# Semantic web annotation for relationship instances

Myo Myo Naing

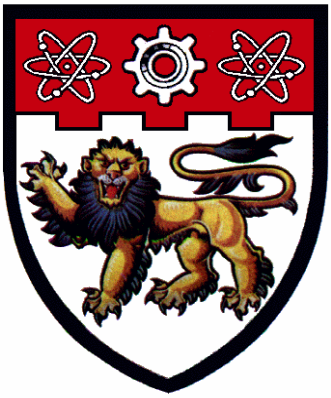
2005

Myo Myo Naing. (2005). Semantic web annotation for relationship instances. Doctoral thesis, Nanyang Technological University, Singapore.

https://hdl.handle.net/10356/2397

https://doi.org/10.32657/10356/2397

# Semantic Web Annotation for Relationship Instances



**MYO MYO NAING**

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

2005

# Semantic Web Annotation for Relationship Instances

**Myo Myo Naing**

**School of Computer Engineering**

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2005

# Acknowledgments

I would like to express my gratitude to the following organizations and people.

First of all, I have to thank **University of Computer Studies, Yangon (UCSY), Myanmar** and **Nanyang Technological University (NTU), Singapore** for providing me the opportunity and scholarship to pursue my PhD research.

I would like to thank my supervisor, **Assoc. Prof. Lim Ee Peng**, for his guidance, encouragement and invaluable advice in research matters even though he was away in Hong Kong for a year and a half. He has helped me very much in improving my ideas, presentations and writings. Without his guidance, the thesis would not have been completed.

I would also like to thank **Assoc. Prof. Ng Wee Keong** for his valuable suggestions at the weekly seminars which have helped me to improve my presentation skills, and broaden my knowledge. I would also like to express my appreciation to **Assoc. Prof. Roger H. L. Chiang and Asst. Prof. Dion Goh Hoe Lian** for reviewing my writeups, and providing useful feedback and amendments.

Thanks also go to all the graduate students in CAIS whom I worked together with during the last three years. Special thanks must be given to **Sun Aixin**, **Liu Zehwa**, **Khin Myo Win**, **Ong Kok Leong**, **Woon Yew Kwong David**, and **Wang Yida** for sharing their knowledge and opinions throughout my studies.

In addition, I need to acknowledge **Mr. Lai Chee Keong** and **Mdm. Tan Lay Choo**, the technicians of CAIS for their technical assistance in my research.

My greatest appreciation goes to **my parents** and family members for their encouragement, kindness and generous support while I was away from home. Their sacrifice and encouragement give me strength to pursue my PhD research.

Finally, I express my gratitude to others who have helped me one way or other in my research.

# Contents

# List of Figures

# List of Tables

# Summary

Heterogeneity and autonomy of Web sites have made information extraction, information retrieval, and data mining on Web information very complex. With a lack of semantics about the Web sites and their Web pages, it is very difficult to analyze Web data and to develop useful applications for them. Web annotation refers to the task of assigning additional semantics to objects on the Web so as to improve their accessibility to automated programs. When multiple Web sites share a common information domain, their annotations can be based on a common semantic structure known as *ontology*, and such an annotation process is also known as **Semantic Web Annotation (SWA)**. With SWA, an ontology shared by different Web annotations can serve as the common structure to extract, process, and integrate the annotated Web objects.

In this research, we analyze the features of different SWA languages and techniques in order to provide an organized view on the existing state-of-the-art SWA research, to identify potential areas of research, and to discuss promising ways of enhancing SWA. From our survey, we realized that the existing SWA approaches only annotate the content of individual Web pages. The annotation of link structures between related Web pages has been largely neglected. We therefore proposed an approach to annotate link structures between Web pages as link chains of relationship instances. In our approach, pairs of related Web pages are known as relationship instances and the link or chain of links among them are known as **link chains**. Link chains are a kind of attribute information for relationship instances. When multiple link chains of the same relationship share common sub-paths, we call these sub-path the *shared-path*. We also propose a normalization based technique to reduce redundant link chain

annotation by finding possible shared-paths. We also design annotation schema for representing annotation instances incorporating link chains.

Unfortunately, manual annotation of link chains without the prior knowledge of link structures at a Web site is tedious and labor intensive causing much time and errors. We therefore define the **link chain extraction problem** and propose a semi-automatic link chain extraction method. Our link chain extraction method semi-automatically extracts link chains using *sequential covering algorithm*. We have applied the proposed extraction method on two well-structured Web sites. The experimental results show that our method performs well even with a small number of training examples.

Finally, our research also include the implementation of CORE, a working prototype system for semantic instance searching and browsing. CORE supports keyword based queries on concept instances that are Web pages and relationship instances represented as pairs of Web pages. The browsing component of CORE facilitates navigation from source pages (source concept instances) to related target pages (target concept instances) by exploiting the extracted link chain information.

# Chapter 1

# Introduction

## 1.1 Semantic Web Annotation

The World Wide Web (WWW) provides abundant information for sharing and dissemination. As enormous number of Web sites operated by autonomous organizations provide a wide range of Web content, locating useful information on the Web can be a taunting task. Web sites and their Web pages are often structured in different styles and formats making them more eye-pleasing but also harder for their content to be automatically extracted and processed. Without a good semantic knowledge about the Web sites and pages, it is also difficult to build sophisticated applications around the Web.

Lee, Hendler and Lassila described the **Semantic Web** as a Web with more intelligent services such as meaningful content-based discovery of information sources, e-business, fusion of information from multiple sites, etc., over which data, documents, or portions of documents can be processed directly or indirectly by machines [LHL01]. Unlike WWW, the purpose of Semantic Web is to give well-defined meaning to the Web sites and Web pages so that they becomes machine processable, rather than merely human browsable. Semantic Web therefore aims to define a layer of metadata above the existing Web. This layer of metadata will provide Web applications the necessary knowledge for understanding Web sites and Web pages.

The creation of metadata for the Semantic Web is known as **Web annotation** [KK01]. Web annotation refers to attaching any comments, notes, explanations,

references, examples, advices, correction or any other type of remarks to the Web pages or parts of them. Annotations can be external (i.e., without modifications to the original pages) or internal (i.e., embedded in the original pages). Examples of Web annotation systems are ComMentor [RMW95], Annotator[1], CoNote [DH95] and Futplex [Hol96] (Please see [HLO99] for more details). The annotations created using these systems lack of some common semantic structures pertinent to the underlying domains of the Web content.

Studies on Web annotation have concluded that annotation without a controlled and well-defined model cannot reap any intended benefits because this just would build another layer of data which share the same heterogeneity and autonomy characteristics with the underlying Web [BG01]. Thus, attempts have been made to create Web annotations based on some well-defined semantic structures or models, known as **ontologies**. As ontologies have been used in many application areas such as knowledge engineering [Gai97, UG96], database design and integration [SCD+97, MKSI98], information retrieval and extraction [MNS03, Wel98, McG98], they have been defined differently for various applications. They, on the other hand, share the common aim of standardizing the vocabulary and semantic structure used for organizing and exchanging information. From our point of view, ontology is a specification of semantically interrelated entities within a given domain. In the following, we give the formal definition of ontology.

**Definition 1.1 (Ontology)** *An* **ontology** *consists of 5 elements* $\langle C, A^C, R, A^R, H \rangle$*, where* $C$ *represents a set of concepts;* $A^C$ *represents a collection of attribute sets, one for each concept;* $R$ *represents a set of relationships;* $A^R$ *represents a collection of attribute sets, one for each relationship; and* $H$ *represents a concept hierarchy.*

*Each concept* $c_i$ *in* $C$ *represents a set of objects of the same kind, and can be described by the same set of attributes denoted by* $A^C(c_i)$*. Each relationship* $r_i(c_p, c_q)$ *in* $R$ *represents a binary association between source concept* $c_p$ *and target concept* $c_q$*, and the instances of such a relationship are pairs of* $(c_p, c_q)$ *concept objects. The*

[1] *http://www.foresight.org/WebEnhance/Annotate.html*

*attributes of $r_i$ are denoted by $A^R(r_i)$. $H$ is a concept hierarchy derived from $C$ and it is a set of parent-child (or superclass-subclass) relations between concepts in $C$. $(c_p, c_q) \in H$ if $c_q$ is a subclass, or sub-concept, of $c_p$.*

Note that the above ontology definition is very similar to that of conceptual schema in traditional databases. However, there are several key differences especially when associating Web information with an ontology:

- Attributes are not mandatory items for the concepts and relationships in the ontology. In other words, it is possible to have attribute values not assigned to some instances belonging to the same concept. Similarly, one may not find values existing in all attributes of the instances belonging to the same relationships. The database conceptual schema however requires every object instance to have values for all their attribute.

- The domains of concept and relationship attributes are not fixed. In a database conceptual schema, it is implicit that each attribute has a fixed domain of values. Such requirement does not exist in ontologies.

- In a database conceptual schema, each entity type (or concept) is required to have some attribute(s) serving as the key of the entity instances. In an ontology, keys are not required. An implicit identifier is instead adopted for every concept instance and relationship instance.

In the following, we give a simple Movie ontology example.

**Example:**

$Movie\_Ontology = \langle\ C_{movie}, A^C_{movie}, R_{movie}, A^R_{movie}, H_{movie}\ \rangle$ where

- $C_{movie} = \{Movie,\ Cast,\ Actor,\ Director,\ Producer,\ ScreenWriter\}$

- $A^C_{movie} = \{A^C_{movie}(Movie), A^C_{movie}(Cast), A^C_{movie}(Actor), A^C_{movie}(Director),$
  $A^C_{movie}(Producer), A^C_{movie}(ScreenWriter)\}$
  $A^C_{movie}(Movie) = \{title,\ releasedate,\ distributor,\ url,\ category\}$

$A^C_{movie}(Cast) = \{name,\ gender,\ url\}$
$A^C_{movie}(Actor) = \{age,\ hometown,\ \ldots\}$
$A^C_{movie}(Director) = \{address,\ \ldots\}$
$A^C_{movie}(Producer) = \{phone,\ \ldots\}$
$A^C_{movie}(ScreenWriter) = \{language,\ \ldots\}$

- $R_{movie} = \{Acted\text{-}By(Movie,\ Actor),\ Directed\text{-}By(Movie,\ Director),$
  $Produced\text{-}By(Movie,\ Producer),\ Written\text{-}By(Movie,\ ScreenWriter)\}$
- $A^R_{movie} = \{\ A^R_{movie}(Acted\text{-}By),\ A^R_{movie}(Directed\text{-}By),\ A^R_{movie}(Produced\text{-}By),$
  $A^R_{movie}(Written\text{-}By)\}$
  $A^R_{movie}(Acted\text{-}By) = \{year\}$
  $A^R_{movie}(Directed\text{-}By) = \{year\}$
  $A^R_{movie}(Produced\text{-}By) = \{date\}$
  $A^R_{movie}(Written\text{-}By) = \{year\}$
- $H_{movie} = \{(Cast, Actor), (Cast, Director), (Cast, Producer), (Cast, ScreenWriter)\}$

The example Movie ontology is shown as an UML class diagram in Figure 1.1.

Using ontologies as predefined semantic structures, Web annotators interpret Web information as concepts and relationships within the ontologies, and create annotations for them [SMH01]. To distinguish such an annotation approach from the rest, we call this ontology-based **Semantic Web Annotation (SWA)**. Compared to the conventional Web annotations, SWA gives more concise, uniform and integrated semantics to Web information. For example, by using the movie ontology depicted in Figure 1.1, homepages of actors and movies at a movie Web site can be annotated as the instances of *Actor* and *Movie* concepts respectively and they are known as *concept instances*. Likewise, a pair of actor and movie homepages could be annotated as an instance of *Actor-Of* relationship if that actor appeared in the movie. Such a pair of web pages having direct or indirect links among them is known as a *relationship instance*.

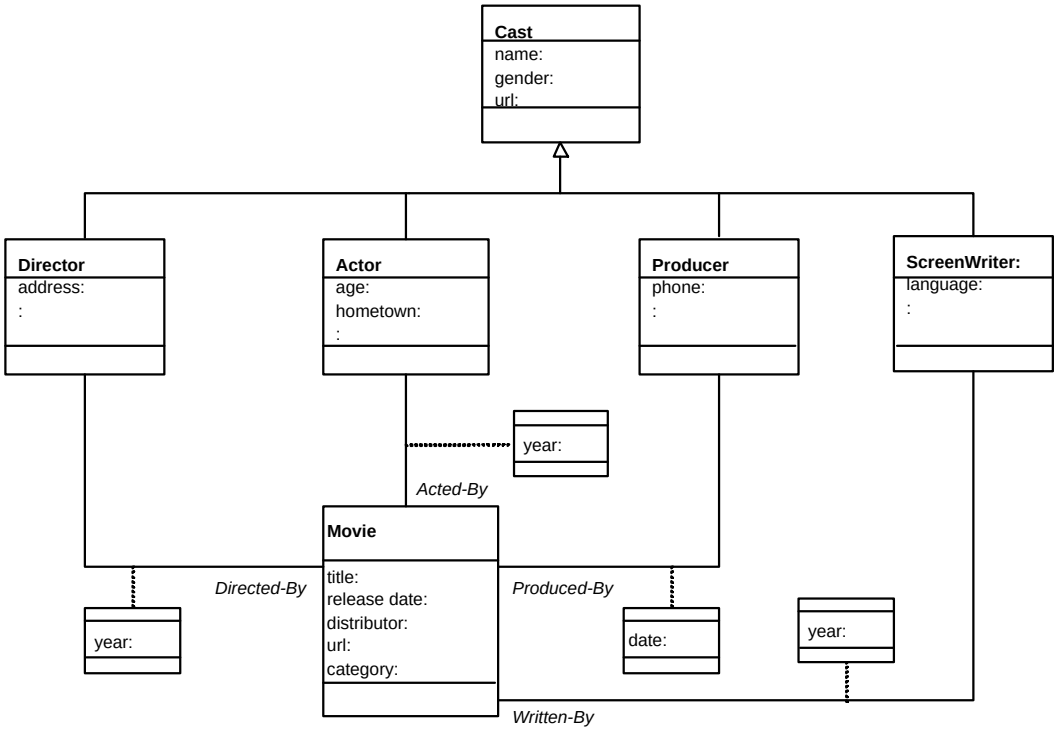



Figure 1.1: A Movie Ontology Example

SWA can be categorized into two types based on the content of annotation instances. The first annotates Web objects with information external to the Web objects [KK01]. The second type of SWA annotates the content (part or whole) of Web objects with attributes of some appropriate concepts [Hef01, DEFS99]. The creation of annotation instances in the former requires manual annotation as the external information must be provided by human annotators. The creation of annotation instances in latter can be done semi-automatically or automatically using various machine learning techniques by extracting information from the Web objects. In this dissertation, we only address SWA of the latter type.

