiki 原始碼

```
As of [[2005]], the total resident population of London was
estimated [[population:=7,421,328]].

Greater London covers an area of [[area:=609 square miles]].
```

程式碼 2.3: Wikipedia 人物傳記模版

```
{{Infobox Biography
| subject_name   = 名稱
| image_name     = 圖片名稱（不須手工加上連結）
| image_size     = 圖片大小
| image_caption  = 圖片說明
| date_of_birth  = 出生日期
| place_of_birth = 出生地點
| date_of_death  = 去世日期
| place_of_death = 去世地點
| occupation     = 身份
| spouse         = 配偶
}}
```

在提醒使用者一篇關於特定主題的文章中，該具備哪些屬性方面，Semantic Wikiepdia 採用了 MediaWiki 本身所具備的模版功能來處理。在實際使用上，使用者必須先至模版列表查詢是否有適當的模版，並在撰寫文章時於原始碼中插入模版，並填寫正確的屬性值。

舉例而言，當使用者在維基百科上撰寫人物傳記時，可先以至模版列表查詢到如程式碼 2.3 的模版資料[4]，並且將這個模板複製到目前正在撰寫的文本中，最後再把模板中等號右側的說明，改寫成正確的屬性值。

要特別說明的是，在 MediaWiki 的系統中，模版是由系統管理員所建立，只有

[4] http://zh.wikipedia.org/w/index.php?title=Template:Infobox_Biography&variant=zh-tw

程式碼 2.4: Semantic Wikipedia 關聯語法範例

```
'''London'' is the capital city of [[capital of::England]]
and of the [[is capital of::United Kingdom]].
```

特定權限的使用者可以建立模版，一般的編輯者只能夠使用系統管理員已經建立好的模版。

2.2.3 關聯

除了利用上一小節的屬性來加註結構化的資料外，Semantic Wikipedia 亦提供了關聯的方式來表達條目與條目之間語意上的關聯，來補强原始 MediaWiki 系統僅以超鏈結來連接兩個條目的不足之處。

舉例來說，若在描述關於倫敦的條目中，寫到了「倫敦是英格蘭與聯合王國的首都」這件事實，在傳統的 Wiki 系統中，使用者會利用 Wiki 本身提供的超連結功能，將 England 與 United Kingdom 連結到相關的條目中，請參照第 8 頁中的原始碼 2.1。

而在 Semantic Wikipedia 系統中，使用者可在超鏈結的語法中，加入關鍵字描述兩者之間的關聯，例如在原始碼 2.4 中，明確地以關鍵字 capital of 以及 is capital of 來指明倫敦與英格蘭和聯合王國間的關聯。

2.2.4 Semantic Wikipedia 文章原始碼解析

上述的兩小節中，我們針對了 Semantic Wikipedia 如何讓使用者在編輯文章時加註屬性與關聯等方式進行了討論，但由於所有的屬性與關聯，都是直接由使用者在編輯文章時加註在原始碼中，為了要將屬性與關聯儲存為機器可讀的格式，Semantic Wikipedia 必需在使用者對文章進行存檔時，解析文章原始碼，並取出屬性與關聯等資料。

圖 2.1 為 Semantic Wikipedia 的系統架構，使用者編寫完成文章的原始碼後，

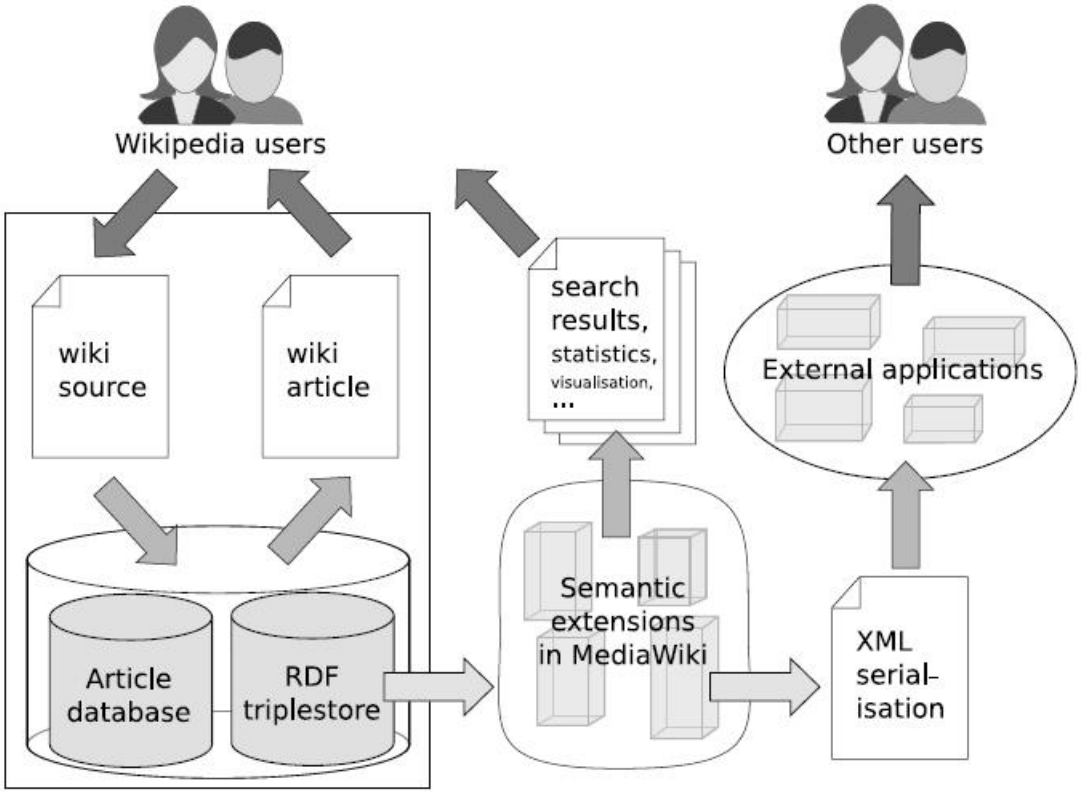

圖 2.1: Semantic Wikipedia 系統架構 [8]

系統會解析文章的原始碼，將文章內容存入 Article database 中，而屬性與關聯的資料則存入 RDF triplestore 資料庫中，以利系統進行分析，進而產生諸如搜尋與統計結果等資料。

2.2.5 問題探討

Semantic Wikiepdia 採用於文章原始碼中加註後設資料的方式，來解決傳統 MediaWiki 系統無法讓文章內結構化資料具有機器可讀性的問題。然而在 Semantic Wikipedia 中，仍有些許的問題尚未解決。

原作者在論文中提到，由於屬性與關聯的名稱都是由使用者自行命名，因此仍會產生同名異義與異名同義等問題，而其解決的方式，則是讓每一個屬性與關聯的名稱，也鏈結至一個頁面，使用者可以在其頁面中加註其用途，避免其他使用者誤用。[8]

除了同名異義外，Semantic Wikipedia 並沒有所謂「概念」的層次，因此使用者可以標示出「台北是倫敦的首都」這樣的事實，但是我們可以知道，這樣的事實是不正確的，因為台北必須是某個「國家」的首都，而不能是「城市」的首都。

我們亦發現，在 Semantic Wikipedia 此系統中，所有屬性與關聯都必需在文章中提及，而且使用者無法在編輯文章時，立刻就知道需要填入哪些值。相反的，使用者必須先查詢模版列表，找到適合使用的模版，才能透過模板知道應該填寫哪些屬性，也容易造成資料漏填的問題。

最後，雖然 Wiki 强調文章原始碼的簡潔易懂，但對於第一次接觸的使用者而言，仍然需要一段時間的學習，才能夠理解其語法，對於使用者的親和力仍有不足。

2.3 Drupal

2.3.1 Drupal 簡介

上述兩節介紹了 Wiki 此類型的內容管理系統，以及如何在 Wiki 的架構下讓文章內容具有機器可讀性。然而 Wiki 的主要應用是協同文件撰寫，沒有嚴格的權限控管，並且是以文章為主體，無法涵概所有的內容管理應用，因此在這一小節中，我們將會介紹另一套內容管理系統 Drupal。

Drupal 是一套開放原始碼的內容管理系統，提供了完整的角色權限控管機制以及多樣化的功能模組，根據其官方網站說明，Drupal 適合應用於下列類型的網站架設[9]：

- 社群入口網站
- 論壇網站
- 公司網站
- Intranet 應用
- 個人網站與網誌
- 電子商務網站

與 Wiki 不同，Drupal 是一個較為通用的內容管理系統，提供了諸如新聞、線上手冊、部落格與討論區等功能。

Drupal 亦採用了模組的概念，在官方提供的版本中，僅有最基礎的核心模組與常用的延伸模組。核心模組負責權限控管以及使用者管理此類的網站基礎功能，若使用者需要增加網站的功能，則可以安裝其他延伸模組[10]。

在實際應用上，包括國立高雄師範大學圖書館、清華大學哲學研究所、破報與勞苦網等，均採用 Drupal 進行架設與管理，而亦有正體中文的 Drupal Taiwan 社群成立，提供了技術文件、翻譯與討論等[5]。

[5]使用 Drupal 之正體中文網站列表可於 `http://drupaltaiwan.org/image/tid/23` 取得

2.3.2 技術堆疊

在這一小節中，我們將會簡單地介紹 Drupal 的設計理念以及系統架構。

Drupal 在設計時，就將客製化列為目標之一，而客製化的方式是採用覆寫核心模組的功能與增加新的模組，而不是直接更改核心模組的程式碼。此外，Drupal 也採用內容與外觀分離的方式，透過佈景主題的功能，同樣的內容可以用不同的外觀呈現[10]。

由於 Drupal 採用了此種設計思維，因此提供了相當完整的 API 給開發者，開發者只需專注在如何利用 Drupal 所提功的 API 與核心模組進行溝通，並新增自己所需的功能。至於核心模組等與網站的基本功能相關的程式碼，則交由 Drupal 的開發者進行維護與開發。

圖 2.2 為一個 Drupal 網站的模組概觀，其中粗框的部份，是 Drupal 的核心模組，提供了使用者管理、本土化、事件記錄等基本的功能。而諸如相簿，電子商務網站、所見即得編輯器，則是由加掛的延伸模組所提供。

在技術方面，Drupal 採用了 PHP 做為程式語言，而由於具有資料庫抽象層的關係，Drupal 的後端資料庫可以採用如 MySQL 及 PostgresSQL 等各種不同的資料庫管理系統，圖 2.3 為 Drupal 的技術堆疊，由圖中可以看出，Drupal 能在各種不同作業系統、網頁伺服器與資料庫管理系統組合的平台上運作。

2.3.3 Drupal Hook 簡介

Drupal 使用了控制權反轉的概念來設計，模組開發者只需要遵守一定的規範，撰寫好相關的函式，Drupal 即會在適當的時機呼叫模組開發者所撰寫的程式，改變系統內定的作業程序，此種函數稱為 Drupal Hooks。

透過 Drupal Hooks，模組開發者可以達到在不修改核心原始碼的狀況下，替系統增加新的功能，例如在使用者將文章存入系統內建的資料表的同時，進行額外的處理，將其餘相關資訊存入模組自行定義的資料表中，達成諸如統計、附加資料等功能。

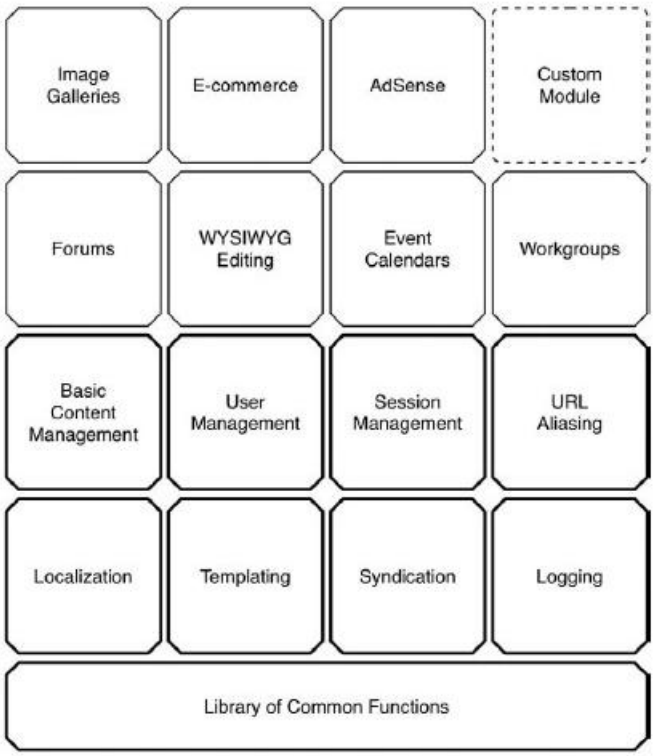

圖 2.2: Drupal 網站模組堆疊 [10]

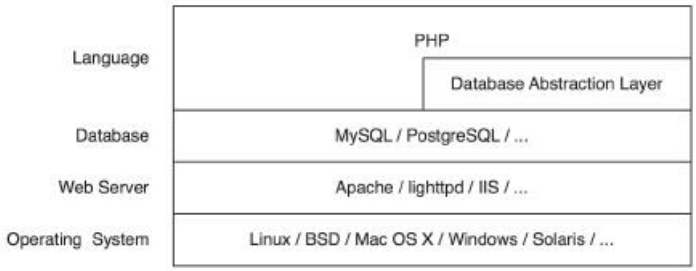

圖 2.3: Drupal 技術堆疊 [10]

