

# Chapter 1

## Introduction

### 1.1 Background

The rapid development in information technology has facilitated the worldwide access to information. Accordingly, a well-designed architecture is required to effectively and efficiently coordinate the dissemination of a large amount of information over the Internet. Having received considerable attention, a digital library (DL) represents an Internet-based architecture capable of accessing information from anywhere. From the perspective of library users, DL is anticipated to provide a transparent access, which means DL search must be semantic to heterogeneous data and unaware for the distribution of DL. Hence, recent research of DL focuses on an integration architecture to access distributed and heterogeneous materials.



The main issue of DL integration is called interoperability. Research in DL interoperability relies on a novel architecture to solve the technical, semantic and legal aspects of interoperability [43]. Paepcke et al. categorizes the research of DL interoperability into four issues [50]. The first refers to the management of information. Early federated systems created global modeling (such as Dublin Core) to simplify the integration of DL. Then the research focused on creating mediation when integrating content. The second refers to the presentation to users. Enabling technology was on distributed display, like bitmap, postscript...etc. Hereafter, networked documents and distributed animation drew the attention to present information in different environment. The third refers to communication among parts of the overall system. This research aimed to the component interconnects and currently to the naming knowledge interchange. The fourth is for the operation and control of a

system's action. The present remote computation technology, like CORBA, DCOM and Java, are used to improve the coordination among independently executing components. Conclusively, as the long-term goal of interoperability, DL would operate by allowing independent model, formal and language to declare their function arbitrarily and engage in peer-to-peer interaction.

From the perspective of autonomy, the two sides in the spectrum of DL interoperability are centralized and decentralized. A centralized DL architecture reconciles DL by concordant middleware. This architecture leaves interoperability machinery outside independent DL but requires large amount mediation to transform different DL information into a canonical model. In contrast to centralization, a decentralized architecture interacts DL with a language which communicates the semantics, structure and operations of all materials among different DL without prior arrangement and mediators. Apparently, it appears that these two kinds of integration architecture come into one common factor – both of them use metadata to interoperate heterogeneity. In a centralized architecture, metadata are used for information description and as a basic unit to translate different materials. In a decentralized architecture, metadata work as a communication basis among different DL.

General speaking, metadata are treated as sub-ordinations of DL information and managed in two layers – physical object layer and metadata layer. Physical object layer represents real objects stored in DL and metadata layer constitutes the composition and semantics to represent how objects are composed in physical object layer. However, this two layer architecture leads DL integration into critical challenges. The first challenge is the lack of structure consideration in conventional representation in physical layer. The structure information of metadata contains implicit semantics to DL activities like data search, semantic inference, and is helpful to integrate data. The second challenge is distributed DL complicate the DL integration. This challenge occurs because the distributed metadata are inconsistent in

formats and semantics. To overcome these challenges, a richer data model and additional functions are required to represent and manage distributed metadata.

## **1.2 Related work**

### **1.2.1 Digital library architecture**

DL integration has been investigated in much research [1, 5, 6, 9, 18, 33, 34, 36, 41, 46, 55]. This research focuses on how varied services are integrated and how to alleviate the effort in DL collaboration [35]. To consider the autonomy infringement in cooperating DL, it falls into two types: external mediation (centralization) and specification-based interaction (decentralization) [50]. External mediation approach locates interoperability machinery outside independent DL and reconciles DL by a concordant middleware. The middleware (sometimes called proxy or mediator) receives native materials and translates them into canonical format. Research including Digital library Initiative (DLI) Phase I & II undertakes broad research on DL infrastructure [4, 15]. Stanford University participates in DLI I & II and develops a proxy-based system, called InfoBus, to extend the current internet protocols with a suite of higher-level information management protocols [50]. Although mediation approach is particularly strong and easy in supporting the criteria of autonomy, the lack of use and scalability makes it copious to add new components.

Alternatively, specification-based interaction describes the language that communicates the semantics, structure and operations of all materials among different DL components without prior arrangement and mediators. For example, the Agent Communication Language (ACL) defined Knowledge Interchange Format (KIF), Knowledge Query and Manipulation Language (KQML) for interaction. Other peer-to-peer architecture also uses languages to communicate with each other [14, 52]. In the specification-based interaction architecture, much functionality required in DL is encapsulated as population of modular, goal-oriented,

specialized agents [13]. The Michigan University provided agent technology to construct DL [12, 57]. Nevertheless, specification-based solution specializes in autonomous owing to the separation of functionality and data, but lacking of centralized facilities complicates the interactive communication and thoroughly degrades the performance.

## **1.2.2 Metadata model**

Metadata enhancing the power to model DL materials and provides additional semantic and structural annotations for original data [38]. Neuroth et al. categorizes metadata into eight types according to the nature of described digital objects [47]. Many researches have addressed the using of metadata to describe resources in DL, include Warwick Framework [37], Dublin Core (<http://purl.oclc.org/dc/>), Resource Description Framework (RDF, <http://www.w3.org/TR/PR-rdf-schema/>), and several others [5, 8, 31, 40]. The Warwick Framework is a conceptual model of metadata that aggregates metadata packages into containers and then relates these packages to each other. This framework separates the management and responsibility of specific metadata and allows access to various different sets of metadata. The container technology has influenced subsequent developments, including Dublin Core and RDF. Dublin Core, which defines 15 basic metadata elements, focuses mainly on resources for Internet-based applications. RDF provides a standard means of representing metadata by XML, employing statements to describe properties of and relationships among items on the Web. Many studies applied RDF to resource discovery [30]. Multimedia Description Framework (MDF) uses RDF to describe multimedia contents [21].

The structure information of metadata is emphasized in the research of automatic metadata extraction. To extract metadata automatically from structured data or documents, it requires a complete model to describe the structure and an algorithm to extract information as metadata. The tag tree patterns are used to represent structure of metadata in Miyahara's

approach and a data mining approach is used to extract frequent structure in web documents [44]. Hirokawa et al. adopts a tag sequence to describe structure of web document and extracts metadata by template approach [20]. Han et al. analyzes the structure of document by classifying the header of research papers into 15 classes [19]. Arasu et al. proposes a formal model to represent structure data and builds template to extract metadata [2].

### 1.2.3 Metadata architecture

Sharing metadata architecture in distributed DL drives the use of metadata to expand semantic relationships [7]. Open Digital Library [16], INRIA [3] and 5S Model [17] apply metadata to describe, derive and federate services in distributed DL. To efficiently manage metadata in distribution environment, considerable attention paid towards metadata architecture has enabled metadata management across distributed DL services [48, 49]. Well-designed metadata architecture should support not only the effective manipulation of native metadata but also the autonomous management of distributed metadata [1, 11, 42]. In mediation schemes, metadata describe and translate information. The metadata architecture proposed by Stanford University, called InfoBus, is one realization of mediation architectures. Infobus uses five service layers to enable uniform access to distributed heterogeneous information resources and services [4, 53]. In InfoBus, *Attribute Model Proxies* model metadata as first class objects, and *Metadata Repository* deposits and indexes metadata that describe which metadata can be provided. These two kinds of metadata help Stanford DL to seamlessly communicate their metadata. Metadata in DL interactive architecture also attempts to facilitate metadata communication. The agent-based architecture helps DL agents to locate desired metadata. The University of Michigan proposed an agent-based architecture, called University Michigan Digital Libraries (UMDL). UMDL refines the basic agent architecture to satisfy the needs for an open information economy. UMDL expresses agents using ontological semantics and employs metadata to represent information. UMDL communicates agents with

each other by Knowledge Query and Manipulation Language.

### 1.3 Goal

The previous section has mentioned the interoperability is treated in four aspects: the management, presentation, communication and operation. However, rare of them discusses the interoperability from the standpoint of metadata. There are four factors to consider metadata as a critical element in coordinating DL:

- The rich expression power solves the semantic inconsistency.
- The easy transformation interacts metadata in diverse formats.
- The structure information derives semantic relationships.
- The economic aspect in structure facilitates integration.

The four trusts motivate this dissertation to consider metadata as the critical element to integrate DL. *Model-Extraction-Query* (abbreviate to *MEQ*) model conceptualizes the access of DL from data, service and semantics view respectively in this dissertation. Therefore, a novel metadata architecture called *M-Architecture@DL* is proposed to integrate DL seamless by managing metadata. *M-Architecture@DL* extends conventional two-layer metadata into three-layer metadata architecture. In *metadata modeling layer*, a language is required to represent metadata and solve the metadata inconsistency in distributed DL. In the following section, *Metadata Model Language (MML)* is introduced with rich modeling power and translation mechanism. *MML* describes the data format, service capabilities, structure presentation and translation rules among different metadata. The language syntax of MML is XML (eXtensible Markup Language), and the data model of MML is a simplified data model revised from RDF with further extending its functions. Based on MML, the second layer *data extraction layer* is responsible for collecting data from distributed DL and encapsulated result into *MML*. In this layer, data are automatically extracted according to the common structure of

individual DL service. An automatic application called *Metadata Extractor* is presented. The third layer, called *semantic query layer*, exploits structure information to derive relationships among metadata, which can solve the semantic ambiguity in DL search. In this layer a *Content and Service Inference Model* is proposed. Summarily, the separation of *M-Architecture@DL* into three-layer architecture helps the DL integration independently and obtains more permanent and explicit knowledge in DL access. Independent DL is easy to declare its capabilities and communicate with each other.

The aim of this dissertation is to establish novel metadata architecture to integrate DL. This work focuses on two research topics:

- Research on metadata representation: Propose novel data model to represent metadata semantically. This model elaborates the transformation of metadata.
- Research on metadata integration architecture: Propose DL architecture to manage metadata in distributed DL. This architecture facilitates the management of metadata from model, semantics, query and integration to seamless access DL.

The rest of this dissertation is organized as follows. Chapter 2 describes three-layer metadata architecture to manage metadata in distribution environment. Chapter 3 presents metadata model to represent the content, the semantics and the structure. Chapter 4 proposes a data extraction algorithm to collect DL metadata from semi-documents with common structure. Chapter 5 presents Content and Service Inference Model to semantically search by inferring structural relationships among metadata. Chapter 6 conducts experiments in virtual union catalog system. Chapter 7 draws conclusion and future direction.

## Chapter 2

### Digital Library Integration Architecture

#### 2.1 Conceptualization

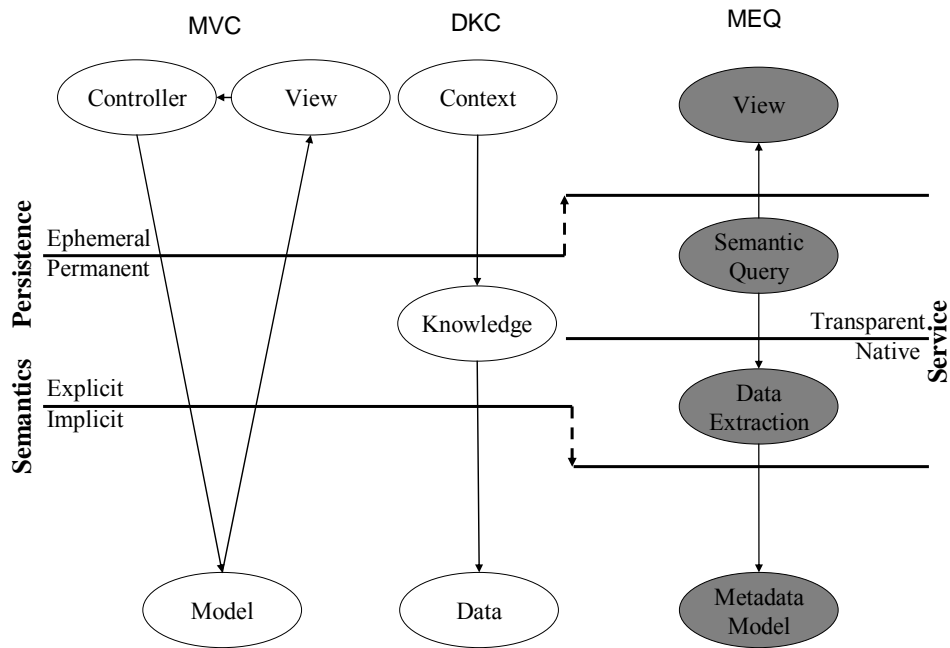


Figure 1. Application model of DL

Conventional access to collaborative services is *Model-View-Controller (MVC)* model (see Figure 1). In *MVC* model, a user contacts *Controller* directly to obtain *Model* and return as *View*. In this model, no information keeps after the access finished. To obtain more useful information in the process of data access, *Data-Knowledge-Context (DKC)* model creates knowledge layer between context and data to extract more knowledge [28, 51]. Explicit knowledge extracted from each access maintains permanently and can be reused even the access finished. However, in DL distributed environment, knowledge level can be further extended from the perspective of metadata. Sharable metadata architecture allows to proceed



metadata directly in place of the original content. The later binding of content enhances the efficiency when access large amount of digital content. Moreover, semantic query across heterogeneous DL provides transparent access to automatically extract content from unspecified services. The *Model-Extraction-Query (MEQ)* model can forward persistence line and backward semantics line to obtain more permanent and explicit knowledge than DKC model. The separation from services creates a new line to distinguish the service transparency and native service extraction. Therefore, based on *MEQ*, DL integration architecture called *M-Architectre@DL* is proposed to help distributed DL access.

## 2.2 Scenario

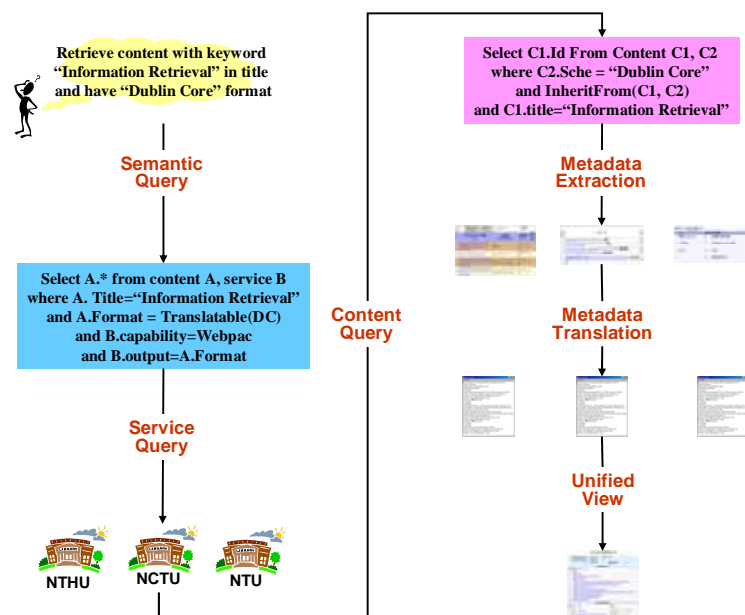
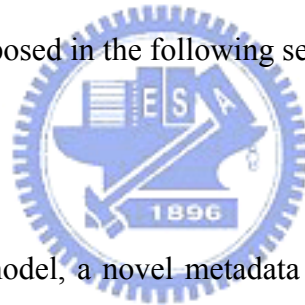


Figure 2. The scenario to query across distributed DL

The main activity of DL is to search information from services (like Webpac) and combines the result to user. Therefore, the purpose of DL integration is to provide a semantics query to retrieve information transparently. Figure 2 displays a scenario to query across distributed and heterogeneous DL. A user wants to retrieve content with specified format and

keyword. Intuitively, the user is now required to know where the desired content and services are. The user is only required to describe what the nature of the content (such as the format, semantics, etc.), what the functionality of the services (such as the integration service, output with Dublin Core format, etc.) and what are the specified criteria (such as the “Information Retrieval” in title field). The semantic query takes the responsibility to find out where the services are automatically and then dispatches the content query to individual service. Next, information for each local service are extracted automatically and encapsulated into metadata. Moreover, the extracted metadata have heterogeneous formats and are required to translate into the same format. Metadata with different semantics are needed to clarify in this step. Finally, metadata are integrated as a single view and return to the user. The scenario reveals the rationales and research topics to design well metadata architecture in the DL integration. Three-layer architecture is proposed in the following sections.

### 2.3 *M-Architecture@DL*



By following the *MEQ* model, a novel metadata architecture for DL integration, called *M-Architecture@DL*, is proposed herein (*M* stands for *Metadata*). *M-Architecture@DL* integrates DL by facilitating semantics search from the perspective of metadata [26, 27]. This architecture contains three layers, *metadata modeling layer*, *data extraction layer*, and *semantic query layer* (see Figure 3). *Metadata modeling layer* provides rich expression power to represent metadata and their structure. A translation mechanism is presented to solve format heterogeneity. In this layer, *Metadata Modeling Language (MML)* is implemented. *Data extraction layer* extracts data from DL services according to the common structure. An extraction algorithm is proposed to automatically retrieve data without prearrange with librarians and structuralize the result into MML format. In this layer, *Metadata Extractor* is implemented. *Semantic query layer* derives fifteen relationships between DL services and content to enhance the query semantics. Manipulation operations are introduced in the query

statement to solve semantic heterogeneity. In this layer, Content and Service Inference Model (CSIM) are proposed.