SWA clearly brings much benefits to Web applications that require query processing, browsing and navigation of Web information compared to conventional Web annotation. SWA allows users to formulate more expressive queries on annotation instances and annotated Web objects. More meaningful results can therefore be returned. This is because SWA allows users to formulate queries based on the concepts,

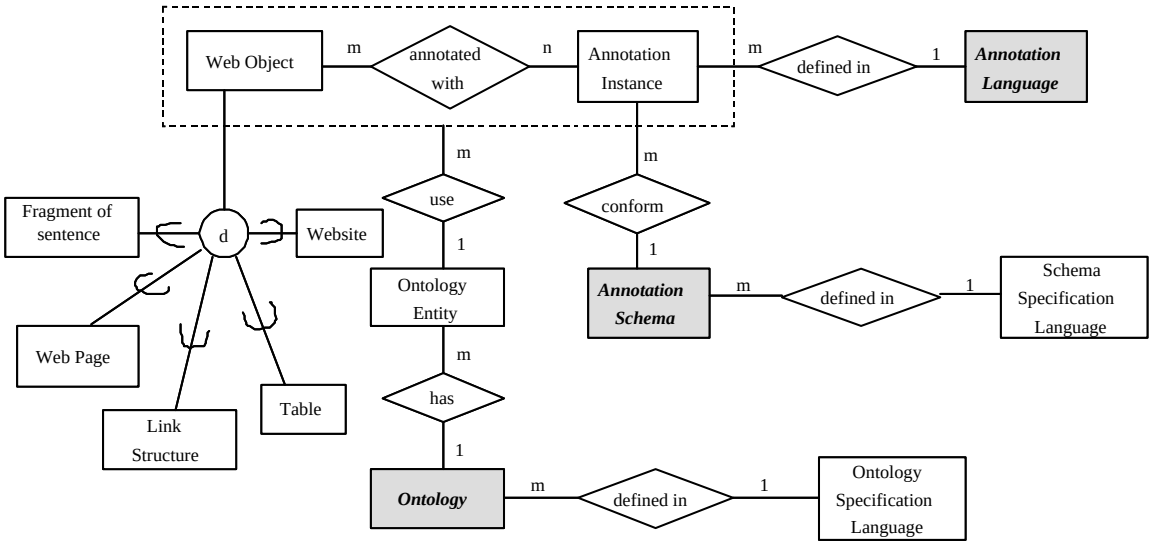



Figure 1.2: Semantic Web Annotation Framework

relationships and attributes from some ontology and to specify query predicates using the ontological knowledge [DEFS99, MI01, MKSA00]. SWA also allows better interpretation of Web object semantics as they are browsed providing a semantically richer context for Web users and application [DBDEM03, QK04].

## 1.2 Semantic Web Annotation Framework

In this section, we present a generic semantic Web annotation framework. The framework aims to describe the different entities involved in SWA and their inter-relationships. The framework represented as an extended ER diagram is shown in Figure 1.2. It includes 3 basic components required in a semantic Web annotation scheme: ontology, annotation language and annotation schema.

In our framework, Web objects are the items to be annotated. A Web object can be a Web site, a Web page, or a part of Web page such as a sentence, anchor element, or table. An annotation instance can be created for a single Web object or a set of Web objects. The latter is required when the Web objects are closely related and may be better to be annotated together. An example is the annotation of both

personal and official homepages of a person since they describe the same person. In the following, we briefly describe the components of our SWA framework.

- Ontology:

  To adopt SWA, an ontology must be used in the creation of annotation instances. There are many ways an ontology can be defined. In our framework, an ontology consists of ontology entities. Ontology serves as the metadata schema for organizing semantically related data on the Web. An ontology is essentially the conceptual model about the Web objects to be annotated. Consisting of mainly concepts and relationships, an ontology can be defined using some specification language and be shared among Web annotations. An ontology specification language is used to define ontologies. As we create an annotation instance, the annotated Web object is associated with some ontology element. This association is recorded within the annotation instance.

- Annotation Language:

  Annotations are defined using some annotation languages. Using the same annotation language and ontology, one can create many different annotation instances for the same Web object. Each annotation language provides the necessary construct to describe the annotated Web objects, mapping them to elements in the ontology(ies). A standardized annotation language also facilitates the sharing of annotations among different annotation systems.

  Figure 1.2 shows that multiple annotation instances (or simply annotations) can be created for each Web object, but each annotation instance is represented in some annotation language. Ideally, one would like to see all annotation instances created using the same annotation language. Nevertheless, the reality is quite different since each annotation language provides different set of annotation features. There are various annotation languages adopted by different annotation systems such as Ontobroker [FDES98], SHOE [Hef01], XML [BPSM00], XOML [CDH+00], RDF [KC04], OCML [Mot98], LOOM [Mac90], etc.. Some

of them are Web-based but others are not. We will survey the existing Web based annotation languages in Chapter 2.

- Annotation Schema:

  Using the same annotation language and ontology, one can create different annotation instances for the same Web object. Hence, to unify the representations, annotation schema is introduced. Annotation schema describes how the annotation instances are formatted. In other words, annotation schema is a common template for instantiating annotation instances. With such a template, it is then possible for any application to parse and extract information from annotation instances. Note that annotation schema itself can be represented in some annotation schema languages such as XML Schema [TBMM01] and RDF Schema [BG02].

## 1.3 Semantic Web Annotation Tasks

Given a set of ontologies $O = \{O_1, O_2, \cdots, O_n\}$ where $O_l = \{C_l, A_l^C, R_l, A_l^R, H_l\}$ and a set of annotated items $I = \{i_1, i_2, \cdots, i_m\}$, where annotated items refer to Web objects to be annotated. SWA consists of the following three sub-tasks to be performed:

- *Concept Assignment Task*

  The assignment of annotated items to concepts can be represented by a set of mapping functions, one for each ontology. We denote the assignment for ontology $O_l$ by a mapping function $f_l : I \rightarrow 2^{C_l}$.

- *Relationship Assignment Task*

  The assignment of annotated item pairs to relationships can be represented by a set of mapping functions, one for each ontology. We denote the assignment for ontology $O_l$ by a mapping function $g_l : I \times I \rightarrow 2^{R_l}$. Note that $g_l(i_j, i_k) = r_l(c_s, c_t)$ only if $f_l(i_j) = c_s$, and $f_l(i_k) = c_t$. In other words, the assignment of

annotated item pairs to relationships must be consistent with the assignment of their pair members to concepts.

- *Attribute Assignment Task*

  Attribute assignment consists of two types: the assignment of literal strings to concept attributes and the assignment of literal strings to relationship attributes.

  Let $\Sigma$ be an alphabet of characters, and $\Sigma^*$ denotes the set of all possible strings constructed from $\Sigma$ contained in a Web object. The assignment of literal strings to the concept attributes can be represented by a mapping function $p_l : I \times A_l^C \rightarrow \Sigma^*$. Note that $p_l(i_j, c_t.a) = w$ only if $f_l(i_j) = c_t$ and $a \in A_l^C(c_t)$.

  The assignment of literal strings to the relationship attributes can be represented by a mapping function $q_l : I \times I \times A_l^R \rightarrow \Sigma^*$. Note that $q_l(i_j, i_k, r_l.b) = w$ only if $g_l(i_j, i_k) = r_l(c_s, c_t)$ and $b \in A_l^R(r_l)$.

Using the movie ontology example, we illustrate SWA results as follows and its equivalent diagram is given in the Figure 1.3.

**Example:**

Let $O_1$ be the $Movie\_Ontology$. Let $i_1$, $i_2$, $i_3$ and $i_4$ be the homepages of a movie "The Princess Diaries", an actor "Mandy Moore", a director "Gary Marshall", and a producer "Whitney Houston", respectively. The following are the possible assignments constituting the SWA tasks.

- *Concept Assignment Task*

  $f_1(i_1) = Movie$

  $f_1(i_2) = Actor$

  $f_1(i_3) = Director$

  $f_1(i_4) = Producer$

  The meaning of the above assignments is self-explanatory.

- *Relationship Assignment Task*

  $g_1(i_1, i_2) = Acted\text{-}By(Movie, Actor)$
  $g_1(i_1, i_3) = Directed\text{-}By(Movie, Producer)$

  The assignment $g_1(i_1, i_2)$ indicates that the annotated item $i_1$ is related to annotated item $i_2$ by the *Acted-By* relationship. This describes that the movie "The Princess Diaries" has "Mandy Moore" as one of the actors/actresses. The assignment $g_i(i_1,i_3)$ is specified in the similar manner.

- *Attribute Assignment Task*

  $h_1(i_1, Movie.url) =$ "$http://movies.yahoo.com/shop?d = hv\&id = 1804857894$
  $\&cf = info\&intl = us$"
  $h_1(i_2, Actor.name) =$ "*Mandy Moore*"

  The above assignments specify a URL attribute for $i_1$, the *Movie* concept instance, and a name attribute value for $i_2$, the *Actor* concept instance.

  $l_1(i_1, i_2, Acted\text{-}By.year) =$ "*2001*"
  $l_1(i_1, i_4, Produced\text{-}By.date) =$ "*August 3, 2001*"

  The above assignments specify a year attribute value for the relationship instance between $i_1$ and $i_2$, and a date attribute value for the relationship instance between $i_1$ and $i_4$.

In order for a widespread use of SWA, one has to consider how to automatically or semi-automatically create annotation instances from the large pool of objects in WWW. In other words, creating annotation instances in SWA is very much like populating the ontology with instances. Unfortunately, this task is difficult and non trivial as the Web data are in diverse format for human consumption, not application software.

Recently, several attempts have employed machine learning techniques to semi-automatically create annotation instances [MNS03, ECJ+99, VVDK+01]. Most of them focused on constructing *concept instances* rather than *relationship instances* as

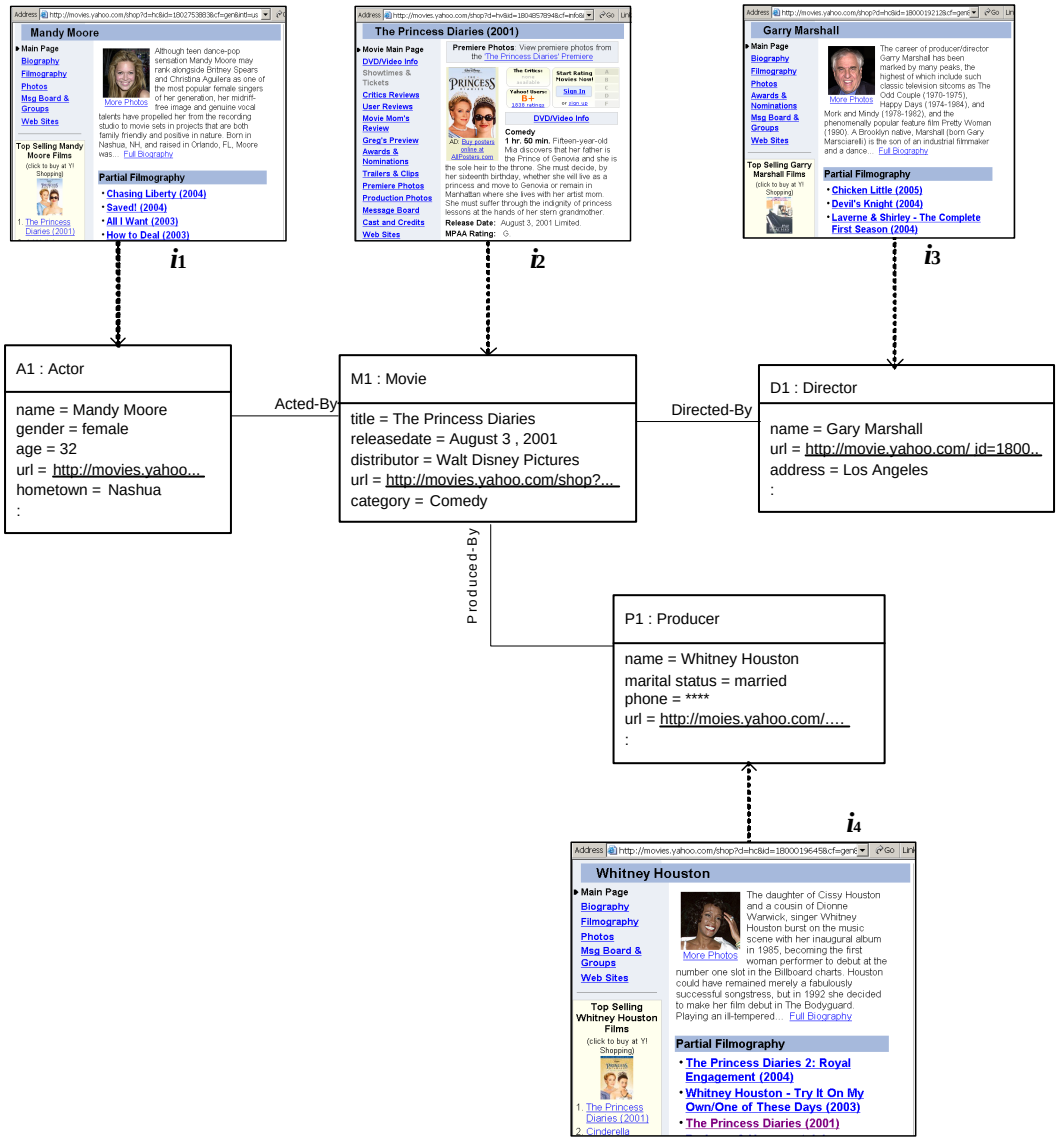



Figure 1.3: Example instances of Movie-Ontology

finding the latter is much more complex. The reason is that a computer program cannot easily tell a pair of items are semantically related to each other without some background knowledge about the items. For example, to find out the relationship instances, one first needs to find out the Web pages that are instances of source concept (e.g.,*Movie*) and target concept (e.g.,*Actor*). Among the many candidate pairs of source and target concept instances, not every source concept instance is semantically related to every target concept instance by some given relationship (e.g.,*Acted-By*).

In this case, human experts are required to examine each candidate pair. This task is time consuming and error prone. In this thesis, we therefore focus on annotating the relationship instances. More specifically, we aim to find pairs of Web pages from some given Web site that can be annotated with relationships.

## 1.4 Research Objectives and Contributions

In SWA, there are still many research issues to be addressed. In this thesis, we contributed to the solutions of the following 3 research objectives.