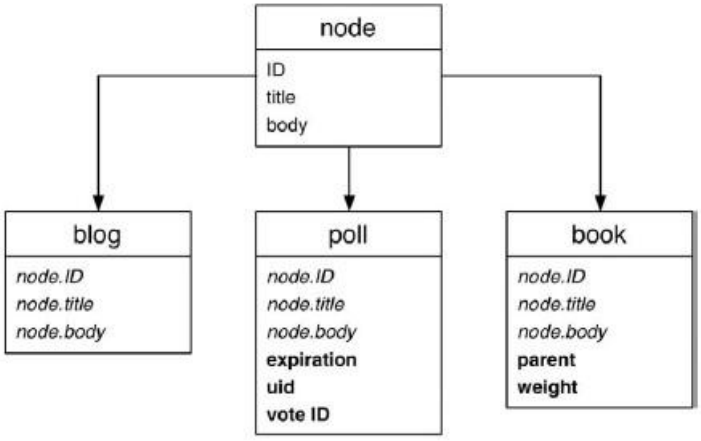

圖 2.4: Node 與其子類別 [10]

2.4 Drupal 後設資料相關模組

我們在 2.3 節中提到，Drupal 具有完整的功能、豐富的 API 以及跨平台等優點，然而 Drupal 對於後設資料的處理，也有其限制。

因此在這一節中，我們將會針對一些已經被廣泛採用的 Drupal 模組進行介紹，並分析這些模組尚有哪些缺陷，無法達成我們在 1.3 節中所期望的目標。

2.4.1 Node 模組

在 Drupal 中，Node 是一筆內容資料，例如一篇部落格的文章、投票、線上文件手冊都是 Node 的一種。Node 模組則是 Drupal 核心中內附最基礎的內容管理功能模組，大部份進階的內容管理延伸模組，均是透過延伸 Node 模組或是複寫其中的功能來實作[10]。

在 Node 模組中，Node 是最基本的內容類型 (Content Type)，其中僅定義了如 ID、標題與內文等最基礎的資料欄位，做為父類別。其他進階的內容類型，例如 Drupal 核心內附的部落格 (Blog)、投票 (Poll)、文件手冊 (Book) 等內容模組，其內容類型均繼承自 Node，如圖 2.4 所示。

在圖 2.4 中，我們可以觀察到 blog、poll 與 book 三個不同的內容類型，都由 node 繼承了 ID、標題與內文的屬性。

其中投票具有期限 (expiration)、發起投票的使用者 (uid)，以及本次投票編號 (voteID) 等額外的欄位。相對的，由於線上手冊需要記錄章節的關係，以及在同一章下不同小節的先後順序，因此具有 parent 與 weight 這兩個欄位。

在實作上，開發者可以自行撰寫 Drupal 延伸模組，並且在資料庫的設計上，存放內容的資料表僅存放標題、內文外的資料，並且擁有一個 nodeID 欄位，且該欄位為一參考到 Node 模組中 Node 資料表裡 ID 欄位的外鍵。

在建立完資料庫綱要後，開發者可以利用 2.3.3 小節中提到的 Drupal Hooks，向 Drupal 註冊新的內容類型，並且複寫原有的 Node 模組中的功能（例如在新增內容時將資料插入到適當的資料表，在顯示時從適當的資料表中取得資料並顯示），達到新增內容類型的目的。

雖然由於 Drupal Hooks 及資料庫設計上的彈性，讓模組開發者可以透過撰寫模組並繼承原有 Node 模組的方式來建立新的內容類型，但缺點則是每次建立新的內容類型，都得重新撰寫程式，無法讓不會寫程式的使用者建立新的內容類型。

2.4.2 Content Construction Kit 模組

在上一小節中，我們介紹了如何在 Drupal 中利用撰寫模組的方式建立新的內容類型，也提到了此種方式的缺點。

Content Construction Kit（簡稱 CCK）為一 Drupal 延伸模組，透過這個延伸模組，使用者只需要在 Drupal 網站的管理介面中進行操作，即可自行定義新的內容類型與所需欄位，底層相關資料表的建立，則由 CCK 自動處理。

當啓用 CCK 模組後，使用者可以在管理介面中新增內容類型，並且設定該內容類型除了基本的標題、內文外，還需要哪些欄位。

舉例而言，若我們新增了一種內容類型是存放「CPU 簡介」的相關資料，做為產品資料庫所用，那麼我們除了標題以及內文之外，還需要諸如 CPU 製造商、時脈

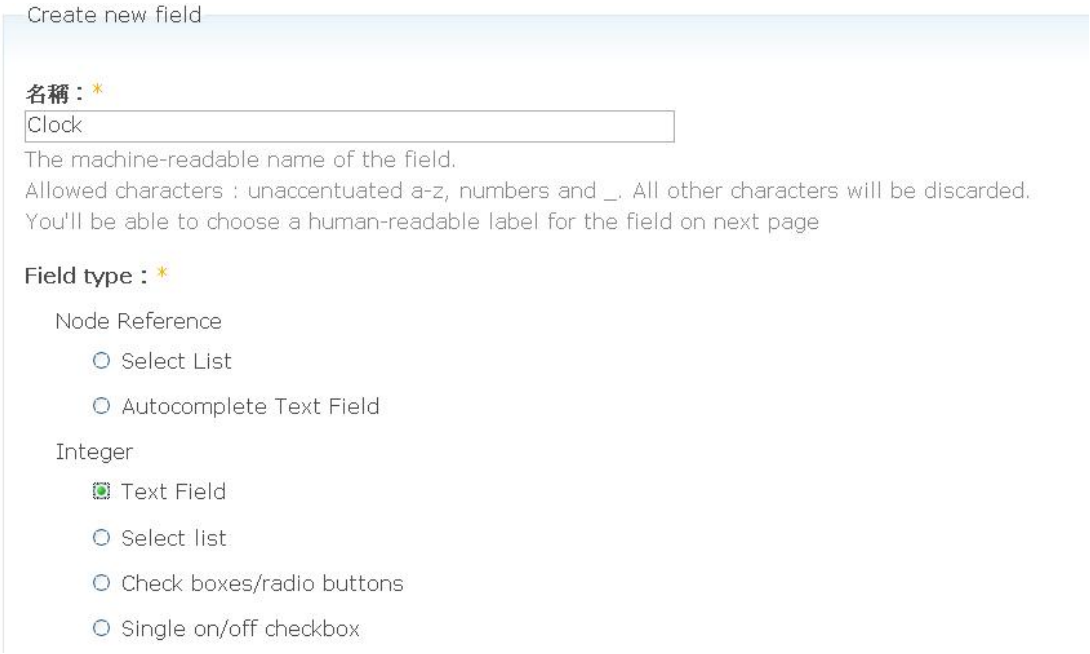

圖 2.5: 利用 CCK 建立新的欄位

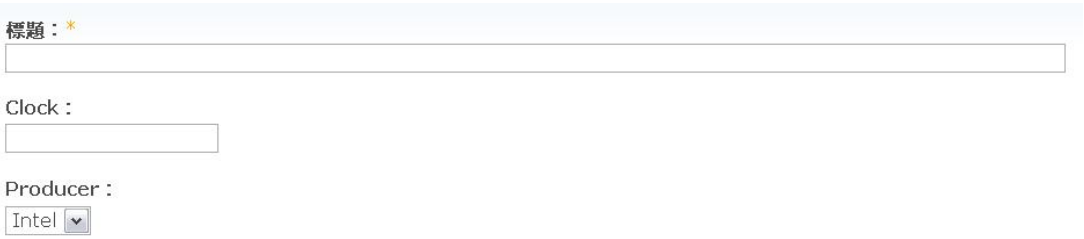

圖 2.6: 由 CCK 所建立的內容類型與新增的欄位

等欄位供讀者參考。

要達成這樣的功能，系統管理者只需要在安裝 CCK 模組後，於管理介面中輸入欄位名稱，選擇適當的資料型態與輸入方式，並且設定該欄位是否為必填欄位等資訊即可。圖 2.5 即為建立 CPU 時脈欄位的畫面，在此例中我們選擇資料型態為整數，輸入方式為文字方塊。

設定完成後，當使用者新增一筆內容類型為「CPU 簡介」的內容時，除了標題、內文等 Node 模組本身就具有的欄位外，我們透過 CCK 新增的欄位，例如 Producer 與 Clock 欄位也會顯示在表單中，供使用者填入適當的值，如圖 2.6 所示。

然而利用 CCK 建立新的內容類型，雖然系統會自動顯示應該填寫的欄位，解決

2.2 一節中 Semantic Wikiepdia 中使用者不易得知所需填寫的屬性的問題，但 CCK 亦有其缺點存在。

由於 CCK 所設定的欄位是跟隨內容類型，而且彼此間無法再繼續繼承或延伸，因此每一種後設資料的組合，都必需建立一個新的內容類型，當後設資料的組合增加時，內容類型也會隨之增加，導至系統不具實用性。

舉例而言，一部動畫作品可能會同時具有電視版、劇場版與 DVD 等三種版本，這三種版本的後設資料有些是相同的，例如作品名稱等，但也有些後設資料會有不同，例如電視版需要記錄播放集數、劇場版會需要記錄上映的戲院或票房，而 DVD 則需要記錄售價。此外，也不是所有的動畫作品都會具有這三種版本。

然而在 CCK 的架構下，使用者無法建立一個單一的「動畫」的內容類型，讓使用者選填是否要填寫電視版、劇場版與 DVD 版的後設資料。相反的，使用者只能新增三種內容類型：「電視動畫」、「動畫電影」、「動畫 DVD」，並且在這三個內容類型中分別新增一筆針對同樣的動畫作品做描述的資料，造成資料的重複及不一致。

2.4.3 Taxonomy 模組

Taxonomy 模組是 Drupal 核心內附的分類系統，透過這套系統，使用者可以自行定義 Flat、Hierarchical 以及 Multiple Hierarchical 等三不種同類型的分類方式，以下將分別敘述三種分類方式。

在 Flat 的分類方式中，所有的分類沒有任何階層關係，是獨立存在的，Hierarchical 則是子分類僅屬於一個父分類；而在 Multiple Hierarchical 的分類關係中，一個子分類則可以屬一個以上不同的父分類，圖 2.7 與 2.8 分別為 Hierarchical 與 Multiple Hierarchical 兩種不同的分類方式的示意圖[10]。

在 Taxonomy 模組的定義裡，圖 2.7 與圖 2.8 中，所有分類的名稱，如 Object-Oriented、PHP 等均稱為一個 Term。而我們也可以發現，由於 PHP 既可以歸類為物件導向程式語言，亦可歸類為程序導向程式語言，若透過 Multiple Hierarchical 的分類方式，則可以適當地表達這種概念。

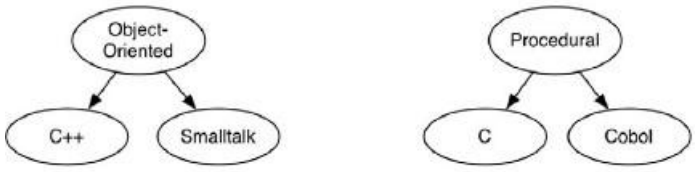

圖 2.7: Hierarchical 分類示意圖 [10]

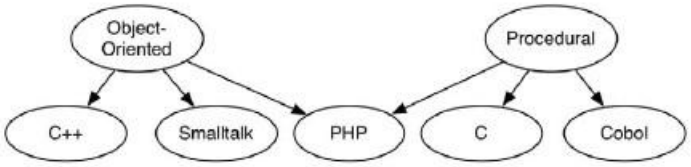

圖 2.8: Multiple Hierarchical 分類示意圖 [10]

除了分類方式外，使用者亦可設定哪種內容類型與此分類是有關的，以及該內容類型是否可以屬於多個分類等設定。當設定完成後，使用者在建立內容時，便可選擇該內容屬於哪一個分類中。

例如一篇內容型類型為「IDE 開發環境」的文章，可以依照其支援的程式語言歸類，而一個 IDE 又可以支援多個不同的程式語言。

如圖 2.9 中的範例所示，使用者將程式語言的分類建好後，可以直接在表單中選擇文章所提及的 IDE 開發環境支援哪些程式語言。

當使用者儲存此筆資料後，即可透過程式語言分類去找尋到適當的 IDE 開發環境。同樣的，若一個 IDE 同時支援 C++ 與 Small Talk 時，則使用者不論是瀏覽 C++ 或 Small Talk 分類，都可以找到該 IDE 的資料。

除了階層式的架構外，Taxonomy 模組亦支援 Related Terms 的功能，使用者可以將 Related Terms 視為類似字典或 API 文件中的 See Also 項目[10]。

舉例而言，在上述程式語言的分類中，我們知道 C++ 是 C 語言的超集，因此我們可以設定 C++ 的 Related Terms 為 C 語言，而 C 語言的 Related Terms 為 C++。