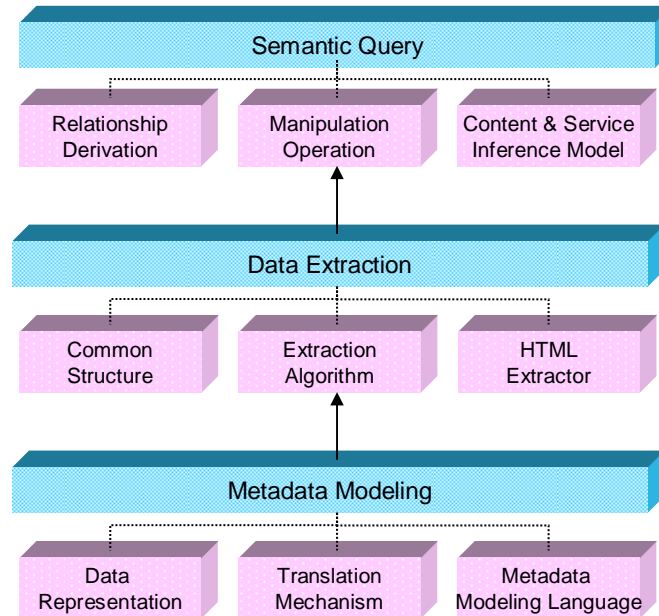


Figure 3. *M-Architecture@DL*

*M-Architecture@DL* exchanges information easily and transparently in distributed DL. Each layer thoroughly separates responsibility to manage DL metadata and accomplish the task as follows:

- *Metadata Modeling Layer.* This layer contains rich modeling power to describe metadata semantically. Metadata structure are constructed by two constructors – tuple and set constructors. Based on the structure expression, translation is easy to elaborate by adding operations among metadata in the translation template. In this layer, a metadata language called *Metadata Modeling Language (MML)* is implemented. Toolkits for MML edition and translation are provided to help MML metadata manipulation. Format interoperability is obtained in this layer.
- *Data Extraction Layer.* This layer collects data from distributed DL and encapsulated

result into MML metadata. In this layer, data are extracted automatically according to the common structure for individual DL service. A transparent metadata extractor called *Metadata Extractor* is provided to extract metadata from semi-structure documents (like HTML documents). This extractor extracts data transparently and enhances protocol interoperability in DL integration.

- *Semantic Query Layer*. This layer provides semantics query to obtain metadata with as much semantics. In this layer, additional data structures (ontology tables) are created to maintain semantics from DL and support semantic manipulation. The semantics query across distributed DL is called *Content and Services Inference Model (CSIM)*. CSIM derives 15 relationships between two DL factors, content and services, and defines functions to manipulate these relationships. Applying CSIM in DL significantly improves semantic queries and alleviates the administrative load when query from DL. Semantic and service interoperability are achieved in this layer.

The separation of *M-Architecture@DL* into three-layer architecture contains several advantages in DL integration. Metadata modeling layer provides rich modeling power and translation mechanism which enhance the format interoperability. Data extractor layer that extracts metadata transparently can interoperate DL without the pre-arrangement to access each DL affiliation. Therefore, access protocol interoperability is achieved. Semantic query layer constructs additional ontology structures to clarify semantics from DL and derive relationships from different services. The semantics query across distributed DL is called *Content and Services Inference Model (CSIM)* to derive 15 relationships between two DL factors, content and services, and defines functions to manipulate these relationships. Applying CSIM in DL significantly improves semantic and alleviates the administrative load when query from DL. In this layer, semantic and service interoperability are achieved.

## Chapter 3

### Metadata Modeling

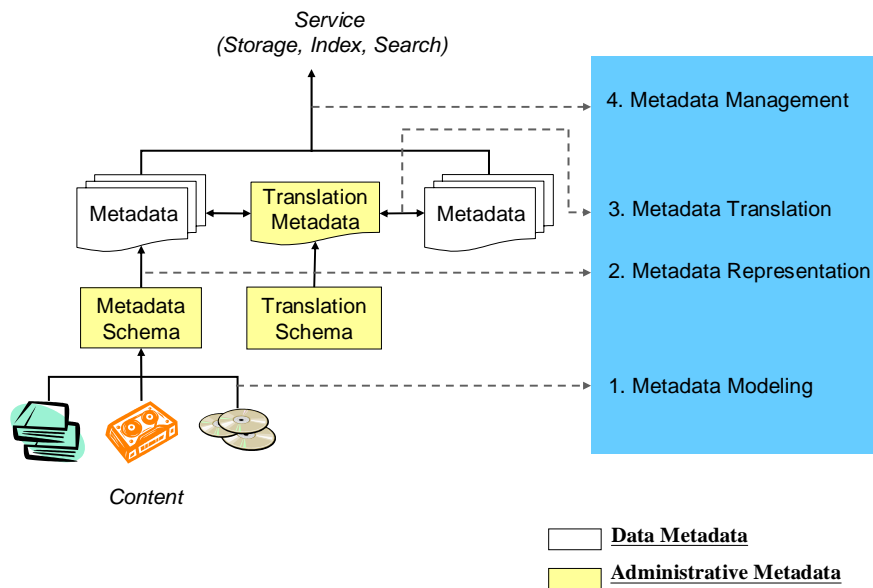


Figure 4. Construction of metadata

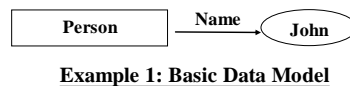
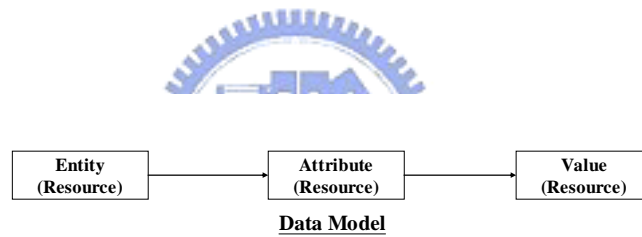
The construction of metadata contains four steps (see Figure 4). To begin with metadata modeling, the original content is initially described by a schema (*Metadata Schema*), which shows how the metadata is composed (*Metadata Structure*). Hereafter, the content are represented as metadata according to the metadata schema. Furthermore, different metadata can be translated according to translation rules (*Translation Metadata*). In this model, metadata is categorized into two types: data metadata and administrative metadata. The metadata encapsulated content information is called data metadata. Administrative metadata is an intermediation for system and only available by system. For example, translation metadata is one kind of administrative metadata to guide how one metadata can be translated into another. Finally, both data and administrative metadata are used by metadata management to

improve interoperability. In chapter 2, an integration architecture is introduced to interoperate DL seamlessly.

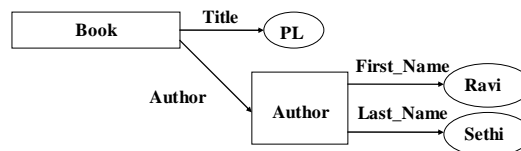
This chapter aims to construct a well-defined metadata model to describe metadata semantically and elaborates metadata when they are translated with each other. In this chapter, a metadata model is formally defined. This metadata model contains three major specifications:

- A data model with formal description to model real-world entities.
- A structure expression to derive semantics
- A translation mechanism to conquer interoperability gap

### 3.1 Data model



**Example 1: Basic Data Model**



**Example 2: Hierarchical Data Model**

Figure 5. Data model (Extend from Warwick Framework)

The data model of metadata is constructed by three types of *Resources* - *Entity*, *Attribute* and *Value*, which is extended from Warwick Framework [37] (see Figure 5) and referred to

the definition of RDF. For example 1 in Figure 5, a Person (*Entity*) has a Name (*Attribute*) called John (*Value*).

The basic unit in data model is *Resource*. Each resource is annotated by ten optional properties. This definition standardizes the metadata communication and management. The formal definition of data model is as follows:

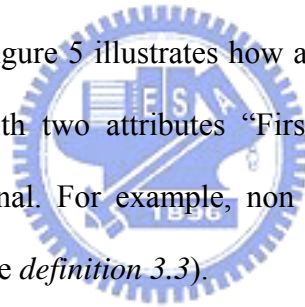
**Definition 3.1 (*Resource*):** A *Resource*  $R=\{Id, Name, Type, Doc, Ver, MO, Obl, Def, Lang, RA\}$  is the fundamental unit of metadata with ten basic properties for the description of metadata where,

- *Id* is the unique identifier of a *Resource*. *Id* is generated from a handle server and can be determined as a combination of primary key referred in relational database, a URI or a naming service which can identify a resource.
- *Name* is the name of a *Resource*. *Name* is an alternative way to reference *Resource* when *Id* reference is non-meaningful for metadata reader. The reference for *Name* can be hierarchical.  $Name_1.Name_2.....Name_n$  represents the hierarchical reference for a resource  $Name_n$  insides the scope of  $Name_{n-1}$  to  $Name_1$ .
- *Type* is the schema type of a *Resource*. *Type* means how to refer each component of the source described in *Metadata Schema*.
- *Doc* is the description of a *Resource*. *Doc* contains the comment of the resource for readability.
- *Ver* is the version of a *Resource*. *Ver* indicates the version of the resource.
- *MO* is the maximal occurrence of a *Resource*. *MO* represents the repeatability of the resource. *MO* is an important property to construct *Metadata Structure*.
- *Obl* is the obligation of a *Resource*. *Obl* indicates if the resource is required to be presented.

- *Def* is the definition of a *Resource*. *Def* contains a list of ontology terms that clearly represent concepts and the essential nature of the resource. Moreover, the domain range of this is also described here.
- *Lang* is the language of a *Resource*.
- *RA* is the registry authority of a *Resource*. *RA* describes where the resource registers. *RA* can be the repository of the resource or somewhere the resource is in residence.

The *Name* and *Identifier* properties are treated as the identification of resources. The both ways of hierarchical reference to resource, *Name* and *Identifier*, are allowed to increase the flexibility for metadata access. *Type* property associates metadata to their schema. The repeatability of metadata is subsumed in *MO* property.

The second example in Figure 5 illustrates how a hierarchical resource “*Book*” refers to another resource “*Author*” with two attributes “*First\_Name*” and “*Last\_Name*”. Notably, these ten properties are optional. For example, non properties except *Id* are necessary to represent value of metadata (see *definition 3.3*).



**Definition 3.2 (Schema):** A Resource of Schema *S* is a triple  $\{Ent, Incs, Atts\}$  to represent how metadata is constructed:

- $Ent \subseteq Resource$ , *Ent* depicts information of Schema. In *Ent*, the *Type* attribute contains the type name to be referred in Metadata
- $Incs \subseteq (Atts_a, Atts_c)$ , *Incs* represents the relationships between *Ent* and *Atts*. Each pair  $(Atts_a, Atts_c)$  indicates that *Atts\_c* is the subtype of *Atts\_a*. *Incs* is assumed to be acyclic.
- $Atts \subseteq Resources$  is the set of Attribute. In *Atts* resource, the *Type* property represents the name of *Atts*. The *Type* property of *Atts* can be another *Schema*. A precedent symbol “@” concatenating with a value indicates that the schema exists in another schema with the value behind @.

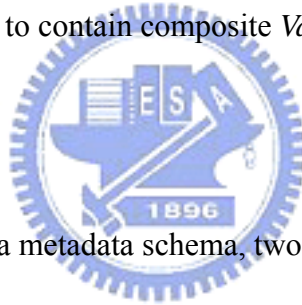


A *Schema* represents how a metadata is encrypted. In *Schema*, each *Atts* can refer to another *Resource*, which means this data model is hierarchical. However, acyclic reference is forbidden. In addition, because this data model is referred from RDF, this implies that instances of *Schema* can be expressed by the Entity-Relationship model as well.

**Definition 3.3 (Metadata):** A *Metadata* is a triple  $\{Ent, Sche, Values\}$  to represent how to encapsulate data by *Schema*:

- $Ent \subseteq Resource$ , *Ent* describes information of *Metadata*. The value of *Type* attribute in *Ent* is the name of *Metadata*.
- $Sche \subseteq Schema$ , *Sche* represents the *Schema* associate with the *Ent*.
- $Values \subseteq (Attribute, Resource)$ , each pair  $(A_l, V_l)$  represents that  $V_l$  is the value of *Attribute*  $A_l$ .  $V_l$  is allowed to contain composite *Values* corresponding to *Sche*.

### 3.2 Structure expression



To denote the structure of a metadata schema, two kinds of constructions are defined [2]:

**Definition 3.4 (Structure):** Given a *Schema*  $S$ , the structure of  $S$  can be represented by the following two constructors:

- **Tuple constructor (TC).** A *tuple constructor*  $TC$  of a *Schema*  $S$  is an ordered list constructed by the union of  $S.Incs$  with the same precedent value in  $S.Incs.Atts_a$ . For example, the set of  $S.Incs$   $(\tau, c_1), (\tau, c_2), \dots, (\tau, c_n)$  are represented as  $TC_\tau = \{c_1, c_2, \dots, c_n\}_\tau$ . The subscript  $\tau$  is the name of  $TC$ .
- **Set constructor (SC).** A *set constructor*  $SC$  of a *Schema*  $S$  is an occurrence type with the same  $S.Atts.MO$  larger than one. For example, the set of  $S.Atts$ ,  $c_1.MO=i, c_2.MO=i \dots c_n.MO=i$ , are represented as  $\langle c_1, c_2, \dots, c_n \rangle_\tau^i$ . The subscript  $\tau$  represents the name of  $SC$  and  $i$  represents the maximum occurrence.

On the other hand, the *tuple constructor* can be obtained from the union of *Incs* with attribute  $MO = 1$  in the same attribute level. The *set constructor* can be obtained from the attributes when their  $MO$  larger than 1. The  $\tau$  is given as the name in *tuple* and *set constructors*.

The representation of *Structure* is to facilitate the reference to some part of *Schema*. Referring to *Schema* easily is an important basis for schema translation. The *Structure* of *Schemas* in two example of Figure 5 are  $\{Name\}_{Person}$  and  $\{Title, \langle First\_Name, Last\_Name \rangle^n_{Author}\}_{Book}$  respectively.

### 3.3 Translation service

**Definition 3.5 (Translation Service):** Given two *Schema*  $S_{Source}=(\cup TC_i \cup SC_i)$  and  $S_{Dest}=(\cup TC_j \cup SC_j)$ :

- **Translation Service (TS):** A *translation service* contain a set of translation rules such that  $S_{Dest}. TC_i = op (S_{Source})$ .  $op$  represents a set of metadata operations that can manipulate the source metadata into the destination.

The  $op$  is supported depending on different schema types. For example, the numeric type attributes can support math operations, like add, subtract, multiple, divide... etc.; the string type attributes can support concatenate and subtract operations to translate source schema into destination. As in Figure 5, the translation of book authors into person type with comma between different authors can be expressed as:

$$TC_{Person.Name} = SC_{Author}((SC_{Author}.First\_Name+” “+SC_{Author}.Last\_Name)+”, “).$$

### 3.4 Implementation (Metadata Modeling Language – MML)

```
MML ::= ['<'mmlPrefix'>' resources* '</'mmlPrefix'>']
mmlPrefix ::= 'Model' | 'Model_Instance'
resources ::= properties* | attributeItems* | stringvalue
properties ::= '<'propertyTag'>' propertyValue '</'propertyTag'>'
propertyTag ::= 'MML_Name' | 'MML_Type' | 'MML_Doc' | 'Identifier' |
    'MML_Version' | 'MML_RA' | 'MML_Language' |
    'MML_Definition' | 'MML_Obligation' | 'MML_MO'
propertyValue ::= 'string' | 'integer' | 'float' | 'boolean' | stringvalue
attributeItems ::= '<attributeTag'>' resource '</attributeTag'>'
attributeTag ::= 'Attribute' | attributename
```

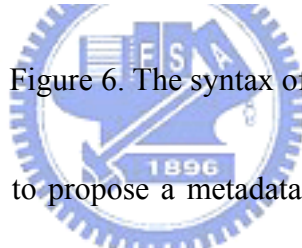


Figure 6. The syntax of MML

The aim of this section is to propose a metadata language to model real-world entities and to improve interoperability. *Metadata Modeling Language (MML)*, which owns rich modeling power and translation mechanism, is implemented to improve interoperability among DL. The data model of MML metadata is hierarchical and applies the container technology. The fine-grained data model elaborates information with semantics and translation mechanism to exchange DL content. MML is designed as an interoperability basis to integrate DL and contains sufficient information for metadata management. The language syntax of MML is XML (eXtensible Markup Language), and revised from RDF with further extending its functions. A MML metadata contains a ten properties meta-information adopted from ISO 11179 specification and standardization of data elements (in section 3.1). Figure 6 shows the syntax of MML metadata.

Book Schema	Author Schema	Book Metadata
<pre> &lt;Model&gt; &lt;MML_Name&gt;Book&lt;/MML_Name&gt; &lt;MML_Type&gt;Book&lt;/MML_Type&gt; &lt;MML_Doc&gt;This is a model of a book&lt;/MML_Doc&gt; &lt;Identifier&gt;Book_ID1&lt;/Identifier&gt; &lt;MML_Version&gt;Version_1&lt;/MML_ Version&gt; &lt;MML_RA&gt;NCTU_DB&lt;/MML_RA&gt; &lt;MML_Language&gt;English&lt;/MML_L anguage&gt; &lt;MML_Definition&gt;Unlimited&lt;/MML _Definition&gt; &lt;MML_Obligation&gt;Unlimited&lt;/MM L_Obligation&gt; &lt;MML_MO&gt;Unlimited&lt;/MML_MO&gt; &lt;Attribute&gt; &lt;MML_Name&gt;Title&lt;/MML_Name&gt; &lt;MML_Type&gt;String&lt;/MML_Type&gt; &lt;MML_Doc&gt;The title of a book&lt;/MML_Doc&gt; &lt;/Attribute&gt; &lt;Attribute&gt; &lt;MML_Name&gt;Author&lt;/MML_Nam e&gt; &lt;MML_Type&gt;@Author&lt;/MML_Typ e&gt; &lt;MML_Doc&gt;Describe the writer of a book&lt;/MML_Doc&gt; &lt;/Attribute&gt; &lt;/Model&gt; </pre>	<pre> &lt;Model&gt; &lt;MML_Name&gt;Author&lt;/MML_Nam e&gt; &lt;MML_Type&gt;Author&lt;/MML_Type &gt; &lt;MML_Doc&gt;This is a model of an author&lt;/MML_Doc&gt; &lt;Identifier&gt;Author_ID&lt;/Identifier &gt; &lt;MML_Version&gt;Version_1&lt;/MML _Version&gt; &lt;MML_RA&gt;NCTU_DB&lt;/MML_RA&gt; &lt;MML_Language&gt;English&lt;/MML_ Language&gt; &lt;MML_Definition&gt;Unlimited&lt;/MM L_Definition&gt; &lt;MML_Obligation&gt;Unlimited&lt;/MM L_Obligation&gt; &lt;MML_MO&gt;Unlimited&lt;/MML_MO &gt; &lt;Attribute&gt; &lt;MML_Name&gt;First_Name&lt;/MML_ Name&gt; &lt;MML_Type&gt;String&lt;/MML_Type&gt; &lt;MML_Doc&gt;The first name of an author&lt;/MML_Doc&gt; &lt;/Attribute&gt; &lt;MML_Name&gt;Last_Name&lt;/MML_ Name&gt; &lt;MML_Type&gt;String&lt;/MML_Type&gt; &lt;MML_Doc&gt;The last name of an author&lt;/MML_Doc&gt; &lt;/Attribute&gt; &lt;/Model&gt; </pre>	<pre> &lt;Model_Instance&gt; &lt;MML_Type&gt;Book&lt;/MML_Type&gt; &lt;Identifier&gt;Book_ID2&lt;/Identifier&gt; &lt;Book&gt; &lt;Title&gt;PL&lt;/Title&gt; &lt;Author&gt;Author_ID2&lt;/Author&gt; &lt;/Book&gt; &lt;/Model_Instance&gt;  &lt;Model_Instance&gt; &lt;MML_Type&gt;Author&lt;/MML_Type&gt; &lt;Identifier&gt;Author_ID2&lt;/Identifier &gt; &lt;Author&gt; &lt;First_Name&gt;Ravi&lt;/First_Name&gt; &lt;Last_Name&gt;Sethi&lt;/Last_Name&gt; &lt;/Author&gt; &lt;/Model_Instance&gt; </pre>

Figure 7. Example of MML schema and metadata

According to the data model in previous section, MML can represent into MML Schema and MML Metadata. MML Schema represents the metadata attribute model as a first-class object. MML metadata is the actual data object encapsulated by MML Schema format. Referring to the example 2 of Figure 5, the MML metadata are illustrated in Figure 7. In Figure 7, attribute “Title” was embedded into the “book” resource to be a basic type of string. All the properties are optional to simple attribute. The second attribute ‘Author’ is a hierarchical reference to other resource by “Type” property. The precedent character ‘@’ refers to a schema which name is ‘Author’.

