Semantic Object Framework – A language independent approach to integrate semantic web and object oriented programming

Abstract

物件導向程式設計是目前軟體開發的主流，然而語意網與物件導向程式設計的整合，卻存在許多問題必須解決，這些待解的問題包括: 將 RDF API 抽象化以物件方式操作，全自動將資料轉爲 RDF 格式，支援多種程式語言的架構，用敘述句描述類別或屬性的語意，支援類別和屬性間的繼承與異質資料查詢，資料和語意的一致性檢驗. 之所以需要解決上述種種問題，主要是因爲物件導向類別關係的描述能力比 RDF 有更多侷限，導致物件導向類別跟語意網類別彼此無法直接對映. 雖然最近已經有針對特定程式語言製作的物件導向 RDF API 試圖減輕開發時的工作量，但這類 API 往往只解決了上述問題的某一部分，缺乏完整的解決方案. 本論文提出 Semantic Object Framework (SOF), SOF 結合了物件導向設計和語意網彼此的優點，並使用內嵌註解的作法來描述類別與屬性的語意關係，以解決上述所提到的所有問題.

1.Introduction

語意網擴充了 WWW 的描述能力，其基本概念是對資料定義其語意關連性，使這些加工後的資料可讓自動工具(機器)分享及處理，這有利於各種應用能有效的找到所需資料、執行自動化、整合、及再利用。RDF 是語意網的資料模型表示法，一個基礎的 RDF 文件是由三個元素組成一個 statement，這三個元素分別是: Subject, Predicate，以及 Object，而目前用來操作 RDF 的 API 多數都是以 Triple Oriented API 爲基礎操作三個元素組成的 statement.

開發語意網應用時，必須在程式語言當中處理 RDF 資料模型. 然而目前主流的軟體開發方法是採用物件導向程式設計，其實並不合適用來處理語意網資料. 現代物件導向程式設計大多採用 Model View Controller (MVC) 作爲責任切割的架構來設計類別，最常見的實作則是將屬於 Model 類別的資料物件轉換爲 record 形式儲存在關連式資料庫. 然而 RDF 採用 Triple Oriented 組成的 statement 作爲資料格式的基礎，這與 MVC Model 類別的格式彼此差異甚多，而且物件導向類別缺乏屬性之間的語意關係描述，因此難以自動將資料物件轉換爲 RDF 提供語意查詢. 若要將龐大的既有資料轉換爲 Triple Oriented 方式儲存,

無論在效能與成本上都是非常嚴苛的考驗.

本論文提出了 SOF 將物件導向設計和語意網緊密的整合在一起，程式開發者不需要學習兩套不同的概念，只需要運用物件導向設計方法，就可以輕易的把資料物件發佈爲 RDF 格式，並且根據類別和屬性的語意關係，進行異質資料查詢.

在第 2 節我們將檢視現存的語意網開發技術有哪些，以及這些既有方案的優缺點，讓讀者對目前的語意網開發技術有一個全貌的瞭解. 第 3 節我們將討論物件導向設計和語意網結合需要解決哪些問題，才算一個全面性的解決方案. 第 4 節描述了 SOF 的 Architecture 以及 Class Diagram，讓讀者瞭解此系統的內部設計原理. 第 5 節實際運用 Gmail 通訊錄與 ThunderBird 通訊錄整合查詢爲範例，介紹如何以 SOF 將資料物件全自動發佈爲 RDF 格式，並且進行異質資料物件合併查詢. 第 6 節描述了未來研究方向，而第 7 節是本論文的結論.

2.Survey

以下將針對 Jena[引用 Jena 文獻], ActiveRDF[引用 ActiveRDF 文獻], D2R[引用 D2R 文獻], 與 EClass[引用 EClass 文獻] 四種開發語意網的解決方案進行背景介紹，這四種方案對開發者提供了不同層次的選擇，由於他們彼此之間有顯著的差異，因此在介紹完這四種方案之後，讀者將對目前開發語意網的方案有何優缺點更能全面的瞭解.

使用 Jena 的解決方案

Jena 是目前最受歡迎的 Java RDF API，可運用 Triple Oriented API 方式對 RDF 資料進行讀寫及查詢. 這個方案的主要優點是對 RDF 低階操作有完整的支援，使用者眾多，因此實作產品較穩定，但是由於是低階 API，因此缺乏與物件導向程式設計的整合，在使用上必須仔細的描述每一個操作步驟，無法用物件的抽象化概念思考.

使用 ActiveRDF 的解決方案

ActiveRDF 是基於 Ruby 語言的 RDF 物件導向 API，他將基礎的 Triple Oriented API 進一步抽象化，採用物件導向的手法來操作 RDF 文件，而底層實現仍是將資料轉換後儲存在 Triple Oriented Storage，其用意在簡化 API 呼叫時的難度，這類物件導向 API 方案，目的並非提供全自動將既有的 Model 物件資料發佈爲語

意網提供查詢，另外由於 API 實作上的限制，這類解決方案只能提供給單一程式語言使用，無法橫跨多種不同程式設計語言.

使用 D2R 的解決方案

D2R 是個很有特色的解決方案，他的概念是將關連式資料庫中的紀錄，直接轉換爲 RDF 格式，以利進行 RDF 的讀寫或查詢，由於操作的對象是資料庫，所以 D2R 適用於任何程式設計語言，只要將資料庫 Table 與 RDF 之間的對映關係指定清楚, D2R 會全自動進行格式轉換的動作，省去程式設計師轉換資料的麻煩. 然而也因爲 D2R 的操作對象是資料庫，因此並沒有提供物件導向的包裝，無法使用物件的方式對資料進行操作.

使用 EClass 的解決方案

EClass解決方案提出了一種改變Java或C++語法的點子，將原本Java語言當中的 Class 宣告改爲叫做 EClass，直接改變語言的定義. 大寫字母E是擴充的意思，因爲EClass當中允許開發者定義屬性彼此之間的語意關係. 然而要去改變已經被廣爲使用的程式語言語法，現實上會有一定的阻礙，而要去除這個阻礙則必須不影響現有程式語言的語法定義，又能夠對類別和屬性之間的語意關係進行描述. 另外EClass解決方案中尚未提及跨異質資料物件的查詢功能，這也需要未來進一步加強.