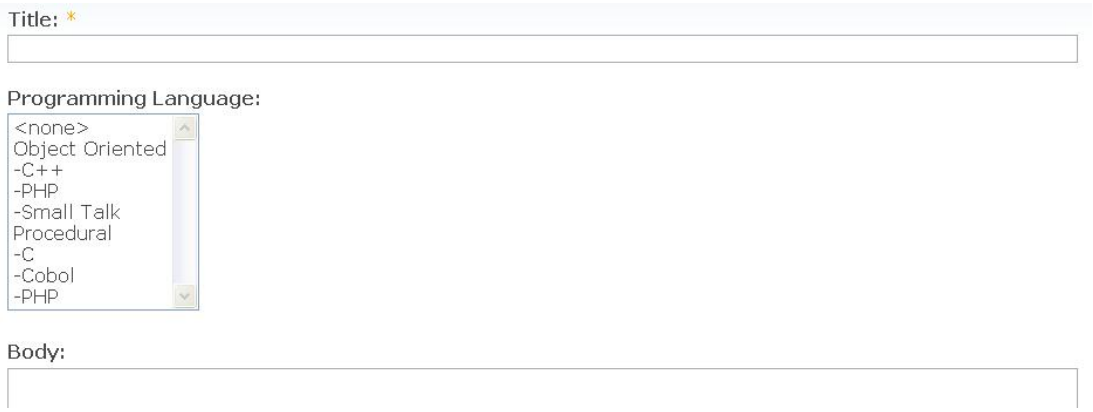

圖 2.9: 替內容選取分類

然而 Taxonomy 模組最大的問題，在於並未清楚表達 Terms 之間的關聯。在階層的架構中我們或許可以預設其為 is-a 或 has-a 的關聯，但在上述 Related Terms 的範例方面，我們僅知道 C++ 語 C 語言是有關聯的，但卻無法得知切確的關係（C++ 是 C 語言的超集，C 語言是 C++ 的子集）。

此外要注意的是，Taxonomy 模組裡所謂的「關聯」，僅僅針對分類本身而言，並無法指定內容與內容間的關聯。

舉例而言，雖然使用者可以將「政府」這個分類設定為「國家」的子分類，並且將其視為 has-a 的關聯，但卻無法明確地將某篇描述「中華民國政府架構」的內容，與描述「中華民國」的內容進行關聯的設定。

2.5 Ontology

在上述的章節中，我們討論了其他系統如何在現行內容管理系統新增屬性與關聯等後設資料處理的功能，然而這些系統都仍有各式各樣的缺陷，並無法完整解決問題。因此在這一小節中，我們將針對 Ontology 這個概念進行探討，並分析是否能夠將 Ontology 應用於內容管理系統中。

Ontology 一詞源於哲學領域，係為以一系統化方式描述實體萬物的理論。而在人工智慧的領域中，Ontology 最早由 Neches 及其同事所定義，其定義如下[11]：

> An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary.[12]

Ontology 的特性在於將特定知識領域概念化，並以圖形的方式呈現，而在進行概念化的過程中，則以「類別（物件）」、「屬性」、「關聯」三者所組成。類別代表了一個總體性的概念，「屬性」定義類別的特性，而「關聯」則表達類別與類別間的互動。而雖然目前 Ontology 有各式各樣的定義，然而大部份均認同下列的幾項原則[13]：

1. 由類別（物件）、屬性與關聯組成。
2. 物件可以擁有屬性。
3. 物件與物件間可以有各式各樣的關聯。
4. 屬性與關聯可隨著時間改變。
5. 在特定的時間下可以發生不同的事件。
6. 事件會造成件物屬性的改變。
7. 物件可以參與特定的程序 (Process)。
8. 物件可以有不同的狀態。
9. 物件可以分為更小的單位物件 (Parts)。

與物件導向相似，Ontology 具有「類別」的概念，類別僅定義了物件所應具有的屬性，而實際的屬性值則視個別的物件而定。

雖然上述列出了九項 Ontology 的特性，但經過我們整理後，認為只需要第一至四項基礎的功能，即可充份的讓內容管理系統合理地表達屬性與關聯的後設資料。

透過採用簡化過後的 Ontology，我們可以在不增加使用者學習負擔的情況下，利用通用的方式來對關聯及屬性進行描述。同時由於 Ontology 本身即與物件導向類似，將概念與實體層次分開討論，因此可以避免如第 2.2.5 小節中所提及的 Semantic Wikipedia 在表達關聯時的錯誤。

2.6 小結

在 2.2 與 2.4 兩節中，我們探討了 Semantic Wikipedia 以及 Drupal 中如何處理屬性後設資料與設定內容間的關聯，以及其作法的優缺點，表 2.2 與表 2.3 為一簡單的比較。

由表 2.2 與表 2.3 中可以看出，在彈性方面，Semantic Wikipedia 採用直接於文章內容原始碼中加入註記的方式，讓使用者在輸入屬性等後設資料與設定內容之間的關聯時，擁有較大的彈性，而且不需要顧及文章的內容類型（因為 MediaWiki 本身並無內容類型的設計），但同時也產生了使用不易以及同名異義等問題。

相較之下，Drupal 的 CCK 與 Taxonomy 模組則專注於讓使用者容易建立自己的內容類型後設資料以及設定分類間的關聯，但 CCK 模組必須針對每種不同的後設資料組合產生不同的內容類型，而 Taxonomy 模組則無法適當表達出每一個 Terms 之間實際的關聯，也無法表達個別內容間的關聯。

在接下來的章節中，我們會介紹本系統如何在彈性與易用性間取得平衡，以及如何利用 Ontology 解決同名異義與處理不同內容之間關聯的表達。

表 2.2: 各系統屬性處理方式

系統	輸入方式	優點	缺點
Semantic Wikipedia	內嵌於文章原始碼	• 彈性大 • 不需考慮內容類型 • 具單位轉換功能	• 使用者需學習特殊語法 • 無法提供使用者應輸入屬性列表 • 屬性具有同名異義等問題
Drupal 模組	由模組開發者決定	• 彈性大	• 需額外撰寫程式
Drupal CCK 模組	HTML 表單	• 設定簡單 • 輸入容易	• 屬性組合與內容類型相關

表 2.3: 各系統關聯處理方式

系統	輸入方式	優點	缺點
Semantic Wikipedia	內嵌於文章原始碼	• 彈性大 • 可描述內容與內容間實際的關聯	• 使用者需學習特殊語法 • 屬性具有同名異義等問題
Drupal Taxonomy 模組	HTML 表單	• 操作介面容易 • 可定義同意字詞	• 無法確切表達 Related Terms 之間的關聯意義 • 僅能描述分類間的關聯，而無法描述內容間的關聯

第三章　系統設計

3.1　簡介

我們曾在 2.3 一節中提到，Drupal 具有諸如跨平台、功能完整，以及容易擴及撰寫功能模組等優點，因此本系統將會以 Drupal 做為基礎的平台，以 Drupal 功能模組的方式開發。

由於採用撰寫功能模組的方式實作，我們可以專注在實現我們所提出的新概念與功能，而不需處理諸如使用者管理、權限控管等基本功能，此外使用者也可以很容易地將本模組套用進現有的網站，而不需要重新學習另一套內容管理系統。

本功能模組的主要目的，則在於讓使用者透過此延伸模組，可以用 Ontology 的方式定義內容與內容間的關聯，並且在建立內容時填寫後設資料，以且解決上述第二章相關研究中現有系統的缺點。

在設計理念方面，我們採用了 Ontology 的概念，並且使 Ontology 與內容類型分離，因此 Drupal 中任何內容類型的資料，都可以使用單一的 Ontology 進行後設資料的處理。同時由於納入了 Ontology 的關係，使用者可以輕易地表達內容與內容之間的關聯，進而解決 Drupal Taxonomy 模組中所具有的缺點。

操作上界面的設計上，我們採用了 HTML 表單的方式，當使用者建立新的內容時，系統會自動尋找所需填寫的屬性值，並顯示在畫面上供使用者填寫，以解決 Semantic Wikiepdia 中使用者需學習特殊語法以及無法得知需要填寫哪些屬性的缺點。

關於本系統所採用的 Ontology 架構，我們會在之後的章節中進行詳細的說明。

3.2 名詞定義

為了方便往後的說明，在此節中，我們會簡單地定義在系統中所使用到的專有名詞，以及之間的關聯。透過這一節的說明，讀者可以對本系統的組成有基本的了解。

3.2.1 Node

在本研究中，Node 泛指 Drupal 中的 Node 模組內建的內容類型，以及所有繼承 Node 的內容類型，例如 2.4.1 小節中提到的 blog、poll、book 以及利用 CCK 模組所建立的內容類型，都視為 Node 的一種。

3.2.2 Ontology

在本系統中， Ontology 被定義為一個 unconnected 的有向圖形。在此圖中，每一個節點稱為一個 Concept，而邊則視為 Relation，同時邊上是具有標籤的，說明 Concept 與 Concept 間的關聯為何。

舉例而言，圖 3.1 是一個在本系統中合法的 Ontology，其中描述了 Java 與 Small Talk 分別是一種物件導向的語言；C++ 同時是物件導向語言也是程序導向語言、C 語言是程序導向的程式語言；而 Haskell 則是一種 Functional Programming 程式語言。除此之外，在此 Ontology 中，我們也標示了 C++ 具有與 C 的向下相性。

值得注意的地方，在於我們於每個邊上都明確地指定其關係：Java 和 Small Talk 與物件導向的關係為「是一種」，而 C++ 與 C 的關聯則是「向下相容」，代表 C 語言是 C++ 的子集，在 C++ 中可以出現 C 的語法，反之 C 語言無法使用 C++ 的功能。

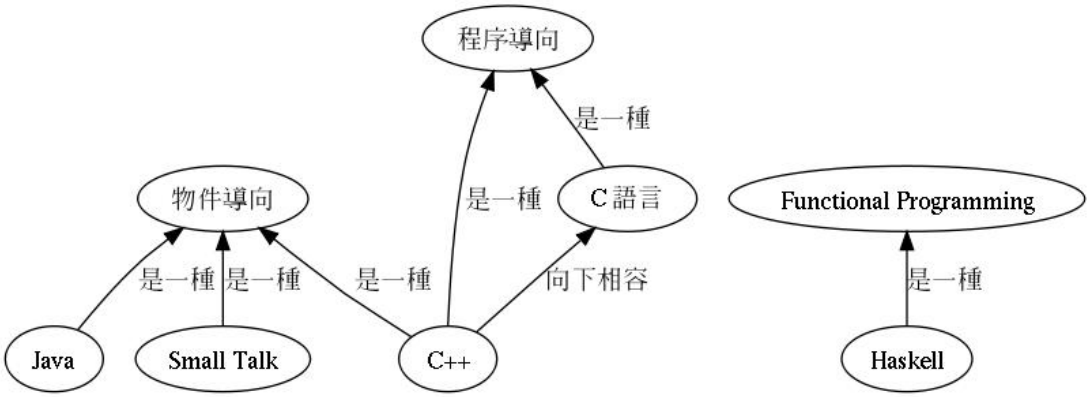

圖 3.1: 描述程式語言的 Ontology

同時讀者也應該注意到，在本例中此 Ontology 為一個 unconnected 圖形，這在我們的系統中是合法的。

3.2.3 Concept

在本系統中， Ontology 圖形上的每一個節點，稱之為 Concept，可以視為一種概念上的總稱。

Concept 是一個總體的集合，而非特定的實體，作用在於規範此 Concept 中 Instance 所應具有的屬性 (Slot)，以及與其他 Concept 之間的關聯，類似於物件導向中的「類別」的概念。關於 Slot 與 Instance 的定義我們在之後會做詳細的說明。

以第 30 頁圖 3.1 為例，各種不同的程式語言都有不同版本的編譯器，因此我們可將 Java 與 C 語言視為兩個不同的 Concept，而特定的程式語言編譯器，例如 JDK 與 GCC，則分別是 Java 與 C 語言這兩個 Concept 的 Instance。

3.2.4 Slot

為了簡化，我們在圖 3.1 中省略了 Concept 可以擁有的屬性的這一項特性，實際上每一個 Concept 可以擁有零到多個屬性，每一個屬性則稱為一個 Slot。

與 Concept 相同，Slot 並不實際存放屬性的值，而是定義諸如屬性名稱、資料

型態與最大最小值等資料，目前本系統定義 Slot 可以擁有的資料型態為字串、整數、浮點數與內容參考.....等，未來可視情況增加新的資料型態。

要特別介紹的是內容參考這項資料型態，內容參考指的是該屬性欄位會利用超鏈結的方式，指引使用者閱讀到系統內已經存在的內容資料。

透過這樣的方式，將可以增加本系統的彈性。例如雖然本系統並未支援「圖片」這種資料型態的屬性，但由於圖片在 Drupal 中也是 Node 的一種，因此使用者可以在「國家」這個 Concept 中新增「地圖」的欄位，並將其型態設為內容參考。

如此一來，往後的使用者在建立屬於「國家」此種內容類型的內容時，便可將「地圖」欄位指定成該國地圖的內容，而讀者在閱讀該篇內容時，即可透過超鏈結開啓該國的地圖。

3.2.5 Relation

本系統中，Ontology 圖形上的邊稱為 Relation，用來描述不同 Concept 之間的關聯。例如在第 30 頁圖 3.1 中，C++ 與 C 的關聯即是「C++ 向下相容 C 語言」。

值得注意的是，在本系統中，Ontology 上的邊代表的僅是概念之概念間關聯。舉例而言，在上例中我們僅指出了 C++ 可以向下相容 C 語言，但並未明確地指出哪一個版本的 C++ 編譯器可以向下相容哪個版本的 C 語言編譯器所接受的 C 語言原始碼。