In MML, interoperability is achieved by iterative translation. Figure 8 shows how substantial data can be iteratively translated into the target format via translation service. This characteristic implies that the source metadata are not required to translate into target ones directly if there is sufficient translation rules. MML translation service receives the source metadata, refers to the translation rule metadata, and finally translates into target format.

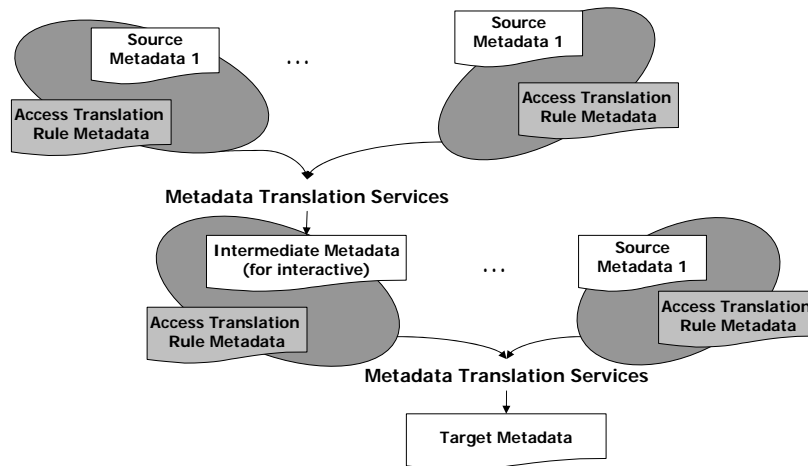


Figure 8. Iterative translation for MML interoperability

**Definition 3.6 (MML Translation Service):** A *MML Translation Service TS* is the task of translating a set of metadata with the same schema *Source* into another metadata with the schema *Dest*. This translation is carried out pertaining a set of translation rules *TRs* to depicts how to operate the *Source* schema into *Dest*. *TS* contains three parameters (*Source*, *TRs*, *Dest*) to describe how to translate *Source* Metadata into *Dest* Metadata via *TRs* Metadata where,

- $Source \subseteq Resource$ , *Source* contains the target MML Metadata
- $TRs \subseteq Resource$ , *TRs* are abbreviated as the translation rules which show how to translate one MML Metadata into another. Each translation rule element in *TRs* is a quadruple (*Ent*, *From\_Attrs*, *Op*, *To\_Attrs*) where,
  - $Ent \subseteq Resources$ , *Ent* indicates the translation rule metadata that the following attributes originated.
  - $From_Attrs \subseteq Resources$ , *From\_Attrs* represent the attribute names to be translated.
  - *Op* represents the operating function supported between attributes.

- $To\_Attrs \subseteq Resources$ ,  $To\_Attrs$  represent the attribute names to be translated into.
- $Dest \subseteq Resources$ ,  $Des$  represents the destination when the source metadata is translated.

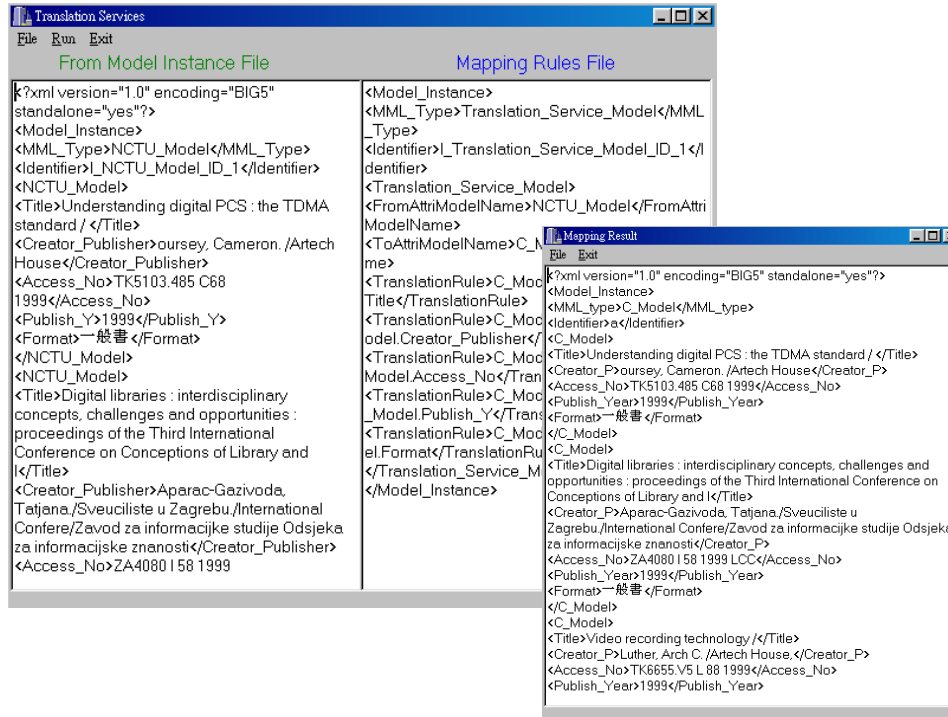


Figure 9. MML translation service

The *MML translation service* is accomplished by translation rule metadata (*TRs*). Translation rule metadata describe how two heterogeneous metadata are transformed into the same data model. *TS* only supports one-way translation. On the other hand, the inverse translation is not allowed in the same *TRs*. The translation rule contains three attributes in their schema, *From\_Metadata*, *To\_Attribute\_Model* and *Translation\_Rules*. In the *Translation\_Rules* template, MML supports basic arithmetic operations, such as add, sub, muliplex and divide... etc., and basic string operation, such as concatenate... etc. Figure 9 is an example to translate NCTU catalog metadata into a canonical model. The left screen shows the original data and the translation rule. After the translation, the target metadata is displayed in the right screen.

MML translation service can express by MML structure in both tuple and set constructors. The expression  $Constructor_i.Attribute_j$  indicates the attribute  $j$  in constructor  $i$ . As the example in Figure 5, the expression of name in author by name in person is  $Person.Name=Author.First\_Name+$  “ $+Author.Last\_Name$ ”. Notably, the *Author* constructor is set constructor which can contain multiple authors in the same book. Hence the expression can be updated into  $Person.Name= Person.Name +$ , “ $+Author.First\_Name+$ ” “ $+Author.Last\_Name$  to add comma in additional authors.

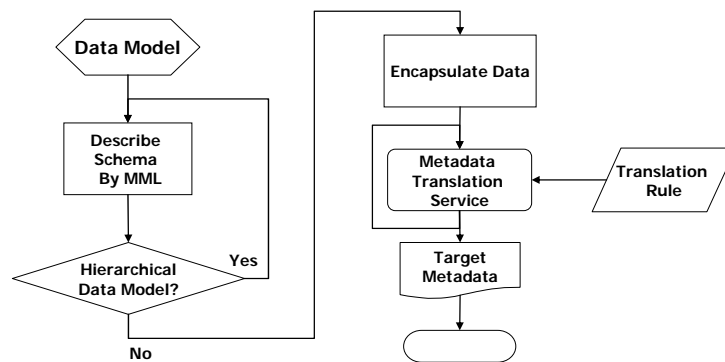


Figure 10. The workflow for metadata interoperability

MML containing translation services conquers information gap and provides more flexible model than others. Figure 10 illustrates the workflow to create MML source metadata and translate into target metadata. User describes the schema first by MML schema and iteratively defines the hierarchical schema until all attributes are defined. Next, the data are encapsulated into MML metadata and apply the translation rule to translate into target schema. By following this process, metadata heterogeneity is easy to solve.

### 3.5 Comparison

	<b>MML</b>	<b>RDF</b>	<b>Infobus</b>
<b>Data Model</b>	Extend RDF with ID and name reference	Hierarchical data model	Hierarchical data model
<b>Translation</b>	Translation template with operations	Non-supported	Leave the translation responsibility to user
<b>Syntax</b>	XML	XML	Proprietary
<b>Object Reference</b>	Identifier with name hierarchy	URI	Non-supported

Table 1. Comparison of MML and current research

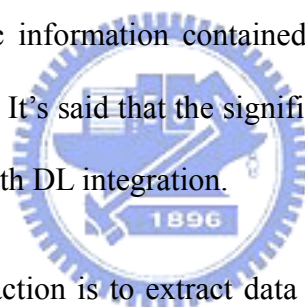
Table 1 shows the comparison of MML, RDF and Stanford Infobus. MML has richer data model than Infobus and extends RDF function by adding identity and name reference for distributed metadata. Additionally, MML provides translation mechanism by translation template with basic operations, which is different from RDF and Infobus who leave translation responsibility to users. Third, MML and RDF adopt XML as syntax when the syntax of Infobus is proprietary. This strategy keeps MML and RDF with standardization. In the object reference, MML use both name and id reference to metadata which is more flexible than RDF and Infobus.



## Chapter 4

### Data Extraction

The main activity of DL integration is to search information from distributed DL. Understandably, the easiest way to obtain information is from a library service through HTTP protocol. It is worth noting that the output for a single search service contains similar structure in every query. This output is called documents with common structure. Common structure documents are defined as the documents whose embedded structure and the semantics are identical. Structure documents with common structure are always generated in library Web-based applications, like a Web-based OPAC system. Once the common structure is analyzed in advance, semantic information contained in these documents can be extracted based on the previous analysis. It's said that the significant pre- and post-processing overhead can be eliminated in dealing with DL integration.



The purpose of data extraction is to extract data automatically from output of each DL search services (for example, the Webpac systems) and then package the result into MML format. In this chapter, an extraction algorithm called *Metadata Extractor* is proposed [24]. A (semi-) structure document is first analyzed to obtain the structure hierarchy and assigned level ID for each critical element of the structure. After labeling the semantics, the iteration process extracts the common structure from the incoming documents automatically. By means of this method, DL access is transparent and automatic through HTML protocol.

#### 4.1 Structure hierarchy

Each (semi-) structure document (like HTML, XML... etc.) can be analyzed into a *Structure Hierarchy*. A *structure hierarchy* is a tree where each node represents a *critical*

*element* (a node with hierarchy information). In *structure hierarchy*, a *critical element* connects with its child critical element nodes with *level property* (if two *critical elements* are hierarchical) and connects with sibling with *parallel property* (if two critical elements are in the same level). As shown in Figure 11, a *structure hierarchy* is illustrated according to an *Auxiliary table* (define *critical tags* and *level/parallel properties* of the tags). In this figure, Level 1 (Root) contains several Level 2 children, and each Level 2 node contains Level 3 children, which are also encompassed in the Root node. To clarify the features in a *structure hierarchy*, two properties are defined.

Auxiliary Table	
Up-Level Tag	Down-Level Tag
<Html>	<Head><Body>
<Body>	<Table><Ol><Ul><Dir> <P>
<Frameset>	<Frame>
<Table>	<Tr>
<Tr>	<Td>
<Td>	<Ol><Ul><Dir> <P>
<Ol><Ul><Dir>	<Li>
<Li>	<Table><Ol><Ul><Dir> <P>
<hr>	<Table><Ol><Ul><Dir> <P>

```

<HTML>
<TABLE>
<TR><TD>Title</TD><TD>First_Name
</TD><TD>Last_Name</TD></TR>
<TR><TD>...</TD><TD>...</TD><TD>...</TD></TR>
<TR><TD>...</TD><TD>...</TD><TD>...</TD></TR>
<TR><TD>...</TD><TD>...</TD><TD>...</TD></TR>
</TABLE>
</HTML>

```

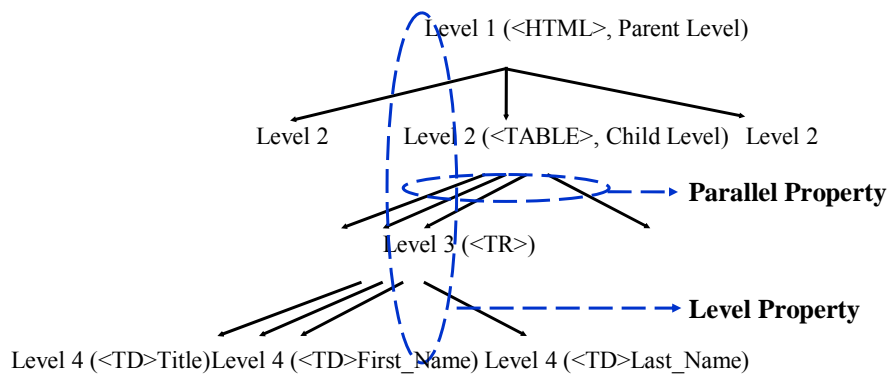


Figure 11. Parallel and level properties

**Definition 4.1. (Level Property - LP)** A and B are two nodes of *structure hierarchy*. LP holds when A is an ancestor (upper-level) of B. (i.e. B is decedent (sub-level) of A and A’s content contains B’s in the document.)

**Lemma 4.1. (Transitive rule of LP)** If A have LP with B and B have LP with C in the same *structure hierarchy*, then LP holds for A and C.

According to Definition 4.1, A's content contains B's and B's content contains C's in the document. It is straightforward that A's content contains C's in the hierarchy tree. In other words, A is an upper-level of C in the structure hierarchy. As a result, *LP* holds for A and C.

**Definition 4.2. (Parallel Property - PP)** A and B are two nodes of a structure hierarchy. *PP* holds when A and B are at the same level in the *structure hierarchy*. (i.e. A's content parallels with B's.)

**Lemma 4.2. (Transitive rule of PP)** If A and B have *PP* and B and C also have *PP* in the same structure hierarchy, then *PP* holds for A and C.

Because A is at the same level as B and B is at the same level as C in the structure hierarchy, it is clear that A is at the same level as C in the structure hierarchy. As a result, *PP* holds for A and C.



#### 4.1.1 Level-ID assignment

Each *critical element* is assigned a *Level-ID* in the hierarchy tree. *Level ID* is a cascaded positive nature number separated by '.' with the following format:

$$Level_1-ID.Level_2-ID.Level_3-ID. \dots$$

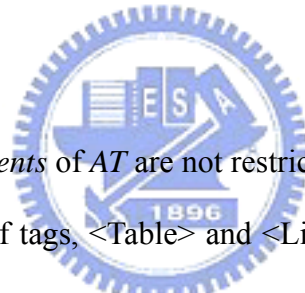
Each *Level<sub>i</sub>-ID* is a positive nature number. For any two *Level-ID* in the same *Level-ID* sequence, we call *Level<sub>i</sub>-ID* is ancestor of *Level<sub>j</sub>-ID* if  $i < j$ . Several important characteristics in *Level-ID* are:

- According to *Lemma 4.1*, *LP* holds in the same *Level-ID* sequence. (For example, *Level ID* 1.1.2 holds *LP* in both 1.1.2.1 and 1.1.2.1.1.)
- According to *Lemma 4.2*, *PP* holds if two *Level-ID* are in the same level and contain the same parent *Level-ID* (For example, *Level-ID* 1.1.2 holds *PP* with 1.1.3 and 1.1.4)

because they have the same parent 1.1.)

#### 4.1.2 Auxiliary table

For assigning *Level-ID* to structure documents, an *Auxiliary Table (AT)* is required to identify *critical elements* and hierarchical property between elements (in Table 2). To take an example of HTML documents, Table 2 depicts a sample *AT* for HTML. (it is easy to extend the approach to other kinds of structure documents, like XML/SGML.) In Table 2, HTML elements (tags) are categorized into Up-Level Tags and Down-Level Tags. *LP* holds for an Up-Level Tag *A* and a Down-Level Tag *B* if *A* and *B* are at the same row of *AT* (For example, by referring to the Row ID 4 of Table 1, we know that *LP* holds for <Table> and <Tr>). Furthermore, *PP* holds for two Down-Level Tags *A* and *B* if *A* and *B* are at the same row of *AT*.



Actually, the *critical elements* of *AT* are not restricted to HTML tags and can be adjusted. For example, only two types of tags, <Table> and <List>, are used in *AT* if only two critical elements have to be analyzed in the structure documents. In addition, for XML or SGML documents, users can create their own structure tag sets and put these tags into *AT*.

