以 Ontology 為基礎之內容管理系統

許洛豪
國立暨南國際大學　資訊管理學系
brianhsu.hsu@gmail.com

俞旭昇
國立暨南國際大學　資訊管理學系
ssyu@ncnu.edu.tw

摘要

Drupal 為一開放源碼的內容管理系統，可以做為諸如社群入口網站、數位典藏及知識分享等應用，同時由於核心架構設計上的彈性，相當容易透過撰寫模組以加強網站功能。

有鑑於目前市面上內容管理系統對於後設資料及資料關聯性表達能力的不足，本研究設計了一套 Drupal 的 Ontology 延伸模組，使用者可以透過此套模組，定義適用於任何資料型態的 Ontology，以表達內容的後設資料與相關性。

此外，我們亦提出了一套演算法，讓使用者可以從龐大的 Ontology 圖形中，截取其中的一個連接元件做為子知識領域，有效減少使用者所需要處理的資料量。

關鍵字：內容管理系統、Ontology

以 Ontology 爲基礎之內容管理系統

摘要

Drupal 爲一開放源碼的內容管理系統，可以做爲諸如社群入口網站、數位典藏及知識分享等應用，同時由於核心架構設計上的彈性，相當容易透過撰寫模組以加強網站功能。

有鑑於目前市面上內容管理系統對於後設資料及資料關聯性表達能力的不足，本研究設計了一套 Drupal 的 Ontology 延伸模組，使用者可以透過此套模組，定義適用於任何資料型態的 Ontology，以表達內容的後設資料與相關性。

此外，我們亦提出了一套演算法，讓使用者可以從龐大的 Ontology 圖形中，截取其中的一個連接元件做爲子知識領域，有效減少使用者所需要處理的資料量。

關鍵字：內容管理系統、Ontology

1. 研究目的

本研究的主要目的，在於建立一套 Drupal 內容管理系統的延伸模組，讓使用者可以在建立內容時填寫後設資料，並且表達內容與內容之間的關聯，以解決目前市面上內容管理系統對於後設資料處理及語意表達支援不足的缺點。

在使用上，本系統設計成當使用者建立 Drupal 中任何內容型態的資料時，系統會自動提示需要輸入的屬性值，方便使用者進行後設資料的建立。在實作方面，我們採用了 Ontology 的概念，並且使 Ontology 與內容型態分離，因此 Drupal 中任何內容型態的資料，都可以指定成 Ontology 上的任何一個 Concept。

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 系統內建的 Taxonomy 分類模組及 Content Construction Kit 內容型態設計模組所具有的缺點。

關於本系統做爲開發平台的 Drupal 內容管理系統以及所採用的 Ontology 架構，則會在之後的章節中進行詳細的說明。

2. Drupal 簡介

2.1. Drupal 簡介

Drupal 是一套開放源始碼的內容管理系統，提供了完整的角色權限控制以及多樣化的功能模組，根據其官方網站說明，Drupal 適合應用於下列的網站架設：[1]

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

Drupal 採用了模組化的概念，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。核心模組是諸如使用者管理等網站的基礎功能，若使用者需要增加網站的功能，則可以安裝其它延伸模組。[2]

在實際使用上，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，同時亦有正體中文的 Drupal Taiwan 社群成立，提供了技術文件、翻譯與討論等。[1]

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

2.2. 技術堆疊

由於本系統爲一 Drupal 的延伸模組，因此在這一小節中，我們將會簡單地介紹 Drupal 的設計理念以及系統架構，以幫助讀者了解本系統的整體架構。

Drupal 在設計時，就將客製化列爲目標之一，而客製化的方式是採用覆寫核心模組與增加新的模組，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。[2]

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 所提供的 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 1 爲一個 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。

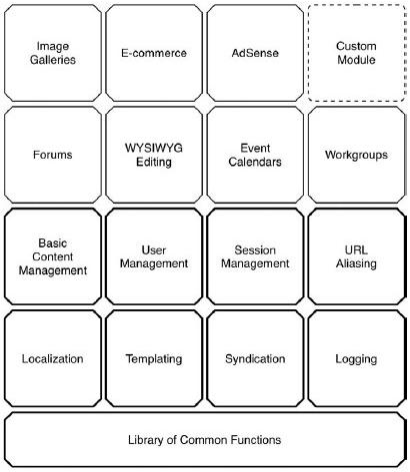

圖 1: Drupal 網站模組堆疊 [2]

在技術方面，Drupal 採用了 PHP 做爲程式語言，且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、PostgreSQL 等各種不同的資料庫管理系統。