至於如何表達「G++ 向下相容 GCC」（假設使用者將 GNU Compiler Collection 中的 GCC 與 G++ 視為不同的編譯器）的事實，我們將會在稍後的章節中提到。

此外，在本系統中我們內建了 is kind of 以及 extends 這兩種 Relation，這兩個 Relation 永遠同時成立，在第 30 頁圖 3.1 中，我們省略了 extends 的 Relation 線段。

3.2.6 Concept Instance

在本系統中，使用者可以將一個 Node 指定到到 Ontology 圖形內零到多個不同的 Concept 上，定義到 Ontology 上的 Node 即稱為 Concept Instance。

假設我們將第 30 頁圖 3.1 視一個程式語言編譯器的 Ontology，那麼一篇描述 JDK 1.6 的內容就是圖中 Java 這個 Concept 的其中一個 Concept Instance。

但是相對的，GNU Compiler Collection 同時支援 C 語言、C++ 與 Java，並無法只分類到其中的一項。為了解決此種問題，在本系統中，使用者可以將 Node 定義於零到多個不同的 Concept 上，因此我們可以指定 GCC 是同時位於 C 語言、C++ 與 Java 上，而 GCC 就同時是 C 語言、C++ 與 Java 這三個 Concept 的 Concept Instance。

3.2.7 Slot Instance

當使用者把 Node 定義於 Concept 上時，即會產生 Slot Instance，例如若使用者在 Ontology 上定義了「城市」這個 Concept 擁有面積、人口數等 Slot 後，當使用者創建一個新的 Node 「倫敦」，並將該 Node 定義到「城市」這 Concept 時，此時便會產生兩個 Slot Instance：倫敦的面積以及倫敦的人口數。

Slot Instance 與 Slot 的不同之處，在於 Slot 僅定義了屬性名稱與資料型態等後設資料，而 Slot Instance 則是屬性實際的值，同時 Slot Instance 不能單獨存在，必定是隨著 Node 和 Concept 的組合而定。

例如在 2.2.2 小節中提及的倫敦人口數為 7,421,328，面積為 609 平方英里的敘述，在本系統中即可用倫敦是「城市」這個 Concept 的 Instance，而「人口數」與「面積」則是「城市」這個 Concept 的 Slot 來描述。

至於實際上的屬性值 7,421,328 與 609 則分別為「城市－倫敦－人口數」以及「城市－倫敦－面積」等組合的 Slot Instance。

特別值得注意的是，由於 Slot Instance 永遠是由 Concept、Concept Instance 以

及 Slot 的組合來定義，因此 3.2.6 小節中將同一個 Node 定義於不同 Concept 的情況下，Slot Instance 也不會混淆。

例如若圖 3.1 中在 Java 與 C 語言和 C++ 均定義了「規格標準」這個 Slot，且使用者將 GCC 同時定義在 Java、C 語言與 C++ 這三個 Concept 上時，那麼總共會擁有三個 Slot Instance，分別是「Java－GCC－規格標準」、「C－GCC－規格標準」、「C++－GCC－規格標準」。

在使用者介面的設計上，系統會自動將同一個 Concept 下的 Slot 組合成同一個群組，讓使用者可以很清楚地知道哪一個屬性屬於哪個 Concept。

3.2.8 Relation Instance

在 3.2.5 小節中，我們提到了 Ontology 上 Concept 與 Concept 間的 Relation 僅代表了概念上的相關性，確並未明確指出是哪個 Concept Instance 與哪個 Concept Instance 有關。

在本系統中，使用者可以指定 Node 與 Node 間的關聯，但僅限於當兩個 Node 都被定義在 Ontology 的 Concept 上，且兩個 Concept 有邊相連的時候，此時這個關聯即稱為 Relation Instance。

舉例而言，若使用者將 GCC 定義於圖 3.1 的 C 語言這個 Concept，將 G++ 定義於 C++ 的 Concept，那麼使用者在編輯 G++ 這個 Node 時，由於 C++ 有一個「向下相容」的邊連到 C 這個 Concept，因此使用者將會在編輯介面上看到一個「向下相容」的下拉式選單，而該下拉式選單中則會出現 GCC 或其他使用者定義成 C 語言這個 Concept 的 Node，使用者即可透過這樣的選擇，表達出「G++ 向下相容 GCC」的事實。

與 Slot Instance 類似，Relation Instance 是由「Concept A－Concept Instance A－Relation－Concept B－Concept Instance B」所組成，因此將同一個 Node 定義於不同的 Concept 上並不會產生混淆。

另外本系統亦支援多對多的 Relation Instance，例如若已知 G++ 向下相容 Turbo C 與 GCC 所接受的 C 語言原始碼，而 Turbo C 與 GCC 均被定義在 C 這個

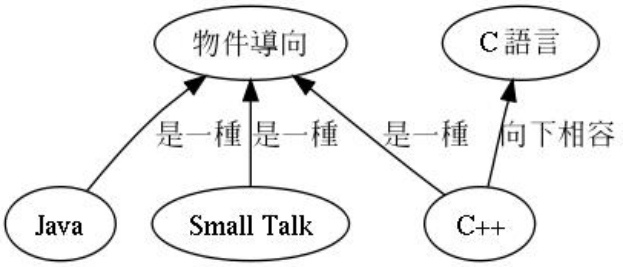

圖 3.2: 透過 OO 這個 Tag 取出的子知識領域

Concept 上，那麼 G++ 可以同時擁有兩個「向下相容」的 Relation Instance，分別指到 Turbo C 與 GCC 這兩個 Concept Instance。

3.2.9 Tag

在本系統的設計上，Ontology 是一個 unconnected 圖形，也只有一個圖形，因此本身並無將不同知識領域的 Ontology 存放在不同圖形的設計，但使用者可以將 Ontology 中的每個連通元件都視為一個獨立的知識領域。

除了上述如何界定不同的知識領域的問題外，隨著使用者的增加與 Ontology 的建立，如何讓使用者可以只針對 Ontology 中其中一部份進行處理，也會是本系統在設計上的考量之一。

為了解決這些問題，我們引入了 Tag 的概念，使用者可以建立 Tag，並且將其指定到 Ontology 中的一個 Concept 上，之後系統將會透過演算法建立出一個從該 Concept 開始的子知識領域 Ontology，而透過這個方式建立的子知識領域 Ontology 均是一個 connected 圖形。

舉例而言，當使用者建立了 OO 這個 Tag，並且將其起始點標示為第 30 頁圖 3.1 中「物件導向」這個 Concept 時，將會產生如圖 3.2 的子知識領域 Ontology。

讀者可以發現，在圖 3.2 中，C 語言依然被納入 Ontology 中，這是因為 C 語言與 C++ 具有直接的關聯，而 C++ 又是從「物件導向」這個 Concept 所衍生出的。

相對的，由於使用者是將該 Tag 標記於「物件導向」上，因此「程序導向」、Functional Programming 以及 Haskell 不會出現在子知識領域 Ontology 中。

關於本系統建立子知識領域 Ontology 的演算法詳細內容，請參閱 4.4 節。

3.3 系統架構

3.3.1 系統技術堆疊

本系統的技術堆疊如圖 3.3 所示，其中虛線框的部份為 Drupal 本身所提供的相關函式庫。

其中 Drupal Form API 提供了與 HTML 表單相關的功能，諸如建立 HTML 表單、驗證使用者輸入及呼叫適當的回呼函式處理使用者輸入等；Druapl Schema API 則提供了針對不同資料庫管理系統均通用的資料結構定義，使模組開發者不需要針對不同的資料庫管理系統撰寫不同的資料庫綱要。

在設計上，為了方便往後的擴充以及將使用者介面與內部邏輯分離，因此將系統切分為三個不同的層次，分別是 DB 處理介面、Ontology 類別函式庫與 Ontology Drupal 模組。

其中 DB 處理介面負責進行資料表外鍵處理等功能，類別函式庫針對內部的資料結構進行處理，並對最上層的 Ontology Drupal 模組提供相對應的 API，而 Ontology Drupal 模組則負責與 Drupal 核心進行溝通，並提供使用者操作介面。

3.3.2 功能元件

在 Ontology 類別庫中，我們將其細分成各個不同的元件，負責處理不同的功能，其架構如圖 3.4 所示。

其中 Concept、Slot、Relation 元件分別負責處理 Ontology 中 Concept、Slot 以及 Relation 的新增、修改、刪除，以及設定 Concept 所具有的 Slot 以及建立 Concept 之間的關聯等。

而 Instance 元件則負責建立 Node 與 Ontology 間的關係，當關係建立後，再利

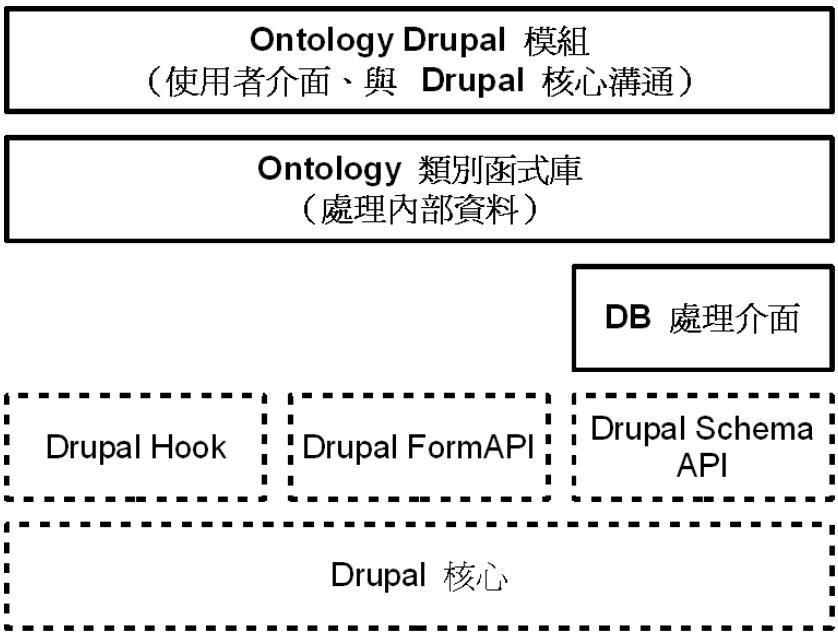

圖 3.3: Ontology 模組技術堆疊

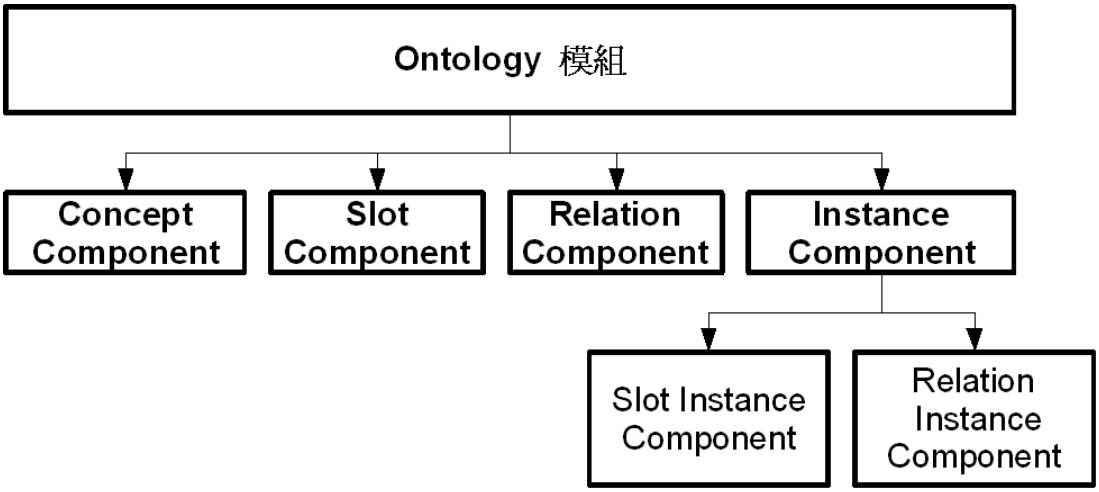

圖 3.4: 系統功能元件圖

用 Slot Instance 與 Relation Instance 元件進行諸如 Node 的後設資料的建立，或是建立 Node 與 Node 間的關聯等功能。

3.4 系統特色

3.4.1 易用性

在許多針對使用者是否採用某樣科技的模型研究中，易用性經常是模型中的變數之一，例如在 Pitt 所提出的 IS 成功模式、Fred D. Davis 於 1986 年所提出的科技接受模式以及 Kwon 與 Zmud 針對 IS 成敗影響的研究中，均將易用性納入為其評估的項目之一。

因此我們希望讓使用者可以用已經具備的網頁瀏覽器操作技巧，就可以達到建立 Ontology 與輸入屬性值等必要的動作，讓使用者不需要學習其他持別的操作方式或語法，以增加本系統的易用性。

在系統的設計與實作上，所有相關的操作均是以 HTML 表單或是配合其他 GUI 操作的方式進行，因此使用者不需要另外學習特殊的語法就可以達到建立各種 Ontology 以及內容的目的。

同時，當使用者建立一篇內容，並且選取該內容在 Ontology 中對應的 Concept 時，系統即會自動產生所需填寫的屬性的 HTML 表單，以及讓使用者選取要建立 Relation Instance 的對象。