Auxiliary Table (AT)	
Up-Level Tag	Down-Level Tag
<Html>	<Head><Body>
<Body>	<Table><Ol><Ul><Dir> <P>
<Frameset>	<Frame>
<Table>	<Tr>
<Tr>	<Td>
<Td>	<Ol><Ul><Dir> <P>
<OL><UL><DIR>	<Li>
<Li>	<Table><Ol><Ul><Dir> <P>
<hr>	<Table><Ol><Ul><Dir> <P>

Table 2. Auxiliary table

---

```

0 Level-ID_Assignment_Algorithm (in HTML D, out Level_ID ){
1     Var Structure_Hierarchy C;
2     For each tag in D{
3         C=Check_Structure_Hierarchy_(tag);
4         IF (C = "Positive-LP")
5             Down_Level (tag); //Level Property holds
6         Else If (C = "Negative-LP")
7             Up_Level(tag) until accurate level;
8             Increment(tag); //Level Property holds
9         Else IF (C = "PP")
10            Increment(tag); //Parallel Property holds
11        Else
12            Do nothing;
13    }
14    End
15 }

```

---

Figure 12. Level-ID assignment algorithm

The AT is manipulated by the *Level-ID* assignment algorithm to label *critical elements* in structure documents. The *Level-ID* assignment algorithm only assigns each *critical element* a *Level-ID*. There are four basic operations in the *Level-ID* assignment algorithm:

- *Down\_Level(tag)*: Extend *Level-ID* to a level down and assign the extended *Level-ID* item as 1 if no this level exists before.
- *Up\_Level(tag)*: Shrink *Level-ID* to a level up.
- *Increment(tag)*: Add one in the last *Level-ID* item of the current *Level-ID*.
- *Check\_Structure\_Hierarchy(tag)*: Check the current and previous tags if they are at the same row in AT (Check if LP holds).

If yes and current tag is Up-Level tag and previous is Down-Level tag,  
return negative-LP.

If yes and current tag is Down-Level tag and previous is Up-Level tag,  
return positive-LP.

If yes and the two tags are both Down-Level tags,  
return PP.

Otherwise return false.

The *Level-ID* assignment algorithm is illustrated in Figure 12.

## 4.2 Common structure

A *common structure* is a structure expression (defined in section 3.2) to extract structure document with semantics. For example, the structure expression in Figure 5 can be expressed as:

$$\{Title^{i.1}, \langle First\_Name^{i.j.2}, Last\_Name^{i.j.3} \rangle_{j=2-m}^{Author}\}_{i=1-n}^{Book}$$

The number in the upright position represents the *Level ID* which indicates the position of *critical element* in the document. Each name of set or tuple constructor is given as the metadata semantics. By following the expression, the incoming documents can obtain “Title” of Book from Level ID 1.1 to n.1, and the “Author” for Book i can be obtained from Level ID i.2.2 and i.2.3 to Level ID i.m.2 and i.m.3.

## 4.3 Implementation

According to Lemma 4.1 and Lemma 4.2, two major conclusions for *Level-ID* approach are:

- All structure documents with common structure have the same *Level-ID* if they have the same structure root and *Auxiliary Table*.
- *Critical elements* have the same level among structure documents with common structure are parallel.

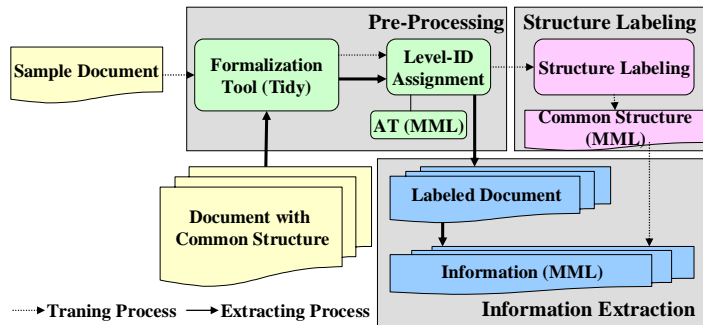


Figure 13. Data extraction for common structure

Based on these two conclusions, an extractor called *Metadata Extractor* for structure documents is implemented. Figure 13 illustrates metadata extraction for structure documents with common structure. In this figure, metadata extraction is divided into two parts: *training process* and *extracting process*. The *training process* uses a sample document to label common structure and saves the result in a common structure file with MML format. The *extracting process* extracts information from documents with the same structure by using of the common structure file. The extracted metadata is also saved in MML format. There are three major phases in this architecture: *pre-processing phase*, *structure labeling phase* and *extraction phase*. The following sections describe the detail.

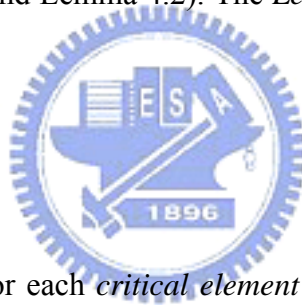
#### 4.3.1 Pre-processing phase

The pre-processing phase reformulates structure documents into a formalized and labeled format. It is composed of two important modules, the formalization tool and *Level-ID* assignment.

**Formalization Tool (TIDY):** Mark-up errors such as unbalanced opening and closing tags often appear in structure documents. Especially in HTML documents, the format is not strictly well-formed (compared with other markup languages, like XML). A formalization tool fixes up documents into a scrupulous form. TIDY, developed by Dave Raggett in W3C (<http://www.w3c.org/>), is a free utility to tidy up sloppy editing into nicely layout markup. In the pre-processing process, TIDY is employed as formalization tool to correct human editing errors.

**Level-ID Assignment:** Structure documents cleaned up by Tidy are sent to *Level-ID* assignment, and each *critical element* is labeled with a *Level-ID*. When structure documents with common structure are parsed, the same Auxiliary Table will give the same *Level-ID* in these documents (Lemma 4.1 and Lemma 4.2). The *Level-ID* assignment is referred to Figure 12.

#### 4.3.2 Structure labeling phase



After *Level-ID* is given for each *critical element* in the sample document, the structure labeling phase records the *Level-ID* and the associated semantic of marked elements into a common structure file with MML format. The marked elements represent the portions of the common structure that will be extracted in the later extracting process. Figure 14 is our labeling tool for structure labeling. In Figure 14, tags with *Level-ID* from 2.1.2 to 2.1.6 are marked and semantics are given for each marked tag. Then the marked tags and their semantics are encapsulated into MML format and save as a common structure file.





Figure 14. Semantics label

### 4.3.3 Data extraction phase

After structure documents with common structure are processed by the pre-processing phase, the extraction algorithm extracts data by referring to the common structure file produced in the structure labeling phase. The final results are encapsulated into MML format. Figure 15 demonstrates a *Virtual Union Catalog System (VUCS)* which the catalog information came from different DL automatically by using *Metadata Extractor*. The later binding of structure according to the auxiliary table compromises the versatile structure of Webpac systems, which is different from several related virtual union catalog systems [22, 28].

The VUCS by using *Metadata Extractor* is a more flexible way to integrate Webpac systems than Open Archives Initiative (OAI) approach (<http://www.openarchives.org/>). OAI provides a standard way to harvest information from OAI compatible services. However, not all data providers have the OAI interface. *Metadata Extractor* integrates data without

prearrangement with data providers and enables an alternative way to integrate heterogeneous data.

**國立交通大學虛擬聯合目錄**

西文期刊聯合目錄     交通大學    查詢字串: 
  
 中文期刊聯合目錄     台灣大學
   
 政治大學

國立政治大學	
序號	書名 (1-12 之 136)
1	<a href="#">Catalog</a>
2	<a href="#">Catalog Age</a>
3	<a href="#">Catalog And Announcements 1965 1966</a>
4	<a href="#">Catalog Directory</a>
5	<a href="#">Catalog For Journal Of Chinese Agricultural Engineering Vol 1 Vol 401955 1994</a>
6	<a href="#">Catalog Of African Government Documents And African Area Index</a>
7	<a href="#">Catalog Of Ancient Books</a>
8	<a href="#">Catalog Of Books And Scientific Periodicals 1967</a>
9	<a href="#">Catalog Of Books Collected In Industrial Library Of China Productivity And Trade Center</a>
10	<a href="#">Catalog Of Books Energy Committee Moea</a>
11	<a href="#">Catalog Of Books In The Cepd Library</a>
12	<a href="#">Catalog Of Chinese Coins</a>

西文期刊聯合目錄			
序號	JSSN	刊名	出版項
1	0740-3119	<a href="#">Catalog Age</a>	New Canaan, Conn. : Catalog Age Pub. Corp.
2	1043-4852	<a href="#">Catalog of New Foreign and International Law Titles</a>	Ann Arbor, Mich. : Ward and Associates



Figure 15. Virtual union catalog system

## Chapter 5

### Semantic Query


Considerable attention has been paid to DL architecture's enabling: DL queries across distributed DL services and interaction between the two most important elements of DL, content and services [39, 45]. Content represents the materials stored in a DL, including texts, images and videos. A service is an application that performs specific functions and interacts with users via specific interface. However, conventional keyword-based queries cannot clarify two factors in distributed DL – content heterogeneity and service capabilities. Content heterogeneity requires determining the format and semantics of two pieces of content. Service capabilities require locating the suitable services to accomplish the task. Fortunately, it is important to note metadata support semantic queries in many ways. First, the modeling of metadata has comprehensive semantics regarding content and services. These semantics benefit the provision of accurate information in response to semantic query. Second, relationships between metadata of content and services can be derived from metadata; manipulating these relationships yields further semantics. Third, metadata can be easily stored and indexed to support retrieval, since they have a formal structure. Conclusively, metadata that describe semantic information about DL content and services motivates the derivation of semantic relationships among metadata.

Given content and services, several questions may be raised. Does one type of content have the same format as another? Can two services perform the same task? Is one type of content produced (or manipulated) by a service? A model that formalizes the relationships between content and services is required to answer these questions. Semantic query layer in *M-Architecture@DL* seeks to formulate the structural relationships among metadata

concerning DL content and services, to assist the extension of semantic DL queries. This chapter proposes a *Content and Service Inference Model (CSIM)* to elucidate the interaction of metadata, in terms of the structural relationship. CSIM defines 15 relationships between DL content and services. Using CSIM, the result of a DL query is extended by embedding relationships into the query predicates. For example, for a user who wants to retrieve content with the format “Dublin Core”, CSIM returns content with the format “Dublin Core” and derives the content with other relationships, such as that which can be translated into “Dublin Core” (the “Translatable” relationship), and that which contains the same semantics but with different formats (the “Homonymous” relationship).

## **5.1 Content and service inference model (CSIM)**

### **5.1.1 Relationships between content and services**



Content and services are two integral aspects of DL. Content is the materials stored in DL, which can be produced and processed by services. The types of content include web pages, library holding records, and multimedia data (like texts, images, and videos). A service is a software application that: 1. receives one type of content 2. accomplishes specific works 3. outputs one type of content. In other words, a service performs specific functions and interacts with users via input and output interfaces. In this study, both services and content are represented via metadata to facilitate the interaction between them. A novel framework called Content and Service Inference Model (CSIM) is proposed to derive relationships between content and services from their metadata [25]. CSIM raises DL queries to a semantic level by using content semantics, service capabilities, and the relationships between content and services.

Basically, the metadata of content contains schema, and a set of semantic description. The metadata of a service includes the input, output and the statement for its capabilities.

According, a total of 15 relationships between content and services are defined in Figure 16.

These relationships are directional and categorized into four types:

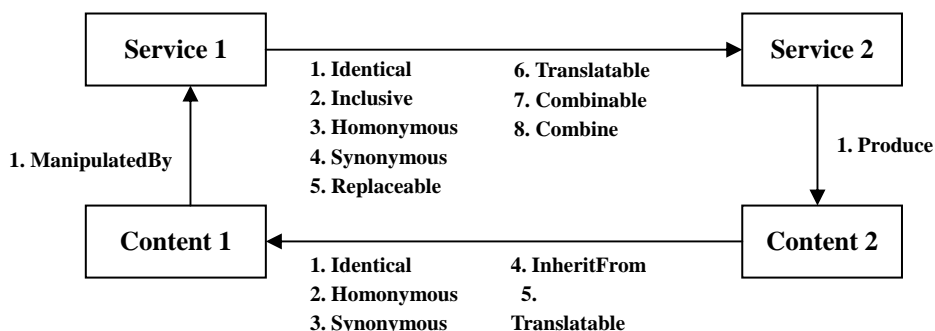


Figure 16. Relationships between content and services

**Service to Service.** Two services interact with each other according to their capabilities and input/output interfaces. Eight relationships of this type are defined - *Identical*, *Inclusive*, *Homonymous*, *Synonymous*, *Replaceable*, *Translatable*, *Combinable*, and *Combine*. For example, two services are "*Synonymous*" if they have the same capabilities but different input/output interfaces. Moreover, one service is "*Inclusive*" of another if the first service contains more capabilities than the other.

**Content to Content.** Two pieces of content relate with each other based on their semantics and schemas. Five relationships of this type are defined - *Identical*, *Homonymous*, *Synonymous*, *InheritFrom*, and *Translatable*. For example, two pieces of content are "*Identical*" if they have identical semantics and format.

**Service to Content.** A service can produce content. One relationship of this type is defined - *Produce*. For example, a WebPAC system may produce a data set in the Dublin Core format.

**Content to Service.** Content can be produced by a service. One relationship of this type

is defined - *ManipulatedBy*. For example, various data sets can be manipulated by a virtual union catalog system to create an integrated view.

The total 15 relationships in CSIM are proposed after thoroughly examining all possible relationships between content and services under the CSIM data model and hypothesis. In the following sections, formal definitions are provided.

### 5.1.2 Basic definitions

In the following, the metadata of content and services is formally defined.

**Definition 5.1 (Content).** A piece of Content,  $C$ , is a quadruple  $\{Id, Schema, Presentations, Semantics\}$  [40] where,

1.  $Id$  is the identifier of  $C$ .
2.  $Schema$  is the schema of  $C$ .  $Schema$  is a quadruple  $\{Ent, Incs, Attributes, Associations\}$ , where
  - 2.1  $Ent \subseteq Names$  is the name of a content schema.
  - 2.2  $Incs \subseteq (Names, Names)$ . Each pair  $(e_1, e_2) \in Incs$  indicates that  $e_1$  is a subtype of  $e_2$ .  $Incs$  is stored in  $CIT$  (defined below) and assumed to be acyclic.
  - 2.3  $Attributes \subseteq Names$  is the set of attribute names.
  - 2.4  $Associations \subseteq (Association\_name, Entity\_name1, Entity\_name2, Cardinality1, Cardinality2)$  is the association set, which indicates the cardinality between two entities.
3.  $Presentations \subseteq Names$  is the set of the presentation interfaces of content  $C$ .
4.  $Semantics \subseteq \mathbf{CST}$  is the set of semantics of content  $C$ .  $CST$  is defined below.



**Definition 5.2 (Service).** A Service,  $S$ , is a quadruple  $\{Id, Capabilities, Outputs, Inputs\}$ , where

1.  $Id$  is the identifier of  $S$ .
2.  $Capabilities \subseteq \mathbf{SCT}$  is the set of service capabilities.  $\mathbf{SCT}$  is defined below.
3.  $Outputs \subseteq \text{Names}$  are the output schemas of service  $S$ .
4.  $Inputs \subseteq \text{Names}$  are the input schemas accepted by  $S$ .

Additional data structures are required for the formal definition of CSIM [56].

1. **Content Semantics Table (CST).** CST contains ontological terms to identify the semantics of content. Dublin Core and MARC are two examples of ontological terms in CST. The ontology in CST is hierarchical; an ascendant term covers the semantics of a descendent term.
2. **Content Inheritance Table (CIT).** CIT maintains the schema hierarchy of content (*Incs* attribute of *Schema* in *Definition 3.1*). For example, the fact that schema A is a subtype of schema B can be represented as (A, B) in *CIT*.
3. **Service Capability Table (SCT).** SCT contains ontological terms to define possible service functionalities. The ontology in SCT is hierarchical; an ascendant term has more general capability than a descendent term. For example, a service that uses CORBA to implement a virtual union catalog system will have two capabilities - *CORBA\_Distributed\_System* and *Virtual\_Union\_Catalog\_System*, where *CORBA\_Distributed\_System* and *Virtual\_Union\_Catalog\_System* are the descendent terms of *Distributed\_System* and *Catalog\_System* respectively.
4. **Translation Rule Table (TRT).** TRT stores the rules for translating between two pieces of content. A rule  $R$  for translating between two pieces of content is expressed as a quadruple:  $\{Id, FromSchema, ToSchema, Rules\}$ , where

1. *Id* is the identifier of *R*.
2. *FromSchema*  $\subseteq$  *Names* is the name of the source schema.
3. *ToSchema*  $\subseteq$  *Names* is the name of the target schema.
4. *Rules*  $\subseteq$  (*FromAttributeName*, *ToAttributeName*, *TranslationRule*) are the rules for translating between specific attributes.

CSIM exploits the translation mechanism proposed in [23] for content translation. If a translation rule *T* exists in *TRT* where *T.FromSchema*=*C*<sub>1</sub> and *T.ToSchema*=*C*<sub>2</sub>, then *C*<sub>2</sub>=Translate(*C*<sub>1</sub>, *T*).

5. **Content and Service Repository (CSR)**: CSR is a repository that stores the metadata of content and services, including the access methods for the CSIM metadata framework to retrieve content and services.

### 5.1.3 Definitions of content and service relationships

As mentioned above, four types of 15 directional relationships between content and services exist - content to content, service to service, content to service, and service to content. This section formally explicates each relationship.

**Definition 5.3 (Content to Content).** Given two pieces of content  $C_1 = \{Id_1, Schema_1, Presentations_1, Semantics_1\}$  and  $C_2 = \{Id_2, Schema_2, Presentations_2, Semantics_2\}$ , the possible relationships between *C*<sub>1</sub> and *C*<sub>2</sub> are as follows.