1. Most existing SWA schemes annotate Web pages or fragments of Web pages with concepts and concept attributes [Ste01, Hef01, FAD+99, ME00]. The annotation of relationship instances usually treats the URI of the target concept instance as an attribute value of a source concept instance. This is done by marking up the anchor link to target Web page in the source Web page. In practice, it is not always true that anchor links to target Web pages appear in the source Web pages. There are other link structures between source pages and target pages. Since these link structures are essential information about pairs of Web pages that are semantically related, it is therefore important to design the representation of link structures for related Web pages. In this thesis, we propose to annotate link structures as link chains. Link chains are a kind of attribute information for relationship instances. We also propose an annotation scheme to incorporate link chains in SWA and a normalization based technique to reduce redundant link chain annotation [NLG02](See Chapter 3).

2. SWA is time consuming when manually done. It is necessary to explore methods to annotate concept and relationship instances automatically or semi-automatically. In this research, we formally define the *link chain extraction problem* that extracts link chains using some training examples from a Web site. We also develop a method that uses sequential covering algorithm and a sophisticated generalization and factoring algorithm to derive the different extraction rules

for anchor elements in the link chains of relationship instances [NLG03] (See Chapter 4).

3. Finally, with SWA, one would like to be able to access annotated Web more effectively. One research challenge is therefore to design and implement a search engine and browser for semantically annotated Web objects. We have developed a search and browsing tool known as Core to assist users to quickly search and navigate the concept and relationship instances using a single user interface. In Core, a new Web search model has been introduced to support both concept and relationship queries. Using Core browser, users can navigate related Web pages by exploiting link chains regardless of the number of intermediate pages among them [NLC05] (See Chapter 5).

## 1.5 Organization

The organization of the thesis is as follows.

- In Chapter 2, we present the literature survey of the current state-of-the-art SWA languages and techniques.
- In Chapter 3, we propose to annotate link information between related Web pages as link chain information and introduce our proposed annotation schema.
- In Chapter 4, we propose the link chain extraction problem and describe a link chain extraction method.
- A concept and relationship instance search and browsing tool, Core, is presented in Chapter 5.
- Finally, we conclude the thesis in Chapter 6.

# Chapter 2

# Literature Survey

In this chapter, we survey different semantic Web annotation (SWA) languages and techniques.

## 2.1 Languages for Semantic Web Annotation

Language for representing SWA is an essential component of our proposed framework. There are several languages that can be used to represent annotations on Web objects. We will focus on only annotation languages that are designed to be used on the Web. There are other non-Web based annotation languages (e.g. OCML [Mot98]) but will not be covered here. By representing annotations using Web-based languages, the annotations themselves can exist on the Web together with the annotated Web objects. Furthermore, using Web as the common medium, annotations can be easily shared among different users. However, these Web languages provide different features that distinguish the language abilities in representing annotations.

Note that there are languages that can be used to specify ontologies. Some of these Web languages can also be used to represent Web annotations. Since our focus is not on ontologies but on ontology-based SWA, we will not discuss such languages in this section. A survey of such ontology specification languages can be found in [CGP00].

Before we examine the different languages for Web annotations, we first describe the features to look for in such annotation languages. Based on these features, a comparison between the languages can be carried out.

(i) **Ability to support annotation schemas:** While annotation languages ensure that Web annotations are machine readable, they alone could not describe how the annotations are formatted. The description of the format of annotations is known as a **annotation schema**. Typically, an annotation schema is created for each ontology used in annotations. When an annotation language supports annotation schemas, it provides a reference to the annotation schema, or provide language constructs to define annotation schemas. For annotation languages that do not support annotation schemas, it is the responsibility of applications to interpret annotations based on some prior knowledge about the annotation formats. Such a prior knowledge could be possible if either there is only one single standard format for formatting the annotations, or the annotation formats have been hard-coded into the parser modules.

(ii) **Independence of annotation instances from annotated Web objects:** The independence of annotation instance refers to whether the created Web annotations can be represented separately from the annotated Web objects. In some annotation languages, the Web annotations can be defined and stored separate from the annotated Web objects. This demonstrates full independence between the annotation instances and the annotated Web objects. On the other hand, there are annotation languages requiring Web annotations to be embedded in the annotated Web documents or objects. Such languages are said to be dependent on the annotated Web objects.

(iii) **Ability to assign Web objects to concepts in ontologies:** An important part of SWA is about assigning Web objects or annotated items to concepts in the given ontologies. The type of Web objects to be annotated could be a Web page, group of Web pages, or even a Web site. Most annotation languages support annotation of Web pages or fragments of Web pages but there are only a few that support annotation of other forms of Web objects (e.g. group of Web pages or Web site).

(iv) **Ability to assign pairs of Web objects to relationships in ontologies:** Another important part of SWA is about assigning pairs of Web objects to relationships in ontologies. The support of annotations involving relationships varies for different annotation languages. In some cases, an annotation language may fully support annotations involving relationships. In others, only limited or even no support may be given.

(v) **Ability to assign content of Web pages to attributes of concepts/ relationships in ontologies:** Assignment of values to concept and relationship attributes is part of SWA. In some annotation languages, the values to be assigned could be derived from the content of the Web objects (e.g., Web pages). There are two ways to support this task: (1) enclosing the Web segment with tags (elements) and (2) annotating the URI of the Web segment. The ability to explicitly assign values to attributes of concepts and relationships is found in all existing annotation languages.

Table 2.1 gives overall comparisons of the Web-based annotation languages. In the following, we group them under three categories, namely HTML-based, XML-based and RDF-based annotation languages.

### 2.1.1 HTML-based Annotation Languages

HTML provides a `META` tag to represent metadata information within the header of Web pages [Sul02]. A `META` tag allows one to assign property and value information to a Web page in the following format:

⟨`META name=`*property name* `content=`*value*⟩

While not originally designed for annotation, `META` tag can be used in a limited way to support annotations. `META` tag is clearly not expressive enough to define annotation schemas. It however can be used to reference an annotation schema. `META` tag can also be used to assign its host Web page to concepts in an ontology. For example, ⟨`META name="Movie.concept" content="Actor"`⟩ assigns the host Web page to the *Actor* concept in the Movie ontology.

| | **Annotation Schema Support** | **Independence of Annotation Instances** | **Concept Assignment** | **Relationship Assignment** | **Attribute Assignment** |
|---|---|---|---|---|---|
| **Pure Meta Tag** [Sul02] | No support | Annotation instances must be part of annotated web objects | Only limited support- Web page annotation only | Similar to attribute assignment | HTML labels as attribute values |
| **Ontobroker** [FAD+99] | No support | Annotation instances must be part of annotated web objects | Web Page annotation only | Similar to attribute value assignment | Portions of web pages can be assigned as attribute values |
| **SHOE** [Hef01] | No support | Annotation instances must be part of annotated web objects | Web page annotation only | Similar to attribute value assignment | Attribute values must be explicitly given |
| **Onto Web** [BG01] | No support | Annotation instances can be independent | Any web object with URI can be annotated | Similar to attribute assignment | Yes, Using URI |
| **Ontology Based Annotation Project** [CDH+00] | Yes | Annotation instances can be independent | Any name entity in a Web document can be annotated as concept instance | Similar to attribute assignment | No |
| **Onto-Annotate** [SMH01] | Yes, Using RDF(S) | Annotation instances are stored in a local database | Web Page annotation only | Similar to attribute assignment | Yes, Using URI |
| **Annotea** [KK01] | Yes, Using RDF(S) | Annotation instances are stored in a RDF database | Any web objects with URIs or referenced by XPointer can be annotated as concept instances | Similar to attribute (property) assignment | Yes, Using URI |

Table 2.1: Comparison of Web-based Annotation Languages

META tags must be embedded in the annotated Web pages. In other words, the annotation instances are not independent from the annotated Web pages. META tags also cannot assign group of Web pages or Web sites to concepts in ontologies. META tags can relate two Web pages together by treating relationships similar to attributes except that URLs of the related web pages are used as content. For example, to relate a Actor homepage with a Movie homepage, the following META statement can be used.

⟨META name="Movie.Actor.Actor-Of"
content="http://movies.yahoo.com/shop?d=hv&id=1808415480&cf=info&intl=us"⟩

META tag can assign concept and relationship attribute values as illustrated by the following statements.

⟨META name="Movie.Actor.name" content="Mandy Moore"⟩
⟨META name="Movie.Actor.Actor-Of.year" content="2003"⟩

Instead of using the pure HTML (i.e., META tag) approach, the Ontobroker [FAD+99] and SHOE [Hef01] Web annotation projects have proposed their own annotation languages by extending HTML with additional language constructs. In the following, we examine the key features of the two annotation languages based on the earlier five language features.

- **Annotation Schema Support**:

  In Ontobroker, the notion of annotation schema does not exist. Ontobroker adopts a single standard format for representing Web annotations. Hence, such Web annotations can be parsed by any applications supporting the standard format.

  SHOE, like Ontobroker, does not support annotation schemas. Hence, there is only a single standard format for representing Web annotations.

- **Independence of Annotation Instances from Annotated Web Objects:**

  In Ontobroker, annotation instances are embedded in the annotated Web pages by extending the anchor tag ⟨A⟩ with the onto attribute. For example, to annotate a Web page at '$http://movies.yahoo.com/shop?d=hc\&cf=gen\&id=$

1802753883&$intl = us$' to be an instance of the *Actor* concept, the following anchor element is added to the Web page.

⟨a onto = "'http://movies.yahoo.com/shop?d=hc&cf=gen&id=1802753883 &intl=us':Actor"⟩ ⟨/a⟩

SHOE also requires all annotation instances to be embedded in the corresponding annotated Web pages. Instead of extending the anchor element for annotation purposes, SHOE adds a few more special tags for representing annotation instances, e.g., ⟨INSTANCE⟩,⟨CATEGORY⟩, ⟨RELATION⟩, etc.. For example, to annotate a Web page to be an instance of the *Actor* concept in the Movie ontology, the following element is added.

⟨CATEGORY NAME = "Movie.Actor"⟩

- **Assignment of annotated items to concepts of ontology:**

  Ontobroker supports Web page annotation only.

  SHOE supports Web page annotation only too.

- **Assignment of annotated item pairs to relationships of ontology:**

  Ontobroker annotates relationships between a pair of Web pages by adding relationship information into an anchor element with the `onto` attribute. For example, to assign the *Acted-By* relationship between a movie homepage and its actor's homepage (with $url_{actor}$ as its URL), the following anchor element is embedded in the former page.

  ⟨a onto = "page[Acted-By= href]" href=$url_{actor}$ ⟩ ⟨/a⟩

  In the above tag, `page` indicates that the host Web page is the target of annotation and `href` indicates that the actor instance will be identified by its URL. Note that relationships can be established between two annotated Web pages even when they do not have hyperlink connections between them.

  SHOE annotates relationships in a way very similar to Ontobroker. The element used is shown below.

  ⟨RELATION NAME="Movie.Acted-By"⟩

⟨ARG POS=TO VALUE = $url_{actor}$⟩

⟨/RELATION⟩

- **Assignment of content of Web pages to attributes of concepts and relationships in ontology:**

  Ontobroker allows anchor tags to be used to markup portions of Web page content as attribute values of some concepts or relationships. For example, to markup the name of actor in an Actor homepage, the following anchor element is added to enclose the actor name value in the Web page.

  ⟨a onto = "page[name= body]"⟩Mandy Moore⟨/a⟩

  In the above example, the name value (i.e., "Mandy Moore") in the original Web page has been marked up. Hence, the name value will appear as an anchor label after tagging.

  SHOE uses its special tags to assign attribute values to concept and relationship instances. Instead of marking up portions of Web page content as attribute values, all attribute values must be explicitly specified. For the same example, SHOE uses the following elements.

  ⟨RELATION NAME="Actor.name"⟩

  ⟨ARG POS=TO VALUE = "Mandy Moore"⟩

  ⟨/RELATION⟩

### 2.1.2 XML-based Annotation Languages

XML (Extensible Markup Language) is a markup language developed by the XML Working group of the World Wide Web Consortium(W3C) for specifying structured documents and data on the Web [BPSM00]. It allows the creation of documents with customized markup tags so that the semantic structures of the documents can be made explicit for parsing and the document content can be easily extracted. However, XML alone does not define the standard set of tags and the document structure. A way

to introduce a standard set of tags to XML documents is by the use of either XML Schema [TBMM01] or DTD (Document Type Definition)[BBC+98].

Figure 2.1 depicts an annotation schema defined for Movie ontology using XML Schema. Figure 2.2 shows an annotation represented in XML using the annotation schema.

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://www.movie.org"
          xmlns="http://www.movie.org">
  <xsd:complexType name="Movie">
        <xsd:sequence>
            <xsd:element name="title" type="xsd:string"/>
            <xsd:element name="category" type="xsd:string"/>
            <xsd:element name="url" type="xsd:anyURI"/>
        </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Actor">
         <xsd:sequence>
              <xsd:element name="name" type="xsd:string">
              <xsd:element name="gender" type="xsd:string">
              <xsd:element name= "url" type= "xsd:string">
        </xsd:sequcence>
  </xsd:complexType>
   :
   :
</xsd:schema>
```

Figure 2.1: XML Schema for part of Movie Ontology

The purpose of a XML Schema or a DTD is to specify a common template to be followed by all XML documents created using the XML Schema or DTD. There are many similarities between XML Schema and DTD although XML Schema has a few additional advantages such as the support of data types.

In the following, we comment on the use of XML to annotate Web objects in two XML-based annotation languages. The first has been adopted by the Onto Web Project [BG01], and another by the Ontology Based Annotation Project by Czejdo *et al* [CDH+00, CS00].