3.Problem Statement

本節將探討 Semantic Web 與物件導向設計結合會遭遇的問題有哪些，並且針對既有方案已經解決了哪些問題做出一張比較表，讓讀者可以對照每種解決方案之間的主要差異. 以下將逐點討論語意網與物件導向設計結合時，有哪些必須支援的功能特點:

3.1 將 RDF API 抽象化以物件方式操作

低階的 RDF API 雖然提供了完整的 RDF 讀寫與查詢操作，但是開發者無法運用物件方式操作 RDF 資料，因此在開發時程上會較長，程式碼也會冗長而難以維護，因此採用物件導向設計方式，把 RDF API 抽象化，可以讓開發者更有效率的寫出具有生產力的程式碼.

SOF 爲解決此問題,允許開發者採用物件導向 API 進行查詢,並且查詢結果也以資料物件的方式回傳.

3.2 全自動將資料轉爲 RDF 格式

RDF API 雖然可以把資料儲存到 Triple Store 進行語意查詢，但是必須由開發者親自一一進行把資料物件轉換爲 Triple Store 格式的動作，而這是非常耗時費力的繁瑣流程，若有一種開發架構能將資料物件全自動轉換爲 RDF 格式發佈，能夠省下大量的開發時間.

SOF 爲解決此問題,允許開發者不用負擔將資料轉換爲 RDF 的責任,而將資料物件發佈爲 RDF 格式由 SOF 全自動完成,大幅減輕開發者負擔. 另外,SOF 也提供了一個內建 Web Server 可以讓第三方程式使用 HTTP Protocol 讀取最新的 RDF 格式資料.

3.3 支援多種程式語言的架構

目前，分別針對各種程式語言，已經有一些物件導向 API 可以操作 RDF 文件 [引用 Python, Ruby ActiveRDF, 相關 API]，然而這些實作都是針對特定語言，彼此之間並沒有一致性.

SOF 爲解決此問題,其語法並沒有特別綁定某一種物件導向語言，由於 SOF 是採用註解的方式對類別與其屬性進行語意描述,SOF 的 Parser 在各種程式語言僅需要做少部份修改即可重複使用. 而程式設計師只要學一套 SOF 的用法，就可以應用在各種程式語言上面.

3.4 用敘述句描述類別或屬性的語意

將語意網與物件導向設計結合，最自然的方式，就是能夠對類別或者屬性進行語意描述，在定義類別時，一併將語意關係表示清楚，這樣就能夠在稍後對資料物件進行語意查詢了. 然而要對類別或屬性進行語意修飾，通常必須修改程式語言的語法才能辦到，如何在不影響原有程式語言語法的情況下，又能修飾類別或屬性的語意關係，是很重要的一點.

SOF 爲解決此問題,採用了內嵌註解的方式,允許使用部分 RDF 與 OWL 語法來修飾物件導向類別或屬性的語意關係.

3.5 語意描述檔與類別定義隨時保持同步

在目前的語意網實作中,有一些解決方案是提供獨立的 RDF 語意描述檔案,去補充修飾既有資料之間的關連性，然而這會造成實作上必須隨時保持各檔案之間同步更新,否則可能造成不一致的錯亂結果.

過去最有名的例子是程式碼 API 說明文件與程式碼本身彼此是獨立的檔案,所以經常造成說明文件來不及更新導致充滿了過時的錯誤描述.

爲了將程式碼說明文件與程式碼本身徹底的結合在一起，避免彼此之間的不一致問題, JavaDoc 採用註解的方式內嵌在程式碼當中，讓程式設計師可以很方便的保持 JavaDoc 與程式碼之間的一致性. [引用 JavaDoc 技術][引用 PyDoc 技術]

SOF 爲解決此問題,運用類似的原理,也採用內嵌於程式碼的方式,讓程式碼和語意描述隨時保持同步.

3.6 支援類別和屬性間的繼承與異質資料查詢

在不同的資料庫當中經常發生 column name 不同但意義相同的情況，例如電子郵件在資料庫 A 的 column name 可能叫做”email”,但在資料庫 B 卻叫做”mail”. 若要對這兩個不同的資料庫進行合併查詢所有人的電子郵件，就必須先定義清楚讓電腦知道 email 與 mail 其實在語意上是指相同的事物. 而目前缺乏一個良好的架構能夠在物件導向程式碼當中,對類別與屬性定義其語意關係,以便讓系統能自動處理屬性名稱不同但意義相同的查詢功能.

除此之外,合併查詢異質資料來源還會發生:查詢結果的資料物件當中，混合著隸屬於不同類別的資料物件，此時必須有個機制可以讓物件導向程式碼能分辨資料物件的所屬類別，並可根據類別的不同對其屬性進行各別處理.

SOF 爲解決此問題,允許利用註解讓屬性之間擁有語意繼承關係,並且允許開發者藉此進行異質資料物件的合併查詢.

3.7 資料和語意的一致性檢驗

當我們針對類別或屬性進行了語意修飾之後，就有可能發生資料物件跟語意彼

此衝突的情況，例如在一個帳號管理系統當中，我們如果指定了一個 Email 的值只能隸屬於一個唯一帳號，那麼當有兩個帳號擁有同一個 Email 值的時候，就是衝突發生了．一個好的解決方案必須能夠輕易的找出這種衝突，確保資料和語意的一致性沒有問題.

SOF 爲解決此問題,提供了查詢非法資料物件的 API,讓開發者可以對資料物件進行語意一致性檢查.

SOF 與既有解決方案的比較表(Jena/ActiveRDF/D2R/EClass)

本論文提出的 Semantic Object Framework(SOF)，正是爲了完整解決上述提到的 7 個問題所設計的．請參考[表 1]的內容，在該表當中針對 5 種語意網開發方案是否能有效解決上述 7 個問題，有詳盡的列表比較，其中 X 代表無法解決該問題，而 O 代表可以解決該問題.