透過這樣的方式，我們希望可以解決 Semantic Wikipedia 中使用者需要學習特殊語法，以及無法確切得知該內容應該要填寫哪些屬性的問題。

3.4.2 泛用性

在 2.4.1 小節中我們曾經提到，Drupal 核心僅內附的 Node 模組僅具最基本的內容類型，但網站管理者也可以藉由 CCK 模組的功能，在不撰寫程式的情況下建立

新的內容類型。

另一方面，目前 Drupal 社群的開發者，也已經開發出諸如處理圖片的 Image 模組、處理影像的 Video 模組，以及其他各式各樣不同的內容類型功能模組，我們希望使用者在使用本系統時，可以延用舊有的內容類型處理模組，而非針對本系統再次撰寫與設計新的內容類型。

舉例而言，若使用者想要建立一個影片資料庫，並且依據影片類型設定不同的後設資料，在本系統中，使用者只需安裝好 Drupal Video 模組與本系統，並且將 Ontology 建立後，即可將 Drupal Video 的任何內容指定到 Ontology 中的某個 Concept 上。

透過這樣的方式，我們希望能達到泛用的目的，同時使用者也可以保留原有模組建立內容的操作習慣，亦可替已經存在於系統的內容直接新增後設資料，而非從頭建立所有的內容。

在實作上，由於包括特定內容類型模組裡定義的內容類型，或使用者透過 CCK 所建立的內容類型，均繼承自 Node 這個資料表，並且具有 Node ID 這個獨一無二的屬性，因此我們只需要讓 Concept Instance 與 Node ID 具有關聯，即可達到上述支援所有內容類型的目的。

3.4.3 權限控管

Drupal 核心本身即具有完整的角色權限控管相關機制，且藉由 Drupal Hooks 的架構，外來模組可以自行定義新的權限，供系統管理者設定，模組則可以檢查使用者是否具有特定的權限，以決定接受或拒絕使用者的操作。

為了達成使用上的多樣性，我們定義了如表 3.1 的相關權限，系統管理者可以依據這些權限，調整系統的開放程度。

表 3.1: 權限列表

名稱	權限類別	對象	說明
Ontology Read	讀	Ontology	使用者是否可以讀取 Ontology
Instance Read	讀	Node	使用者是否可以讀取 Node 中與 Ontology 相關的後設資料
Concept Write	寫	Concept	使用者是否可以新增或修改 Concept
Slot Write	寫	Slot	使用者是否可以新增或修改 Slot
Relation Write	寫	Relation	使用者是否可以建立、修改、刪除 Relation
Concept Instace Write	寫	Node	使用者是否可以新增、修改、刪除 Node 所屬的 Concept
Slot Instance Write	寫	Node	使用者是否可以修改 Node 所具有的 Slot Instance
Relation Instance Write	寫	Node	使用者是否可以建立或刪除 Node 與 Node 間的 Relation Instance

第四章　系統實作

4.1　資料庫

4.1.1　資料庫實體關聯模型

圖 4.1 為本系統之資料庫實體關聯模式圖 (Crow's Foot E-R Diagram)，左側 Drupal Core 為我們所參考到的 Drupal 核心資料表，右側的 Ontology Module 區塊則是本系統新增的資料表。

以下將針對圖中的資料表以及資料表與資料表間的關係逐一進行說明：

Concepts

此資料表可對照至 3.2.3 小節 Ontology 定義中的 Concept，儲存使用者所設定的 Concept 名稱 (usTitle) 及描述 (usDesc)。

Slots

此資料表可對照至 3.2.4 小節 Ontology 定義裡的 Slot，所需要儲存的資料有 Slot 的名稱 (usTitle)、描述 (usDesc)、資料型態 (tiType)、可接受的最大 (maximal) 與最小值 (minimal)，若該 Slot 的資料型態為單選或多選列表，則還需要記錄列表內的選項 (usVList)。

SlotOfConcept

由於本系統中 Concept 與 Slot 的關係為多對多，在將資料表進行正規化後，需要額外的資料表來記錄此多對多的關係，SlotOfConcept 此資料表便是用來記錄兩者的多對多關係。

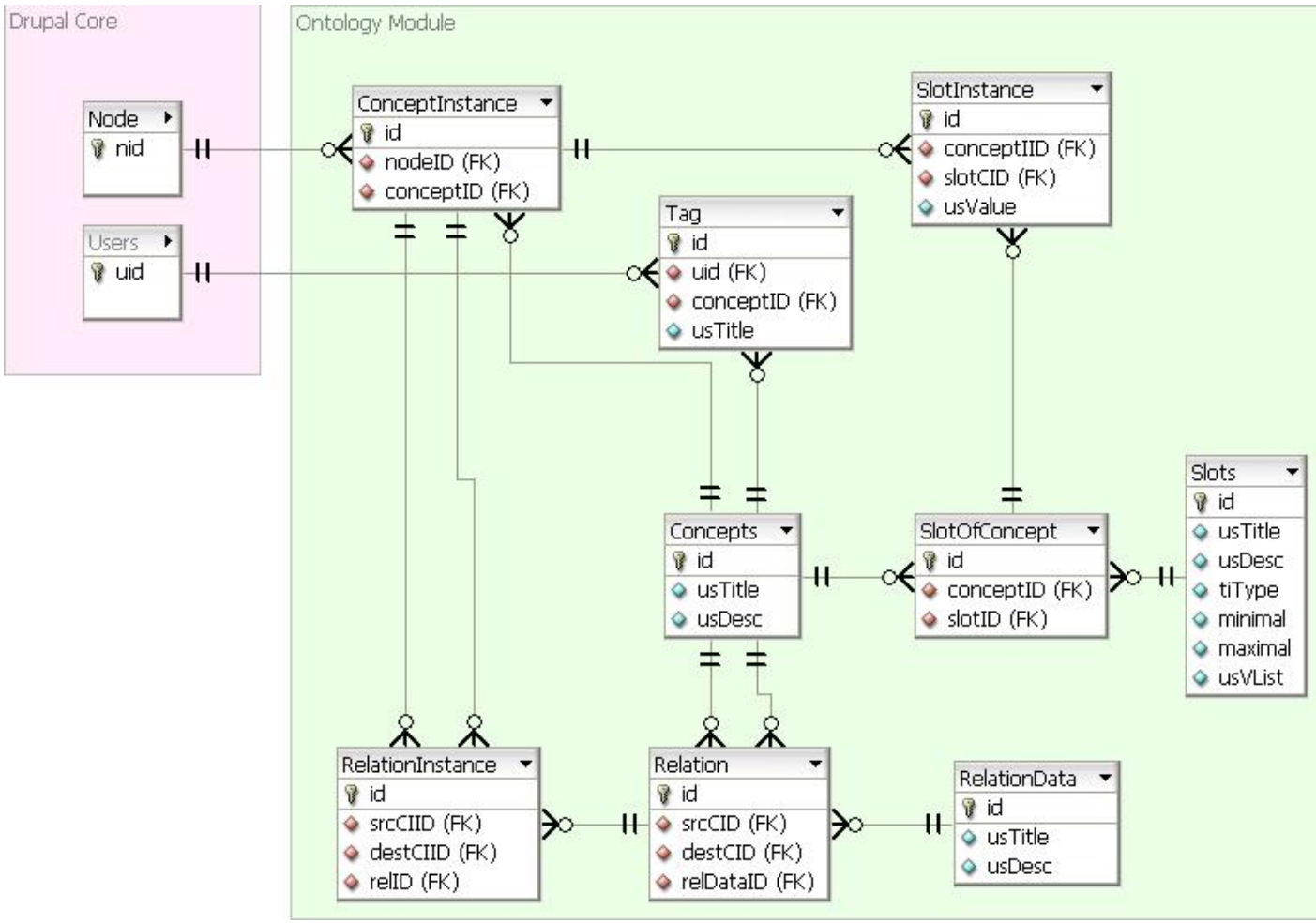

圖 4.1: 資料庫實體關聯模式圖

在這個資料表中，我們記錄了某個 Concept (conceptID) 所具有的 Slot (slotID)，而 conceptID 及 slotID 分別是參考到 Concepts 與 Slots 資料表的外鍵，此外 (conceptID, slotID) 的組合在資料表中需唯一。

RelationData

這個資料表可對照到 3.2.5 小節中 Relation 上的標籤，記錄了該標籤的名稱 (usTitle) 以及描述 (usDesc)。

Relation

此資料表可對照至 3.2.5 小節裡 Ontology 圖形裡的邊，也就是 Relation。此處所需要記錄的資料有該 Relation 的起始 Concept (srcCID)、結束的 Concept (destCID)，以及該 Relation 的標籤 (relDataID)。

要注意的是，此處的 srcCID 與 destCID 是 Concepts 資料表的外鍵，relDataID 為 RelationData 資料表的外鍵，同時 (srcCID, destCID, relDataID) 的組合在此資料表中需唯一。

ConceptInstance

此資料表用來表示 3.2.6 小節中 Ontology 定義裡的 Concept Instance，需要記錄的有該 Concept Instance 的 Node 編號 (nodeID) 以及所指定到的 Concept 的編號 (conceptID)。

在這個資料表裡，nodeID 為參考到 Drupal 核心中 Node 資料表的外鍵，且為可選的一對多關聯，也就是一個 Node 可以有零到多個 Concept Instance，而一個 Concept Instance 一定要有而且只有一個 Node。

同樣的，conceptID 為參考到 Concepts 資料表的外鍵，且亦為可選的一對多關聯。此外 (conceptID, nodeID) 的組合在資料表中需唯一。

SlotInstance

此資料表可對應到 3.2.7 小節中所定義的 Slot Instance，儲存 Concept Instance 的實際屬性值。

在這個資料表中，我們需要記錄其對應的 Concept Instance (conceptIID) 以及是哪個 Concept 的哪個 Slot (slotCID)，最後還需要記錄該屬性的值 (usValue)。

其中 conceptIID 為參考到 Concept Instance 的外鍵，而 slotCID 則為參考到 SlotOfConcept 資料表的外鍵。

RelationInstance

這個資料表用來記錄 3.2.8 小節中所定義的 Relation Instance，所需要記錄的資料有該 Relation Instance 的起始 Concept Instance (srcCIID)、目的 Concept Instance (destCIID) 以及屬於哪個 Relation (relID)。

其中 srcCIID 與 destCIID 為參考到 ConceptInstance 資料表的外鍵，relID 為參考到 Relation 資料表的外鍵，同時 (srcCIID, destCIID, relID) 的組合在此資料表中需唯一。

Tags

此資料表對應到 3.2.9 小節中所定義的 Tags，由於在系統的設計上，我們讓每個使用者所設定的 Tag 彼此有所區分，因此我們除了需要記錄該 Tag 的名稱 (usTitle)、Concept 起始點 (conceptID) 外，還需要記錄該 Tag 屬於哪個使用者 (uid)。

其中 conceptID 為參考到 Concepts 資料表的外鍵，而 uid 則是參考到 Drupal 核心使用者資料表的外鍵。

4.1.2 資料庫綱要

在上一小節中，我們針對本系統中的資料庫設計進了了概略性的討論，分析需要哪些資料表及欄位，本小節則進一步探討每個資料表實際的資料表綱要。

在實作上，為了讓本模組在不同的資料庫管理系統中都能使用，我們採用 Drupal 內建的 SchemaAPI[1] 進行建立資料表的工作，而不直接使用 SQL 命令。

表 4.1 至 4.9 為本系統內所有的資料表綱要，在表中的 serial 資料型態，是指該欄位為流水編號，相當於 MySQL 中的 auto_incerment 功能。

其中值得注意的，是在 Slots 資料表中關於 usValue 的定義，在此處 usValue 的定義為當該 Slot 的資料型態為單選或多選列表時，此欄位會存放使用者所定義的可

[1] `http://drupal.org/node/146843`

表 4.1: Concepts 資料表綱要

欄位	資料型態	可為 NULL	說明	備註
id	serial	否	流水號	主鍵
usTitle	varchar(50)	否	標題	
usDesc	text	是	描述	

表 4.2: Slots 資料表綱要

欄位	資料型態	可為 NULL	說明	備註
id	serial	否	流水號	主鍵
usTitle	varchar(50)	否	標題	
tiType	varchar(20)	否	資料型態	
minimal	varchar(20)	是	最大值限制	
maximal	varchar(20)	是	最小值限制	
usValue	text	是	候選值列表	
usDesc	text	是	描述	

選項目，以換行符號為每一個項目的分隔。

此外，在 SlotInstance 資料表中，我們採用 text 做為 usValue 欄位的資料型態，這是因為 PHP 本身是弱型別的程式語言，為了簡化資料表的複雜度，我們採用在程式內部進行型態轉換的方式，而不直接在資料表中指定不同的資料型態。

表 4.3: RelationData 資料表綱要

欄位	資料型態	可為 NULL	說明	備註
id	serial	否	流水號	主鍵
usTitle	varchar(50)	否	標題	
usDesc	text	是	描述	

表 4.4: Relation 資料表綱要

欄位	資料型態	可為 NULL	說明	備註
id	serial	否	流水號	主鍵
srcCID	integer	否	關聯來源	Concepts.id 的外鍵
destCID	integer	否	關聯目的	Concepts.id 的外鍵
relDataID	integer	否	關聯資料	RelationData.id 的外鍵