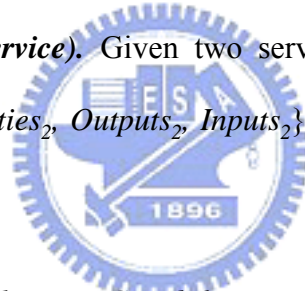
1. **Identical**(*C*<sub>1</sub>, *C*<sub>2</sub>) iff *Schema*<sub>1</sub> = *Schema*<sub>2</sub> and *Semantics*<sub>1</sub> = *Semantics*<sub>2</sub>.
2. **Homonymous**(*C*<sub>1</sub>, *C*<sub>2</sub>) iff *Schema*<sub>1</sub> = *Schema*<sub>2</sub> but *Semantics*<sub>1</sub>  $\neq$  *Semantics*<sub>2</sub>
3. **Synonymous**(*C*<sub>1</sub>, *C*<sub>2</sub>) iff *Schema*<sub>1</sub>  $\neq$  *Schema*<sub>2</sub> but *Semantics*<sub>1</sub> = *Semantics*<sub>2</sub>
4. **InheritFrom**(*C*<sub>1</sub>, *C*<sub>2</sub>) iff (*Schema*<sub>1</sub>, *Schema*<sub>2</sub>)  $\in$  *Incs*<sub>2</sub>



$C_1$  and  $C_2$  have the InheritFrom relationship if and only if the schema pair ( $Schema_1$ ,  $Schema_2$ ) exists in CIT. Namely, the schema of  $C_1$  inherits from  $C_2$ . Given a sequence of content  $C_1$  to  $C_n$ , such that any two consecutive pieces of content have the InheritFrom relationship, these pieces of content exhibit the transitive property.

5. **Translatable**( $C_1$ ,  $C_2$ ).  $C_1$  and  $C_2$  have the *Translatable* relationship if  $C_1$  can be translated into  $C_2$  by means of specific translation rules. In other words,  $C_1$  and  $C_2$  are translatable if and only if there exists a translation rule  $T \in TRT$  such that  $T.FromSchema = Sch_1$  and  $T.ToSchema = Sch_2$ . Moreover, given a sequence of content  $C_1$  to  $C_n$ , such that any two consecutive pieces of content have the Translatable relationship, these pieces of content exhibit the transitive property.

**Definition 5.4 (Service to Service).** Given two services  $S_1 = \{Id_1, Capabilities_1, Outputs_1, Inputs_1\}$  and  $S_2 = \{Id_2, Capabilities_2, Outputs_2, Inputs_2\}$ , the possible relationships between  $S_1$  and  $S_2$  include the following.



1. **Identical**( $S_1$ ,  $S_2$ ) iff  $Capabilities_1 = Capabilities_2$
2. **Inclusive**( $S_1$ ,  $S_2$ ) iff  $Capabilities_1 \subseteq Capabilities_2$
3. **Homonymous**( $S_1$ ,  $S_2$ ) iff  $Outputs_1 = Outputs_2$ ,  $Inputs_1 = Inputs_2$ , but  $Capabilities_1 \not\subseteq Capabilities_2$
4. **Synonymous**( $S_1$ ,  $S_2$ ) iff  $Outputs_1 \neq Outputs_2$  or  $Inputs_1 \neq Inputs_2$ , but  $Identical(S_1, S_2)$
5. **Replaceable**( $S_1$ ,  $S_2$ ) iff  $Inclusive(S_1, S_2)$ ,  $Outputs_1 = Outputs_2$  and  $Inputs_1 = Inputs_2$ .
6. **Translatable**( $S_1$ ,  $S_2$ ) iff  $Inclusive(S_1, S_2)$ ,  $\exists T_1, T_2, T_3, T_4$  in TRT such that  $Translate(Outputs_1, \{T_1\}) = Translate(Outputs_2, \{T_2\})$  and  $Translate(Inputs_1, \{T_3\}) = Translate(Inputs_2, \{T_4\})$ .
7. **Combinable**( $S_1$ ,  $S_2$ ) iff  $Inputs_1 = Outputs_2$

**8. Combine**( $S_c, \{S_i\}$ ) combines a set of services  $\{S_i\}$  ( $1 \leq i \leq n$ ),  $\forall i$  **Combinable**( $S_i, S_{i-1}$ ) (that is,  $Outputs_1 = Inputs_2, Outputs_2 = Inputs_3, \dots, Outputs_{n-1} = Inputs_n$ ), into a new service  $S_c = \{Id_c, Capabilities_c, Outputs_c, Inputs_c\}$ , where

1.  $Id_c$  is the identifier of  $S_c$
2.  $Capabilities_c = Capabilities_1 \cup Capabilities_2 \cup \dots \cup Capabilities_n$
3.  $Outputs_c = Outputs_n$
4.  $Inputs_c = Inputs_1$

Given a sequence of services  $S_1$  to  $S_n$ , a new service  $S_c$  can be generated when any two successive services  $S_i$  and  $S_{i-1}$  have the Combinable relationship. The new service has a new Id, and the capabilities include all the capabilities of  $S_1$  to  $S_n$ . The new service has the same input interface as the first service ( $S_1$ ), and the same output interface as the final service ( $S_n$ ).

**Definition 5.5 (Service to Content).** Given a service  $S = \{Id_s, Capabilities_s, Outputs_s, Inputs_s\}$  and content  $C = \{Id_c, Schema_c, Presentations_c, Semantics_c\}$ ,  $S$  and  $C$  have the *Produce*( $C, S$ ) relationship iff  $Schema_c = Outputs_s$  or *Translatable*( $Outputs_s, Schema_c$ ). In other words, this definition determines if  $S$  can produce  $C$ .

**Definition 5.6 (Content to Service).** Given a content  $C = \{Id_c, Schema_c, Presentations_c, Semantics_c\}$  and a service  $S = \{Id_s, Capabilities_s, Outputs_s, Inputs_s\}$ ,  $C$  and  $S$  have the *ManipulatedBy*( $C, S$ ) relationship iff  $Schema_c = Inputs_s$  or *Translatable*( $Schema_c, Inputs_s$ ). In other words, this relationship determines whether  $C$  can be manipulated by  $S$ .

#### 5.1.4 Manipulating operations

*CSIM* applies manipulating operations to the above four types of 15 relationships. Manipulating operations are of two types -  $\pi$  operations and  $\Pi$  operations.  $\pi$  operations assess whether the relationship between the given content and service is. If a specific

relationship exists, the operation returns TRUE, otherwise it returns FALSE. *II operations* return the corresponding content or services satisfying the specified relationship.

*$\pi$  operations* check if the given content and service have the relationship specified in the  *$\pi$  operations*. A total of five  *$\pi$  operations* are defined:

- $\pi_{\text{Schemas}}$ : Given two pieces of content, A and B, A  $\pi_{\text{Schemas}}$  B refers to CIT and returns TRUE if A and B have the *InheritFrom*(A, B) relationship .
- $\pi^{\sigma}_{\text{Schemas}}$ : Given two pieces of content, A and B, A  $\pi^{\sigma}_{\text{Schemas}}$  B refers to TRT and returns TRUE if A and B have the *Translatable*(A, B) relationship .
- $\pi_{\text{Semantics}}$ : Given two pieces of content, A and B, A  $\pi_{\text{Semantics}}$  B refers to CST and returns TRUE if A and B have the *Identical*(A, B) relationship .
- $\pi_{\text{Capabilities}}$ : Given two services A and B, A  $\pi_{\text{Capabilities}}$  B refers to SCT and returns TRUE if A and B have the *Inclusive*(A, B) relationship .
- $\pi^{\sigma}_{\text{Capabilities}}$ : Given a service A and a set of services Bs, A  $\pi^{\sigma}_{\text{Capabilities}}$  Bs refers to SCT and returns TRUE if the *Inclusive*(A, Bs) relationship holds, or one of the relationships holds: the *Combinable*(A, Bs), *Replaceable*(A, Bs) or *Translatable*(A, Bs). In other words, A  $\pi^{\sigma}_{\text{Capabilities}}$  Bs determines whether service A can be replaced by a series of services Bs according to one of the following four conditions.
  - Bs contain all the capabilities of A;
  - A can be combined by a set of services into Bs;
  - A can be replaced by a set of services into Bs;
  - Bs contain all the capabilities of A, but Bs also can be translated into the same input and output schemas as A.

The algorithms corresponding to the five  *$\pi$  operations* can be referenced in Appendix.

$\Pi$  operations return the content or services conforming to the specified relationship. Four categories of  $\Pi$  operations exist:  $\Pi^c$ ,  $\Pi^s$ ,  $\Pi^{sc}$  and  $\Pi^{cs}$ , with respect to the four types of relationships defined in Section 5.1.3.

$\Pi^c$  operations return the content that conforms to the specified relationship.

- $\Pi^c_{\text{Translatable}}$ : Given content A,  $\Pi^c_{\text{Translatable}}$  determines the content that can be translated into A by direct or transitive translations.
- $\Pi^c_{\text{InheritFrom}}$ : Given content A,  $\Pi^c_{\text{InheritFrom}}$  determines the content that is inherited from A by direct or transitive inheritance.
- $\Pi^c_{\text{Identical}}$ : Given content A,  $\Pi^c_{\text{Identical}}$  determines the content that satisfies the *Identical* relationship with A.
- $\Pi^c_{\text{Homonymous}}$ : Given content A,  $\Pi^c_{\text{Homonymous}}$  determines the content that satisfies the *Homonymous* relationship with A.
- $\Pi^c_{\text{Synonymous}}$ : Given content A,  $\Pi^c_{\text{Synonymous}}$  determines the content that satisfies the *Synonymous* relationship with A.

$\Pi^s$  operations return the services that exhibit the specified relationship.

- $\Pi^s_{\text{Identical}}$ : Given a service A,  $\Pi^s_{\text{Identical}}$  determines the services that exhibit the *Identical* relationship with A.
- $\Pi^s_{\text{Inclusive}}$ : Given a service A,  $\Pi^s_{\text{Inclusive}}$  determines the services that exhibit the *Inclusive* relationship with A.
- $\Pi^s_{\text{Homonymous}}$ : Given a service A,  $\Pi^s_{\text{Homonymous}}$  determines the services that exhibit the *Homonymous* relationship with A.
- $\Pi^s_{\text{Synonymous}}$ : Given a service A,  $\Pi^s_{\text{Synonymous}}$  determines the services that exhibit the *Synonymous* relationship with A.

- $\Pi^s_{\text{Replaceable}}$ : Given a service  $A$ ,  $\Pi^s_{\text{Replaceable}}$  determines the services that exhibit the *Replaceable* relationship with  $A$ .
- $\Pi^s_{\text{Translatable}}$ : Given a service  $A$ ,  $\Pi^s_{\text{Translatable}}$  determines the services that exhibit the *Translatable* relationship with  $A$ .
- $\Pi^s_{\text{Combinable}}$ : Given a service  $A$ ,  $\Pi^s_{\text{Combinable}}$  determines the services that exhibit the *Combinable* relationship with  $A$ .

One  $\Pi^{\text{sc}}$  operation,  $\Pi^{\text{sc}}_{\text{Produce}}$ , is defined. Given a service  $S$ ,  $\Pi^{\text{sc}}_{\text{Produce}}$  returns the content that satisfies the *Produce* relationship with  $S$ .

One  $\Pi^{\text{cs}}$  operation,  $\Pi^{\text{cs}}_{\text{ManipulatedBy}}$ , is defined. Given content  $C$ ,  $\Pi^{\text{cs}}_{\text{ManipulatedBy}}$  returns all the services that satisfy the *ManipulatedBy* relationship with  $C$ .

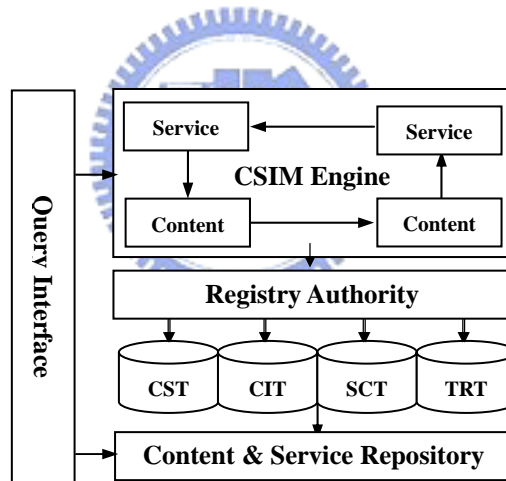


Figure 17. CSIM architecture

The corresponding algorithm for each operation is listed in Appendix. Figure 17 depicts the architecture to apply *CSIM* in DL queries. A *query interface* receives semantic queries and dispatches the queries to the *CSIM Engine*. The *CSIM Engine* parses the query predicates and applies the algorithms described above to solve semantic DL queries by referring to the *Registry Authority*. . The *registry authority* contains access methods for all content and service metadata to expedite the access to the metadata. Furthermore, the *registry authority* comprises a set of ontological tables. These ontological tables present a common vocabulary used for the

ontological hierarchies that define content semantics and service capabilities. Index structures for *CST*, *CIT*, *SCT* and *TRT* are stored in the *Content & Service Repository (CSR)*, to accelerate metadata retrieval. The ontology proposed herein consists of vocabulary with hierarchical structure, and an ascendant ontological concept implies all descendant concepts.

## 5.2. Semantic digital library query

Applying *CSIM* to DL supports powerful semantic queries in content and service retrieval. In *CSIM*, content and service semantics can be elaborated more finely than conventional keyword-based approaches because relationships defined in the previous section can be added in query statement. Using this abundant semantic information, *CSIM* accurately retrieves results and derives alternative answers that conventional approaches cannot do. For example, in response to a query for content with particular semantics, *CSIM* can retrieve the content not only in the same schema hierarchy and with identical semantics, but also in a different format, such as synonymous content. Content with various schemas, which are translatable into a single schema, can also be retrieved. Furthermore, in a semantic service query, *CSIM* can infer a list of recommendations to suggest that a user concatenates available services to create the desired service, using the combinable and translatable relationships.

### 5.2.1 Query language

The query language for *CSIM* is SQL-like. It consists of three main clauses.

1. *Select Clause*: The select clause contains the attributes of the content or service to be retrieved. The mode of attributes in a query can be set to *EXACT* or *AMBIGUOUS*. An *EXACT* query returns the attributes that exactly satisfy the given predicate without being translated or combined. For example, if one user wants to exactly retrieve the content with the data format, “Dublin Core”, the query statement can be set to “*Select EXACT C.Id From*

Content  $C$  where  $C.Schema = \text{“Dublin Core”}$ ”. An *AMBIGUOUS* query recommends answers that have the same semantics as those specified attributes. As in the preceding example, the query can be set to *“Select AMBIGUOUS C.Id From Content C where C.Schema = “Dublin Core”*”. The query will return three types of answer: 1. the content with the data format “Dublin Core”. 2. the content which is not in data format “Dublin Core” but can be translated into “Dublin Core” format. 3. The content which is not in data format “Dublin Core” but in the same hierarchy of “Dublin Core” in CIT. The absence of the attribute mode indicates the default query mode “EXACT”.

2. *From Clause*: This clause specifies the content or service from which a user seeks. For example, a user wants to retrieve information from one piece of content  $C$  and two services  $S_1$  and  $S_2$ .

3. *Where Clause*: This clause states conditional expressions that consist of the content or services given in the *From Clause*. In this clause, a set of Boolean operators (NOT, AND, OR), and a set of relationships defined in Section 5.1.3 are applied. For example, if a user wants to retrieve a service  $S_1$  whose input is produced by another service  $S_2$  and whose output data schema is Dublin Core, then the *Where Clause* is *“  $S_1.Input = ManipulatedBy(S_2)$  AND  $S_1.Output = \text{“Dublin Core”}$ ”*. Basically, the syntax of a conditional expression in this clause is like that of traditional SQL-like language.

### 5.2.2 EXACT and AMBIGUOUS query

Semantic queries are based on the relationships between content and services in CSIM. Semantic queries encompass two types - *EXACT query* and *AMBIGUOUS query* (in Figure 18). An *EXACT* query inquires the services or content that precisely satisfies the given predicates, without inferring other relationships. An *AMBIGUOUS* query returns the service or content that can be inferred from the translatable or combinable relationships, as well as the

service or content with the same semantics as those specified attributes. Notably, an *AMBIGUOUS* query yields more results but takes more time to respond. Both content and service queries are illustrated by “Basic” and “Advanced” query. “Basic” query are in standard SQL statement and can be used in conventional query interface. “Advanced” query contain CSIM manipulation functions in their query predicate, which are able to derive more sophisticated semantic relationships.

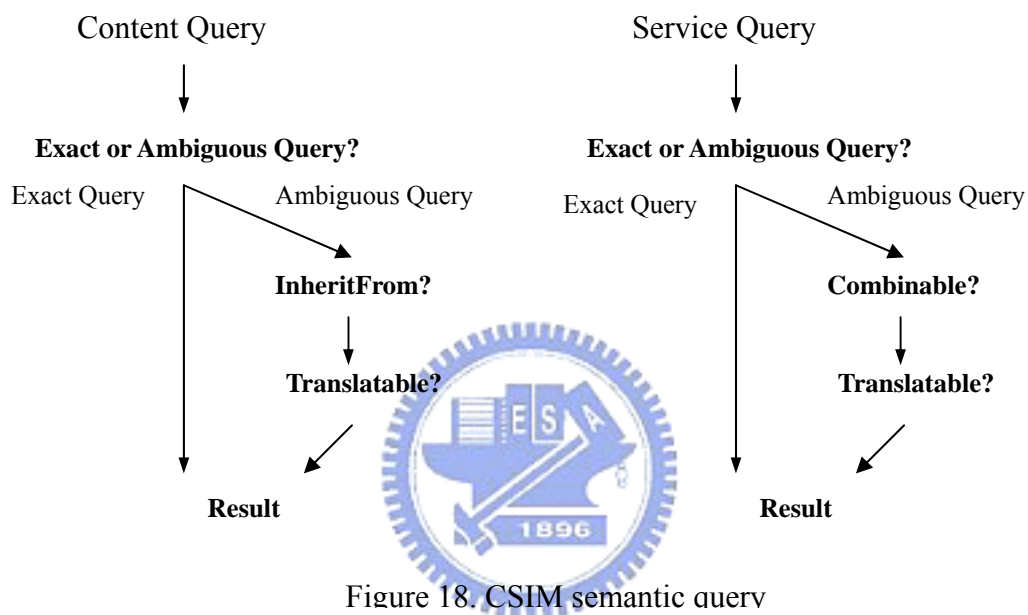


Figure 18. CSIM semantic query

## 1. Content query

A content query inquires about the content meeting the requirement specified in the query. A user can specify an *EXACT* or *AMBIGIOUS* query. An *EXACT* query returns the content that entirely satisfies the query, which means no inference is employed to obtain the result. An *AMBIGIOUS* query returns all the content that can have the same semantics specified in the query. Here, the term "can" means that the content may be translated into, or inherited from the target content.