表 1: 5 種語意網開發方案的功能比較表

Problem	Jena	ActiveRDF	D2R	EClass	SOF
3.1	X	O	X	O	O
3.2	X	X	O	O	O
3.3	X	X	O	X	O
3.4	X	X	X	O	O
3.5	X	X	X	O	O
3.6	X	X	X	X	O
3.7	X	X	X	X	O

4.Solution

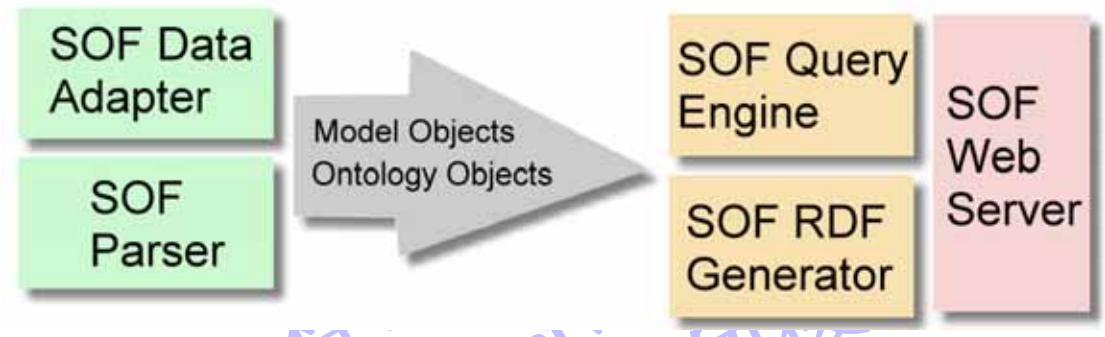
本節提出 SOF 架構來解決第 3 節所提到的 7 個問題，並運用架構圖來說明 SOF Architecture 如何達成其設計目標，本節的最後藉由異質通訊錄查詢範例，讓讀者體驗 SOF 如何成功整合物件導向設計和語意網技術.

SOF Architecture

以下將介紹 SOF 的架構，包含五個主要 module 的用途介紹，以及他們彼此之間

input/output 關係的運作流程說明.

[Diagram 1] Five SOF primary modules



SOF 由 5 個主要的 module 構成, 請參考 [Diagram 1], 其中在箭頭部分的 model objects 是包含了資料內容的資料物件, 而 ontology objects 當中則包含了類別和屬性之間的語意關係, 以下分別一一說明 5 個 module 的用途.

SOF data adapter

此 module 的功能是讀取各種不同的 data source, 以便能夠將這些 data source 轉換爲 model objects. data source 可以是 CSV 檔案格式, 或者資料庫中的記錄, 或者讀取各種 proprietary 資料的 API. 程式設計師也可以自己撰寫其他 data source 的 data adapter, 只要可以將 data source 轉換爲 model objects 即可符合 SOF 的資料處理需求, 而 SOF 提供了彈性的架構, 在未來允許支援各種資料來源格式.

SOF data adapter 的輸出是 model objects, 是將資料以物件的方式呈現, model objects 當中包含了所有實際的資料內容, 也就是每筆資料屬性的值. 而 model objects 將成爲 SOF Query Engine 和 SOF RDF Generator 的輸入參數.

SOF parser

此 module 的功能是 Parse 程式碼註解當中的 SOF 敘述句, 以便產生 ontology objects. 爲了能夠提供語言中立的特性, SOF parser 必須支援各種主流的物件導向語言, 然而每一種語言的註解寫法不盡相同, 所以 SOF 採用了可跨程式語言的撰寫語法, 以便讓 SOF 敘述句能融合在各種不同程式語言的註解當中, 而又能讓所有程式語言都共用同一份 SOF parser 程式碼.

SOF parser 的輸出是 ontology objects, 是將類別和屬性的語意關係採用物件的

表示法呈現, ontology objects 將會成爲 SOF RDF Generator 與 SOF Query Engine 的輸入參數.

SOF RDF generator

此 module 目的是將 model objects 輸出爲 RDF 格式, 以便讓其他第三方的程式碼可以讀取 RDF 格式的資料. 由於 ontology objects 當中已經記錄了 model objects 彼此之間的語意關係, 所以最後能產生包含語意關係的 RDF 格式檔案.

SOF query engine

此 module 目的是提供物件導向 API 進行跨異質 data source 的合併查詢, 除了查詢的 API 是採用物件導向的風格之外, 查詢結果也會以物件陣列集合的方式回傳給開發者. 由於回傳的 model objects 陣列可能分別隸屬於多個不同類別, 所以也必須提供適當的類別轉型 API 來處理格式轉換上的問題.

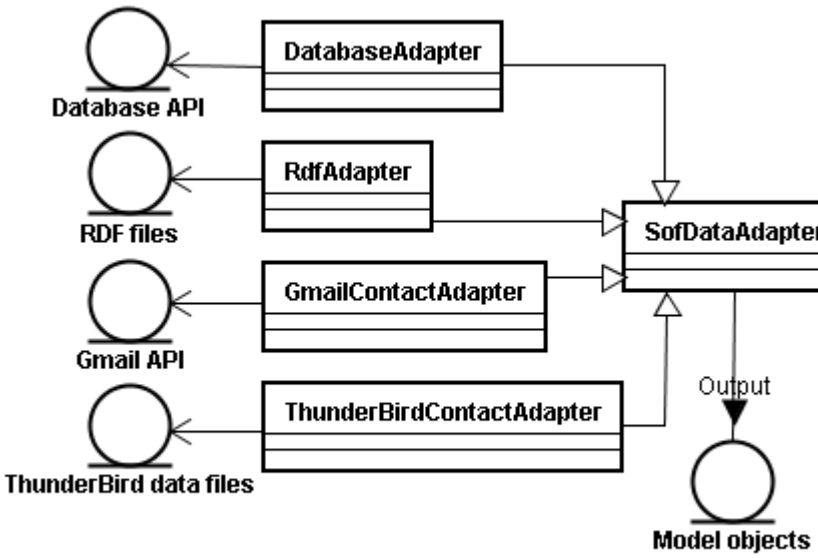
SOF web server

此 module 目的是提供一個 HTTP Protocol 的入口,讓第三方程式可以讀取最新的 RDF 文件, 由於 SOF 是採用動態轉換的方式將資料轉換爲 RDF, 所以只要是從 SOF web server 讀取的 RDF 文件, 都能確保反應了最新的 model objects 內容變化, 不用擔心資料一致性的問題.