```
<?xml version="1.0"?>
<Movie xmlns ="http://www.movie.org"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.movie.org/ Movie.xsd">
  <Actor>
        <name> Many Moore </name>
        <gender> female </gender>
        <url> http://movies.yahoo.com/shop?d=hc&cf=gen&id=1802753883&intl=us</url>
        <hometown> Nahua </hometown>
  </Actor>
  :
  :
</Movie>
```

Figure 2.2: XML annotation instance of *Actor* concept

- **Annotation Schema Support:**

  Onto Web Project uses purely XML to annotate Web pages. Unlike the other projects, Web pages here are assumed to be XML documents. Since XML Schema and DTD are not used, the annotation language does not support annotation schemas. Instead, it supports a fixed set of tags for assigning elements in a XML document to concepts and attributes in the ontology, thus keeping the annotation information easy to parse.

  In Ontology Based Annotation Project, an annotation language called XOML (Extended Ontology Markup Language) is introduced to annotate Web pages. XOML can be defined with different DTDs allowing it to annotate Web pages with different ontologies. It therefore supports annotation schema.

- **Independence of Annotation Instances from Annotated Web Objects:**

  In Onto Web Project, annotation instances can be stored in a separate repository. Hence, annotation instances are independent from the annotated Web objects.

  In Ontology Based Annotation Project, XOML annotations are generated through a series of steps of extracting concept and relationship information from the annotated Web pages. The final XOML annotations are designed to be independent from the annotated Web pages.

- **Assignment of annotated items to concepts of ontology:**

  Onto Web Project deals with annotation of XML documents (HTML documents are first converted to XML format for annotation) and any element in an XML document can be annotated as a concept instance. The annotated elements can be identified by their uniform resource identifiers (URI).

  Unlike the other projects, Ontology Based Annotation Project assumes that multiple concept instances can appear within a Web page. Text fragments are assigned to concepts in the ontology using NLP technique.

- **Assignment of annotated item pairs to relationships of ontology:**

  In Onto Web Project, the relationship instances in a XML document can be annotated by treating them as properties of some concept instances. The related concept instances can be identified by their URIs.

  Ontology Based Annotation Project extracts relationship information from the annotated Web objects and represents them separately using relationship element. The relationship instances are pairs of text fragments.

- **Assignment of content of Web pages to attributes of concepts and relationships in ontology:**

  In Onto Web Project, attribute values can be assigned to concept and relationship instances. The assignment may involve explicit values, or references to some elements in the annotated XML documents.

  Ontology Based Annotation Project does not support assignment of content of Web pages to attributes of concepts and relationships as there is no provision of references to the content of annotated Web pages.

### 2.1.3 RDF-based Annotation Languages

**RDF (Resource Description Framework)** [Mil98, SEMD00], strictly speaking, is a standard for describing resources including Web objects. It is a data model, not

a language. It assumes that every Web object to be described (annotated) must have a URI, and each object can be described by property types and property values. The Web object may be an entire Web page, part of a Web page, a Web site or an object that is not directly accessible via the Web. It describes an annotation by a set of triples each consisting of resource, property type and property value. To adopt the RDF model in an annotation language, we often have to incorporate RDF into XML.

While RDF provides a common model to describe or annotate a Web object, it does not provide a standard vocabularies (that is, the set of property types) to be used in an ontology. Hence, using RDF alone with XML for SWA requires the annotators and annotation users to have implicit agreement on how the annotations are formatted. With RDF schema, RDF(S), such an agreement is not longer needed since RDF schema can encode the ontology information, and can be referred by RDF. An example RDF schema is given in Figure 2.3. The schema example has been used to create an annotation instance shown in Figure 2.4.

```
<rdf:RDF xmlns:lang="en"
        xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns\#"
        xmlns:rdfs:="http://www.w3.org/2000/01/rdf-schema\#">
 <rdfs:Class rdf:ID="Movie">
 </rdfs:Class>
 <rdfs:Class rdf:ID="Actor">
        <rdfs:subClassOf rdf:resource="\#Cast"/>
 </rdfs:Class>
 <rdf:Property rdf:ID="title">
        <rdfs:domain rdf:resource="\#Movie"/>
        <rdfs:range
          rdf:resource="http://www.w3.org/2000/01/rdf-schema\#Literal"/>
 </rdf:Property>
 <rdf:Property rdf:ID="Acted-By">
        <rdfs:domain rdf:resource="\#Movie"/>
        <rdfs:range rdf:resource="\#Actor"/>
 </rdf:Property>
<rdf:RDF>
```

Figure 2.3: RDF Schema for part of Movie Ontology

In RDF Schema, the basic modelling primitives are: *class* for defining concepts, *subclass-of* for parent-child relationships between concepts, *property* for concept attributes, *subproperty-of* for properties of attributes, etc.. All data types of RDF(S) attributes must be literal strings.