圖 2 爲 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。

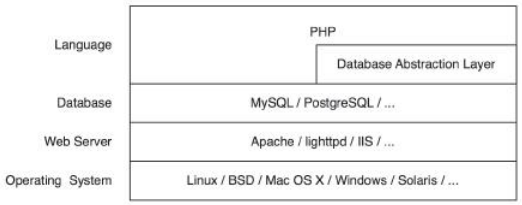

圖 2: Drupal 技術堆疊 [2]

我們的目標是設計一套 Drupal 的模組，以解決目前內容管理系統對於語意及後設資料的支援尚不完整的問題。

爲了說明爲何要開發這套模組，以及如何實現這個架構，在接下來的章節中，我們將會簡單介紹一些 Drupal 裡已經被經常使用的模組，以及這些模組尚有哪些缺失，無法解決我們所提出的問題。

2.3. Drupal Hooks 簡介

Drupal 採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，此種函數稱爲 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。

舉例而言，在本系統中，我們設計了一個 ontology_nodeapi() 函數，將文章的後設資料呈現在螢幕上，在本模組未被啓用下，使用者是無法

看見這些資訊的。但是當使用者將本模組啓用後，Drupal 會在使用者讀取文章時，呼叫本模組內的 ontology_nodeapi() 函數，而此函數會將文章的後設資料顯示於畫面上，讀者即可閱讀到這些後設資料。

2.4. 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 核心中最基礎的內容管理系統模組，大部份的進階內容管理延伸模組，均是透過延伸此模組或是覆寫其中的功能，以達成其所設計的功能。[2]

在 Node 模組中，Node 是最基本的內容型態，其中僅定義了如 ID、標題與內文等相關資料，做爲父類別。其他進階的內容型態，例如部落格文章、投票與線上文件手冊等容型態均繼承自 Node，如圖 3 所示。

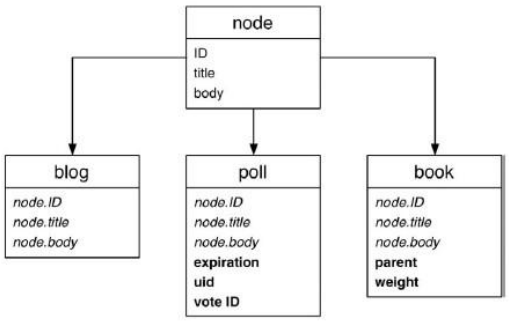

圖 3: Node 與其子類別 [2]

在實作上，開發者可以自行撰寫 Drupal 延伸模組，並且透過 Drupal Hooks 定義新的內容型態及相關操作。而在資料表的設計上，只需新增一 nodeID 的欄位，並且將其設爲參考到 Node 資料表中 ID 欄位的外鍵即可。

雖然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資料型態，但此種方式的缺點在於每次建立新的資料型態，都得撰寫額外的程式碼，無法讓不會寫程式的使用者建立新的內容型態。

2.5. 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型態，也提到了此種方式的缺點，接下來我們將介紹另一套爲了解決這個問題而開發的模組。

Content Construction Kit（簡稱 CCK）是一套 Drupal 的延伸模組，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型態，至於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型態，並且設定該內容型態除了標題與內文外所需要的其它欄位。

舉例而言，若我們新增了一種內容型態『中央處理器簡介』，以做爲產品資料庫所用，那麼我們除了標題以及內文之外，還需要諸如製造商、時脈等欄位供讀者參考。

在 CCK 模組中，系統管理者只需在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爲必填欄位等資訊即可。圖 4 即爲建立處理器時脈欄位的畫面，在此例中我們選擇資料型態爲整數，輸入方式爲文字方塊。

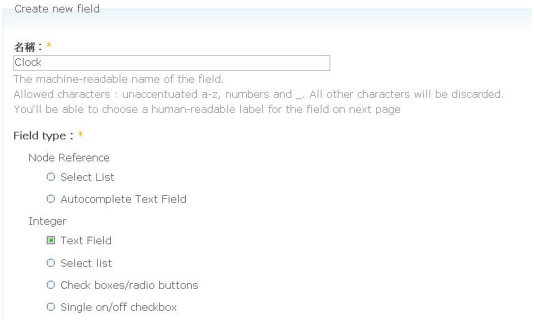

圖 4: 利用 CCK 建立新的欄位

設定完成後，當使用者新增一筆內容型態爲『處理器簡介』的內容時，除了標題、內文等欄位外，我們透過 CCK 新增的欄位，例如製造商與時

脈等欄位也會顯示在表單中，供使用者填入適當的值，如圖 5 所示。

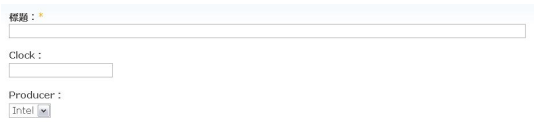

圖 5: 由 CCK 所建立的內容型態與新增的欄位

然而利用透過 CCK 建立新的資料型態，雖然可以讓使用者不需要撰寫程式即可定義需要的欄位，但 CCK 亦有其缺點存在。

由於 CCK 所設定的欄位是跟隨內容型態，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型態，當組合增加時，內容型態也會隨之增加，將會導至系統不具實用性。