註：(srcCID, destCID, relDataID) 為 unique keys

表 4.5: SlotOfConcept 資料表綱要

欄位	資料型態	可為 NULL	說明	備註
id	serial	否	流水號	主鍵
conceptID	integer	否	Concept	Concepts.id 的外鍵
slotID	integer	否	Slot	Concepts.id 的外鍵

註：(conceptID, slotID) 為 unique keys

表 4.6: ConceptInstance 資料表綱要

欄位	資料型態	可為 NULL	說明	備註
id	serial	否	流水號	主鍵
nodeID	integer	否	Node ID	nodes.nid 的外鍵
conceptID	integer	否	Concept ID	Concepts.id 的外鍵

註：(nodeID, conceptID) 為 unique keys

表 4.7: SlotInstance 資料表綱要

欄位	資料型態	可為 NULL	說明	備註
id	serial	否	流水號	主鍵
conceptIID	integer	否	ConceptInstance ID	ConceptInstance.id 的外鍵
slotCID	integer	否	Slot of Concept ID	SlotOfConcept.id 的外鍵
usValue	text	否	屬性值	

註：(conceptIID, slotCID) 為 unique keys

表 4.8: RelationInstance 資料表綱要

欄位	資料型態	可為 NULL	說明	備註
id	serial	否	流水號	主鍵
srcCIID	integer	否	來源 ConceptInstance	Concepts.id 的外鍵
destCIID	integer	否	目的 ConceptInstance	Concepts.id 的外鍵
relID	text	否	關聯意義	Relation.id 的外鍵

註：(srcCIID, destCIID, relID) 為 unique keys

表 4.9: Tags 資料表綱要

欄位	資料型態	可為 NULL	說明	備註
id	serial	否	流水號	主鍵
uid	integer	否	使用者編號	users.uid 的外鍵
usTitle	varchar(50)	否	標題	
conceptID	integer	否	Concept ID	Concepts.id 的外鍵

4.1.3 DB 處理介面

在 Drupal 的系統要求中，並未强制要求使用者使用具有外鍵處理功能的資料庫管理系統，且 SchemaAPI 亦無提功外鍵的設定，因此在本系統中，我們必須自行處理資料表外鍵欄位。

在設計上，由於我們在每一個資料表均有一個流水編號的欄位做為主鍵，因此我們可以撰寫如程式碼 4.1 的函式，利用這個函式，我們可以取出資料表內所有外鍵為特定值的記錄。

程式碼 4.1: 取得特定外鍵欄位的記錄

```
/**
 * 取得資料表內外鍵欄位為特定值的記錄
 *
 *   @param   string    $table 要查詢的資料表
 *   @param   string    $fkField 外鍵欄位名稱
 *   @param   integer   $fkID 外鍵值
```

```
 *  @param  string  $pkField 此資料表主鍵欄位名稱
 *
 *  @return array 所有記錄的主鍵值
 */
function getFKID ($table, $fkField, $fkID, $pkField = 'id')
{
    $list = array();

    $result = db_query ("SELECT $pkField FROM $table ".
                        "WHERE $fkField=%d", $fkID);

    while ( $id = db_fetch_array($result) ) {
        $list[] = $id['id'];
    }

    return $list;
}
```

我們透過第 41 頁圖 4.1 的 ER 模式圖，可整理出刪除資料表內的記錄時，需要同時清除哪些參考到將被刪除的記錄的外鍵。

而有了上述 4.1 的函式後，系統在進行資料表內記錄的刪除時，必須依照下列的定義，先將其他資料表內參考到此筆資料的記錄先行刪除。此列表中，第一層為欲刪除的記錄所在的資料表，第二層為可能有參考至此筆資料的外鍵的資料表，括號內則為外鍵的欄位名稱。

- Concept
  - Tags (conceptID)
  - ConceptInstance (conceptID)
  - SlotOfConcept (conceptID)
  - Relation (srcCID)
  - Relation (destCID)
- ConceptInstance

  - RelationInstance (srcCIID)
  - RelationInstance (destCIID)
  - SlotInstance (conceptIID)
- Slots
  - SlotOfConcept (slotID)
- SlotOfConcept
  - SlotInstance (slotCID)
- Relation
  - RelationInstance (relID)

經過上述的處理後，我們即可確保當使用者刪除某筆記錄後，其他資料表內參考到此記錄的相關資料也會同時刪除，必免資料的不一致。

4.2 系統類別庫

4.2.1 慣例說明

由於 PHP 是屬於弱型態的程式語言，我們並沒有辦法直接從函式的原型得知每一個參數所屬的資料型態。

此外，即便是相同的資料型態，其所代表的意義也會有所不同，例如同樣是字串的資料型態，還可再分為由使用者輸入的原始資料（不安全字串），以及經過程式跳脫特殊字元後，可以直接輸出至瀏覽器的安全字串[14]。

為了進行這樣的區分，讓程式碼及函式原型更具有輔助錯誤檢查的功能，我們在類別函式庫中的函式及參數命名，在必要時會以應用匈牙利命名法命名[2]。

在此命名法中，函式或參數的開頭會有特定的字首，代表該參數所代表的分類，例如 us 開頭的參數代表不安全的字串，而經過跳脫特殊字元後的安全字串，則

[2] http://www.joelonsoftware.com/printerFriendly/articles/Wrong.html

會以 ss 開頭。表 4.10 為我們所定義的應用匈牙利命名法字首，以及其所代表的意義。

表 4.10: 函式及參數字首意義

字首	資料型態	說明
us	String	使用者輸入的原始資料字串（不安全字串）
ss	String	經過跳脫特殊字元，可輸出至瀏覽器的安全字串
usa	Array of String	不安全字串陣列
st	Slot Object	Slot 物件
sta	Array of Slot Object	Slot 物件陣列
si	Integer	Slot ID
sil	Associative Array	Key 為 SlotID，Value 為該 Slot 是否被選擇
ca	ConceptInstance Object	ConceptInstance 物件
cna	Array of ConceptInstance Object	ConceptInstance 物件陣列
cii	Associative Array	Key 為 ConceptInstance ID，Value 為 ConceptID
nta	Associative Array	Key 為 NodeID，Value 為 Node 標題
ti	String	Slot 資料型態 ID
tn	String	Slot 資料型態名稱
tna	Array of String	Slot 資料型態名稱陣列
cc	Concept Object	Concept 物件
cca	Array of Concept Object	Concept 物件陣列
cta	Array of String	Concept 標題陣列
rl	Relation Object	Relation 物件
ri	Integer	Relation ID
rla	Array of Relation Object	Relation 物件陣列
rli	Array of String	Relation 標題陣列

4.2.2 Concept 類別

在本類別函式庫中，Concept 用來處理 Ontology 內與 Concept 相關的行為，諸如建立與刪除 Concept，取得 Concept 的相關資料等。

Concept 類別提供了如表 4.11 的成員函式以及表 4.12 的類別函式。

表 4.11: Concept 成員函式

函式原型	說明
Concept __construct (int $id)	建構子
int getID ()	取得此 Concept 的 ID
string usGetDesc ()	取得此 Concept 的描述
string usGetTitle ()	取得此 Concept 的標題
Concept update (string $usTitle, string $usDesc, array $silSlots)	更新此 Concept
void deleteFromDB ()	將此 Concept 自系統中刪除
array ntaGetNodeList()	取得所有屬於此 Concept 的 Node
array staGetSlots ()	取得此 Concept 所具有的 Slot
array rlaGetSubdomain ()	取得所有指到此 Concept 的 is kind of 關聯

表 4.12: Concept 類別函式

函式原型	說明
Concept ccCreateConcept (string $usTitle, string $usDesc, array $silSlots)	建立新的 Concept
array ccaGetConceptList (int $pager = NULL)	取得系統內所有 Concept
array ctaFromCca (array $ccaConcepts)	將 Concept 陣列轉換為 Concept 標題陣列

4.2.3 Slot 類別

Slot 類別用來處理 Ontology 中的 Slot，即與 Concept 有所關聯的屬性，該類別提供了諸如建立、更改、刪除 Slot 等功能，此類別提供表 4.13 及 4.14 中的函式供使用者使用。

表 4.13: Slot 成員函式

函式原型	說明
Slot __construct (int $id)	建構子
int getID ()	取得 Slot ID
string usGetTitle ()	取得 Slot 標題
string usGetDesc ()	取得 Slot 描述
float getMaximal ()	取得此 Slot 的最大值定義
float getMinimal ()	取得此 Slot 的最小值定義
float getMaxInstance ()	取得目前此 Slot 實際擁有的最大值
float getMinInstance ()	取得目前此 Slot 實際擁有的最小值
array usaGetVList ()	取得候選值列表
void setSlotCID (int $slotCID)	設定 Slot of Concept ID
int getSlotCID ()	取得 Slot of Concept ID
Slot update (string $usTitle, string $tiType, int $minimal, int $maximal, string $usVList, string $usDesc)	更新 Slot
void deleteFromDB ()	刪除此筆 Slot
string checkInstanceValid (string $usValue)	檢查使用者所輸入的值是否符合 Slot 的規範
array createForm (string $usValue = NULL, $disabled = NULL)	建立供 Drupal 使用的 HTML 表單資料結構

表 4.14: Slot 類別函式

函式原型	說明
boolean createSlot (string $usTitle, string $tiType, mixed $minimal, mixed $maximal, stirng $usVList, string $usDesc)	建立新的 Slot
array staGetSlotList (int $pager = NULL)	取得系統內所有 Slot

4.2.4 Relation 類別

此類用來表示 Ontology 中的 Relation，Relation 所代表的義意則存放於 RelationData 中，本類別提供了如表 4.15 與 4.16 的函式可供使用。

表 4.15: Relation 成員函式

函式原型	說明
Relation __construct (int $relID)	建構子
int getID ()	取得 Relation ID
int getRelDataID ()	取得 RelationData ID
string usGetRelDataTitle ()	RelationData 的標題
string usGetRelDataDesc ()	RelationData 的描述
int getSrcCID ()	取得起點的 Concept ID
string usGetSrcCTitle ()	起點 Concept 的標題
int getDestCID ()	取得終點的 Concept ID
string usGetDestCTitle ()	終點 Concept 的標題
boolean deleteFromDB ()	刪除此 Relation

表 4.16: Relation 類別函式

函式原型	說明
boolean connectConcept (int $srcCID, int $destCID, int $relDataID)	建立 Relation
array rlaGetRelList (int $pager = NULL)	取得系統內所有 Relation

4.2.5 RelationData 類別

本類別對應於圖 4.1 資料庫實體關聯圖中的 RelationData 資料表，代表關聯的意義，提供了如表 4.17 的類別函式可供使用。

表 4.17: RelationData 類別函式

函式原型	說明
int riCreateRelData (string $usTitle, string $usDesc)	建立新的 RelationData
array rliGetTitleList ()	取得所有 RelationData ID 與標題的關聯陣列

4.2.6 ConceptInstance 類別

ConceptInstance 類別處理 Ontology 中 ConceptInstance 的相關資料，同時也做為 Ontology 模組內 ConceptInstance 與 Drupal 核心中 Node 資料結構的橋接。本類別提供了如表 4.18 與 4.19 所列的函式。

表 4.18: ConceptInstance 成員函式

函式原型	說明
ConceptInstance __construct (int $id)	建構子
array createSIForm (boolean $disabled = NULL)	建立供 Drupal 用的 Slot/Relation Instance HTML 表單資料結構
int getID ()	取得此 ConceptInstance 的 ID
int getConceptID ()	取得此 ConceptInstance 所屬 Concept 的 ID
int getNodeID ()	取得與此 ConceptInstance 相對應的 Node 的 ID
array getSlotInstanceList ()	取得此 ConceptInstance 的 Slot Instance

表 4.19: ConceptInstance 類別函式

函式原型	說明
void appendConceptForm (array &$form)	將 Concept 表單附加到現有的表單上
int ciidFromNid (int $nodeID, int $conceptID)	將 NodeID 轉成 Concept Instance ID
array ciiGetConceptOfNode (int $nodeID)	取得所有指向 Node 的 Concept Instance
array cnaFromCii (array $ciiList)	將 Concept Instance ID 陣列轉換成 Concept Instance 物件陣列
void nodeDelete (int $nodeID)	當 Node 被刪除時，將會呼叫此函式清除 Ontology 內的相關資料
void nodeInsertUpdate (int $nodeID, array $ctaConceptList)	當 Node 被修改時，會呼叫此函式更新 Ontology 內的相關資料

4.2.7 RelationInstance 類別

RelationInstance 類別提供了表 4.20 與表 4.21 的函式供使用者呼叫，此類別用來處理 Ontology 中與 RelationInstance 相關的後設資料。

表 4.20: RelationInstance 成員函式

函式原型	說明
RelationInstance __construct (int $id)	建構子
int getID ()	取得 RelationInstance ID
int getRelID ()	取得相對應的 RelationID
string usGetRelDataTitle ()	取得 RelationData 的標題
int getDestCIID ()	取得終點的 Concept Instance ID
int getSrcCIID ()	取得起點的 Concept Instance ID

表 4.21: RelationInstance 類別函式

函式原型	說明
array getCIRelIList (int $srcCID, int $relID)	取得從 $srcCID 啓始，關聯為 $relID 的 Node ID 與標題的關聯陣列
array getNodeRelInstance (int $nodeID)	取得 Node 所有的 Relation Instance
void update (int $srcNodeID, array $relInstanceList)	更新 RelationInstance