SOF Class Diagram

以下將介紹 4 個主要 module 的 class diagram, 而 SOF web server 僅是單純的提供外界 request 的介面,所以在此略過其 class diagram 的解說.

[Diagram 2] Class diagram of SOF data adapter

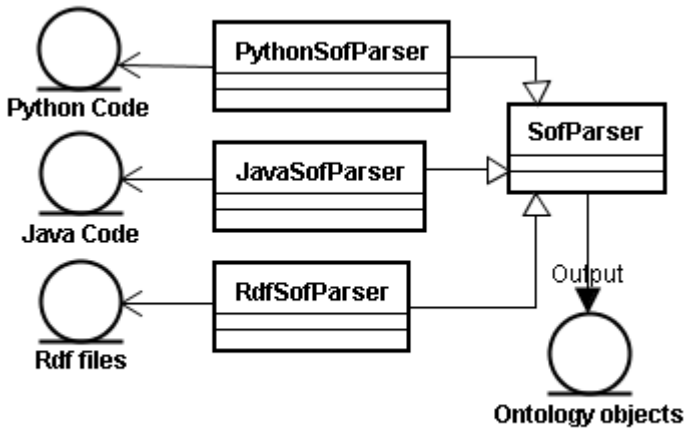


SOF data adapter 的輸入端可以是各種不同的 data source，經過轉換將這些資料輸出爲 model objects 格式之後，便可以運用物件導向 API 讀寫這些 model objects.

請參考 [Diagram 2]，我們可以看到有四種不同的 SOF data adapter，分別是: DatabaseAdapter 透過資料庫 API 讀取紀錄, RdfAdapter 讀取 RDF 格式的資料檔案, GmailContactAdapter 透過 Gmail API 讀取通訊錄資料，最後一個 ThunderBirdContactAdapter 則讀取 ThunderBird 的通訊錄資料檔案格式. 而這四種不同的 SOF data adapter 都繼承 SofDataAdapter 類別，以便讓他們擁有共同的操作方法.

在物件導向程式設計當中, Model-View-Controller (MVC) 設計模式，是很常見的一種手法 [引用 MVC 相關論文]，而這其中 Model 代表了資料本身. SOF data adapter 最後的輸出格式指的就是 MVC 當中的 Model. 通常 model objects 都提供了物件屬性的讀取與寫入等操作方法.

[Diagram 3] Class diagram of SOF parser

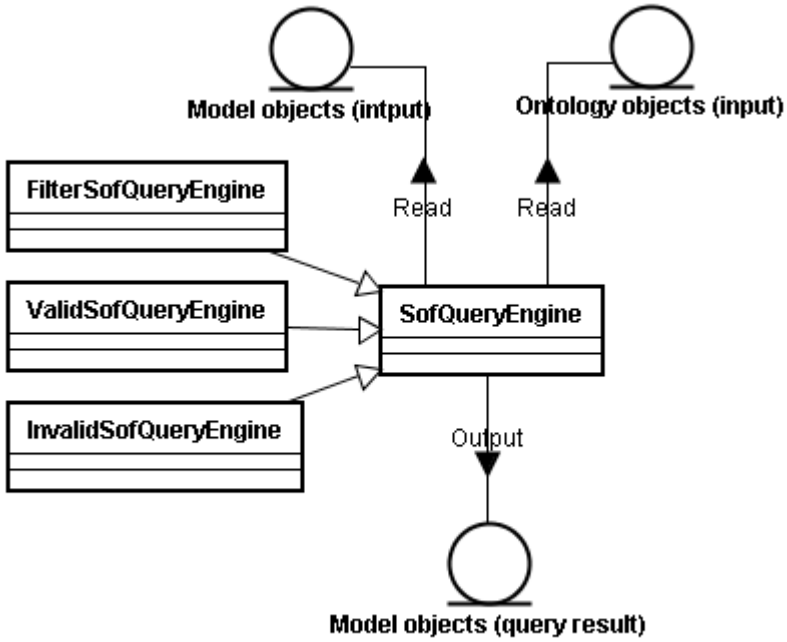


SOF parser 的輸入是各種不同的程式語言 source code，這些 source code 當中包含了 SOF 敘述句，這些 SOF 敘述句當中描述了類別與屬性彼此之間的語意關係，而 SOF parser 會將 SOF 敘述句轉換爲 ontology objects 並輸出.

請參考 [Diagram 3]，我們可以看到有三種不同的 SOF parser，分別是 PythonSofParser 負責讀取 Python code, JavaSofParser 負責讀取 Java code, 而 RdfSofParser 負責讀取 Rdf 檔案格式中的類別語意關係. 這三個類別都繼承 SofParser 類別，並且可以在 SofParser 當中實作這三個類別共用的程式碼.

由 SOF parser 所輸出的 ontology objects 記錄了類別和屬性之間的語意關係，並且以物件的表示法呈現，這些 ontology objects 若和 model objects 同時使用，則可以對 model objects 進行跨異質資料來源的語意查詢.