## 1.1 Basic content query

**Example:** Determine the content with the schema of “Dublin Core”.

**Query Statement:** *Select C.Id From Content C where C.Schema = “Dublin Core”*

**Algorithm:** ContentQuery(Attributes A, Content C){

1. Locate content  $c$ . Let  $c \in CSR$ , and  $c.Id = C.Id$ ,  $c.Schema \pi_{Schema} C.Schema$ ,  $c.Presentations = C.Presentations$ , and  $c.Semantics \pi_{Semantics} C.Semantics$ ;
2. If  $A \in AMBIGUOUS$ , for each  $r \in \Pi_{Translatable}(r, C)$ ,  $c \leftarrow c \cup r$ ; (The symbol “ $\leftarrow$ ” indicates “assign the value”.)
3. For each  $k \in c$ , return  $k.A$ ;

## 1.2 Advanced content query

**Example:** Determine the content inherited from the “Dublin Core” schema.

**Query Statement:** *Select C<sub>1</sub>.Id From Content C<sub>1</sub>, C<sub>2</sub> where C<sub>2</sub>.Schema = “Dublin Core” and InheritFrom(C<sub>1</sub>, C<sub>2</sub>)*

**Algorithm:** AdvancedContentQuery(Attributes A, Contents C, Relationship R){

1. Locate content  $c$ . Let  $c \in CSR$ , and  $c.Id = C.Id$ ,  $c.Schema \pi_{Schema} C.Schema$ ,  $c.Presentations = C.Presentations$  and  $c.Semantics \pi_{Semantics} C.Semantics$ ;
2. If  $A \in AMBIGIOUS$ , for each  $r \in \Pi^c_{Translatable}(r, C)$ ,  $c \leftarrow c \cup r$ ;
3. For each  $i \in \Pi(R)$   
$$c \leftarrow c \cup \text{ContentQuery}(Id, i)$$

4. For each  $k \in c$ , return  $k.A$ ;

## 2. Service query

A service query inquires about the services meeting the requirement specified in the query. A user can specify the query to be *EXACT* or *AMBIGIOUS*. An *EXACT* query returns the services that entirely satisfy the query. An *AMBIGIOUS* query determines all the services that can have the same capabilities specified in the query. Here the term “can have” implies that the services can be concatenated or translated into the target service.

### 2.1 Basic service query

**Example:** Determine the services with the service capability “Catalog\_System”.

**Query Statement:** *Select S.Id From Service S where S.Capabilities = “Catalog\_System”*

**Algorithm:** ServiceQuery(Attributes A, Services S){

1. Locate service s. Let  $s \in CSR$ ,  $s.Id=S.Id$ , and  $s.Capabilities \pi_{Capabilities} S.Capabilities$ ;
2. If  $A \in AMBIGIOUS$ , for each  $r \in \Pi_{Translatable}^s(r, C.Schema)$ ,  $c \leftarrow c \cup r$ ;
3. For each  $k \in c$ , return  $k.A$ ;

### 2.2 Advanced service query

**Example:** Determine the services that have the same capabilities with “Catalog\_System”.

**Query Statement:** *Select S<sub>1</sub>.Id From Service S<sub>1</sub>, S<sub>2</sub> where S<sub>2</sub>.Capabilities = “Catalog\_System” and Inclusive(S<sub>1</sub>, S<sub>2</sub>).*

**Algorithm:** AdvancedServiceQuery(Attributes A, Services S, Relationship R){

1. Locate service s. Let  $s \in CSR$ ,  $s.Id=S.Id$ , and  $s.Capabilities \pi_{Capabilities} S.Capabilities$ ;
2. If  $A \in AMBIGIOUS$ ,

Locate service  $rs \in CSR$  where  $S.Capabilities \pi_{Capabilities}^{\sigma} rs.Capabilities$

$c \leftarrow c \cup rs;$

3. For each  $i \in R$

$c \leftarrow c \cup ServiceQuery(id, i)$

4. For each  $k \in c$ , return  $k.A$ ;

### 5.2.3 Ranking function

A result of a CSIM semantic query can be classified into one of the following types:

1. **Exact match.** The result conforms to the query predicate without additional translation, inheritance, or combination.
2. **Ambiguous match.** The result is a recommendation that may not completely satisfy all query predicates, but can satisfy the predicates by translating, inheriting, or combining available services or content.

Because the results of a query may not totally fulfill the user's requirements, a ranking function is proposed to evaluate the fitness of the results of a query. The ranking function  $W$  is separated into  $W_{Content}$  and  $W_{Service}$ , and defines as follows;

**Ranking function  $W_{Content}(Content A, Content Results)$**  (Equation 1)

$=1$  if  $A \pi_{Schemas} Results$  and  $Num(Results)=1$

$=\prod(1-T_i)$  if  $A \pi_{Schemas}^{\sigma} Results$  and  $Num(Results)>1$

**Ranking function  $W_{Service}(Service A, Service Results)$**  (Equation 2)

$=1$  if  $A \pi_{Capabilities} Results$  and  $Num(Results)=1$

$$=(\sum W_{Results}^i * \Pi(1-T_i)) / (Num(Results)-Num(T_i))$$

if  $A \pi_{Capabilities} Results$  and  $Num(Results)>1$

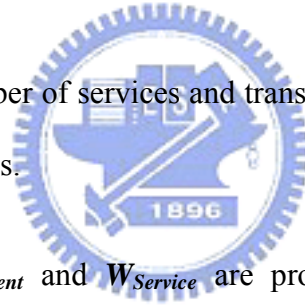
where

$W_{Results}^i$  :  $Num(Results^i.Capabilities)/Num(A.Capabilities) * W_{Result}^S$

$W_{Result}^S$  : Service weight of  $Result^i$ . The larger weight represents the service is easier to compose in the result.

$T_i$  : Overhead of the translation rules to produce  $Results$

$Num(Results)$  : The number of services and translation rules. The less number of results is, the larger rank of the result is.



Ranking functions  $W_{Content}$  and  $W_{Service}$  are proposed to rank content and services, respectively. The ranking follows the number of semantic concepts or capabilities that meet the query predicates, and the amount of content and services that are combined to yield the result.

**Example:** Assume a service A with five capabilities; we apply CSIM in the semantic query and obtain that A can be concatenated by three services  $S_1$ ,  $S_2$ , and  $S_3$  with two translations T1 and T2. Each of the three services owns three service capabilities of A, and the union of their capabilities includes all the service capabilities of A. If all of these services have  $W^S=1$  and the two translation rules have a 10% and 20% overhead respectively, what is the rank of this concatenation?

**Result:** In this example,  $W_{Results}^1$ ,  $W_{Results}^2$  and  $W_{Results}^3$  are  $3/5=0.6$ .  $T_1$  and  $T_2$  are 0.1 and 0.2, respectively. Applying Eq. 2 yields the rank of this concatenation as  $(0.6*1+0.6*1+0.6*1) * ((1-0.1) * (1-0.2)) / (5-2)=0.432$

### 5.3 Implementation

A prototype system is implemented (in Figure 19). This system supports advanced semantic DL queries for users to retrieve content and services. Librarians can consult this system to determine reusable components in DL before they start establishing new services. This system includes four main areas.

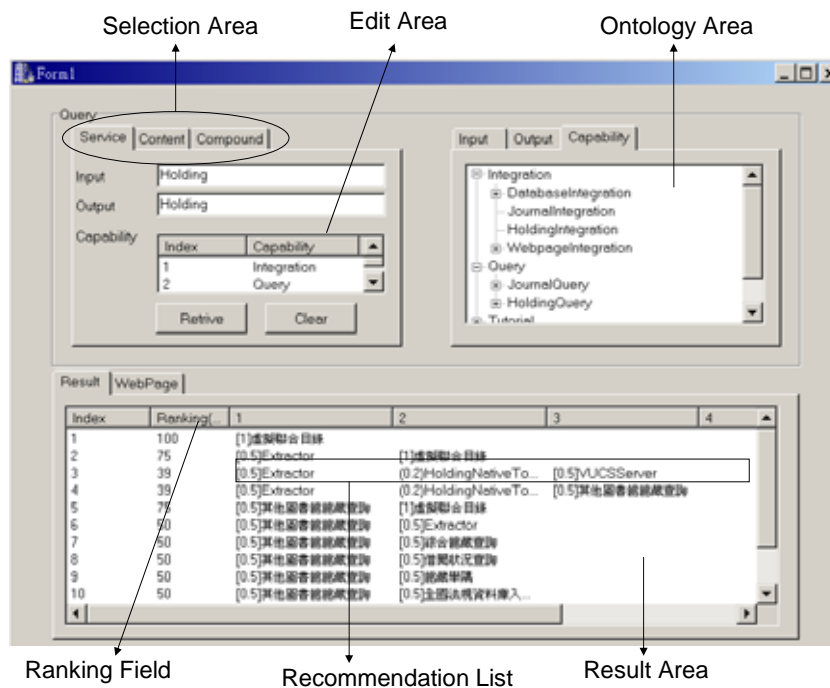


Figure 19. CSIM prototype system

1. **Selection area:** The selection area allows the user to specify which one of the three query types is to be issued - service query, content query, and compound query. Compound query performs complex service and content queries to retrieve content or services.

2. **Edit area:** The edit area allows the user to specify the query predicates, including input/output schemas and semantics of content, and service capabilities. They are selected from the ontology area.
3. **Ontology area:** The ontology area contains all ontological hierarchies defined in Section 5.1.2 The ontology is developed by domain experts.
4. **Result area:** The result area includes the results that satisfy the predicates in the edit area. Each result is expressed as a recommendation list with one or more items. A list with one item indicates that the item exactly matches the specified predicates; on the other hand, a list with more than one item indicates that the users can combine these items together to obtain new content/service that satisfies the specified predicates. The ranking of each result is also given. A numeral in front of each service of the recommendation list denotes the percentage to fit the specified search capabilities. The result area presents a set of recommendation lists to advise the user; nevertheless, the system leaves the task of confirming the feasibility of the recommendation list to the user.

The example illustrates in Figure 19 conducts a query to retrieve *virtual union catalog system*. The input and output interfaces are specified as “holding” format. The “*holding*” format, which is the most superior item in one content schema ontology means the service to be returned should contain any kinds of library holdings. The capabilities are specified as “*Integration*” and “*Query*”, each of which is the most superior item in one capability ontology. Consequently, the answer area shows not only the VUCS@NCTU service (Index 1 of the result area in Figure 19), but also a recommended list to suggest another virtual union catalog system (Index 3 of the result area in Figure 19) by combing three services: an extractor from structure documents, a translation service to translate native data schemas into a canonical schema, and an integration service to combine distributed extractors.

## Chapter 6

### Experiments

#### 6.1 Experimental set-up and approach

The idea of *M-Architecture@DL* is verified in this chapter. A series of experiments were conducted to examine the performance of semantic query. The experiments contained service and content queries, which explored the structural relationship of the metadata of services and content.

The subjects of service queries were the 81 services in Digital Library of National Chiao Tung University. The services were divided into six categories – Tutorial (TU), Query (QU), Service (SE), Database (DB), Journal (JO), and Holding (HO). The metadata of each service was given by experts and contained keywords that delineated the service capabilities. Experiments on service queries were conducted to compare the conventional keyword-based approach, the *CSIM* approach, and the *CSIM* approach with service inference. The keyword-based approach returned the services the description of whose capabilities exactly matched the specified keywords. The *CSIM* approach involved the *AMBIGUOUS* semantic query without enabling the  $IT_{Combinable}$  manipulation function; this approach conducted a query to refer to related ontological terms for service capabilities and returned answers that would be inherited or translated into the desired services. *CSIM* with service inference involved the *AMBIGUOUS* semantic query with enabling the  $IT_{Combinable}$  manipulation function. This approach returned the answer of the *CSIM* approach; in addition, a recommend list that could compose the required service from existing services was returned as well. For a digital library that decomposed services into reusable components, *CSIM* with inference advised the system developer to construct new services by combining the existing

components. In this manner, a lot of development effort could be saved.

To evaluate content queries, a *virtual union catalog system (VUCS)* [10], which harvested over 20 WebPAC systems of Libraries in Taiwan, was adopted. The issue of various data schemas in the WebPAC systems complicated the mapping of schema attributes in response to a content query. To handle this issue, conventional *VUCS* systems involved user intervention in the design phase. This design strategy led to a much less flexible system and required user intervention in the system design. *CSIM* referred to the ontology stored in the *CIT* (Content Inheritance Table) to extend the hierarchical relationships between attributes of schemas. Furthermore, *CSIM* used the ontology of semantics stored in the *CST* (Content Semantics Table) to solve the problems of attribute semantics (such as those involving *Synonymous* and *Homonymous* relationships). The experiments retrieved four attributes (title, subject, author, and publisher) of the content satisfying the query. The keyword-based approach returned the content contained the specified keywords within these fields (not all WebPAC systems contains all of these fields). The *CSIM* approach employed the ontological tables (both *CST* and *CIT*) to map various attributes of schemas into the same attribute if they were at the same level of the ontological hierarchy. Additionally, the *CSIM* approach also exploited the ontological tables and *TRT* to convert heterogeneous attributes into the content suitable for the requested attributes. The keywords used in content queries were randomly selected. The top k answers were calculated to average the performance in the four attributes.

## 6.2 Experimental metrics

Two metrics, *Accuracy* and *Coverage*, were used to evaluate both the service and content query and thereby elucidated the effectiveness of *CSIM*. *Accuracy* represents the effectiveness of the returned answers to be correct. *Coverage* represents the effectiveness of the returned correct answers to be included in the entire correct answers. For the service query, the



*Accuracy* ( $A_m(s)$ ) and *Coverage* ( $C_m(s)$ ) are defined as

$$A_m(s) = \frac{(|\text{total services in } F_s| \cap |\text{total services in } F_{\text{ideal}}^s|)}{|\text{total services in } F_s|}$$

$$C_m(s) = \frac{(|\text{total services in } F_s| \cap |\text{total services in } F_{\text{ideal}}^s|)}{|\text{total services in } F_{\text{ideal}}^s|}$$

where  $m$  indicates the method to be examined, which can be keyword-based, *CSIM* and *CSIM* with inference.  $s$  represents the examined service category. In these formulas,  $F_s$  represents the services in each service category returned by the query;  $F_{\text{ideal}}^s$  represents all the services in category  $s$ , the services in concept  $s$ , and the services returned by *CSIM* with inference whose ranks exceed the threshold  $T_{\text{service}}$ :

$$F_{\text{ideal}}^s = \text{the services of category } s \cup \text{the services of concept } s \cup \text{the services returned by CSIM with inference whose ranks exceed } T_{\text{service}}$$

The threshold  $T_{\text{service}}$  discards the results that are cascadedly translated by too more translation rules between the input and output interfaces. The aim of  $T_{\text{service}}$  is to control the efficiency of the service query. For the content query, the *Accuracy* ( $A_m(c)$ ) and *Coverage* ( $C_m(c)$ ) are defined as

$$A_k(c) = \frac{(|\text{total content in } F_c| \cap |\text{total content in } F_{\text{ideal}}^c|)}{|\text{total content in } F_c|}$$

$$C_k(c) = \frac{(|\text{total content in } F_c| \cap |\text{total content in } F_{\text{ideal}}^c|)}{|\text{total content in } F_{\text{ideal}}^c|}$$

where  $m$  indicates the method to be examined, which can be keyword-based, *CSIM* or *CSIM* with inference.  $c$  represents the top  $k$  content returned by VUCS@NCTUDL. In these

formulas,  $F_c$  represents the response to a content query.  $F_{ideal}^c$  represents all the content returned by the keyword-based approach, and the content returned by CSIM whose ranks exceed the threshold  $T_{content}$ :

$$F_{ideal}^c = \text{the content returned by keyword-based approach} \cup \text{the content returned by CSIM whose ranks exceed } T_{content}$$

The threshold  $T_{content}$  discards the results that are cascadedly translated by too more translation rule between two content. The aim of  $T_{content}$  is to control the efficiency of the content query.

The percentage of the improvement of CSIM over the keyword-based approach was used to demonstrate the performance of CSIM in content query. The *Improvement* is defined as:

$$Improvement_{Accuracy} = (A_m^{CSIM}(c) - A_m^{Keyword-based}(c)) / A_m^{Keyword-based}(c)$$

$$Improvement_{Coverage} = (C_m^{CSIM}(c) - C_m^{Keyword-based}(c)) / C_m^{Keyword-based}(c)$$

### 6.3 Experiment results

Figure 20 and 21 present the *Accuracy* and *Coverage* of the service query. Only one translation is allowed in their input/output schemas between two services. Both figures indicate that the *CSIM* approach outperforms the keyword-based approach. In all service categories, the keyword-based approach has poor *Accuracy* and *Coverage* because these services do not explicitly contain the searched keywords in their capabilities. The *CSIM* approach dramatically improves the performance with regard to both *Accuracy* and *Coverage*. This finding shows that exploiting concept approaches (like *CSIM*) is useful for semantic DL queries. Notably, the *CSIM* approach with inference outperforms pure *CSIM* in the ‘‘HO’’ category because the former approach can recommend users to combine existing services to

generate additional ones such as the union of WebPAC system and the electronic journal databases. This result may encourage libraries to spend less effort by integrating available ones on developing add-on services.