```
<rdf:RDF xmlns:lang="en"
        xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns\#"
        xmlns:movie:="URL of MovieOntoRDF\#">
 <rdf:Description rdf:ID="How to Deal">
        <rdf:type rdf:resource = "#Movie"/>
 </rdf:Description>
 <rdf:Description rdf:ID="Mandy Moore">
        <rdf:type rdf:resource = "#Actor"/>
 </rdf:Description>
 <rdf:Description about="#How to Deal">
        <movie:Acted-By rdf:resource="#Mandy Moore">
 </rdf:Description>
 <rdf:Description about="#How to Deal">
        <movie:url rdf:resource="http://movie.yahoo.com/shop?d=hv&id=1808415480&cf=info&intl=us">
 </rdf:Description>
</rdf:RDF>
```

Figure 2.4: RDF instance example of Movie Ontology

The annotation projects using RDF-based annotation languages include OntoAnnotate [EMSS01, SMH01], and Annotea [KK01]. In the following, we analyze the annotation languages used in the two projects.

- **Annotation Schema Support:**

  OntoAnnote is the ontology-based semantic annotation tool in which underlying annotation language is RDF and RDF Schema. Hence, their own annotation schema is supported in this environment by using RDF Schema. Reference to annotation schema use the following:

  ⟨`xmlns:`
  `ka2="http://www.semanticWeb.org/ontologies/ka2-onto-2000-12-07.rdfs"`⟩

  The annotation language in Annotea is based on RDF. RDF Schema is used to describe the properties of annotations. It therefore supports annotation schema and reference to annotation schema based on W3C standards.

- **Independence of Annotation Instances from Annotated Web Objects:**

  In OntoAnnotate, annotations instances can be separate from annotated items, and are therefore independent from the latter.

  In Annotea, since annotations are separate from the annotated items and can

be stored in either local or shared annotation servers, annotation independence is supported.

- **Assignment of annotated items to concepts of ontology:**
  In OntoAnnotate, not only HTML documents but also Microsoft office documents like Excel and Word documents can be annotated as concept instance by citing the object identifier http://example.com/some/person:rst as shown in the following example.
  ⟨ka2:Actor rdf:ID="http://example.com/some/person:rst"⟩ ⟨/ka2:Actor⟩
  In Annotea, any Web object with URIs or referenced by XPointers can be annotated as concept instances.
  ⟨a:annotates r:resource=http://example.com/some/page.html"/⟩

- **Assignment of annotated item pairs to relationships of ontology:**
  In OntoAnnotate, a relationship instance is represented by a property with a Web object as its value. The URI of the Web object is assigned to the property as shown in the following example.
  ⟨ka2:director⟩
  http://movies.yahoo.com/shop?d=hc&id=1807745570&cf=gen&intl=us
  ⟨/ka2:director⟩
  In Annotea, relationship instances are assigned by specifying their URIs as properties values. For example,
  ⟨a:body r:resource= "http://www.example.com/mycomment.html"/⟩

- **Assignment of content of Web pages to attributes of concepts and relationships in ontology:**
  In OntoAnnotate, attribute instances are explicitly assigned as literal strings. For example,
  ⟨ka2:firstName⟩Clare⟨/ka2:firstname⟩
  ⟨ka2:lastName⟩Kilner⟨/ka2:lastName⟩

In Annotea, users are allow to explicitly assign the attribute values. Users can also select some texts in the annotated items to assign attribute values.

⟨d:creator⟩Clare Kilner⟨/d:creator⟩

## 2.2 Semantic Web Annotation Techniques

With large volume of information on the Web, manual semantic annotation is expensive and time consuming. Automated techniques are therefore required to address the three tasks of SWA. In this section, we survey several Web classification, information extraction (IE) and Natural Language Processing (NLP) techniques that are relevant to SWA.

### 2.2.1 Web Classification Techniques

Classification is a process of organizing objects into pre-defined classes or categories [SLN03]. In classification, user labelled objects are used as training examples to train classifiers for classifying the unseen objects/unlabelled objects. Each object is represented by a set of features and the classifiers are trained to discriminate the features for different classes or categories.

In SWA, the purpose of *concept assignment task* is to assign a collection of Web objects to the concepts in the ontology. When the annotated items are Web pages, Web page classification techniques can be adopted. When the objects to be classified are Web sites, Web site classification can be used instead.

Most of the Web page and Web site classification techniques were derived from *text classification*. The features in text classification techniques are solely based on the text content (e.g. words or terms) of documents [CS99, KS97, NGL97]. The text features are obtained by using different weighting schemes such as *set-of-words* [KS97, SS00, SL01, SLN03], *bag-of-words*, and *Term Frequency Inverse Document Frequency (TFIDF)* [SB88, Seb02].

### Web Page Classification

In Web page classification, the Web objects to be classified are Web pages. Unlike plain text documents, Web pages contain not only text but also the HTML tags, links and meta data. Therefore, in Web page classification, in addition to text content, HTML tags, anchor text and links provide interesting features for classifying Web pages.

The HTML tags are the basic structural elements in a Web page. They include headings, titles, paragraphs, tables, lists, links, etc.. Web page classification based on HTML tags treat words appearing in different structural elements as separate features. Words are given different weights depending on which the structural elements they appear in. For example, a word appearing in the title elements may be given 5 times more important than the same word appearing in the anchor element, and 10 times more important than the same word appearing in a paragraph. Sun *et al* reported that good accuracy can be achieved by adding anchor texts as additional features [SLN02].

The use of anchor text features is based on two observations: (1) Text alone is insufficient for a reliable classification and (2) The anchor texts associated with the incoming links of Web pages carry useful information [Fur99]. For example, a department homepage may be the target of several links containing anchor text such as "computer science department", "CS department", "Department of CS", etc.. Based on these observations, Furnkranz proposed to use three kinds of information in representing each Web page: the anchor text (i.e. the text on the link), the paragraph where the anchor text appears and the headings of the section where the link occurs [Fur99]. Each type of information forms one feature-set. For each feature-set, a decision rule classifier is built and a Web page is classified based on the outcome of these classifiers. Experiments have shown that his method outperforms the classifier based on text alone. However this method is not applicable when a Web page does not have any link pointing to it. Several anchor text features have also been adopted by other researchers [YSG02, SLN02, GTL+02].

Web page classification based on concept of neighboring Web pages was investigated in [CDI98, OML00, LG03]. Neighboring Web pages of a page refer to pages that are one link away. The concepts assigned to neighboring Web pages can be used as features. In Web page classification proposed by Chakrabarti *et al*, the concept labels assigned to a Web page can reinforce the concept labels of its neighboring Web pages using a probabilistic framework [CDI98]. Their experiments have shown that the use of neighboring Web pages reduced classification errors significantly for a small set of YAHOO! pages. Another classification technique based on the concept of neighboring Web pages is proposed by Oh *et al* [OML00]. Their technique first classifies Web pages with the largest number of links and this provides concept hints to the neighboring Web pages. The assigned concepts are in turn used as hints and propagated to other neighbor Web pages. Their technique was experimented using Naïve Bayes classifier on a collection of Korean articles. They achieved higher $F_1$ measures than the technique that do not consider the concept of neighboring Web pages. In Web page classification technique proposed by Lu and Getoor's, each Web page is classified by combining the probability based on content features and the probability based on link distribution [LG03]. The link distribution of a Web page is represented by a histogram or counts of the occurrences of different concepts of incoming, outgoing and co-citation links for that page. Their experiments have shown that the classification based on both content and link distribution outperforms the classification based on the content alone.

Craven and Slattery proposed a new classifier known as Foil-Pilfs that combines Naïve Bayes classifier and First Order Inductive Learner (Foil) [CDF+00, CS01]. In their approach, Web pages are classified as *concept instances* and the pairs of Web pages are classified as *relationship* instances. The Naïve Bayes classifier is first used to classify Web pages using text features only. The concept(s) assigned by Naïve Bayes classifier are later augmented by the Foil classifier to recognize *concept instances* and relationships among them [CSN98]. Foil [QCJ93] is used as a relational learner using *relational path finding* method [RM92]. It is capable of describing hyperlink patterns

that occur across multiple pages in terms of a graph structure. The experiments on WebKB dataset have shown that the Foil-Pilfs classifier provides more accurate classification results than the Naïve Bayes classifier that use only text components in the Web pages [CS01].

### Web Site Classification

In Web site classification, the Web objects to be classified are Web sites. Very few works have been conducted in this area.

Ester *et al* define a Web site of internet domain $D$ as a directed graph $G_D(N, E)$ where $N$ denotes a set of Web pages within the domain $D$ and $E$ represents the links between pages [EKS02]. Based on this definition, a Web site can be represented in three approaches: *superpage approach*, *topic frequency vector* and *a page tree*.

In the *superpage approach*, a Web site is represented as a single virtual Web page consisting of the union of all pages from the Web site. In this approach, a Web site is treated as one large HTML document and standard Web page classification techniques are directly applied on it without considering page structure. Using superpage approach, Ester *et al* presented Web site classification using Naïve Bayes classifier on a dataset consisting of 82842 Web pages from 207 Web sites [EKS02]. Pierre also employed the superpage approach using both text and meta data components of Web pages and obtained better classification accuracy [Pie01].

*A topic frequency vector* approach represents a Web site by a vector of keyword frequencies. In this approach, the content of each Web page is assigned a keyword taken from a pre-defined vocabulary (e.g. company, places and opening hours, products and services, etc.) using traditional Web page classification techniques. The underlying assumption in this approach is that words in a page can only derive the concept label of the page but not the entire Web site. A vector of keyword frequencies is therefore derived to represent a Web site. Ester *et al* conducted experiments on the same dataset using Naïve Bayes and decision tree classifiers and achieved higher accuracy than the superpage approach [EKS02].

In the *page tree* approach, a Web site is represented as a tree consisting of Web pages and links among them. Starting from the homepage of a Web site, a page tree of a Web site is constructed by using breadth-first search on all the Web pages while ignoring the links to pages that have already been visited. The basic idea is that most Web sites have hierarchical structures, they normally provide general information in the homepage which contains links to the pages that contain more specific information. Ester *et al* represent each page by a keyword. The *page tree* of a Web site is therefore a tree of keywords. By classifying the *page tree* of a Web site using a Markov tree model, they achieved high accuracy compared to the previous two approaches [EKS02].

### 2.2.2 Information Extraction Techniques

Several information extraction (IE) techniques are adopted when annotated items are fragments of Web pages/literal strings. There are many information extraction (IE) techniques that have been developed[1]. Instead of covering all the methods, our survey in this section will only focus on the work related to Web information extraction.

Depending on the data slots to be extracted and the number of Web pages involved, Web extraction methods can be classified using the taxonomy given in Figure 2.5 [LRN02, Mus99]. As shown in the taxonomy, Web extraction methods can support *single-slot* or *multi-slot extraction*. The former generates extraction rules that can only extract one or more data values of the same type and they are not semantically related to one another. Multi-slot extraction methods generate rules that are able to extract one or more tuples such that each tuple consists of a set of two or more attributes (or slots).

The single-slot Web extraction methods include Rapier [CM98] and SRV [Fre98]. The multi-slot Web extraction methods can be further divided depending on the number of pages containing the slots forming a tuple. In normal circumstances, all slots of a tuple can be found within a single page. An example is a Web page

[1]A listing of information extraction systems can be found at http://www.isi.edu/info-agents/RISE/.

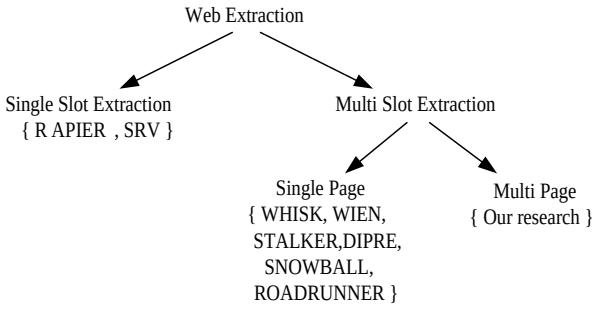



Figure 2.5: Taxonomy of IE Methods

containing a table of product items and their price information. When a tuple has slots deriving from multiple pages, we have a *multi-page multi-slot* Web extraction problem. An example is a Web page containing a table of product item names (i.e., product name slot) each appearing in an anchor element leading to the Web page giving more detailed fields (e.g., product size, price, weight, etc., slots) about the product item. Almost all multi-slot Web extraction methods are designed for single pages. In other words, they assume that all slots of a tuples can be found in a single Web page. These include WHISK [Sod99], WIEN [Kus00], STALKER [MMK01], DIPRE [Bri98], SNOWBALL [AGP+01] and RoadRunner [CMM01]. Very few research is found on *multi-page multi-slot* Web extraction.

### Single-Slot Web Extraction

Rapier generates extraction rules that consist of three parts: *a pre-filler*, *a filler* and *a post-filler* patterns [CM98]. The *pre-filler* and *post-filler* patterns indicate the immediate preceding and following delimiters of the item to be extracted. The *filler* pattern describes the structure of the item to be extracted. The structure imposes constraints on the length, part of speech and semantic class of the items. The extraction rule learning is based on inductive logic programming.

In SRV, extraction rules are learnt from labelled sequences from training documents [Fre98]. SRV has extraction rules based on features defined over tokens. The features include simple properties of the item to be extracted, e.g., *word*, *numeric*,

and *punctuation*, and relational properties that involve both the item to be extracted and its neighboring items. The extraction rules are learnt using a top-down relational rule learning algorithm. SRV has been used in the WEB→KB project to extract attribute values of Web pages belong to some concepts [CDF+00].

### Single-Page Multi-Slot Web Extraction

Multi-slot Web extraction is studied in several IE research projects. WHISK adopts a top-down rule induction approach to generate extraction rules for information contained in a Web page or a free text document [Sod99]. From a given set of annotated training Web pages, WHISK generates extraction rules in the form of regular expression patterns to match Web page segments.

STALKER proposed a single-page multi-slot Web extraction using single-slot extraction rules [MMK01]. It assumes that some examples have been labelled in the training Web pages, and an Embedded Catalog (EC) Tree is given by the user to describe the hierarchical page structure. STALKER generates extraction rules that derive tuples from a Web page by combining extracted slots belonging to the same tuple(s). STALKER's extraction rules are general enough to handle documents with slightly different formats by allowing rules to have disjunctive predicates. The learning algorithm for these extraction rules is based on sequential covering.

WIEN is another rule induction based Web extraction method [Kus00]. An automatic labelling algorithm is developed to obtain the training examples. From the labelled Web pages, extraction rules of *HLRT* format (containing a Head delimiter, a set of Right and Left delimiters of the item to be extracted, and a Tail delimiter) are learnt. Rules adopted by WIEN are similar to that of STALKER but are not designed to handle the nested structure of Web pages as they only use delimiters that immediately precede and succeed the data to be extracted. An inductive learning approach is used to learn the extraction rules.

DIPRE extracts pairs of items from a Web page [Bri98]. DIPRE has extraction rules in the form of five tuples (*order*, *urlprefix*, *prefix*, *middle*, *suffix*). The *order* reflects the order in which the items appear, the *urlprefix* refers to the URL pattern

of those pages that contains the item pairs, the *prefix*, *middle* and *suffix* are the text on the left of the first item, text between the two consecutive items and text on the right of the second item respectively. The extraction rule learning method is based on bootstrapping.

The Web extraction method in SNOWBALL is built on the idea of DIPRE. SNOWBALL generates the rules for extracting pairs of items from a free text document [AG00, AGP+01]. Each extraction rule is a five tuple ⟨*left*, *item1*, *middle*, *item2*, *right*⟩ where *item1* and *item2* are named-entities and *left*, *middle*, and *right* are the patterns for matching the left of the first item, text between the two items and text on the right of the second item respectively. Each pattern is represented by a set of vectors containing the terms. Each vector has a non-zero weight for each term. Matching between a text fragment and a pattern is done by computing the similarity between their respective vectors. Bootstrapping has been used to learn the extraction rules.

RoadRunner proposed an automatic extraction rule learning method [CMM01, CMM02]. The method does not require user specified training examples and does not rely on any a priori knowledge about page structures and content. Its extraction rule is defined as a regular expression that describes the common page structure of the example pages. The rule generation method considers two Web pages at a time, analyze the similarities and dissimilarities between them, progressively refines and generate the extraction rules.

Amilcare[2] is an adaptive information extraction tool which is adaptable to any application domain with natural language text documents [FNR03, CW03]. Amilcare performs information extraction by adding texts with XML tags. Amilcare provides two modes for application development:*system driven* and *user driven*. In *system driven mode*, Amilcare generates rules with user provided training examples and trained the system, generate the rules and test them on unseen documents. The rule learner is based on sequential covering algorithm $LP^2$ [Cir01]. The learner induces two types of rules: *tagging rules* and *correction rules*. A *tagging rule* inserts

[2]http://nlp.shef.ac.uk/amilcare/overview.html

an XML tag in the texts (e.g. <hotel>). The correction rule shifts misplaced tags (inserted by tagging rule) to the correct position. In *user driven mode*, users are allowed to inspect the induced rules and revise them. Amilcare combines with a natural language processing tool GATE [MTC+02, CMBT02] for pre-processing the natural language documents. Amilcare has been adapted in semi-automatic semantic Web annotation systems S-CREAM [HSC02] and MnM [VVMD+02a, VVMD+02b].

### 2.2.3 Natural Language Processing Techniques

Natural language processing (NLP) techniques are adopted for annotating the documents when annotated items are text fragments from natural language text.

NLP techniques include four sub-tasks: text tokenization, sentence splitting, part of speech tagging, and named entity recognition [CMBT02]. *Tokenizer* splits the text documents into tokens such as numbers, punctuation, symbols, and words of different types (e.g all capital, small, initial capital, etc.). Usually, the tokenizer does not need to be modified for different application domains, or text types. *Sentence splitter* is used for identifying sentences in the text document. Usually, a piece of text is segmented into sentences by periods. There are however some other exceptional situations that need to be handled (e.g., abbreviation of names). *Part of speech tagger* performs the grammatical tagging of nouns, verbs, prepositions, etc.. The main challenge in part of speech tagging is to address lexical ambiguities. For example, in the sentence "He books tickets", the word "books" is a third person singular verb, but a plural noun in the sentence "He bought books". Named entity recognition (NER) refers to annotating fragments of a document with entities such as people, places, times and amounts [CS04]. Variants of hidden Markov models have also been widely used to perform the name entity recognition [ZS02, CS04]. Based on these models, both syntactic and semantic rules are generated to recognize the name entities of different types. In order to recognize the name entities of wide application areas, dictionaries and *semantic gazetteers* are used to enhance the semantic rules [PKO+03]. The *semantic gazetteer* keeps the entities with their aliases and descriptions, as well as the lexical resources such as possible first names of male and female persons, etc..

The semiautomatic or automatic annotation projects that employed NLP techniques are MnM[3], ArtEquAKT[4] and Knowledge and Information Management (KIM)[5]. All these projects adopted GATE [CMBT02] for extracting name entities of different domains. The information extraction component Amilcare is adopted in MnM that include GATE as name entity recognizer [VVMD+01, VVMD+02a, VVMD+02b]. ArtEquAKT adopts Apple Pie Parser (APP) [SG95] for splitting sentences. APP outputs are sent to GATE for name entity recognition. WordNet is deployed as additional information source for recognizing the entities that are not recognized by default GATE [KAH+02, AKM+03]. The name entities recognized by KIM are not only the traditional flat types (e.g organization, person, location, money) but also the hierarchical types (e.g., sub-organization of organization, firstname of the person name, etc..) [PKO+03, KPT+04]. In KIM, the grammar rules in GATE are modified to recognize the hierarchical natures of the name entities.

## 2.3 Summary

We have surveyed the state-of-the-art SWA languages and related techniques. Our survey mainly consists of two parts: SWA languages and SWA techniques. Three types of SWA languages are investigated and compared based on different SWA features. We also surveyed Web classification, information extraction and natural language processing techniques that have been adopted in the current SWA solutions.

From our survey, we have learnt that very few research have addressed the semiautomatic annotation of Web objects to relationships in the ontology. In particular, the techniques for annotating and extracting relationship instances are still in nascent stage. Classification techniques are widely used for concept assignment task. Only one project (WebKB) employed classification technique for relationship assignment task. IE and NLP techniques are widely adopted for attribute assignment task. When IE and NLP techniques are used for concept assignment, the concept instances