[Diagram 4] Class diagram of SOF query engine

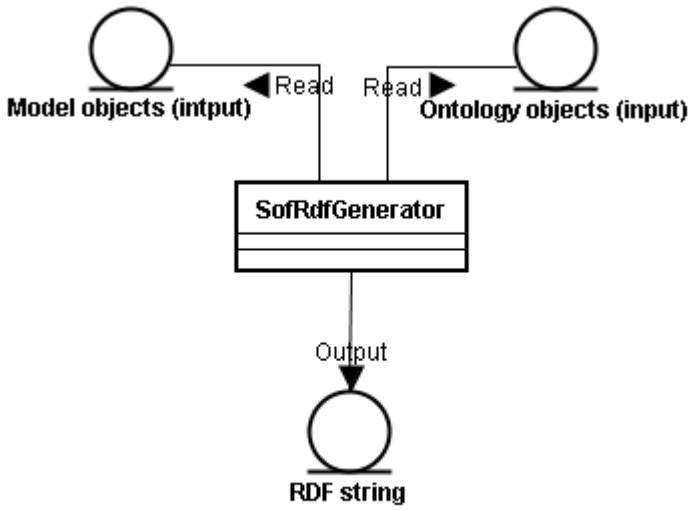


SOF query engine 的輸入是 model objects 以及 ontology objects，在此 model objects 可以是各種不同類別組合而成的物件陣列，而 ontology objects 則可以解釋這些 model objects 彼此之間的繼承關係以及語意關係. SOF query engine 可以接受查詢敘述句，並且以 model objects 的形式輸出查詢結果.

請參考 [Diagram 4]，有三種不同的 SOF query engine 分別是，FilterSofQueryEngine 負責以條件過濾的方式進行語意查詢，ValidSofQueryEngine 負責查詢符合語意規則的 model objects，而 InvalidSofQueryEngine 負責查詢在語意上屬於非法的 model objects. 這三者都繼承 SofQueryEngine，並且輸出結果都是 model objects.

最後產生的查詢結果 model objects，如果是 FilterSofQueryEngine 的輸出，則僅會列出符合查詢條件的 model objects，並且因爲查詢時可以輸入各種不同類別的物件陣列，所以查詢結果也會包含不同類別的物件，由於程式開發者可以用 API 得到 model objects 的原始類別型態，因此如果有需要的話，可以針對不同類別進行特殊處理. 而如果查詢屬於 InvalidSofQueryEngine 的輸出，則 model objects 還會包含爲何該物件被歸類爲非法物件的原因描述，讓開發者可以知道如何修正該錯誤.

[Diagram 5] Class diagram of SOF RDF generator



SOF RDF generator 的輸入分別是 model objects 以及 ontology objects，藉由結合這兩種輸入, SOF RDF generator 可以輸出包含類別和屬性語意關係的 RDF string.

請參考 [Diagram 5]，最後的 RDF string 輸出可以直接儲存成爲檔案，或者讓其他應用程式透過 SOF web server 以 HTTP 存取該 RDF string. 由於 RDF string 是 W3C 標準的格式，並且包含 model objects 資料內容以及 ontology objects 的語意關係，因此只要能處理 RDF 格式的應用程式都能很方便的對 RDF string 進行查詢或資料合併等動作.

5.Examples

本節將採用實際的範例，來說明如何運用 SOF. SOF 提供了兩個主要的功能,第一個是將資料物件全自動轉換並發佈爲 RDF 格式,第二個功能是跨異質資料來源進行語意查詢. 而在本範例中將使用 Gmail 與 ThunderBird 兩種不同軟體的通訊錄資料，進行 SOF 的主要功能示範,由於這兩種通訊錄採用不同屬性名稱，資料格式也不盡相同,如果開發者想要撰寫程式進行跨通訊錄進行查詢的任務,會遭遇許多繁瑣的格式轉換流程. 在此我們採用 Python 語言爲例,利用 SOF 在類別宣告時對 Gmail 與 ThunderBird 通訊錄的屬性增添語意關係. 當語意關係建立完成

之後，分別示範 SOF 的兩項功能,第一項是利用 SOF 自動將兩種通訊錄一併發佈爲 RDF 格式，第二項是允許跨異質通訊錄進行語意查詢.

在註解中使用 OWL 語法定義兩種不同的通訊錄類別

要對兩種不同通訊錄進行合併查詢之前,我們先定義一個叫做 Contact 的類別,讓 Contact 類別擁有兩種通訊錄共通的屬性,並且稍後 GmailContact 與 ThunderBirdContact 將在語意上繼承 Contact 類別.

```
class Contact(Model):
    partOfName=''
    partOfAddress=''
    #owl:InverseFunctionalProperty Contact_email
    email=''
    phoneNumber=''
    #Contact_officePhoneNumber rdfs:subClassOf Contact_phoneNumber
    officePhoneNumber=''
    #Contact_homePhoneNumber rdfs:subClassOf Contact_phoneNumber
    homePhoneNumber=''
    #Contact_mobilePhoneNumber rdfs:subClassOf Contact_phoneNumber
    mobilePhoneNumber=''
    #Contact_faxPhoneNumber rdfs:subClassOf Contact_phoneNumber
    faxPhoneNumber=''
```

由於 Contact 類別在典型的 Model-View-Controller (MVC) 設計模式當中是屬於 Model 資料類別，所以我們宣告 class Contact(Model) 代表 Contact 繼承 Model 類別. 繼承 Model 的資料類別都可以 Serialize 儲存在資料庫，並允許條件式的資料查詢.

partOfName 這個屬性名稱所代表的意義是聯絡人名字，代表聯絡人名字的屬性可能有: 姓/名/中間名/全名/暱稱 等等，我們在此讓 partOfName 代表任何可能的名字片段或全名，稍後如果有其他屬性在語意上繼承 partOfName，就代表該屬性是用來識別聯絡人名字的字串之一.

井號 # 在 Python 語言當中是註解的意思，由於 SOF 的語法是內嵌在註解當中，所以當看到註解當中有包含了 owl: 或者 rdfs: 時，就代表該行敘述爲 SOF 特有的描述語句. #owl:InverseFunctionalProperty Contact_email 採用了 OWL 語法來修飾其語意，說明了當 email 字串相同時，其 Contact 物件代表的一定是唯

一的人，不應該發生 Email 字串值相同時，但是 Contact 物件卻有兩個以上的情況. 如果萬一在資料裡面發生了兩個 Contact 物件都擁有相同的 email 字串時，SOF 有能力將衝突的兩筆 Contact 物件查詢出來，並且交由程式設計師進一步用各種策略來化解資料在語意上違法的情況. 這類的 OWL 敘述將有助於程式設計師運用更豐富的語法來限制資料物件之間的關連性.

email 屬性代表的是電子郵件，然而在不同的通訊錄軟體當中, email 屬性名稱有可能是下列幾種: Email/email/mail/Mail/emailAddress/EmailAddress 這些不同的屬性名稱在語意上是完全相同的，如果我們要跨異質通訊錄把所有的電子郵件屬性值都顯示出來，就必須在語意上讓各種電子郵件的屬性使用 rdfs:subClassOf 繼承 Contact_email 這個屬性.