4.2.8 SlotInstance 類別

SlotInstance 類別的主要功能為設定與取得附屬於 ConceptInstance 的屬性值，本類別提供了表 4.22 所列出的成員函式。

表 4.22: SlotInstance 成員函式

函式原型	說明
SlotInstance __construct (int $conceptIID, int $slotCID)	建構子
bool update (mixed $usValue, int $usUnit, string $tiType)	更新屬性值
string usGetTitle ()	取得屬性標題
string usGetValue ()	取得屬性值

4.2.9 Tag 類別

在本系統中，使用者可以選取某個 Concept 成為 Tag，並產生由此 Concept 所衍生的子知識領域，Tag 類別即負責處理相關的行為。本類別提供了表 4.23 與表 4.24 所列的函式。

表 4.23: Tag 成員函式

函式原型	說明
Tag __construct (int $id)	建構子
string usGetTitle ()	取得 Tag 標題
array extractSubdomain ()	取得子領域
boolean delete ()	刪除此 Tag

表 4.24: Tag 類別函式

函式原型	說明
boolean addTag (string $usTitle, string $conceptID)	新增 Tag
array taGetTagList ()	取得系統內所有 Tag

4.3 Ontology 模組

4.3.1 目錄結構

在 Drupal 模組的實作規格中，僅要求模組提供模組資訊檔 (.info)、安裝檔 (.install) 以及主程式 (.main) 等三項就是一個完整的模組，並未強制要求模組內目錄的架構。

然而為了讓系統的架構更清楚，我們將程式碼切割，並依照一定的目錄架構擺放，以利後續的維護與更新。本系統的檔案架構及說明如下所示：

+ontology/ 模組主目錄

ontology.info 模組資訊檔

ontology.install 模組安裝檔

ontology.main 模組主程式

+src/ 其餘相關程式碼

constant.php 模組常數定義

db.php DB 處理介面

hooks.php Drupal Hooks 實作

load.php 自動載入物件實作

themes.php Theme 實作

utils.php 公用函式

+class/ Ontology 類別函式庫

+page/ 頁面回呼函式

+form/ 表單相關函式

其中 src/class/ 目錄內為 4.2 小節提到的類別函式庫，而 src/page/ 是特定頁面的回呼函式，當使用者瀏覽與 Ontology 模組相關的網址時，系統會呼叫此目錄內的函式，以顯示適當的畫面。

最後，src/form/ 目錄內為 Ontology 相關表單的定義函式，當使用者填寫及操作與 Ontology 相關的 HTML 表單時，系統會依照此目錄內的定義函式顯示、檢查、送出使用者所填寫的表單欄位。

4.3.2 Drupal Hooks

在本系統中，與 Drupal 的溝通方式為透過第 16 頁 2.3.3 小節中所提到的 Drupal Hooks。當我們在系統內實作特定名稱的函式後，Drupal 核心會在適當的時機呼叫這些函式，以取得所需的資料，或是執行特定的作業，以改變原有的行為或介面。

我們所實作的 Drupal Hook 以及其功用，如表 4.25 至表 4.28 所示，其中較為特別的是 ontology_nodeapi() 以及 ontology_form_alter() 這兩個函式，也是本系統的核心所在。

表 4.25: 與模組安裝與反安裝相關的 Drupal Hook 實作

Hook 名稱	檔案	說明
ontology_schema()	ontology.install	利用 SchemaAPI 定義模組所用到的資料庫綱要
ontology_install()	ontology.install	安裝模組並建立資料表
ontology_uninstall()	ontology.install	反安裝模組並刪除所有資料表

表 4.26: 與系統核心功能相關的 Drupal Hook 實作

Hook 名稱	檔案	說明
ontology_menu()	ontology.module	定義主選單、網址與頁面的對應等
ontology_perm()	src/hooks.php	定義本系統所設計的權限

表 4.27: 與內容處理相關的 Drupal Hook 實作

Hook 名稱	檔案	說明
ontology_ndoeapi()	src/hooks.php	處理與 Node 相關的動作
ontology_form_alter()	src/hooks.php	新增或編輯內容時，顯示模組定義的表單

表 4.28: 其他的 Drupal Hook 實作

Hook 名稱	檔案	說明
ontology_theme()	src/hooks.php	設定格式化相關函式
tag_load()	src/load.php	在特定網址時自動載入相對應的 Tag 物件
concept_load()	src/load.php	在特定網址時自動載入相對應的 Concept 物件
relation_load()	src/load.php	在特定網址時自動載入相對應的 Relation 物件
slot_load()	src/load.php	在特定網址時自動載入相對應的 Relation 物件

ontology_nodeapi() 的函式原型為 ontology_nodeapi(&$node, $op, $a3, $a4)，這個函式會在使用者建立、修改、刪除、讀取以及顯示 Node 時被呼叫，而 $op 此參數會被設為使用者正在執行的動作[3]。

而我們則分別針對這六個動作，增加新的處理程序，來處理 ConceptInstance 的相關資料，於是使用者在操作 Node 的同時，也等於是在操作 ConceptInstance，我們所新增的處理程序如下所示：

建立 將使用者輸入的 Ontology 後設資料新增至資料庫

修改 修改統資料庫內與此 Node 相關的紀錄

刪除 刪除資料庫內與此 Node 相關的紀錄

讀取 將資料庫內的 Ontology 後設資料附加至 Node 物件內

顯示 在畫面上顯示與此 Node 相關的 Ontology 後設資料

ontology_form_alter() 則會在系統顯示 Node 編輯表單時被呼叫，此時我們即可以將 Ontology 的相關表單附加至原有的 Node 編輯表單，讓使用者在新增或修改 Node 時，同時進行 Ontology 的相關操作。

透過實作這些函式，我們便可讓上一節所設計的系統類別庫與 Drupal 核心進行溝通，並且讓使用者透過網頁的方式進行 Ontology 後設資料的操作。

4.4 子知識領域演算法

在 3.2.9 中我們曾經提到，使用者可以透過 Tag 來縮小 Ontology 的範圍，取得子知識領域的 Ontology。在這一個小節中，我們將會說明我們在做此動作時所採用的策略與演算法。

在策略上，整個演算法分成兩個部份，第一部份使用基礎的 BFS 演算法配合上一些簡單的判斷，取出圖形中特定的子集，在這個子集中，每兩個節點之間都只會具有唯一一個方向的邊。

[3] http://api.drupal.org/api/function/hook_nodeapi/6

程式碼 4.2 即為此演算法，其中與 BFS 不同的部份，僅在於我們並未將所有的邊都考納入 BFS 的處理範圍。

在第一個節點時，我們僅將與該節點為 extend 關係的節點放入 BFS 的佇列中，因此當演算法結束時，第二層的節點與原始的節點一定是 extend 的關係。

接著，我們在選取其他的邊時，特意將 is kind of 這種 Relation 排除。如此一來，將不會尋訪到上一階層的節點，而有效地將整個 Ontology 的範圍縮小，這也是為什麼在圖 3.2 中，「程序導向」這個 Concept 不會被納入的原因。

程式碼 4.2: 子領域演算法

```
/**
 * Ontology 子領域演算法 (BFS Part)
 * Input:  Concept.
 * Output: A set of arc.
 */
function subdomain ($concept)
{
    $queue = new Queue();

    // 最後回傳之 set of arc.
    // 預設的邊：所有 $concept 所有標為 extend 的邊
    $arcList = $concept->getExtendPath();

    // 將此 Concept 標示為已來訪
    $concept->vistied = true;

    // BFS 初始化：僅處理由 $concept extend 出的 Vertex
    foreach ($arcList as $relation) {
        $queue->enqueu ($relation->to);
    }

    // BFS 演算法
    while ( !$queue->empty ) {
```

```
        $vertex = $queue->dequeue ();

        if ( $vertex->visted ) {
            continue;
        }

        $vertex->visted = true;

        // 取得從 $vertex 往外走，且不為 is kind of 的 relation
        $arcs = $vertex->getArcExcludeIsKindOf ();

        foreach ($arcs as $arc) {
            $arcList[] = $arc;
            $queue->enqueue ($arc->toVertex);
        }
    }

    return $arcList;
}
```

由上述第一部份所產生的圖形，每兩個節點之間均只有一個方向的邊，但例如 extend 這種類型的邊，應該要有相反方向的 is kind of 的邊，因此第二部份的演算法就是在重建此類對稱式的關係。

在這個演算法中，我們僅針對由 subdomain() 所產生的 arcList 進行處理，方式也非常簡單，僅需查詢資料庫中是否有相反方向的邊，若有則將其加入 arcList 中即可，程式碼 4.3 為此部份的演算法。

程式碼 4.3: 重建反向邊演算法

```
/**
 * Ontology 重建反向邊演算法 (Rebuild Reverse Arc Part)
 * Input:  A set of arc.
 * Output: A set of arc.
 */
```

```
function rebuildReverseArc ($arcList)
{
    // 掃過所有 $arcList 中的每一個 arc
    foreach ( $arcList as $arc ) {

        // 找尋相反方向的邊
        $reverse = $arc->reverseArc();

        // 若有相反方向的邊則將其加入 $arcList 中。
        if ($reverse != NULL) {
            $arcList[] = $reverse;
        }
    }

    return $arcList;
}
```

透過以上兩個演算法，我們可以很輕易地建立起由某個特定 Concept 所開始的子知識領域，減少使用者所需要觀注的 Ontology 範圍。

第五章　系統展示

本章我們將簡單地介紹本系統實際操作時的狀況，並以此說明，我們如何在不更改使用者原有內容建立操作習慣的情況下，於系統內建立屬性及關聯的後設資料。

5.1　Ontology 操作

5.1.1　系統概觀

使用者安裝本模組後，會於左側的選單出現一 Ontology 選項，供使用者操作 Ontology，而當使用者點選該選單時，本系統會將使用者已建立之 Ontology 以圖形化方式呈現，供使用者參考，系統畫面如圖 5.1 所示。

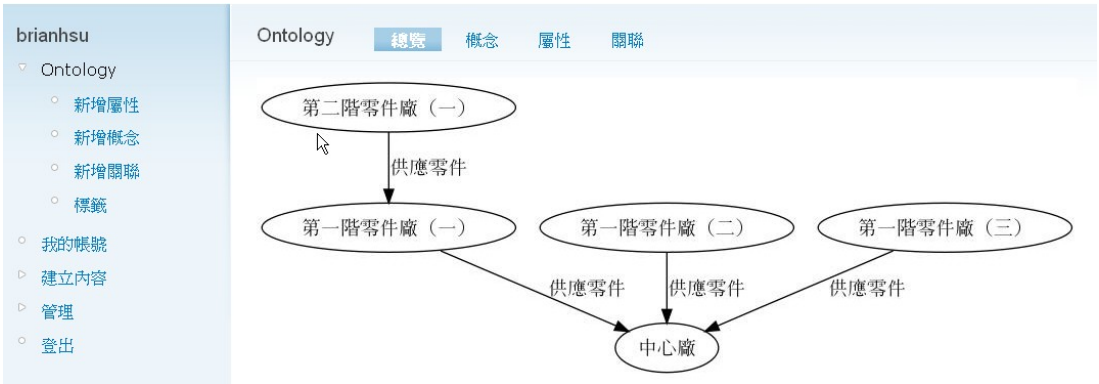

圖　5.1: 系統內 Ontology 概觀圖

5.1.2 新增 Slot

當使用者需要增加系統內的 Slot 時，只需於畫面左側點選「新增屬性」即可，本系統會呈現出新增 Slot 的 HTML 表單，供使用者填寫。使用者可於此表單內設定欲新增 Slot 的資料類型以及最大最小值等。

在圖 5.2 的範例例中，我們新增了一「零件工作溫度」的 Slot ，其資料類型為溫度，無最大最小值之限制。

圖 5.2: 新增 Slot

5.1.3 新增 Concept

當使用者新增完所需使用的 Slot 時，即可以於側的選單點選「新增概念」，以建立 Ontology 中新的 Concept。此時系統會顯示新增 Concept 的表單，使用者可以填寫新 Concept 的說明，以及選取此 Concept 所需使用的 Slot，如圖 5.3 所示。

當新的 Concept 被建立後，在 Ontology 總覽圖中，會立刻出現此新建立的 Concept 節點，如圖 5.4 所示。

圖 5.3: 新增 Concept

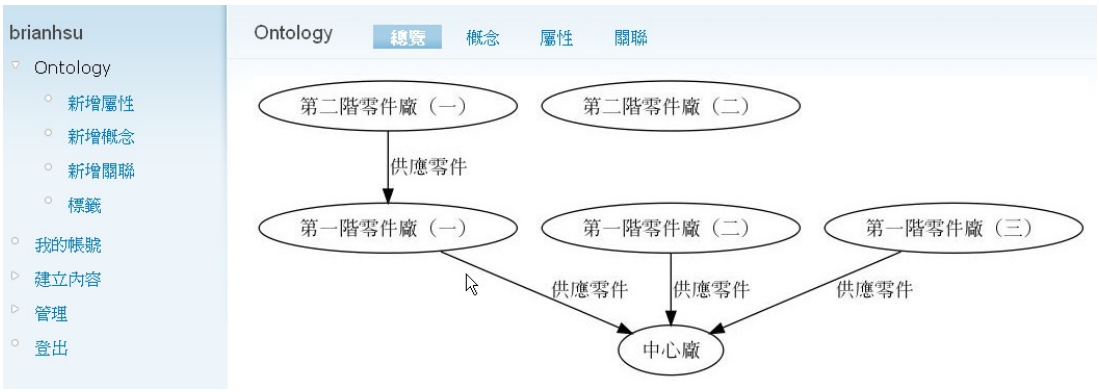

圖 5.4: 新的 Ontology 概觀

5.1.4 新增 Relation

使用者新增 Concept 後，可以利用左側選單的「新增關聯」選項，替系統內的 Ontology Concept 建立 Relation，操作畫面如圖 5.5 所示。在此例中，我們將上一小節建立的「第二階零件廠（二）」此 Concept，與「第一階零件廠（二）」建立 Relation，Relation 意義為「供應零件」，表示第二階零件廠（二）會供應零件給第一階零件廠（二）。