[3]http://kmi.open.ac.uk/projects/akt/MnM/
[4]http://www.artequakt.ecs.soton.ac.uk/
[5]http://www.ontotext.com/kim

are text fragments in Web pages, when they are used for relationship assignment, the relationship instances are pairs of the text fragments in the Web pages. None of them treated the pairs of Web pages as relationship instances except the classification technique proposed by Creven and Slattery. In order for SWA to be widely adopted, more advanced languages and techniques are required to for relationship assignment task. Henceforth, our research will focus on *relationship instance* annotation and extraction.

# Chapter 3

# Modelling and Representing Link Chains for Relationship Instance Annotation

Usually within a well maintained Web site, link structures among the Web pages are important knowledge about the semantic relationships between the pages. A Web site containing domain specific information consists of Web pages and links between them. In this thesis, we view some Web pages as instances of ontological concepts and pairs of related Web pages as relationship instances. As part of annotating relationship instances, we propose to annotate the link structures of related Web page pairs that are relationship instances. More specifically, we model these link structures as link chains and assign them as attributes of relationship instances.

In this chapter, we first give a formal definition of link chain. We then present a relational approach to represent link chain information and propose a normalization approach to identify link chains sharing some common sub-paths to derive *shared-paths* for more compact annotation representation. We also present our proposed annotation scheme and annotation schema.

## 3.1 Overview of Link Chain

Before we give the formal definition of link chain, consider some Web pages from an example movie Web site as shown in Figure 3.1. The figure shows $w_1$, the home-page of the movie with title "How to Deal". It contains a link $l_1$ to another Web

page $w_2$ containing a list of individuals participated in this movie together with the links to their Web pages. Among them is $w_3$, the homepage of an actor "Mandy Moore". Consider the Movie ontology given in Figure 1.1. One can create an annotation instance for $w_1$ by assigning to it the *Movie* concept label. Similarly, another annotation instance is created for $w_3$ by assigning it the *Actor* concept label. In the above example, the link structure connecting $w_1$ to $w_3$ represents an essential piece of information about the *Acted-By* relationship. It allows us to identify relationship instances given their source Web pages. In our research, we model this link chain structure as link chains. Link chains provides the sources where attributes of the associated relationship instance can be extracted.

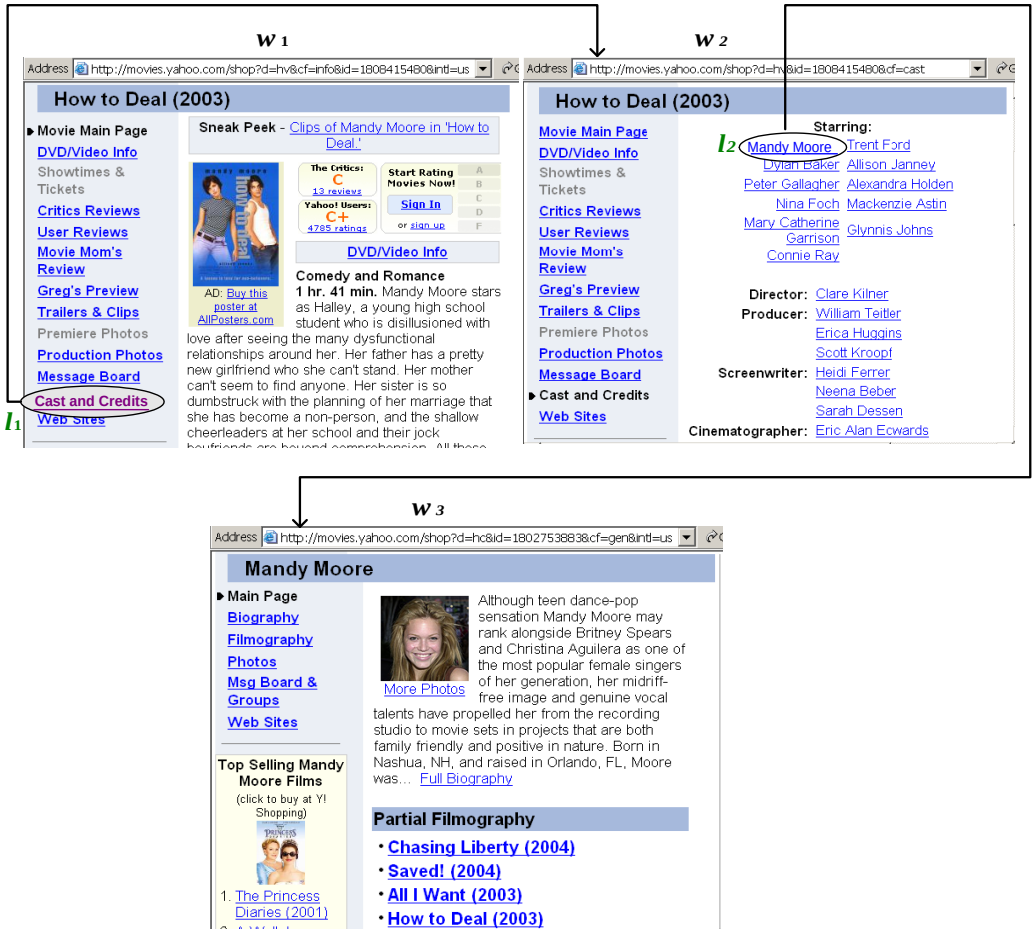



Figure 3.1: An *Acted-By(Movie,Actor)* relationship instance, $(w_1, w_3)$, and its link chain

Table 3.1: Notations

| Symbol | Meaning |
|---|---|
| $w_i$ | Web page |
| $w_i.url$ | URL of $w_i$ |
| $l_j$ | anchor element |
| $l_j.target$ | target URL of $l_j$ |
| $l_j.atext$ | anchor text of $l_j$ |

In the following, we give a formal definition of link chain. Before that, some important symbols to be used mainly in this chapter and parts of others in the thesis are shown in Table 3.1.

Recall that concept instances are homepages of semantic entities, e.g., movies, actors, etc.. Let $C_s$ and $C_t$ be two concepts and $R(C_s, C_t)$ be a relationship. The instances of $R(C_s, C_t)$ are some pairs of Web pages $(w_i, w_j)$'s such that $w_i$ is an instance of $C_s$ and $w_j$ is an instance of $C_t$. We call $w_i$ and $w_j$ the **source concept instance** and **target concept instance** respectively.

**Definition 3.1 (Link Chain)**
*Suppose* $(w_s, w_t)$ *is an instance of a relationship* $R(C_s, C_t)$. *The* **link chain** *lc of* $(w_s, w_t)$ *with respect to* $R(C_s, C_t)$ *is a list of* **chain elements** *denoted by* $\langle (w_1, l_1), (w_2, l_2), \ldots, (w_n, l_n) \rangle$ *such that (i)* $w_1 = w_s$*; (ii)* $l_i$ *is an anchor element in* $w_i$*; (iii)* $l_n.target = w_t.url$*; and (iv)* $\forall 1 \leq i < n, l_i.target = w_{i+1}.url$.

In a relationship instance, the source concept instance may not be directly linked to the target concept instance. When they are indirectly linked, one or more intermediate pages will be included in the link chain to provide the list of anchor elements. In Figure 3.1, the relationship instance $(w_1, w_3)$ of *Acted-By(Movie, Actor)* involves an intermediate page, $w_2$. The link chain of $(w_1, w_3)$, with respect to *Acted-By(Movie, Actor)* is $\langle (w_1, l_1), (w_2, l_2) \rangle$.

### 3.1.1 Applications of Link Chain Information

In our literature survey, we noted that although most existing SWA approaches attempts to annotate relationship between Web pages or elements of the Web pages,

none of them provides the ability to annotate the link structure as useful information between related Web pages. This restricts the association of useful semantics to the links between related Web pages. For example, the *Acted-By* relationship between a movie and an actor homepage given in Figure 3.1 will be annotated by having the URI of $w_3$ inserted as an attribute value in the annotation instance of $w_1$, not the actual link structure connecting $w_1$ and $w_3$ corresponding to the *Acted-By* relationship. We therefore propose to annotate link structure among the related Web pages as attributes of relationship instances.

By annotating link chains, query and navigation among the Web pages can be more intelligently performed as described below:

- Browsing:

  When a user browses a Web page annotated as an concept instance, he or she can examine the relationship semantics of the links in the Web page before deciding which link to be traversed next. In other words, the user can be presented with the relationship(s) associated with each annotated link in the Web page. With the additional relationship information, the user can make more informed and better decision for Web site navigation.

- Query :

  With link chain information annotated, users can now query information associated with links and their assigned semantics. Examples of such queries include:

  - Find relationship instances that are connected by some link chains. The answer to this query will include all the source and target concept instances that are connected via link chains with respect to the given relationship.
  - Find the relevant anchor texts between a particular source concept instance (e.g., movie homepage) and a target concept instance (e.g., actor homepage). The answer to this query will include the anchor texts of chain elements contained in the relevant link chain(s).

– Find Web pages in the link chains of some specified relationship. The answer to this query will include not only the source and target pages but also all the intermediate pages of the link chains. For some queries, the intermediate pages are important information especially when they are the instances of other concepts. For example, consider a link chains associated with a *Supervise(Professor, Student)* relationship instance in University Web site. A professor homepage may have an indirect link to his student homepage via project homepage in which both of them collaborated. In this case, by retrieving the intermediate Web pages, the project they work together can be found.

### 3.1.2 Association of Link Chains with Relationship Instances

We now review the association of link chains with relationship instances. There are two basic associations between link chains and relationship instances: one-to-one or many-to-one.

- *One-to-One Association*: In *One-to-One Association*, there is only one link chain for any relationship instance. The one-to-one association between the given link chains and corresponding relationship instances is shown in Figure 3.2.

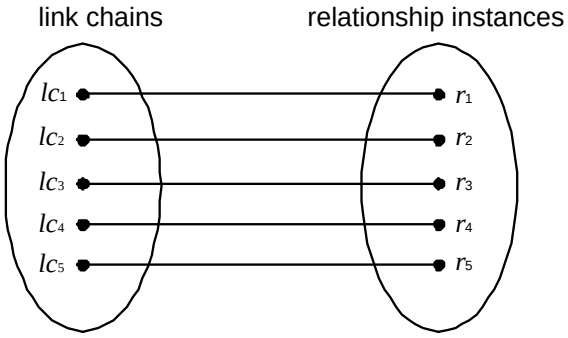



Figure 3.2: One-to-One Association between link chains and relationship instances

- *Many-to-One Association*: In *Many-to-One Association*, there can be multiple link chains for some relationship instance. *Many-to-one* association between link chains and relationship instances is shown in Figure 3.3.

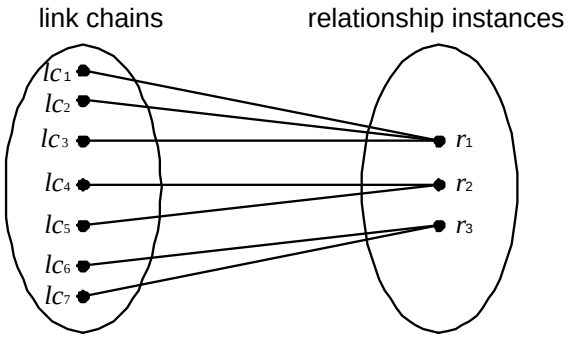



Figure 3.3: Many-to-One Association between link chains and relationship instances

## 3.2 Redundancy and Updating Issues in Link Chain Annotation

Very often, link chains for the same relationship may share some common chain elements. If these chain elements are separately annotated in every link chain containing them, the amount of redundant annotation will be very large. To reduce this redundancy, we introduce the notion of *shared-path*. To systematically derive shared-paths from a given set of link chains, a normalization approach to a table representation of link chains is proposed.

### 3.2.1 Shared-Paths Among Link Chains

A *shared-path* is a sub-path with one or more consecutive chain elements that can be shared by two or more link chains. For example, among the following link chains, $lc_1, lc_2$ and $lc_3$, $\langle(w_{source}, l_n)\rangle$ is a shared-path found in all the 3 link chains.

$lc_1 = \langle(w_{source}, l_n),\ (w_{ip}, l_{n1})\rangle$,

$lc_2 = \langle(w_{source}, l_n),\ (w_{ip}, l_{n2})\rangle$, and

$lc_3 = \langle(w_{source}, l_n),\ (w_{ip}, l_{n3})\rangle$

In another example below, the shared-path $\langle(w_{source}, l_s),\ (w_{ip1}, l_{12}),(w_{ip2}, l_{24})\rangle$ is common between $lc_1$ and $lc_2$. Another shared-path $\langle(w_{source}, l_s),\ (w_{ip1}, l_{13}),\ (w_{ip3}, l_{34})\rangle$ is common between $lc_3$ and $lc_4$. Finally, $\langle(w_{source}, l_s)\rangle$ is also shared-path common to all the 4 link chains. Note that in a set of link chains, shared-paths of different lengths may be found. Moreover, these shared-paths can further share common chain elements.

$lc_1 = \langle(w_{source}, l_s),\ (w_{ip1}, l_{12}),\ (w_{ip2}, l_{24}),\ (w_{ip4}, l_{41})\rangle,$

$lc_2 = \langle(w_{source}, l_s),\ (w_{ip1}, l_{12}),\ (w_{ip2}, l_{24}),\ (w_{ip1}, l_{42})\rangle,$

$lc_3 = \langle(w_{source}, l_s),\ (w_{ip1}, l_{13}),\ (w_{ip3}, l_{34}),\ (w_{ip4}, l_{41})\rangle,$ and

$lc_4 = \langle(w_{source}, l_s),\ (w_{ip1}, l_{13}),\ (w_{ip3}, l_{34}),\ (w_{ip4}, l_{42})\rangle.$

### 3.2.2 Link Chain Table and Functional Dependencies

Given a relationship $R$, the set of link chains associated with the instances of $R$ can be represented in a *link chain table* $L$ that consists of attributes $CE_1, CE_2, \ldots, CE_n$ where $n$ refers to the length of the longest possible link chain in $R$. Each link chain $\langle(w_1, l_1),(w_2, l_2),\ldots,(w_k, l_k)\rangle$ is stored as a tuple in $L_R$, $t_R$, such that $t_R.CE_i = (w_i, l_i)$ $\forall 1 \leq i \leq k$.

For example, suppose we are given the four link chains of some relationship as follows:

$lc_1 = \langle(w_{source}, a),\ (w_{ip1}, b),\ (w_{ip2}, m),\ (w_{ip4}, d),\ (w_{ip5}, h),\ (w_{ip6}, k)\rangle,$

$lc_2 = \langle(w_{source}, a),\ (w_{ip1}, c),\ (w_{ip3}, n),\ (w_{ip4}, e),\ (w_{ip5}, i),\ (w_{ip6}, l)\rangle,$

$lc_3 = \langle(w_{source}, a),\ (w_{ip1}, b),\ (w_{ip2}, m),\ (w_{ip4}, f),\ (w_{ip5}, h),\ (w_{ip6}, k)\rangle,$ and

$lc_4 = \langle(w_{source}, a),\ (w_{ip1}, c),\ (w_{ip3}, n),\ (w_{ip4}, g),\ (w_{ip5}, i),\ (w_{ip6}, l)\rangle.$

The link chain table $L$ can be constructed as shown in Table 3.2.

Since the length of link chains of a relationship may not be the same, NULL values will have to be added to the missing chain elements of link chains with length smaller than the table cardinality. Such NULL values carry the "NOT APPLICABLE" semantics and any comparison between a NULL value and other values (NULL or non-NULL) will return a FALSE value.

Table 3.2: Link Chain Table $L$

| $CE_1$ | $CE_2$ | $CE_3$ | $CE_4$ | $CE_5$ | $CE_6$ |
|---|---|---|---|---|---|
| $(w_{source}, a)$ | $(w_{ip1}, b)$ | $(w_{ip2}, m)$ | $(w_{ip4}, d)$ | $(w_{ip5}, h)$ | $(w_{ip6}, k)$ |
| $(w_{source}, a)$ | $(w_{ip1}, c)$ | $(w_{ip3}, n)$ | $(w_{ip4}, e)$ | $(w_{ip5}, i)$ | $(w_{ip6}, l)$ |
| $(w_{source}, a)$ | $(w_{ip1}, b)$ | $(w_{ip2}, m)$ | $(w_{ip4}, f)$ | $(w_{ip5}, h)$ | $(w_{ip6}, k)$ |
| $(w_{source}, a)$ | $(w_{ip1}, c)$ | $(w_{ip3}, n)$ | $(w_{ip4}, g)$ | $(w_{ip5}, i)$ | $(w_{ip6}, l)$ |

A *functional dependency* ($FD$) is a constraint between two sets of attributes from a link chain table. $FDs$ for a link chain table $L$ are assumed to be specified by a human expert who has designed or learnt the structure of link chains belonging to $L$. Formally, a $FD$, $X \rightarrow Y$, between two set of attributes $X$ and $Y$ of $L$ says that, for any two link chains $lc_1$ and $lc_2$ in $L$ that have $lc_1[X] = lc_2[X]$, $lc_1$ and $lc_2$ must also agree on Y values, i.e. $lc_1[Y] = lc_2[Y]$. We also say that $Y$ is functionally dependent on $X$.

Consider the link chain table $L$ and suppose that the following functional dependencies should hold:

1. $CE_3 \rightarrow CE_2\ CE_5$,
2. $CE_5 \rightarrow CE_2\ CE_3$,
3. $CE_4 \rightarrow CE_1$.

### 3.2.3 Finding Shared-Paths by Normalization

The normalization approach to find shared-paths begins with a link chain table of a relationship and a set of $FDs$. If the link chain table does not satisfy the third normal form ($3NF$), it will be decomposed further into small tables and the latter are further normalized.

The normalization algorithm for link chain table is shown in Algorithm 3.1 **Find-Shared-Paths**. Input to the algorithm are link chain table $L$, a set of candidate keys $K$ and a set of functional dependencies $FDs$. The algorithm mainly consists of two parts: *Normalization* and *Selection*. Lines 3 to 6 perform normalization. The results of normalization are the decomposed tables that satisfy $3NF$. In the relational data model, a table is said to be in $3NF$ if it satisfies the following conditions [EN02]:

**Algorithm 3.1 Find-Shared-Paths($L$, $K$, $FDs$)**

1: Set $A \leftarrow \{L\}$
2: *shared-path-set* $\leftarrow \varnothing$
3: **while** there is a table $S$ in $A$ that is not in $3NF$ **do**
4: Find a functional dependency $X \rightarrow Y$ in $S$ that violates $3NF$
5: $A \leftarrow A - \{S\} \cup \{S - (Y - X), X \cup Y\}$
6: **end while**
7: **for** each decomposed table $S \in A$ **do**
8: Find a candidate key $K'$ and all maximal subsets of attributes in $S$ such that every subset $A'$ consists of consecutive attributes only and $A' \cap K' = \varnothing$
9: *shared-path-set* $\leftarrow$ *shared-path-set* $\cup \{\pi_{A'}\ S\}$
10: **end for**
11: Return *shared-path-set*

- No non-atomic attributes or nested relations exist
- No non-key attribute is functionally dependent on a part of the primary key and
- No non-prime attribute of $L$ is transitively dependent on the primary key.

Nevertheless, in the case of link chain table, the first condition is not applicable since each attribute value is a chain element. Lines 7 to 10 perform the selection task where subsets of consecutive attributes are selected. By projecting the attribute values of each subset, we obtain the set of shared-paths.

We illustrate the above algorithm with the following example. Suppose we are given the link chain table $L(CE_1, CE_2, CE_3, CE_4, CE_5, CE_6)$ as shown in Table 3.2, and a set of functional dependencies $FDs = \{CE_3 \rightarrow CE_2\ CE_5,\ CE_5 \rightarrow CE_2\ CE_3,\ CE_4 \rightarrow CE_1\}$. From $FDs$, we derive that $\{CE_4, CE_5, CE_6\}$ forms a candidate key.

$L$ is not in $3NF$. The decomposed tables using the given functional dependencies are shown in Figure 3.4. In table $L$', only $CE_2$ and $CE_3$ are consecutive attributes. In table $L$", there are none. In table $L$"', only $CE_3$ and $CE_4$ are consecutive attributes. Since $CE_3$ is a key in $L$', we obtain the shared-paths from $L$' by extracting the projected tuples $\langle(w_{ip1}, b),(w_{ip2}, m)\rangle$ and $\langle(w_{ip1}, c),(w_{ip3}, n)\rangle$ as shared-paths.

L'

| $CE_2$ | $CE_3$ | $CE_5$ |
|---|---|---|
| $(wip_1, b)$ | $(wip_2, m)$ | $(wip_5, h)$ |
| $(wip_1, c)$ | $(wip_2, n)$ | $(wip_5, i)$ |

L"

| $CE_4$ | $CE_1$ |
|---|---|
| $(wip_4, d)$ | $(wsource, a)$ |
| $(wip_4, e)$ | $(wsource, a)$ |
| $(wip_4, f)$ | $(wsource, a)$ |
| $(wip_4, g)$ | $(wsource, a)$ |

L"'

| $CE_3$ | $CE_4$ | $CE_6$ |
|---|---|---|
| $(wip_2, m)$ | $(wip_4, d)$ | $(wip_6, k)$ |
| $(wip_2, m)$ | $(wip_4, e)$ | $(wip_6, l)$ |
| $(wip_2, n)$ | $(wip_4, f)$ | $(wip_6, k)$ |
| $(wip_2, n)$ | $(wip_4, g)$ | $(wip_6, l)$ |

Figure 3.4: Decomposed Tables

## 3.3 Proposed Annotation Scheme

In this section, we propose an annotation scheme to incoporate link chain annotation. We mentioned in Chapter 2 that there are existing SWA schemes based on a variety of annotation languages. Most of them adopt some implicit annotation schema commonly agreed by a community of users. They however have not consider link chains in their annotations.

Our annotation scheme is built based on that of Onto Web project[BG01]. The latter defines annotation instances external to the annotated items so that original documents do not need to be modified, and the annotation instances can be stored separately. Our proposed scheme also supports independent or external annotation instances. The annotation instances created are represented in XML very similar to Onto Web. Figure 3.5 depicts the BNF of the proposed annotation scheme. In the figure, each optional element is enclosed by [ and ], and each repeating element by { and }. The * symbol following the repeating element indicates zero or more repeating occurrences, and the $^+$ symbol for one or more repeating occurrences. To

⟨annotation-inst⟩ ::= a-id onto-server inst-url ⟨concept⟩ {⟨relationship⟩}$^+$

⟨concept⟩ ::= c-id c-name {⟨c-attrib⟩}*

⟨c-attrib⟩ ::= name value

⟨relationship⟩ ::= r-id r-name {⟨r-inst⟩}$^+$

⟨r-inst⟩ ::= rinst-id target-inst {⟨r-attrib⟩}* {⟨linkchain⟩}$^+$

⟨r-attrib⟩ ::= name value

⟨linkchain⟩ ::= lc-id {⟨chain-element⟩ | ⟨s-path-ref⟩}$^+$

⟨chain-element⟩ ::= target-url anchor-text

⟨s-path-ref⟩ ::= table-ref s-id

⟨shared-path-table⟩::= name {⟨shared-path⟩}$^+$

⟨shared-path⟩ ::= s-id {⟨chain-element⟩ | ⟨s-path-ref⟩}$^+$

Figure 3.5: BNF of Proposed Annotation Scheme

distinguish non-terminal elements from the terminal ones, the former are enclosed in angle brackets, i.e., ⟨'s and ⟩'s.

We assume each concept instance to be a Web page and each relationship instance to be a Web page pair. Our annotation scheme assigns each annotated Web page a unique annotation id a-id, and stores this a-id as an attribute in an ⟨annotation-inst⟩ element. The ⟨annotation-inst⟩ element also contains the onto-server attribute identifying the server where the required ontology is located. We assume that there is a separate language for representing an ontology and each concept and relationship is given a unique c-id and r-id respectively. The annotation instance consists of the URL of annotated Web page inst-url, a ⟨concept⟩ element containing the concept label assigned to the annotated Web page, and one or more ⟨relationship⟩ elements each annotating the instances of a relationship originating from the Web page. Each relationship element consists of the id of the corresponding relationship entity in the

ontology as r-id, relationship name r-name and one or more relationship instances ⟨r-inst⟩. For each relationship instance, its instance id rinst-id, target concept instance id target-inst, attribute values ⟨r-attrib⟩, and link chains ⟨linkchain⟩ are included.

We annotate the link chain by treating them as a sequence of ⟨chain-element⟩ elements. Each link chain is identified by a unique link chain id lc-id and includes one or more ⟨chain-element⟩ element. We identify each chain element by its target URL, and the anchor text in its anchor element.

In addition, ⟨s-path-ref⟩ element can also be identified within a link chain if shared-path(s) exists in it. Each ⟨s-path-ref⟩ element includes table-ref identifying the URI of shared-path table, and shared-path id s-id to uniquely identify the shared-path in the table.

The ⟨shared-path-table⟩ element contains annotated shared-paths. Each shared-path consists of a unique shared-path id s-id, one or more ⟨chain-element⟩ elements and ⟨shared-path-ref⟩ if its sub-path is further shared.

The core of SWA framework is the annotation schema which defines the template for creating annotation instances. To facilitate sharing of annotation schema among different applications, it is represented using XML Schema [TBMM01]. The XML Schema of our annotation scheme is given in the following.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XMLSchema">
<xsd:annotation>
   <xsd:documentation xml:lang="en">
      Annotation Schema, Copyright 2001 CAIS, NTU. All rights reserved.
   <xsd:documentation>
</xsd:annotation>
<xsd:element name="annotation-inst" type="annotationType"/>
<xsd:complexType name="annotationType">
   <xsd:sequence>
      <xsd:element name="inst-url" type="xsd:anyURI"/>
      <xsd:element name="concept" type="conceptType" maxOccurs="1"/>
      <xsd:element name="relationship" type="relationshipType" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="a-id" type="xsd:string" />
   <xsd:attribute name="onto-server" type="xsd:anyURI"/>
</xsd:complexType>
<xsd:complexType name="conceptType">
   <xsd:sequence>
   <xsd:element name="c-name" type="xsd:string"/>
   <xsd:element name="c-attrib" type="c-attribType" minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
      <xsd:attribute name="c-id" type="xsd:anyURI"/>
</xsd:complexType>
<xsd:complexType name="c-attribType">
```

```
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="relationshipType">
    <xsd:sequence>
        <xsd:element name="r-name" type="xsd:string"/>
        <xsd:element name="r-inst" type = "r-instanceType" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="r-id" type="xsd:anyURI"/>
</xsd:complexType>
<xsd:complexType name= "r-instanceType"/>
    <xsd:sequence>
        <xsd:element name="target-inst" type="xsd:anyURI" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="r-attrib" type="r-attribType" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="linkchain" type="chainType" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="rinst-id" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="r-attribType">
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="chainType" >
    <xsd:sequence>
    <xsd:element name="chain-element" type ="chain-eltType" minOccurs="1" maxOccurs="unbounded" nillable="true"/>
    <xsd:element name="s-path-ref" type="s-path-refType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="lc-id" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="chain-eltType">
    <xsd:attribute name="target-url" type="xsd:anyURI"/>
    <xsd:attribute name="anchor-text" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="s-path-refType">
    <xsd:attribute name="table-ref" type="xsd:anyURI"/>
    <xsd:attribute name="s-id" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="shared-path-table" type="tableType"/>
<xsd:complexType name="tableType">
    <xsd:sequence>
        <xsd:element name="shared-path" type="shared-pathType" minOccurs="1" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>
<xsd:complexType name="shared-pathType" >
    <xsd:sequence>
    <xsd:element name="chain-element" type = "chain-eltType" minOccurs="1" maxOccurs="unbounded" nillable="true"/>
    <xsd:element name="s-path-ref" type= "s-path-refType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="s-id" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
```