其他屬性因爲都很直覺可以理解,因此不再多作描述.

接下來我們看看 GmailContact 如何在語意上繼承 Contact 所定義好的各種屬性.

```
#GmailContact rdfs:subClassOf Contact
class GmailContact(Model):
    #GmailContact_name rdfs:subClassOf Contact_partOfName
    name=''
    #GmailContact_email rdfs:subClassOf Contact_email
    email=''
    #GmailContact_phone rdfs:subClassOf Contact_officePhoneNumber
    #GmailContact_phone rdfs:subClassOf Contact_homePhoneNumber
    phone=''
    #GmailContact_mobile rdfs:subClassOf Contact_mobilePhoneNumber
    mobile=''
    #GmailContact_fax rdfs:subClassOf Contact_faxPhoneNumber
    fax=''
    company=''
    title=''
    #GmailContact_address rdfs:subClassOf Contact_partOfAddress
    address=''
```

#GmailContact rdfs:subClassOf Contact 所代表的意義是, GmailContact 這個類別在語意上繼承了 Contact 類別，所以日後如果有物件查詢命令，要查詢出所有 Contact 資料物件時，繼承自 Contact 類別的 GmailContact 資料物件也會被列入查詢範圍當中，稍後我們可以看到 ThunderBirdContact 也在語意上

繼承 Contact, 因此當開發者要跨 Gmail/ThunderBird 兩種不同通訊錄軟體查詢資料物件時, 只要指定查詢對象是 Contact 類別, 則 SOF 就能根據繼承關係全自動判定 GmailContact 與 ThunderBirdContact 這兩種資料物件也必須納入查詢範圍當中, 這樣就能夠輕易達成跨異質通訊錄進行查詢的目的.

#GmailContact_name rdfs:subClassOf Contact_partOfName 這一行說明了 GmailContact 類別中的 name 屬性語意上繼承 Contact 類別的 partOfName 屬性. 這代表未來如果開發者指定查詢 Contact_partOfName 屬性的字串值時, SOF 會全自動一併查詢 GmailContact_name 屬性的字串值.

接下來需要介紹的敘述句是有關 GmailContact_phone 的多重繼承關係: #GmailContact_phone rdfs:subClassOf Contact_officePhoneNumber 以及 #GmailContact_phone rdfs:subClassOf Contact_homePhoneNumber 代表 GmailContact_phone 這個屬性可能是一種辦公室電話, 也可能會是一種家用電話. 由於 RDF 語法是允許多重繼承關係的, 所以即使是在不支援多重繼承的程式語言如 Java 當中, SOF 仍然允許在語意上讓類別或各屬性進行多重繼承關係的描述. 以 GmailContact_phone 爲例, 未來無論開發者查詢的目標是 Contact_officePhoneNumber 或是 Contact_homePhoneNumber, SOF 都會全自動對 GmailContact_phone 屬性進行查詢.

由於 GmailContact 其他屬性的語意繼承關係非常直觀, 讀者應可以從程式碼片段了解其意義, 因此不再一一解釋. 以下我們再來看看 ThunderBirdContact 如何在語意上繼承 Contact 的範例.

```
#ThunderBirdContact rdfs:subClassOf Contact
class ThunderBirdContact(Model):
    #ThunderBirdContact_mail rdfs:subClassOf Contact_email
    mail=''
    #ThunderBirdContact_givenName rdfs:subClassOf Contact_partOfName
    givenName=''
    #ThunderBirdContact_sn rdfs:subClassOf Contact_partOfName
    sn='' #first name
    #ThunderBirdContact_cn rdfs:subClassOf Contact_partOfName
    cn='' #full name
    #ThunderBirdContact_telephoneNumber rdfs:subClassOf
Contact_officePhoneNumber
    telephoneNumber=''
    #ThunderBirdContact_homePhone rdfs:subClassOf
```

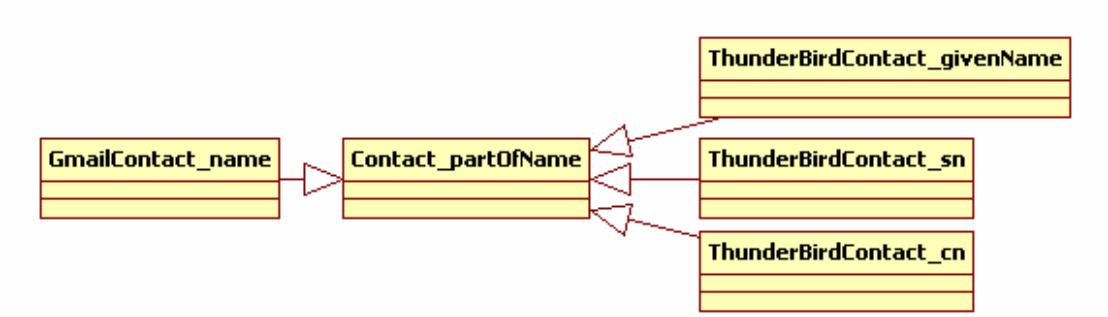
Contact_homePhoneNumber
homePhone=''
#ThunderBirdContact_fax rdfs:subClassOf Contact_faxPhoneNumber
fax=''
#ThunderBirdContact_mobile rdfs:subClassOf
Contact_mobilePhoneNumber
mobile=''
#ThunderBirdContact_homeStreet rdfs:subClassOf
Contact_partOfAddress
homeStreet=''
#ThunderBirdContact_mozillaHomeLocalityName rdfs:subClassOf
Contact_partOfAddress
mozillaHomeLocalityName=''
#ThunderBirdContact_mozillaHomeState rdfs:subClassOf
Contact_partOfAddress
mozillaHomeState=''
#ThunderBirdContact_mozillaHomePostalCode rdfs:subClassOf
Contact_partOfAddress
mozillaHomePostalCode=''
#ThunderBirdContact_mozillaHomeCountryName rdfs:subClassOf
Contact_partOfAddress
mozillaHomeCountryName=''
#ThunderBirdContact_street rdfs:subClassOf Contact_partOfAddress
street='' #street of company
#ThunderBirdContact_l rdfs:subClassOf Contact_partOfAddress
l='' #locality name of company
#ThunderBirdContact_postalCode rdfs:subClassOf
Contact_partOfAddress
postalCode='' #postal code of company
#ThunderBirdContact_c rdfs:subClassOf Contact_partOfAddress
c='' #country name of company
title=''
department=''
company=''
mozillaHomeUrl=''