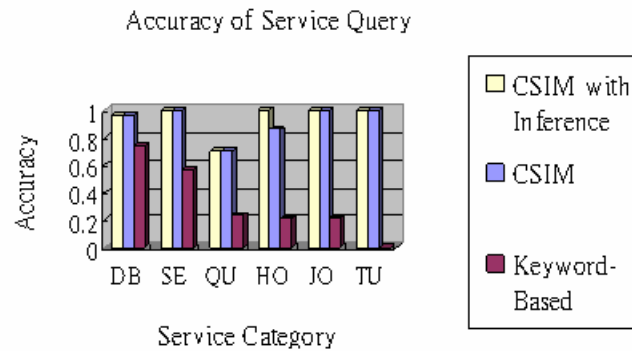


Figure 20. Accuracy of service query

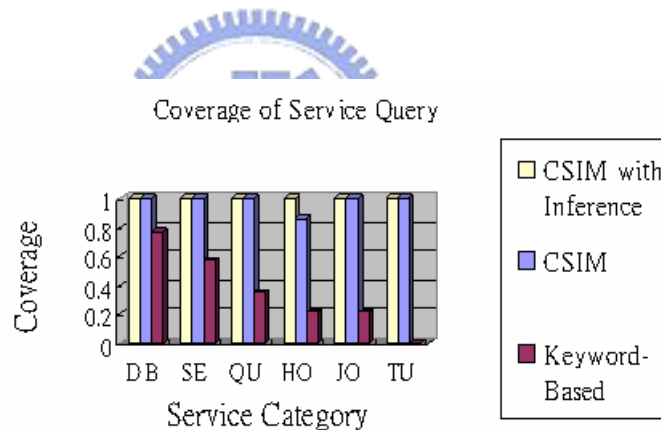


Figure 21. Coverage of service query

For content query, Figure 22 and Figure 23 plot the improvement of the average Accuracy and Coverage with respect to the queries in the four attributes (title, subject, author and publisher). The Top-K in X axis means that the top-k books returned by VUCS are selected as the result. To avoid too many translations between two pieces of content, only one translation is allowed. In Figure 22, CSIM increase 1.2 times performance in average than the keyword-based approach in Accuracy because CSIM refers to CIT and TRT to obtain more

conceptually related attributes of schemas. For example, the “author” field in NCTUDL may appear in other Webpac systems with different name such as creator; in this case, keyword-based approach cannot retrieve the correct results. In Figure 23, the improvement of Coverage indicates that CSIM outperforms 8%~10% to the keyword-based approach. The curve of *Coverage* improvement increases when the returned answers increase because CSIM refers to CST to return those attributes with the same semantics but different format (like Synonyms).

In summary, the performance of semantic DL queries with *CSIM* is highly promising. The *CSIM* model represents a significant improvement in both service and content queries. The *CSIM* model not only enhances the *Accuracy* and *Coverage* of a DL query but also suggests how librarians can integrate available components into a desired service.

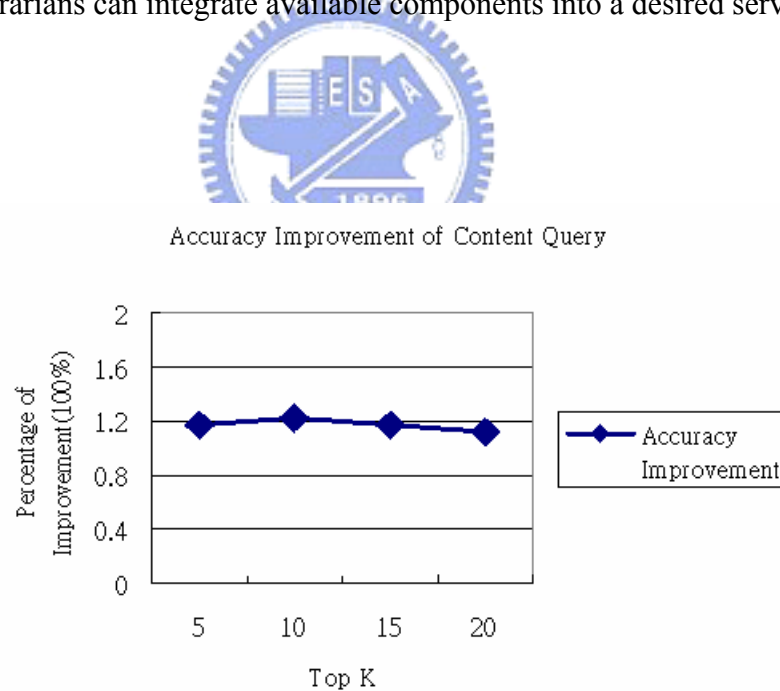


Figure 22. Accuracy improvement of content query

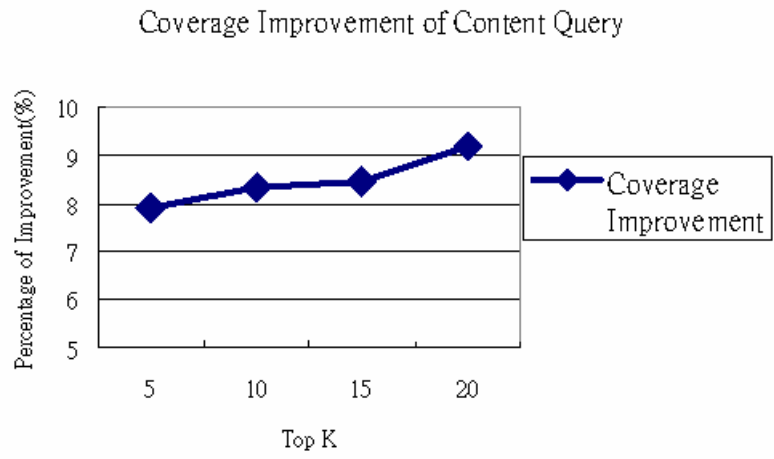


Figure 23. Coverage Improvement of Content Query

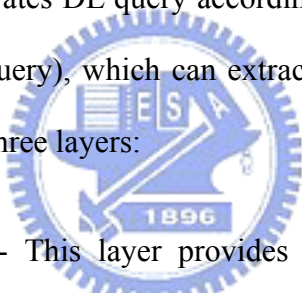


## Chapter 7

### Concluding Remarks

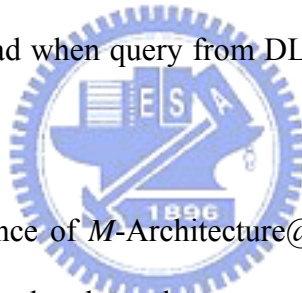
This dissertation addresses the problem of conventional architecture for DL integration, and proposes a novel architecture from the perspective of metadata to integrate DL seamlessly. The metadata integration of DL contains several advantages: metadata model is powerful, metadata transformation is easy, metadata structure is derivable and metadata integration is economic. Based on these reasons, a metadata architecture called *M-Architecture@DL* is proposed to facilitate DL integration.

*M-Architecture@DL* separates DL query according to *MEQ* model (metadata modeling, data extraction and semantic query), which can extract additional knowledge in persistency. *M-Architecture@DL* contains three layers:

- 
- Metadata Modeling - This layer provides rich modeling power and translation mechanism to conquer heterogeneity. A *Metadata Modeling Language (MML)* is proposed in this layer. *MML* adopts *XML* as its syntax and extends *RDF* by adding name hierarchy reference. The format interoperability is achieved by iterative translations. The other profit is that *MML* provides formal expression of metadata structure. This expression is useful for metadata manipulation and integration when metadata structure can be expressed by both tuple and set constructors. Toolkits including a metadata edit tool and a translation tool are provided for the creation of metadata and metadata translations rules.
  - Data extraction - This layer collects data from distributed DL and encapsulates result into *MML* metadata. Data from DL services with similar structure can be extracted into metadata automatically by means of the common structure. This layer

provides a transparent way without prearrange with library affiliations and saves a lot of effort to collect information. *Metadata Extractor* is implemented to extract *MML* metadata from (semi-) structure documents with common structure. The *Metadata Extractor* that extracts metadata transparently can enhance protocol interoperability in DL integration.

- Semantic query - This layer provides query to obtain metadata semantics as much as possible. The semantics query across distributed DL is called *Content and Services Inference Model (CSIM)*. CSIM derives 15 relationships between two DL factors, content and services, and defines functions to manipulate these relationships. CSIM can add manipulation functions in the query statement to obtain more semantic answer. Applying *CSIM* in DL significantly improves semantic query and alleviates the administrative load when query from DL. Semantic interoperability is achieved in this layer.



To examine the performance of *M-Architecture@DL*, both content and service queries are conducted. Experiments results show that semantic queries in *M-Architecture@DL* are highly promising. In service query, significant improvement is achieved because not only the keywords for service capability but also other relationships, like translatable and combinable, are conformed. The content query with CSIM from *Virtual Union Catalog System* has superior performance because semantic heterogeneity is solved by ontology tables.

Summarily, adopting *M-Architecture@DL* to integrate DL can save a lot of administrative effort. When collecting data from DL affiliations, *Metadata Extractor* extracts data from public services, like webpac system, which can save a lot of effort to communicate individual service protocol. The *CSIM* adopts ontology table for semantics to save a lot of effort for user-intervene in semantics heterogeneity. If librarian tries to construct new services, CSIM can also advise a recommended list from existent services to avoid a whole new

construction.

In the future, *M-Architecture@DL* has several works on which to pay attention. In metadata modeling layer, the data model can be integrated to more popular models, such as the E-R model in the field of Database. The combination of MML with Database can facilitate the transformation from databases to metadata, which is practical in many DL applications. Moreover, in data extraction layer, the detection of consistency in common structure is an important issue to prevent the mistaken extraction when the outputs of DL search services are changed. This work is critical when constructing virtual union catalog systems because it is a high risk for the DL to change their Webpac systems in a period of time. The third work will be focused on the efficiency of the semantic query layer. The derivation among content and services may spend a lot of time when the specified content semantics or service capabilities are unclear. A moderate terminate mechanism should be taken to avoid cyclic or endless derivation. Another issue for the semantic query layer is the relationships of CSIM. A more delicate definition of content or a service is anticipated to imply more than 15 relationships among content and services.

The future direction for this study can be focused on the reuse of metadata. The next generation of metadata model requires an information model to deal with the reuse of different metadata. Literature has shown that Cornell University is developing a metadata reusable interface called information network overlay (INO) to provide an extensible knowledge base for an expanding suite of digital library services [54]. INO adopts aggregation model to manipulate metadata and create metadata operations for metadata. “The metadata operation for metadata” facilitates the use of versatile metadata and save a lot of effort when constructing unified search. This research involves several issues, like information model for metadata aggregation, resource description, semantics induction ...etc.



Another observation for metadata in DL integration is that the increasing standards require many mappings to exchange information. For example, giving 10 standards, the total exchange of all standards requires 90 mappings to translate with each other. Therefore, an idea called switching-across is introduced to act as an intermediate among various standards. With an intermediate presented, the optimal solution only requires 10 mappings. To improve the sharing of metadata, the automatic generation of common switching-across is useful in metadata exchange. Administrative effort will be reduced in collaborative search. As a result, using a switching across to channel crosswalks seems to be a well-accepted solution in metadata exchange. This research involves several issues, like schema integration, schema mapping with minimum information loss, schema translation ... etc.

The two directions for future research can be easily added into *M-Architecture@DL*. The aggregation of metadata can be treated as the fourth layer after the returned result of semantic query. The result can be easily integrated by creating metadata to operate data metadata. The switching-across can be treated as additional functions in metadata modeling layer. The performance of *M-Architecture@DL* can be enhanced when the metadata come from many DL and require large amount of translations.

## Bibliography

- [1]. Abiteboul, S., Benjelloun, O., & Milo, T. (2002) Web services and data integration, *Proceedings of the Third International Conference on Web Information and System Engineering*, (pp.1-6), Singapore.
- [2]. Arasu, A., & Garcia-Molina, H. (2003) Extracting structured data from web pages, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, (pp. 337-348), San Diego, California.
- [3]. Arms, W. Y. (1995) Key concepts in the architecture of the digital library, *D-Lib Magazine*, <http://www.dlib.org/>.
- [4]. Baldonado, M., Chang, C. C. K., Gravano, L., & Paepcke, A. (1997) The stanford digital library metadata architecture, *International Journal on Digital Libraries*, 1(2), 108-121.
- [5]. Berners-Lee, T. (1997) Metadata architecture - documents, metadata, and links, <http://www.w3.org/DesignIssues/Metadata/>.
- [6]. Bhoopalam, K., Maly, K., McCown, F., Mukkamala, R., & Zubair, M. (2005) A standards-based approach for supporting dynamic access policies for a federated digital library, *Proceedings of the 8<sup>th</sup> International Conference on Asian Digital Libraries (ICADL2005)*, (pp.242-252), Bangkok, Thailand.
- [7]. Blanchi, C., & Petrone, J. (2001) Distributed interoperable metadata registry, *D-Lib Magazine*, December, 7(12), <http://www.dlib.org/dlib/december01/blanchi/12blanchi.html>.
- [8]. Chan, L. M., & Zeng, M. L. (2006) Metadata Interoperability and Standardization – A Study of Methodology Part I, *D-Lib Magazine*, June, 12(6), <http://www.dlib.org/dlib/june06/chan/06chan.html>.
- [9]. Chen, H. (2005) Digital library development in the Asia pacific, *The 8<sup>th</sup> International Conference on Asian Digital Libraries*, 2005, (pp.509-524), Bangkok, Thailand.

- [10]. Coyle, K. (2000) The virtual union catalog: a comparative study, *D-Lib Magazine*, 6(3), <http://www.dlib.org/dlib/march00/coyle/03coyle.html>.
- [11]. Digital Library Production System - DLPS (1999) Supporting access to diverse and distributed finding aids: A final report to the digital library federation on the distributed finding aid server project, <http://www.umdl.umich.edu/dlps/dfas/dfas-final.html>.
- [12]. Durfy, E. H., Kiskis, D. L., & Birmingham, W. P. (1997) The agent architecture of the university of Michigan digital library, *Proceedings of IEE Software Engineering*, 144(1), (pp.61-71).
- [13]. Derbyshire, D., Ferguson, I. A., Muller, J. P., Pischel, M., & Wooldridge, M. (1997) Agent-based digital libraries: driving the information economy, enabling technologies: infrastructure for collaborative enterprises, *Proceedings of the 6th Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises*, (pp.82-86), Washinton, U.S.A.
- [14]. Ding, H., & Solvberg, I. (2005) Choosing appropriate peer-to-peer infrastructure for your digital libraries, *Proceedings of the 8<sup>th</sup> International Conference on Asian Digital Libraries (ICADL2005)*, (pp.457-462), Bangkok, Thailand.
- [15]. Dushay, N., French, J. C. & Logoze, C. (1999) Using query mediators for distributed searching in federated digital libraries, *Proceedings of the Fourth ACM Conference on Digital Libraries*, (pp. 171-178), Berkeley, U.S.A.
- [16]. Fox, E. A., Suleman, H., & Luo, M. (2002) Building digital libraries made easy: Toward open digital libraries, *Proceedings of the Fifth International Conference on Asian Digital Libraries (ICADL2002)*, (pp. 14-24), Singapore.
- [17]. Goncalves, M. A., Fox, E. A., Watson, L.T., & Kipp, N. A. (2004) Streams, structures, spaces, scenarios, societies (5S): A formal model for digital libraries, *ACM Transactions on Information System*, 22(2), 270-317.
- [18]. Grossman, R., Qin, X., & Xu, W. (1995) An architecture for a scalable,

high-performance digital library, *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems*, (pp. 11-14), Monterey, California.

- [19]. Han, H., Giles, C. L., Manavoglu, E., & Zha, H. (2003) Automatic document metadata extraction using support vector machines, *2003 Joint Conference on Digital Libraries (JCDL'03)*, (pp.37-48), Houston, U.S.A.
- [20]. Hirokawa, S., Itoh, E., & Miyahara, T. (2003) Semi-automatic construction of metadata from a series of web documents, *Australian Conference on Artificial Intelligence 2003*, (pp.942-953), Perth, Australia.
- [21]. Ho, H. I., & Hsiang, J. (2005) Configurable meta-search for integrating Web public access catalogs, *International Conference on Asian Digital Libraries (ICADL2005)*, (pp.317-322), Bangkok, Thailand.
- [22]. Hu, M. J., & Jian, Y. (1999) Multimedia description framework (MDF) for content description of audio/video documents, *Proceedings of the Fourth ACM International Conference on Digital Libraries*, (pp. 67-75), Berkeley, U.S.A.
- [23]. Huang, S. H., Ke, H. R., & Yang, W. P. (2000) Interoperability of cooperative databases with metadata, *The Fourth World Multiconference on Systemics, Cybernetics and Informatics (SCI2000)*, (pp. 169-174), Orlando, U.S.A.
- [24]. Huang, S. H., Ke, H. R., & Yang, W. P. (2000) Information extraction for documents with common structure, *The Third International Conference of Asian Digital Library (ICADL2000)*, (pp. 105-112), Seoul, Korea.
- [25]. Huang, S. H., Ke, H. R., & Yang, W. P. (2005) Enhancing semantic digital library query using a content and service inference model (CSIM)<sup>4</sup>, *Information Processing & Management*, 41, 891-908.