In the schema, the maximum occurrence of conceptType is set to 1, meaning that a Web page can be assigned only one concept label with an annotation instance. If the same Web page is assigned with another concept label, separate annotation instance will be required. For example, when a homepage of an actor is assigned with both *Actor* and *Director*, two annotation instances will be created for the actor homepage.

Although our proposed annotation scheme extends that of Onto Web, the former is still unique for the following features:

- Our scheme provide annotation schema to describe the template of annotation instances. The annotation schema is independent of the ontology used. The resultant annotations can make reference to ontology represented in any specification languages as long as each concept and relationship is assigned a unique identifier.

- The original Onto Web annotation scheme does not support link chains as part of relationship instances. Link chain annotation is however a key feature in our scheme. The notion of shared-paths in link chain is also provided in our scheme.

### 3.3.1 Annotation Example of Link Chain Information

The annotation instance for a movie homepage and an actor hompage using our proposed annotation schema is shown in Figure 3.9 and Figure 3.6 respectively.

In Figure 3.9, the homepage of "How to Deal" movie is assigned the *Movie* concept label. There are three relationship instances in which the movie home page serves as the source concept instance and three different target concept instances are included (i.e., ri201, ri202 and ri301). For *Acted-By* relationship, two annotation instances are annotated (from movie home page to Mandy Moore's homepage and Dylan Baker's homepage respectively). For relationship instance ri201, three link chains are annotated. Among them, two link chains consist of *shared-paths* s11 and s21. For relationship instance ri202, only one link chain is annotated that contain a *shared-path* s11. Two link chains are annotated for relationship instance ri301 and both of them contains *shared-paths* s11 and s21.

*Shared-paths* s11 and s21 can be found in the shared-path tables ST1 and ST2 respectively. The shared-path tables ST1 and ST2 are shown in Figures 3.7 and 3.8 with XML representation. There is only one <shared-path> in both shared-path tables and each <shared-path> contains only one <chain-element>.

```
<annotation-inst a-id = "a112" ontoserver= "http://localhost:8080/ontology/Movie/">
   <inst-url>http://movies.yahoo.com/shop?d=hc&id=1802753883&cf=gen&intl=us</inst-url>
   <concept c-id ="&c002">
      <c-name>Actor</c-name>
      <c-attrib name = "name" value = "Mandy Moore"/>
      <c-attrib name = "gender" value = "female"/>
      <c-attrib name = "age" value = "32" />
      <c-attrib name = "hometown" value = "Nashua, NH"/>
   </concept>
   <relationship r-id = "&r005">
      <r-name>Acted-In</r-name>
      <r-inst rinst-id = "&ri501">
         <target-inst>&a111</target-inst>
         <r-attrib name = "year" value = "2003"/>
         <linkchain lc-id = "&l07" >
            <chain-element target-url = "http://movies...id=18..." anchor-text = "Filmography"/>
            <chain-element target-url = "http://movies...id=18..." anchor-text = "How to Deal(2003)"/>
         </linkchain>
      </r-inst>
   </relationship>
   <relationship r-id = "&r006">
      <r-name>HasBiography</r-name>
      <r-inst rinst-id = "&ri601">
         <target-inst>"&a912" </target-inst>
         <linkchain lc-id = "&l08">
            <chain-element target-url ="http://movies...id=18..." anchor-text ="Biography"/>
         </linkchain>
         <linkchain lc-id = "&l09">
            <chain-element target-url ="http://movies...id=18..." anchor-text = "Full Biography"/>
         </linkchain>
      </r-inst>
   </relationship>
</annotation-inst>
```

Figure 3.6: Annotation Instance of "Mandy Moore" Actor Homepage

```
<shared-path-table name="ST1">
   <shared-path s-id="s11">
      <chain-element target-url="http://us...&cf=cast" anchor-text="Cast and Credits"/>
   </shared-path>
</shared-path-table>
```

Figure 3.7: Shared-path Table ST1

In Figure 3.9, the first chain elements of link chains with lc-id l02, l04 and l05 are retrieved from the referenced shared-path-table ST1 and the first chain elements of link chains with lc-id l03,l06 are retrieved from the referenced shared-path-table ST2.

For many-to-one association between link chains and relationship instances, our annotation schema allows multiple ⟨linkchain⟩ elements to be defined within a single relationship instance (i.e. ⟨r-int⟩ element).

```
<shared-path-table name="ST2">
    <shared-path s-id="s21">
        <chain-element target-url="http://us...&cf=cast" anchor-text="More Cast and Credits..."/>
    </shared-path>
</shared-path-table>
```

Figure 3.8: Shared-path Table ST2

## 3.4 Summary

In this chapter, we propose to annotate the link chain among the related Web pages. Link chain refers to an ordered list of anchor element paths from a source Web page to a target Web page. Link chains provide important information in finding out the semantically related Web page pairs in a Web site. We introduce a formal definition of link chain and annotation of different types of link chains. The annotation of link chains having shared-paths brings the redundancy problem. We therefore propose a method to find the dependency among shared-paths in order to reduce the amount of redundant shared-path annotations. We then propose an annotation scheme in order for creating the annotation instances of not only the concepts, but also the relationships and their corresponding link information.