2.6. Taxonomy 模組

在上述兩小節中，我們簡單地介紹如何在 Drupal 中處理後設資料，但尚未提到如何表達內容與內容間的關聯，以及如何將內容分類，因此接下來我們將會介紹在 Drupal 中如何組織已存在的內容。

Taxonomy 模組是 Drupal 核心所採用的分類系統，透過這套系統，使用者可以選擇採用扁平、階層以及多重階層等三不種同類型的分類方式來組織內容，以下將分別敘述三種分類方式的不同點。

在扁平式的分類方式中，所有的分類沒有任何階層關係，是獨立存在的；階層式則是子分類僅屬於一個父分類，而在多重階層的分類關係中，一個子分類則可以屬一個以上不同的父分類。圖 6 與 7 分別爲階層與多重階層兩種不同的分類方式的示意圖。[2]

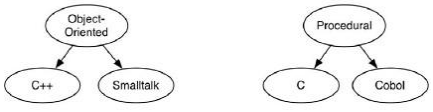

圖 6: 階層式分類法 [2]

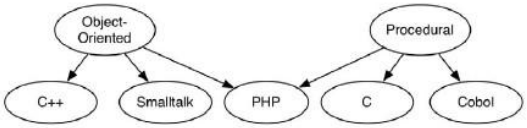

圖 7: 多重階層式分類法 [2]

在圖 6 與圖 7 中，所有分類的名稱，如 Object-Oriented、PHP 等均稱爲一個詞彙。此外我們也可以發現，由於 PHP 既可以歸類爲物件導向程式語言，亦可歸類爲程序導向程式語言，若透過多重階層的分類方式，則可以適當地表達這種概念。

除了分類方式外，使用者亦可設定哪種內容型態與此分類是有關的，以及該內容型態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，即可選擇該內容屬於哪一個分類中。

例如一個『IDE 開發環境』的內容，可以依照其支援的程式語言歸類，而當使用者將程式語言的分類建好後，可以直接在表單中選擇此篇內容所提及的 IDE 開發環境是支援哪些程式語言，如圖 8 所示。此後使用者即可透過每一種程式語言的分類找尋到適當的 IDE 開發環境資料。

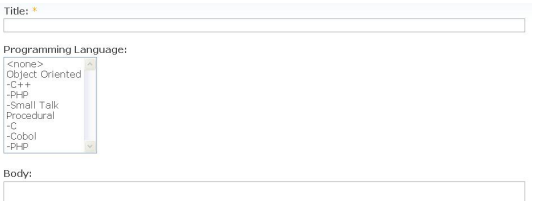

圖 8: 使用者可以選擇已建立好的分類

除了階層式的架構外，Taxonomy 亦支援相關詞彙，使用者可以將相關詞彙視爲類似字典或 API 文件中的 See Also 項目。

舉例而言，在圖 7 程式語言的分類中，我們知道 C++ 是由加強 C 語言而來，因此我們可以設定 C++ 的相關詞彙爲 C 語言。

然而 Taxonomy 中相關詞彙於並無法清楚表達詞彙之間的關聯，例如上述的範例，我們僅知道 C 語言與 C++ 兩者之間是具有關聯的，但卻無法準

確地得知兩者間的關係。

3. 系統簡介

在這一節中，我們將會對本系統的元件組成及架構進行介紹，並且針對所使用到的專有名詞進行較爲嚴謹的定義，讓讀者了解本系統的設計原理及架構，並且解釋爲何透過這樣的設計，可以解決在 2.5 及 2.6 小節中提及的 CCK 模組與 Taxonomy 模組所具有的缺點。

同時，在這一節中我們也會對如何讓使用者在協同編輯環境下，Ontology 不斷擴大的同時，可以只觀注自己有興趣的子知識領域進行介紹。

3.1. Ontology

在本系統中，Ontology 被定義爲一 unconnected directed graph，在此圖中，每一個節點稱爲一個 Concept，邊則被視爲 Relation，同時邊上是具有標籤的，說明 Concept 與 Concept 間的關聯爲何。