當使用者建立此 Relation 後，系統內的 Ontology 概觀圖也會改變，將兩個 Concept 節點以線段連接，並標示出其方向性，如圖 5.6 所示。

圖 5.5: 新增 Relation

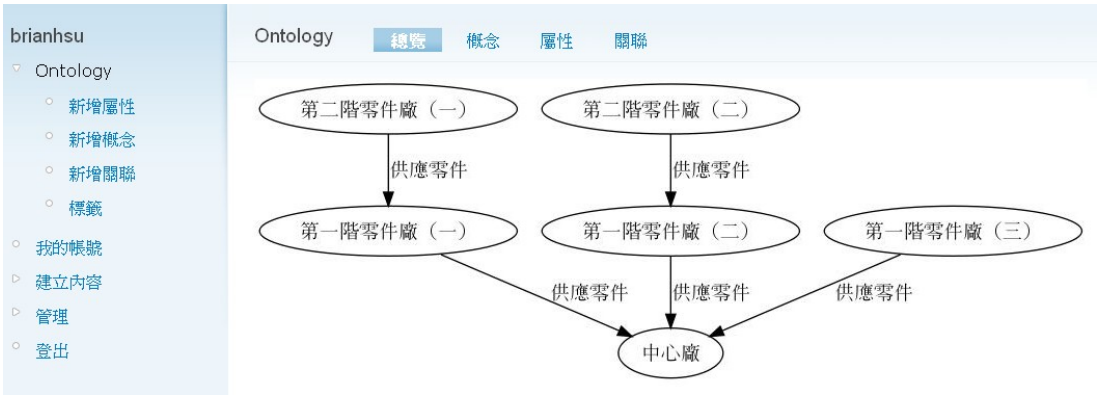

圖 5.6: 新的 Ontology 概觀

5.2 內容與後設資料操作

當使用者建立起系統內的 Ontology 後，即可針對系統內的內容與 Ontology 相互關聯做處理。在這一節中，我們將展示如何新增不同內容類型的文章，並將其指定到同一個 Conceptt 中。

5.2.1 建立 Concept Instance

首先，使用者可以依照 Drupal 原有的操作習慣進行內容的新增，此時在原有的新增內容表單中，將會插入 Ontology 的相關選項，供使用者選擇要指定的

Concept。

在圖 5.7 的範例中，我們建立一筆內容類型為 Page 的文章，並將其指在「第二階零件廠（二）」上，成為一 Concept Instance。

圖 5.7: 建立 Page 內容與選取 Concept

圖 5.8 則是新增 Stroy 此內容類型的畫面，在此處我們亦可看到，不論內容類型為何，系統出現的 Concept 表單都是相同的，使用者可以把任何的內容類型的內容指定到同一個 Concept 上。

圖 5.8: 新增 Story 內容並指定其 Concept

5.2.2 瀏覽後設資料

當使用者建立內容並指定其 Concept 後，該篇內容在呈現時，即會一併呈現 Ontology 內關聯及屬性的後設資料。

以圖 5.9 為例，由於「第二階零件廠（二）」此 Concept 具有「供應零件」、「出貨量」以及「工作溫度」等三項後設資料，因此使用者在瀏覽上一小節所建立的內容時，即會出現這三個欄位。

圖 5.9: 瀏覽後設資料

5.2.3 設定後設資料

具有更改後設資料權限的使用者，可以利用上方的 Ontology 標籤，進入後設資料的填寫表單，並修改相關的後資料，如圖 5.10 所示。

更改後設資料後，下次再瀏覽此筆內容時，下方與 Ontology 相關的後設資料即會顯示更新過後的資料，如圖 5.11 所示。

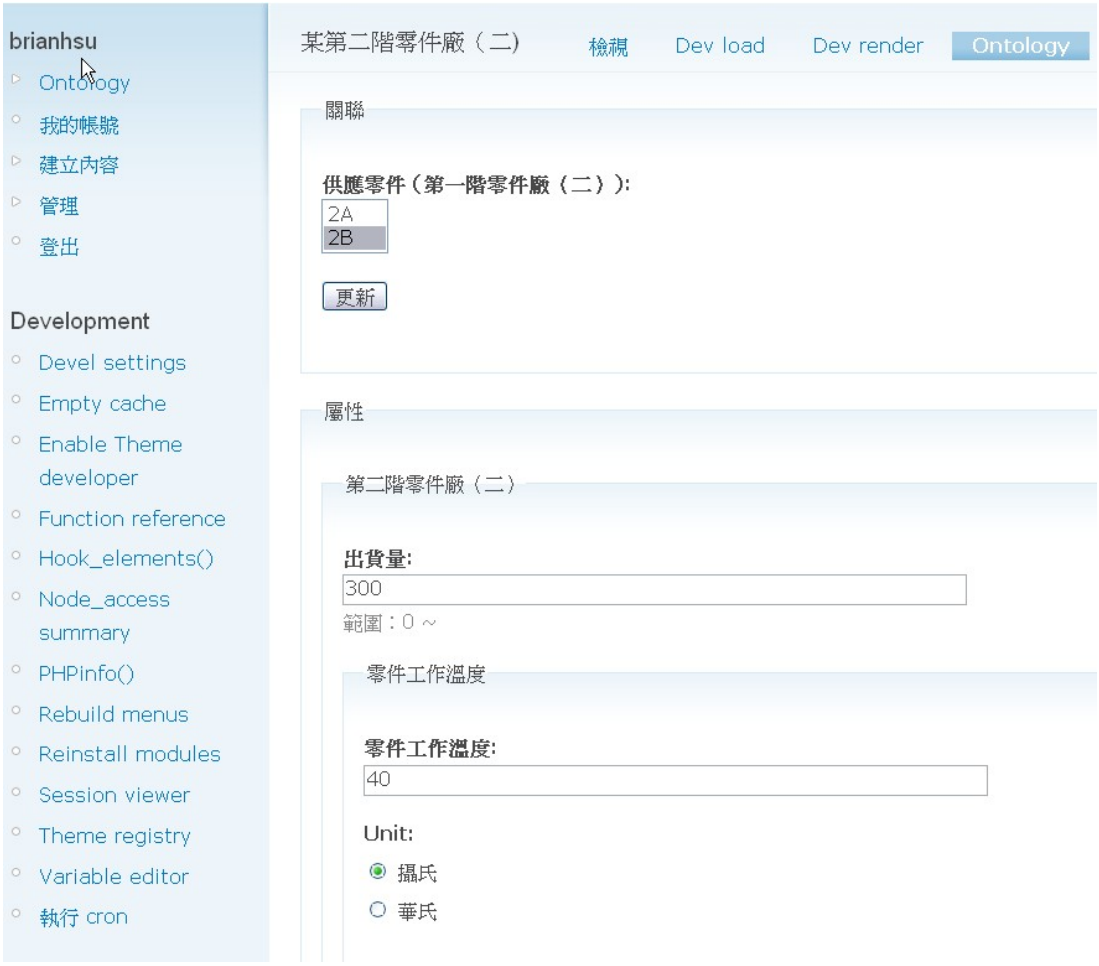
brianhsu
Ontology
我的帳號
建立內容
管理
登出
Development
Devel settings
Empty cache
Enable Theme developer
Function reference
Hook_elements()
Node_access summary
PHPinfo()
Rebuild menus
Reinstall modules
Session viewer
Theme registry
Variable editor
執行 cron
某第二階零件廠（二）
檢視
Dev load
Dev render
Ontology
關聯
供應零件（第一階零件廠（二））:
2A
2B
更新
屬性
第二階零件廠（二）
出貨量:
300
範圍：0 ~
零件工作溫度
零件工作溫度:
40
Unit:
攝氏
華氏

圖 5.10: 設定後設資料

圖 5.11: 瀏覽經設定後的後設資料

第六章　結論

6.1　研究成果與貢獻

在本研究中，我們提出了以 Ontology 解決目前內容管理系統針對後設資料處理不足的問題，並且實做了一套 Drupal 的延伸模組，以證明此概念的可行性。

與 Semantic Wikipedia 相較，我們在後設資料的輸入與建立，係採用 HTML 表單的方式，因此使用上較為簡單易懂。同時由於我們增加了概念的層次，所以能夠解決在 2.2.5 小節中所提到的「台北是倫敦的首都」此類明顯不符合概念的事實的問題。

而與 Drupal 所內建的 Taxnonmy 以及第三方所提供的 CCK 模組相比，我們可以更加精確地描述關聯的意義，同時在泛用性上也比 CCK 來得好。

表 6.1 與 6.2 為本模組與上述三套系統的比較表，從表中可以看出，我們解決了其餘三套系統大部份的問題。

最後，為了讓本模組能夠持續發展，我們將此模組採用 GNU GPL 授權合約釋出，放置在 Google Code 網站上[1]，供其他使用者下載使用，並提供了相關的文件。此外，我們目前也正在申請將此模組放在 Drupal 官方網站的模組列表中。

[1] `http://code.google.com/p/simpleontology/`

表 6.1: 各系統屬性處理方式

系統	輸入方式	優點	缺點
Semantic Wikipedia	內嵌於文章原始碼	• 彈性大 • 不需考慮內容類型 • 具單位轉換功能	• 使用者需學習特殊語法 • 無法提供使用者應輸入屬性列表 • 屬性具有同名異義等問題
Drupal 模組	由模組開發者決定	• 彈性大	• 需額外撰寫程式
Drupal CCK 模組	HTML 表單	• 設定簡單 • 輸入容易	• 屬性組合與內容類型相關
本系統	HTML 表單	• 設定簡單 • 輸入容易 • 可套用至任何的內容類型 • 支援單位換算	

表 6.2: 各系統關聯處理方式

系統	輸入方式	優點	缺點
Semantic Wikipedia	內嵌於文章原始碼	• 彈性大 • 可描述內容與內容間實際的關聯	• 使用者需學習特殊語法 • 屬性具有同名異義等問題
Drupal Taxonomy 模組	HTML 表單	• 操作介面容易 • 可定義同意字詞	• 無法確切表達 Related Terms 之間的關聯意義 • 僅能描述分類間的關聯，而無法描述內容間的關聯
本系統	HTML 表單	• 操作介面容易 • 視覺化呈現 • 可描述關聯間的意義	

6.2 未來研究方向

由於時間上的限制，本系統僅針對 Ontology 的編寫部份進行實作，尚未針對透過 Ontology 對後設資料的搜尋以及推論功能進行研究與實作。

在未來的研究方向上，我們希望能夠利用這套系統，架設產業資料庫並進行實驗，以驗證我們所提出的架構，是否能夠被實際的應用。

此外，我們也希望透過這套系統以及產業資料庫內所建立的資料，進行推論引擎的設計與實作，提供使用者一個更具有輔助及參考價值的系統。

參考文獻

[1] Paul Browning and Mike Lowndes. Content management system. *JISC TechWatch Report*, September 2001.

[2] Klaus Svarre. *What is content management system?* `http://searchsoa.techtarget.com/sDefinition/0,,sid26_gci508916,00.html`.

[3] 中文版維基百科. *Wiki*. `http://zh.wikipedia.org/wiki/Wiki`.

[4] Ward Cunnigham. *The wiki way: quick collaboration on the Web*. Addison-Wesley, 2001.

[5] Ward Cunnigham. *Wiki Design Principles*. `http://c2.com/cgi/wiki?WikiDesignPrinciples`.

[6] 陳立華、徐建初. Wiki:網路時代協同工作與知識共享的平臺. 中國信息報導, 1:51 – 51.

[7] Wiki. *Encyclopaedia Britannica*, 2007.

[8] Max Volkel. Semantic wikipedia. In *Proceedings of the 15th international conference on World Wide Web*, pages 585 – 594, 2006.

[9] Drupal.org. *About Drupal*. `http://drupal.org/about`.

[10] John K. VanDyk and Matt Westgate. *Pro Drupal Development*. Apress, 2007.

[11] Asuncion Gomez Perez. Overview of knowledge sharing and reuse components: Ontologies and problem-solving methods. In *IJCAI-99 workshop on Ontologies and Problem-Solving Methods*, 1999.

[12] T. Finin T. R. Gruber T. Senator R. Neches, R. E. Fikes and W. R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36 – 56, 1991.

[13] VR Benjamins B Chandrasekaran, TR Johnson. Ontologies: What are they? why do we need them. *IEEE Intelligent Systems and Their Applications*, 1999.

[14] Joel Spolsky. Making wrong code look wrong. *Joel on Software*, May 2005. `http://www.joelonsoftware.com/printerFriendly/articles/Wrong.html`.