ThunderBirdContact 類別與 GmailContact 類別在語意上全都是繼承 Contact 類別，讀者可以從 ThuderBirdContact 的眾多屬性中得知該類別比 GmailContact

複雜,尤其是與地址相關的屬性,在 GmailContact 沒有區分住家地址與公司地址,甚至也沒有區分國家/縣市/街道等屬性,GmailContact 僅採用一個 address 屬性代表所有可能的住址字串. 但 ThunderBirdContact 當中有關地址的屬性多達九個, 並且這九個屬性在語意上都繼承了 Contact_partOfAddress.

我們可以進一步來看[Diagram 6], ThunderBirdContact 當中與聯絡人名字有關的屬性有三個, 而 GmailContact 只有一個 name 屬性來代表聯絡人名字. 而 ThunderBirdContact 與 GmailContact 與名字相關的屬性全都在語意上繼承 Contact_partOfName 屬性.

[Diagram 6] 繼承 Contact_partOfName 的屬性示意圖



至此我們已經將 Contact, GmailContact, ThunderBirdContact 三者的語意關係建立完畢, 因爲 SOF 是採用註解的方式把語意描述內嵌於程式碼當中, 所以閱讀程式碼的人可以很容易找到屬性彼此之間的語意對映關係. 接下來無論是要透過 SOF 把資料物件發佈爲 RDF 格式, 或是透過 SOF 對資料物件進行查詢, 都可以很輕易的達成目標.

全自動把通訊錄發佈成爲 RDF 格式

SOF 允許將通訊錄資料物件全自動發佈爲 RDF 格式, 由於 RDF 格式的資料必須能採用 HTTP 方式存取, 所以在 SOF 系統中有一個 SOF Web Server 負責提供 HTTP 進入點, 只要使用適當的 URL 就能抓取對應的資料物件 RDF 格式,例如: http://127.0.0.1:8080/sof/Contact/ 就可以取得所有繼承 Contact 的資料物件的 RDF 格式資料, 此時 GmailContact 與 ThunderBirdContact 的資料物件將混合在同一份 RDF 供開發者讀取. 如果開發者只想單獨取得子類別的 RDF 則採用以下 URL 即可達成, 例如: http://127.0.0.1:8080/sof/GmailContact/ 可單獨取得所有 GmailContact 資料物件的 RDF 格式資料, 而不包含 ThunderBirdContact 的資料物件.

由於 SOF 採用的是動態轉換資料物件爲 RDF 的實作技術，因此每次從 URL 都能夠取得最新的資料變化，而在效能方面可以運用 Cache 技術來提升，如果資料沒有變動時，只要從 Cache 調用上次的 RDF 結果即可，若資料有變動時，才需要重新自動轉換產生一份 RDF 格式的文件.

跨異質通訊錄進行查詢

當 Gmail 與 ThunderBird 兩種通訊錄發佈爲語意網之後，跨兩種不同資料庫來源合併查詢，是語意網最有用的功能之一. 以下的程式碼片段將從所有繼承 Contact 的子類別當中，找出 email 屬性結尾是 nctu.edu.tw 字串的資料物件.

要注意的是, GmailContact 與 ThunderBirdContact 兩者代表 Email 意義的屬性名稱並不相同，在 Gmail 當中 Email 屬性叫做 email，但是在 ThunderBird 當中 Email 屬性叫做 mail，雖然兩者屬性名稱相異，不過卻不影響合併查詢的功能，原因是他們都繼承了 Contact_email 屬性，所以符合的資料物件仍會全部都被找出來.

```
lstContact=Contact.objects.get("email like '%nctu.edu.tw'")
intCounter=0
for contact in lstContact:
    intCounter+=1
    print '=== Contact %s ==='%intCounter
    print 'partOfName:\n      %s'%contact.partOfName
    print 'email:\n      %s'%contact.email
```

上述程式碼片段會從所有繼承於 Contact 的類別中，找出 Email 結尾是 nctu.edu.tw 的資料物件，其中 Contact.objects.get 是進行物件查詢的關鍵敘述句，其語法有點類似資料庫 SQL Command 的 SELECT 命令. 若翻譯成 SQL 命令會類似下面的敘述句:

```
select * from Contact where email like '%nctu.edu.tw'
```

找出符合的資料物件之後，資料物件可能有兩種型態，分別是: GmailContact 或 ThunderBirdContact. 接下來是用一個 for 迴圈，把找到的資料物件其 partOfName 與 email 屬性列印在螢幕上顯示. 執行結果如下:

=== Contact 1 ===
partOfName:
"GmailContact_name":"Bowen Chiu",
email:
"GmailContact_email":"bowen@nctu.edu.tw",
=== Contact 2 ===
partOfName:
"ThunderBirdContact_givenName":"Kao",
"ThunderBirdContact_sn":"Gloria",
"ThunderBirdContact_cn":"Gloria Kao",
email:
"ThunderBirdContact_mail":"gloria@nctu.edu.tw",

執行結果顯示兩筆資料物件，第一筆 Contact 1 是屬於 GmailContact 類別的資料物件, contact.partOfName 這個字串的值是 "GmailContact_name":"Bowen Chiu"，我們可以看到這是一個 key:value pair, 前面的 key 爲 GmailContact_name，可以讓開發者知道後面的值 Bowen Chiu 隸屬於 GmailContact_name.