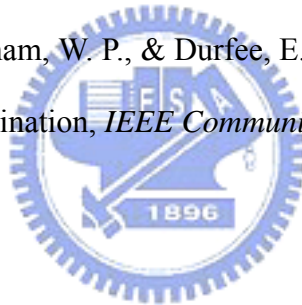
- [26].Huang, S. H., Ke, H. R., & Yang, W. P. (2006) Using metadata to integrate digital libraries by three-layer architecture, *WSEAS Transactions on Computers*, 10(5), 2301-2308.
- [27].Huang, S. H., Ke, H. R., & Yang, W. P. (2006) Metadata architecture for digital library integration, *Proceedings of the 10th WSEAS International Conference on COMPUTERS*, (pp.717-722), Athens, Greece.
- [28].Hung, J. F. (1999) WebOpac and catalog module of library information system, *Master Thesis, Department of Computer Science and Information Engineering, National Chung Cheng University, ChyaYi, Taiwan*.
- [29].Iverson, L. (2004) Collaboration in digital libraries: a conceptual framework, *Proceedings of the 4th ACM/IEEE-CS Joint Conference on Digital Libraries*, (pp.380-380), Tuscon, U.S.A.
- [30].Jenkins, C., Jackson, M., Burden, P., & Wallis, J. (1999) Automatic RDF metadata generation for resource discovery, [http://www.scit.wlv.ac.uk/~ex1253/rdf\\_paper/](http://www.scit.wlv.ac.uk/~ex1253/rdf_paper/).
- [31].Kahn, R., & Wilensky, R. (1995) A framework for distributed digital object services, <http://www.cnri.reston.va.us/cstr/arch/k-w.html>.
- [32].Ke, H. R., Huang, S. H. & Yang, W. P. (2001). The study of interoperability of digital libraries with metadata. *University Library Journal*, 5(1), 49-78.
- [33].Kelapure, R., Goncalves, M. A., & Fox, E. A. (2003) Scenario-based generation of digital library services, *Proceedings of the 7th Conference on Research and Advanced Technology on Digital Libraries (ECDL2003)*, (pp. 263-275), Trondheim, Norway.
- [34].Kim, H., Choo, C. Y., & Chen, S.S. (2003), An integrated digital library server with OAI and self-organizing capabilities, *Proceedings of the 7th Conference on Research and Advanced Technology on Digital Libraries (ECDL2003)*, (pp. 105-112), Trondheim, Norway.

- [35].Kovacs, L., & Micsik, A. (2005) An ontology-based model of digital libraries, *International Conference on Asian Digital Libraries (ICADL2005)*, (pp.38-43), Bangkok, Thailand.
- [36].Kumar, A., Saigal, R., Chavez, R., & Schwertner, N. (2004) Architecting an extensible digital repository, *Proceedings of the Joint Conference on Digital Libraries 2004 (JCDL'04)*, (pp. 2-10), Tucson, Arizona.
- [37].Lagoze, C. (1996) The Warwick framework, *D-Lib Magazine*, <http://www.dlib.org/>.
- [38].Lagoze, C., Kraff, D., Cornwell, T., & Eckstrom, D. (2006) Representing contextualized information in the NSDL, *European Conference on Digital Library*, (pp.329-340), Alicante, Spain.
- [39].Lynch, C., & Garcia-Molina, H. (1995) Interoperability, scaling and the digital libraries research agenda, *Informaiton Infrastructure Technology and Applications (IITA) a Digital Libraries workshop*, <http://www-diglib.stanford.edu/diglib/pub/reports/iita-dlw/main.html>.
- [40].McCray, A. T., Gallagher, M. E., & Flannick, M. A. (1999) Extending the role of metadata in a digital library system, *Proceedings of the IEEE Research and Technology Advances in Digital Libraries, 1999 (ADL99)*, (pp. 190-199), Baltimore, U.S.A.
- [41].McCray, A. T., & Gallagher, M. E. (2001) Principles for digital library development, *Communications of the ACM*, 44(5), 49-54.
- [42].Melnik, S., Garcia-Molina, H., & Paepcke, A. (2000) A mediation infrastructure for digital library services, *Proceedings of the Fifth ACM International Conference on Digital Libraries*, (pp. 123-132), San Antonio, U.S.A.
- [43].Miller, P. (2000), Interoperability, what is it and why should I want it, <http://www.ariadne.ac.uk/issue24/interoperability/intro.html>.
- [44].Miyahara, T., Suzuki, Y., Shoudai, T., Uchida, T., Takahashi, K., & Ueda, Hi (2004) Discovery of maximally frequent tag tree patterns with contractible variables from

- semistructured documents, *Proceedings of Advances in Knowledge Discovery and Data Mining, 8th Pacific-Asia Conference*, (pp.133-144), Sydney, Australia
- [45].Monch, C., & Drobnik, O. (1998). Integrating new document types into a digital library, *Proceedings of the IEEE Research and Technology Advances in Digital Libraries, 1998 (ADL98)*, (pp. 56-65), Santa Barbara, California.
- [46].Napoli, C. D., & Giordano, M. (2004) A service-oriented customizable digital library, *Proceedings of ACM Symposium on Applied Computing (SAC'04)*, (pp. 1730-1731), Nicosia, Cyprus.
- [47].Neuroth, H., & Bargheer, M. (2003) Metadata models - international developments and implementation, *Electronic Information and Communication in Mathematics 2002*, (pp.112-121), Beijing, China
- [48].Nikolaou, C., & Marazakis, M. (1998) System infrastructure for digital libraries: A survey and outlook, *Proceeding of the 25th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM98)*, (pp. 186-203), Jasna, Slovakia.
- [49].Paepcke, A., Cousins, S. B., Garcia-Molina, H, Hassan, S. W., Ketchpel, S. P., Roscheisen, M., & Winograd, T. (1996) Using distributed objects for digital library interoperability, *IEEE Computer*, 29(5), 61-68.
- [50].Paepcke, A., Chang, C. C. K., Winograd, T., & García, M. H. (1998), Interoperability for digital libraries worldwide, *ACM Communication*, 41(4), 33-43.
- [51].Petrou, C., Hadjiefthymiades, S., & Martakos, D. (1999) An XML-based, 3-tier scheme for integrating heterogeneous information sources to the WWW, *Proceeding of Database and Expert Systems Applications on Tenth International Workshop (DEXA99)*, (pp.706-710), Florence, Italy
- [52].Podnar, I., Luu, T., Rajman, M., Klemm, F., & Aberer, K. (2006) A peer-to-peer architecture for information retrieval across digital library collections, *Proceedings of the 10<sup>th</sup> Conference on Research and Advanced Technology on Digital Libraries*

(*ECDL2006*), (pp. 14-25), Alicante, Spain.

- [53]. Roscheisen, M., Baldonado, M., Chang, K., Gravano, L., Ketchpel, S., & Paepcke, A. (1997) The stanford infobus and its service layers, <http://www-diglib.stanford.edu>.
- [54]. Shen, R., Vemuri, N. S., Fan, W., Torres, R.S., & Fox, E. A. (2006) Exploring digital libraries: Integrating browsing searching and visualization, *Joint Conference on Digital Libraries 2006 (JCDL'06)*, (pp.1-10), North Carolina, U.S.A.
- [55]. Weatherley, J. (2006) A web service framework for embedding discovery services in distributed library interfaces, *Joint Conference on Digital Libraries 2005 (JCDL'05)*, (pp.42-43), Colorado, U.S.A.
- [56]. Weinstein, P. C., & Birmingham, W. P. (1998) Creating ontological metadata for digital library content and services, *International Journal on Digital Libraries*, 2(1), 20-37.
- [57]. Weinstein, P. C., Birmingham, W. P., & Durfee, E. F. (1999) Agent-based digital libraries: decentralization and coordination, *IEEE Communications Magazine*, 37(1), 110-115.





## Appendix

Table 3.  $\pi$  operations of CSIM

$\pi$ operations	
$\pi_{Sches}$	<b>Algorithm:</b> $\pi_{Sches}(\text{Content A, Content B})\{$ 1. Locate $A.Sches$ from $CIT$ 2. For each $r \in A.Sches.Incs$ If $r = \langle A, B \rangle$ , Return TRUE 3. Return FALSE
$\pi^{\sigma}_{Sches}$	<b>Algorithm:</b> $\pi^{\sigma}_{Sches}(\text{Content A, Content B})\{$ 1. Locate $A.Sches$ from $CIT$ 2. For each $r \in A.Sches.Incs$ If $\Pi^c_{Translatable}(A,B) \neq \text{NULL}$ , Return TRUE 3. Return FALSE
$\pi_{Sems}$	<b>Algorithm:</b> $\pi_{Sems}(\text{Content A, Content B})\{$ 1. Locate $A.Sems$ and $B.Sems$ from $CST$ 2. For each $r \in A.Sems$ If $r \notin B.Sems$ , Return FALSE 3. Return TRUE
$\pi_{Caps}$	<b>Algorithm:</b> $\pi_{Caps}(\text{Service A, Service B})\{$ 1. Locate $A.Caps$ and $B.Caps$ from $SCT$ 2. For each $r \in A.Caps$ If $r \notin B.Caps$ Return FALSE 3. Return TRUE
$\pi^{\sigma}_{Caps}$	<b>Algorithm:</b> $\pi^{\sigma}_{Caps}(\text{Service A, Service Bs})\{$ 1. Locate $A.Caps$ and $Bs.Caps$ from $SCT$ 2. For each $r \in A.Caps$ If $r \notin Bs.Caps$ or $\Pi^c_{Translatable}(A,B) = \text{NULL}$ Return FALSE 3. If $\Pi^s_{Combinable}(A) \neq Bs$ Return FALSE 4. Return TRUE

Table 4.  $\Pi^c$  operations of CSIM

$\Pi^c$ operations	
$\Pi^c_{Translatable}$	<b>Algorithm:</b> Set $SRC = \{B\}$ to the result, $\Pi^c_{Translatable}(\text{Content A, Content B, SRC})\{$ 1. For each $r \in TRT\{$ Locate $r.FromSche$ from $CIT$ if $r.FromSche \in SRC\{$ $SRC = SRC \cup r$ if $r.ToSche \neq A$ , $\Pi^c_{Translatable}(\text{Content A, r.ToSche, SRC})$ } } 2. return $SRC\}$
$\Pi^c_{InheritFrom}$	<b>Algorithm:</b> $\Pi^c_{Translatable}(\text{Content A})\{$ 1. Set $SRC = \{A\}$ to the result 2. Locate $A.Sches.Incs$ from $CIT$ 3. For each $s \in SRC\{$ For all $r \in CIT$ , if $r \in A.Sches.Incs$ , $SRC = SRC \cup r$ } 4. Return $SRC\}$
$\Pi^c_{Identical}$	<b>Algorithm:</b> $\Pi^c_{Identical}(\text{Content A})\{$

	<ol style="list-style-type: none"> <li>1. Set <i>SRC</i> the the empty result</li> <li>2. Locate <i>A.Sches</i> from <i>CIT</i>, <i>A.Sems</i> from <i>CST</i></li> <li>3. For each <math>r \in CSR</math> <ul style="list-style-type: none"> <li>If <math>r \pi_{Sems} A</math> and <math>r \pi_{Sches} A</math>, <math>SRC = SRC \cup r</math></li> </ul> </li> <li>4. Return <i>SRC</i>}</li> </ol>
$\Pi^c$ Homonymous	<b>Algorithm:</b> $\Pi^c_{Homonymous}(\text{Content } A)\{$ <ol style="list-style-type: none"> <li>1. Set <i>SRC</i> to the empty result</li> <li>2. Locate <i>A.Sches</i> from <i>CIT</i>, <i>A.Sems</i> from <i>CST</i></li> <li>3. For each <math>r \in CSR</math> <ul style="list-style-type: none"> <li>If <math>r \pi_{Sems} A</math> and not (<math>r \pi_{Sches} A</math>), <math>SRC = SRC \cup r</math></li> </ul> </li> <li>4. Return <i>SRC</i>}</li> </ol>
$\Pi^c$ Synonymous	<b>Algorithm:</b> $\Pi^c_{Synonymous}(\text{Content } A)\{$ <ol style="list-style-type: none"> <li>1. Set <i>SRC</i> to the empty result</li> <li>2. Locate <i>A.Sches</i> from <i>CIT</i>, <i>A.Sems</i> from <i>CST</i></li> <li>3. For each <math>r \in CSR</math> <ul style="list-style-type: none"> <li>If not (<math>r \pi_{Sems} A</math>) and <math>r \pi_{Sches} A</math>, <math>SRC = SRC \cup r</math></li> </ul> </li> <li>4. Return <i>SRC</i>}</li> </ol>

Table 5.  $\Pi^s$  operations of CSIM

$\Pi^s$ operations	
$\Pi^s$ Identical	<b>Algorithm:</b> $\Pi^s_{Identical}(\text{Service } A)\{$ <ol style="list-style-type: none"> <li>1. Set <i>SRC</i> to the empty result</li> <li>2. Locate <i>A.Caps</i> from <i>SCT</i></li> <li>3. For each <math>r \in CSR</math> <ul style="list-style-type: none"> <li>If <math>r \pi_{Caps} A</math> and <math>A \pi_{Caps} r</math>, <math>SRC = SRC \cup r</math></li> </ul> </li> <li>4. Return <i>SRC</i>}</li> </ol>
$\Pi^s$ Inclusive	<b>Algorithm:</b> $\Pi^s_{Inclusive}(\text{Service } A)\{$ <ol style="list-style-type: none"> <li>1. Set <i>SRC</i> to the empty result</li> <li>2. Locate <i>A.Caps</i> from <i>SCT</i></li> <li>3. For each <math>r \in CSR</math> <ul style="list-style-type: none"> <li>if <math>r \pi_{Caps} A</math>, <math>SRC = SRC \cup r</math></li> </ul> </li> <li>4. Return <i>SRC</i>}</li> </ol>
$\Pi^s$ Homonymous	<b>Algorithm:</b> $\Pi^s_{Homonymous}(\text{Service } A)\{$ <ol style="list-style-type: none"> <li>1. Set <i>SRC</i> to the empty result</li> <li>2. Locate <i>A.Caps</i> from <i>SCT</i></li> <li>3. For each <math>r \in CSR</math> <ul style="list-style-type: none"> <li>If (<math>r.Input \pi_{Sches} A.Input</math>) and (<math>r.Output \pi_{Sches} A.Output</math>) and not (<math>r \pi_{Caps} A</math>), <math>SRC = SRC \cup r</math></li> </ul> </li> <li>4. Return <i>SRC</i>}</li> </ol>
$\Pi^s$ Synonymous	<b>Algorithm:</b> $\Pi^c_{Synonymous}(\text{Service } A)\{$ <ol style="list-style-type: none"> <li>1. Set <i>SRC</i> to the empty result</li> <li>2. Locate <i>A.Caps</i> from <i>SCT</i></li> <li>3. For each <math>r \in CSR</math> <ul style="list-style-type: none"> <li>If not (<math>r.Input \pi_{Sches} A.Input</math>) or not (<math>r.Output \pi_{Sches} A.Output</math>) and (<math>r \pi_{Caps} A</math>), <math>SRC = SRC \cup r</math></li> </ul> </li> <li>4. Return <i>SRC</i>}</li> </ol>
$\Pi^s$ Replaceable	<b>Algorithm:</b> $\Pi^s_{Replaceable}(\text{Service } A)\{$ <ol style="list-style-type: none"> <li>1. Set <i>SRC</i> to the empty set</li> <li>2. Locate <i>A.Caps</i> from <i>SCT</i></li> <li>3. For each <math>r \in CSR</math> <ul style="list-style-type: none"> <li>If (<math>r.Input \pi_{Sches} A.Input</math>) and (<math>r.Output \pi_{Sches} A.Output</math>) and (<math>r \pi_{Caps} A</math>), <math>SRC = SRC \cup r</math></li> </ul> </li> <li>4. Return <i>SRC</i>}</li> </ol>
$\Pi^s$ Translatable	<b>Algorithm:</b> $\Pi^s_{Translatable}(\text{Service } A)\{$

	<ol style="list-style-type: none"> <li>1. Set <math>SRC</math> to be an empty set to hold result</li> <li>2. Locate <math>A.Caps</math> from <math>SCT</math></li> <li>3. For each <math>r \in CSR</math> { <ul style="list-style-type: none"> <li>If (<math>r \pi_{Caps} A</math>) {</li> <li>Locate <math>rInput</math> and <math>rOutput</math> from <math>TRT</math></li> <li>if (<math>rInput.FromSche=r.Input</math> and <math>rOutput.FromSche=r.Output</math>) {</li> <li><math>SRC = SRC \cup r</math></li> <li>if not (<math>rInput.ToSche = B.Input</math> and <math>rOutput.ToSche = B.Output</math>),</li> <li><math>\Pi^s_{Translatable}(r.ToSche, Content B, SRC)</math></li> <li>}</li> <li><math>SRC = SRC \cup r</math></li> <li>}</li> </ul> </li> <li>4. Return <math>SRC</math></li> </ol>
$\Pi^s_{Combinable}$	<b>Algorithm:</b> $\Pi^s_{Combinable}(\text{Service } A)\{$ <ol style="list-style-type: none"> <li>1. Set <math>SRC</math> to be an empty set to hold result</li> <li>2. For each <math>r \in CSR</math> <ul style="list-style-type: none"> <li>If (<math>r.Input \pi_{Sches} A.Output</math>), <math>SRC = SRC \cup r</math></li> </ul> </li> <li>3. Return <math>SRC</math></li> </ol>

Table 6.  $\Pi^{sc}$  operation of CSIM

$\Pi^{sc}$ operations	
$\Pi^{sc}_{Produce}$	<b>Algorithm:</b> $\Pi^{sc}_{Produce}(\text{Content } A)\{$ <ol style="list-style-type: none"> <li>1. Set <math>SRC</math> to be an empty set to hold result</li> <li>2. For each <math>r \in CSR</math> <ul style="list-style-type: none"> <li>If (<math>r.Output = A</math>), <math>SRC = SRC \cup r</math></li> </ul> </li> <li>3. Return <math>SRC</math></li> </ol>

Table 7.  $\Pi^{cs}$  operation of CSIM

$\Pi^{cs}$ operations	
$\Pi^{cs}_{ManipulateBy}$	<b>Algorithm:</b> $\Pi^{cs}_{ManipulatedBy}(\text{Content } A)\{$ <ol style="list-style-type: none"> <li>1. Set <math>SRC</math> to be an empty set to hold result</li> <li>2. For each <math>r \in CSR</math> <ul style="list-style-type: none"> <li>If (<math>r.Input = A</math>), <math>SRC = SRC \cup r</math></li> </ul> </li> <li>3. Return <math>SRC</math></li> </ol>