舉例而言，圖 9 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 的向下容相性。

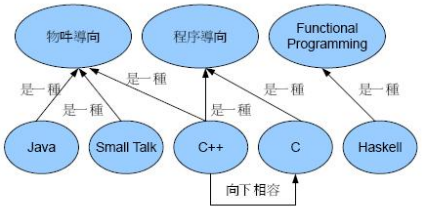

圖 9: 描述程式語言的 Ontology

其中值得注意的地方，在於我們於每個邊上都明確的指定了其關係，Java 和 Small Talk 與物件導向的關係爲『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。

透過在邊上加註其意義，我們可以解決 Drupal Taxonomy 模組相關詞彙功能無法準確描述其意義的缺點。

3.2. Concept

在本系統中，Ontology 圖形上的每一個節點，稱之爲 Concept，可以視爲一種概念上的描述。Concept 是一種總體的集合，而非特定的物體，類似於物件導向中的『類別』，僅定義了屬於此 Concept 的 Instance 具有哪些屬性。

以圖 9 爲例，各種不同的程式語言都有不同的編譯器，因此我們可將 Java 語言、以及 C 語言視爲 Concept，而等定的程式語言的編譯器，則是屬於該 Concept 下的 Instance。

3.3. Slot

爲了簡化，我們在圖 9 中省略了 Concept 可以擁有的屬性的這一項特性，在本系統中每一個 Concept 可以擁有零到多個屬性，每一個屬性稱爲一個 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爲字串、整數、浮點數、列表與內容參考等，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，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，將其型態設爲內容參考。往後的使用者在建立屬於『國家』這個 Concept 的內容時，便可將『地圖』欄位指定成該國地圖，讀者在閱讀該篇內容時，即可透過超鏈結開啓該國的地圖。

3.4. Relation

本系統中，Ontology 圖形上的邊稱爲 Relation，描述不同 Concept 之間的關聯，例如在圖 9 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 上的邊代表的是概念之概念間關聯，舉例而言，在上圖中我們僅指出了 C++ 可以向下相容 C 語言，但並未明確的指出哪一個版本的 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++ 視爲不同的編譯器）的事實，我們將會在稍後的章節中提到。

在本系統中，內建了 is kind of 以及 extends 這兩種 Relation，這兩個 Relation 永遠同時成立，在圖 9 中，我們省略了 extends 的 Relation 線段。

3.5. Node

在本研究中，Node 泛指 Drupal 中的 Node 模組內建的內容型態，以及所有從 Node 繼承的內容型態，例如 2.4 中提到的部落格文章、投票、線上手冊文件以及利用 CCK 模組所建立的內容型態，都視爲 Node 的一種。

3.6. Concept Instance

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

假設我們將圖 9 視爲一個程式語言編譯器的 Ontology，那麼一篇描述 JDK 1.6 的文章就是圖中 Java 這個 Concept 的其中一個 Concept Instance。

但是相對的，GNU Compiler Collection 同時支援 C、C++ 與 Java，並無法只分類到其中的一項。爲了解決此種問題，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 同時位於 C、C++ 與 Java 這三個 Concept 上，而 GCC 就同時是 C、C++ 與 Java 這三個 Concept 的 Instance。

3.7. 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。

而 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相關資料，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。

特別值得注意的是，由於 Slot Instance 永遠是由 Concept、Node 以及 Slot 來定義，因此 3.6 中將 Node 定義於不同 Concept 的情況下，Slot Instance 也不會混淆。

例如倘若在圖 9 中，Java、C 語言和 C++ 均定義了『規格標準』這個 Slot，而當使用者將 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屬性欄位，分別是『Java－GCC－規格標準』、『C－GCC－規格標準』、『C++－GCC－規格標準』。

而在使用者介面上，系統會自動將同一個 Concept 下的 Slot 組合成同一個群組，使用者可以很清楚地知道哪一個屬性是屬於哪個 Concept。

3.8. Relation Instance

在 3.4 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，但並未明確指出是哪個 Instance 與哪個 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。

舉例而言，若使用者將 GCC 定義於圖 9 的 C 這個 Concept，將 G++ 定義於 C++ 的 Concept，那麼使用者在編寫 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 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－Node A－Relation－Concept B－Node 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 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。

3.9. Tag

在本系統的設計上，Ontology 是一個且只有一個 unconnected graph，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 圖形中的每個連接元件都視爲一個獨立的知識領域。

除了界定不同的知識領域外，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。

爲了解決上述的問題，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 開始的子領域 Ontology，而透過這個方式建立的子領域 Ontology 均是一個 connected grpah。