第二筆 Contact 2 的資料物件是屬於 ThunderBirdContact 類別，所以 contact.partOfName 代表的意義比較複雜, contact.partOfName 的值在此相當於一個陣列，用逗點隔開而列出 ThunderBirdContact 當中 givenName, sn, cn, 三個屬性的 key:value pair，這是因爲這三個屬性在 ThunderBirdContact 類別當中都繼承了 contact.partOfName，所以在合併查詢的結果會把三個屬性的 key:value pair 全部都放入 contact.partOfName 的查詢結果.

如果有需要對每種類別提供不同的資料顯示方法，開發者只要根據回傳資料物件的 key:value pair 就能判斷出該物件原本屬於哪個類別，所以在合併查詢顯示時，可以針對不同的類別適當對顯示格式作特殊調整. 或者如果只是想要做簡單的應用，也可以乾脆不調整顯示格式，直接把找到的所有 key:value pair 原原本本的顯示在螢幕上.

我們由此就可以看出 SOF 在跨異質通訊錄查詢上的威力，只需要簡單的於註解上描述屬性之間的繼承關係，就可以分辨屬性名稱不同但意義相同的情況，一次把相符資料全部找出來，非常的便利.

查詢出語意合法或非法的資料集合

由於 RDF 可以針對物件資料彼此的語意關係進行限制，有可能某些資料物件是不符合 RDF 指定的限制條件，而我們在一些情況下，會需要分辨哪些資料是合法或非法的.

舉例來說，如果我們希望能夠找到不重複的郵寄名單，但是 GmailContact 與 ThunderBirdContact 中有可能包含了重複的聯絡人資料，此時就可以運用當初定義的 #owl:InverseFunctionalProperty Contact_email 這個 SOF 敘述句. 在這個敘述句當中限制了 Contact_email 屬性的值必須屬於唯一的 Contact，也就是任意兩個 Contact 資料物件不應該擁有相同的 Email 值. 如果兩個以上的 Contact 資料物件中有相同的 Email 屬性值，在語意上會被視爲其實是同一個 Contact. 藉由這樣的限制，我們可以利用 getInvalid() API，找出有哪些資料物件違反了此原則，並且顯示在螢幕上. 請看以下的程式碼片段:

```
lstContact=Contact.objects.getInvalid()
for contact in lstContact:
    print 'partOfName:%s'%contact.partOfName
    print 'phoneNumber:%s'%contact.phoneNumber
    print 'invalid reason:%s'%contact.getInvalidReason()
```

第一筆非法資料如下:

```
partOfName:
  "GmailContact_name":"Bowen Chiu",
phoneNumber:
  "GmailContact_phone":"+88635727001",
  "GmailContact_mobile":"+886922387002",
invalid reason:validation fail ->
owl:InverseFunctionalProperty(Contact_email,Contact)
```

第二筆非法資料如下:

```
partOfName:
  "ThunderBirdContact_givenName":"Chiu",
  "ThunderBirdContact_sn":"Bowen",
  "ThunderBirdContact_cn":"Bowen Chiu",
phoneNumber:
  "ThunderBirdContact_telephoneNumber":"+88635727001",
```

"ThunderBirdContact_homePhone":"+88638885003",
"ThunderBirdContact_mobile":"+886993288002",
invalid reason:validation fail -> owl:InverseFunctionalProperty Contact_email

我們可以看出來，非法資料物件第一筆是屬於 GmailContact，第二筆是屬於 ThunderBirdContact，雖然這是兩筆不同的資料物件，不過因爲他們的 Email 屬性值相同，所以被判定爲非法資料，在語意上其實應該是同一個 Contact 才對. SOF 成功的橫跨兩種不同通訊錄，找出語意上重複的資料物件，這種功能通常是在列印不重複的郵寄名單時非常重要，可以避免把兩份相同的物品寄送給同一個人. 而 contact.getInvalidReason() 命令甚至還可以顯示該資料物件違反 RDF 語意限制的原因，這樣可以讓開發者進一步進行衝突處理，例如刪除多餘的資料物件，或者乾脆把兩個資料物件合併成一筆.

若要讀取所有合法資料物件的命令，則可以運用 lstContact=Contact.objects.getValid() 把所有合法資料物件放入 lstContact 陣列當中，而這些全都是 Email 屬性不重複的 Contact 資料物件.

6.Future Work

本論文所提出的 SOF 架構可以將資料物件自動發佈爲 RDF 格式，還可以進行異質資料來源合併查詢. 雖然這是非常有威力的類別物件發佈流程，但是由 SOF 概念所延伸的工具模組還不夠齊全，尤其是在 IDE 開發工具的自動化支援部分，未來若是可以結合各種語言的 IDE 開發環境，支援 auto complete，以及動態語法檢查，甚至圖形化表示類別之間的語意關係，讓語意圖跟程式碼之間彼此可以相互同步，利用圖像化的開發環境來指定類別的語意關係，這些都是運用 SOF 概念可以逐步達成的目標.

另外，如果 SOF 敘述句語法寫錯了，或者 SOF 敘述句彼此之間有語意衝突，未來都必須有更嚴謹的檢查工具，自動進行語意衝突的檢查，將檢查結果回報給開發者進行處理. 相信在 SOF 概念逐步受到重視並導入之後，相關開發工具的支援會越來越完整.

7.Conclusion

本論文針對資料物件發佈爲 RDF 文件提供了全自動轉換的 SOF 解決方案，過去要發佈資料物件爲 RDF 時，通常必須手動將所有的資料物件轉換爲 Triple Store 的儲存方式才能發佈，而本論文採用 SOF 作爲內嵌在程式碼中修飾類別或屬性的語意，加強了物件導向語言在類別或屬性關係描述上的不足，除了能夠隨時保持類別關連性描述與程式碼同步, SOF 也提供了程式語言中立的工具組，可支援各種程式語言，不會受到特定語言實作的限制. SOF 提供了一個非常直覺的語意網發佈流程，並允許跨異質資料來源進行查詢，成功結合了物件導向程式設計跟語意網雙方的優點.