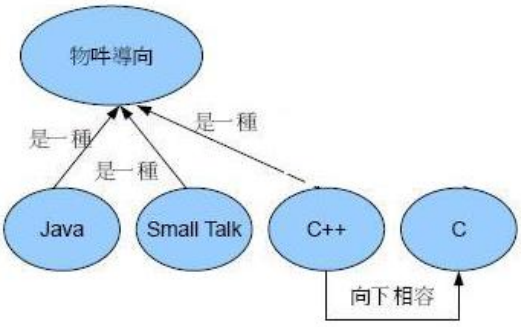

圖 10: 透過 OO 這個 Tag 取出的子領域

舉例而言，當使用者建立了 OO 這個 Tag，並且將其起始點標示爲圖 9 中『物件導向』的這個 Concept 時，將會產生如圖 10 的子領域 Ontology。

讀者可以發現，在圖 10 中，C 語言依然被納入 Ontology 中，這是因爲 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』、Function Pro-

gramming 以及 Haskell 不會出現在子領域 Ontology 中。

關於本系統建立子領域 Ontology 的演算法詳細內容，請參閱第 5.1 小節。

4. 系統特色

4.1. 泛用性

在 2.4 中我們曾經提到，Drupal 核心內的 Node 模組僅具最基本的內容型態，而由於 Drupal 設計上的擴充性，模組開發者可以輕易地建立新的內容型態，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型態。

另一方面，目前 Drupal 社群的開發者，已經開發出處理圖片的 Image 模組、影像的 Video 模組，以及其他各式各樣不同內容型態的模組，我們希望使用者在使用本系統時，可以沿用舊有的內容模組，而不需另外建立新的內容型態。

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 模組與本模組，並且將 Ontology 建立後，即可將 Drupal Video 的任何內容指定到 Ontology 上的某個 Concept 上。

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

在實作上，由於系統內所有的內容型態均繼承自 Node 這個資料表，並且具有 Node ID 這個編號欄位，因此我們只需要建立 Concept 與 Node ID 的關聯，即可達到上述支援所有內容型態的目的。

4.2. 權限控管

Drupal 核心本身即具有完整的角色權限控制等相關機制，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。

爲了達成使用上的多樣性，我們定義了如表 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 1: 權限列表

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

5. 相關演算法

5.1. 子領域演算法

在 3.9 中我們曾經提到，使用者可以透過 Tag 來縮小 Ontology 的範圍，取得子領域 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。

程式碼 1 即爲此演算法，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，我們僅將與該節點爲 extends 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 extends 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，這也是爲什麼在圖 10 中，『程序導向』這個 Concept 不會被納入的原因。

程式碼 1: 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)
{
    $arcList =
        $concept->getExtendPath();

    $concept.vistied = true;

    foreach ($arcList as $rel) {
        enqueu ($rel.to);
    }

    // BFS 演算法
    while ( !$queue.empty ) {
        $vertex = dequeue ();

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

        $vertex->visted = true;

        $arcs =
            $vertex->
            getArcExcludeIsKindOf ();

        foreach ($arcs as $arc) {
            $arcList[] = $arc;
            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() 所產生的邊集合進行處理，方法也非常簡單，僅需查詢資料庫中是否有相反方向的邊，若有則將其加入該邊集合中即可，程式碼 2 爲此部份的演算法。

程式碼 2: 重建反向邊演算法

```
/**
 * Ontology 重建反向邊演算法
 * Input:  A set of arc.
 * Output: A set of arc.
 */
function rebuildRArc ($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 所開始的子領域，減少使用者所需要觀注的 Ontology 範圍。

6. 結論

本研究中，爲了解決 Drupal 內建的 Taxonomy 分類模組以及 CCK 內容型態建立模組上的缺點，我們提出了一套以 Ontology 爲基礎架構的延伸模組。

在這個系統中，使用者可以利用 Ontology 的概念替系統中的內容進行分類，並且替內容加註後設資料並且描述內容與內容間的關聯性，在此同時，我們也保留了相當大的彈性，例如同樣的內容可以同時屬於不同的 Concept。

本系統與其他內容處理模組最大的不同，在於我們將後設資料與內容分離，因此使用者可以沿用舊有的資料與內容模組，同時也可以處理任何的內容型態。

此外，我們亦提出了一個簡單的演算法，讓使用者可以針對複雜的 Ontology 圖形進行過濾，取出其中一個連接元件做爲子知識領域，縮小所需要處理的資料量。

雖然此模組仍在開發初期，但我們期望這個模組能夠有效解決目前市面上的內容管理系統，對於內容的關聯性以及後設資料處理不易等問題。

參考文獻

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

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