Annotating link chain among the related Web pages is a non-trivial task. Once the annotation languages are extended with the capabilities to annotate the link chains, it is then appropriate to develop automatic or semi-automatic techniques to identify the link chain information to be annotated.

```
<annotation-inst a-id = "a111" ontoserver= "http://localhost:8080/ontology/Movie/">
   <inst-url>http://movies.yahoo.com/shop?d=hv&cf=info&id=1808415480&intl=us</inst-url>
   <concept c-id = "&c001">
      <c-name>Movie</c-name>
      <c-attrib name = "title" value = "How to Deal"/>
      <c-attrib name = "category" value = "Comedy and Romance"/>
      <c-attrib name = "releasedate" value = "July 18, 2003"/>
      <c-attrib name = "distributor" value = "New Line Cinema (USA)"/>
   </concept>
   <relationship r-id = "&r002">
      <r-name>Acted-By</r-name>
      <r-inst rinst-id = "ri201">
         <target-inst>&a112</target-inst>
         <r-attrib name = "year" value = "2003"/>
         <linkchain lc-id = "l01">
            <chain-element target-url = "http://movies..." anchor-text = "Mandy Moore"/>
         </linkchain>
         <linkchain lc-id = "l02">
            <s-path-ref table-ref = "&ST1" s-id= "s11">
            <chain-element target-url = "http://movies..." anchor-text = "Mandy Moore"/>
         </linkchain>
         <linkchain lc-id = "l03">
            <s-path-ref table-ref = "&ST2" s-id= "s21">
            <chain-element target-url = "http://movies..." anchor-text = "Mandy Moore"/>
         </linkchain>
      </r-inst>
      <r-inst rinst-id= "ri202">
         <target-inst>&a113</target-inst>
         <r-attrib name = "year value = "2003/>
         <linkchain lc-id = "l04">
            <s-path-ref table-ref= "&ST1" s-id = "s11"/>
            <chain-element target-url = "http://movies..." anchor-text = "Dylan Baker"/>
         </linkchain>
      </r-inst>
   </relationship>
   <relationship r-id = "&r003">
      <r-name>Written-By</r-name>
      <r-inst rinst-id = "ri301">
         <target-inst>&a511</target-inst>
         <linkchain lc-id = "l05">
            <s-path-ref table-ref= "&ST1" s-id = "s11"/>
            <chain-element target-url = "http://movies..." anchor-text = "Heidi Ferrer"/>
         </linkchain>
         <linkchain lc-id = "l06">
            <s-path-ref table-ref = "&ST2" s-id= "s21">
            <chain-element target-url = "http://movies..." anchor-text = "Heidi Ferrer"/>
         </linkchain>
      </r-inst>
   </relationship>
</annotation-inst>
```

Figure 3.9: Annotation Instance of "How to Deal" Movie Homepage

# Chapter 4

# Semi-Automatic Link Chain Extraction for Relationship Instances

In this chapter, we study the problem of extracting link chains of relationship instances from a given Web site. Firstly, we give a formal definition of the link chain extraction problem. We assume that the source Web pages of the instances of some relationship are given and some labelled relationship instances are also available for training purposes. We also assume that the Web site is fairly well structured. By deriving the extraction rules for link chains, we can effectively determine the target Web pages related to the given source pages.

Relationship instances can be obtained by extracting the link chains between the source and target Web pages of the relationship instances. To extract link chains of the related Web page from a Web site, we propose Web information extraction method that semi-automatically extracts link chains using some training examples from a Web site. Web information extraction refers to extracting useful text, numbers or records from Web pages. Several variants of Web information extraction problem can be defined based on the type of information to be extracted and the characteristics of the given Web data collection. Our Web information extraction problem consists of link chain extraction and link chain construction algorithms to extract the semantically related Web information.

To measure the performance of our proposed link chain extraction method, we have conducted a series of experiments on two publicly available Web sites, namely the Yahoo! Movies and DBLP Web sites. Our experiments have shown that the proposed extraction method performed well even with small number of training examples.

## 4.1 Related Work

Our semi-automatic link chain extraction method is related to two areas: information extraction (IE) research and Web site discovery research.

### 4.1.1 Information Extraction

In our literature survey, we have studied that Web extraction problems involve *single-slot* or *multi-slot* depending on the data slots to be extracted. The multi-slot Web extraction problems are further divided depending on the number of pages containing the slots forming a tuple. In normal circumstances, all slots of a tuple are found within a single page. In our literature survey, almost all multi-slot Web extraction methods are designed for single pages. In other words, they assume that all slots of a tuple can be found in a single Web page.

Link chain extraction is considered a multi-page multi-slot Web extraction problem as each link chain is actually a tuple consisting of anchor elements as slots found in different Web pages. The unique characteristics of multi-page multi-slot Web extraction include the following:

- Extraction rules to be learnt must be able to span or navigate multiple pages and extract slots that are to be combined into tuples.
- The accuracy of extraction is likely to deteriorate as the slots to be extracted are located in Web pages further away from the first Web page visited by the extraction rules.
- Each training example involves slots from different Web pages.

### 4.1.2 Web Site Structure Discovery

*Web site structure discovery* refers to the problem of finding the structure and semantics of links among Web pages from a given Web site. This problem is similar to link chain extraction in some ways: (1) they both involve learning the link structures among Web pages; and (2) they both could infer some semantic meaning between Web pages through the links.

Web site structure discovery and link chain extraction however have several key differences:

- Web site structure discovery does not assume any training examples given beforehand. It is therefore an unsupervised learning problem. At the end of Web site discovery, users usually have to analyze the discovered site structure and assign semantics to the structure components. Link chain extraction on the other hand requires some training examples and it involves supervised learning.

- Web site structure discovery covers all Web pages from the site. Link chain extraction is usually confined to some subsets of Web pages involved in some semantic relationship(s). Hence, the latter is expected to return link chains that are interesting.

Sew is a Web site structure discovery method that automatically discovers the skeleton of a Web site [LNL04]. The skeleton refers to the hyperlink structure among the Web pages in a Web site. Two types of Web pages are included in the skeleton: *content pages* (pages providing information content) and *navigation pages* (pages containing links to content pages). Starting from the home page of a Web site, Sew discovers the hierarchical organization of the pages using a top-down approach. The discovery relies on the several domain independent heuristics and features identifying the most important set of links within each page. Sew adopts a data model (Wicaap) to represent the logical views of Web sites[LNLL04].

Creszcenzi *et al* [CMM03] proposed another Web site structure discovery method that automatically discovers the navigational structure of a Web site as a *site model*.

A *site model* is a graph in which nodes are *page classes* and directed edges are defined between *page classes*. A *page class* is a collection of pages having a similar structure. Each link in a Web page is represented by a root-to-link path in the DOM tree representation of the page. A link collection is a set of shared root-to-link paths from a Web page. A directed edge from a *page class* to another indicates that there exists a subset of links from link collections from Web pages of the former page class to the Web pages of the latter page class. The site discovery algorithm accepts the URL of an entry point to the target Web site, traverses a limited portion of the site and produces a Web site structure.

## 4.2 Problem Definition

In this section, we formally define the link chain extraction problem. We have previously given the formal definitions of *concept instance*, *relationship instance* and *link chain* in Chapter 3.

Note that the link chain of a relationship instance may not be unique. These multiple link chains are often designed to facilitate easy navigation. It is also possible to have different link chains (of the same or different relationship instances) sharing common Web pages, and hence the common link elements. For example, in Figure 3.1, the movie "How to Deal" has several other actors (or actresses), and the corresponding relationship instances share the common source and intermediate Web pages. They only differ in the second link elements of their link chains.

When a Web site is well structured or is generated from some backend database, the link chains of instances of the same relationship can share very similar pattern in the way their anchor elements are found in the source and intermediate Web pages. If one knows this pattern well, it can be used to extract *all* instances of the relationship from the Web site which is an important goal to be achieved in Web data extraction and Web wrapper generation research.

We now define the link chain extraction problem as follows:

**Definition 4.1 (Link Chain Extraction)**
*Given a collection of Web pages from a Web site, find the link chains with respect to a given relationship of an ontology.*

The link chain extraction problem as defined above is difficult to solve because there are many possible candidate pairs of Web pages to be examined. We therefore study a more constrained version of the problem defined as follows:

**Definition 4.2 (Link Chain Extraction with Given Source Pages)**
*Given a collection of Web pages from a Web site and a set of Web pages representing the source concept instances of a given relationship, find the link chains with respect to the relationship.*

The additional input in the second problem definition is the set of source concept instances (or Web pages). By adopting this additional input, we break up the original link chain extraction problem into 2 steps: (i) finding the instances of a given concept; and (ii) finding the target concept instances of a given relationship.

The first subproblem will not be an issue if the source concept instances are already known. For example, we may only be interested to know the actors (or actresses) of a specific movie. When the source concept instances are not given, they can be found by conducting Web page classification [BM98, GTL+02] on the Web site. There are several ongoing research to automate the concept assignment process using classification techniques. For example, Craven *et al* proposed a relational learning method to assign Web pages with concept labels [CDF+00]. Sun *et al* further developed a Web unit mining method to assign clusters of Web pages with concept labels [SL03]. The second subproblem, in contrast, has not been studied before. It is closely related to the *Web site structure discovery problem* which aims to derive a logical structure for an entire Web site. The link chain extraction problem however differs in the use of concepts and relationships to model Web pages and their links. We argue that these additional semantic knowledge will help us to focus on only Web pages that are relevant.

Strictly speaking, finding link chains originating from a set of source concept instances is just one of few ways to find target concept instances for a given relationship. We will only confine our research to the Web page content and the links since they represent the primary features.

Compared with other Web data extraction problems that usually refer to extracting data records from a single Web page [CMM01, Fre98, MMK01, Sod99], the link chain extraction problem involves extracting data from multiple Web pages. The conventional Web data extraction focuses on extracting as many data records as possible from a Web page and does not pay attention to the actual data record values. This is however not the case in link chain extraction where the anchor element properties provide important clues to determine if a given anchor element is part of a link chain. Since there are possibly many anchor elements in a Web page and each Web page is a nested document, the link chain extraction problem remains to be a challenging research.

For example, in Figure 3.1, the anchor element with "Cast and Credits" as anchor text is the target object to be extracted for a link chain. Since it appears in a HTML table as a table entry, any Web data extraction method without considering the anchor text pattern will return all entries in the HTML table, instead of just the one with "Cast and Credits". In our proposed link chain extraction method, we therefore adopt a target rule to select the relevant anchor elements based on the anchor text pattern. In other words, we need to apply another level of filtering to determine the relevant anchor elements to be extracted.

In the simple scenario, all link chains of a given relationship have the same length and their $i^{th}$ level anchor elements are found in Web pages sharing the same structure. The $i^{th}$ level anchor element refers to the anchor element of the $i^{th}$ chain element. The link chain extraction method is therefore required to derive only one single extraction pattern for each level. In the more complex scenario, the link chains to be extracted for a given relationship may have different lengths, and their $i^{th}$ level anchor elements are found in Web pages of different structures. To extract such link chains, we must

be able to determine the appropriate number of extraction patterns for each level and learn the patterns accordingly.

In the following sections, we will define the extraction pattern used in our proposed link chain extraction method. This is followed by the learning techniques.

## 4.3 Extraction Rules for Link Chain Extraction

Like other data extraction problems, link chain extraction requires extraction rules to extract link chains for a specific relationship. Since a link chain consists of a list of chain elements each embedded in some Web page, a natural approach is to define extraction rules for each Web page. Once extracted, the chain elements from different Web pages are then combined together to form a link chain.

There are two main issues to be considered in the definition of extraction rules:

- *Each Web page consists of a hierarchical nesting of page elements and the chain element(s) to be extracted is likely to be within some page element nested in the page structure.* We therefore need to define the extraction rules specific to the nested page structure. This also distinguishes our work from those text extraction methods that assume text only pages.

- *There are potentially multiple chain elements embedded in the same Web page each belonging to a different link chain.* Consider $w_2$ in Figure 3.1. It consists of anchor elements linking to the Web pages of different actors (actresses) and these anchor elements are represented as entries in a tabular structure. We call the multiple chain elements appearing as list items in the Web page the *chain element list*. In our proposed method, these chain elements are extracted together using the same extraction rule(s).

To address the above issues, we model the nested structure of a Web page using a *page path* and assign *extraction rules* to each path element. To join chain elements from different Web pages into link chains, *page path pattern* consisting of a list of page paths is required. The detailed definitions of page path and page path pattern are

given in Section 4.3.1. This is followed by the definitions and evaluation of extraction rules in Sections 4.3.2 and 4.5.1.

### 4.3.1 Page Path and Path Patterns

The definition of page path is adapted from that of *embedded catalog* introduced by the STALKER project [MMK01]. The original embedded catalog represents a Web page structure as a tree consisting of basic values (i.e., text, number, strings, etc.) as leaf nodes and composition constructs (i.e., tuple, list, etc.) as internal nodes. By passing extraction rules to the nodes of an embedded catalog, a set of nested tuples can be extracted from the Web page.

Since link chain extraction aims to extract chain elements only, page path involves only a path in the Web page structure and each internal node is a list construct. The formal definition is given below:

**Definition 4.3 (Page Path)**
*The page path of a chain element* $(w_i, l_i)$ *is a sequence of nodes* $(n_0, n_1, \ldots, n_k)$ *where:*
$n_0$ *represents the root element of* $w_i$*;*
$n_k$ *represents the anchor element(s) to be extracted for* $l_i$*.*
$n_i$ *is a list construct for* $0 < i < k$*.*

In the above definition, $n_0$ represents the entire Web page. Each $n_i$ $(0 < i < k)$ represents some grouping (or listing) of Web page elements that contain the anchor elements to be extracted. Page path supports two list constructs, namely the list defined by the $\langle li \rangle$ tag and table entries defined by the $\langle tr \rangle$ tag.

**Definition 4.4 (Page Path Pattern)**
*A* **page path pattern** *of a link chain* $\langle (w_1, l_1), (w_2, l_2), \ldots, (w_n, l_n) \rangle$ *is defined as an ordered list of* **page paths**, $(pp_1, pp_2, \ldots, pp_n)$ *such that each* $pp_i$ *is the page path of* $(w_i, l_i)$*.*

We assume that the Web site from which the link chains are to be extracted is well structured. This assumption is necessary to ensure that the extraction rules derived from a few training examples can be applied to other Web pages at the Web site. A well structured Web site possesses the following two properties:

- The link chains of relationship instances with respect to each relationship are of the same length.
- Web pages from the Web site must have consistent format.

In other words, a well structured Web site has link chains of each relationship sharing the same page path pattern. Examples of well structured Web sites are mainly those managed by organizations. Incidentally, these are also the Web sites that are frequently the targets of Web extraction.

**Example:**

Consider the Yahoo! Movies Web site again. All instances of the *Acted-By(Movie, Actor)* share the page path pattern $(pp_1, pp_2)$ as shown in Figure 4.1. The first page path $pp_1$ indicates that a single anchor element exists within the movie page and it is a part of a chain element to be extracted. The second page path $pp_2$ consists of an internal node that indicates that there is a list of page elements each containing an anchor element to be extracted.

Similar to STALKER, we assume that the page paths are provided before learning the extraction rules. In other words, the page path pattern of link chains of each relationship is also provided. Compared with embedded catalogs, page paths describe much simpler structures in Web pages and are therefore easier to define. The automatic discovery of page paths is beyond the scope of this research and will be mentioned as part of our future work (see Chapter 6).

### 4.3.2 Extraction Rules

A page path pattern models the relevant structure in Web pages containing the chain elements to be extracted. Page path patterns have to be further assigned with extraction rules in order to identify the relevant page elements and chain elements in

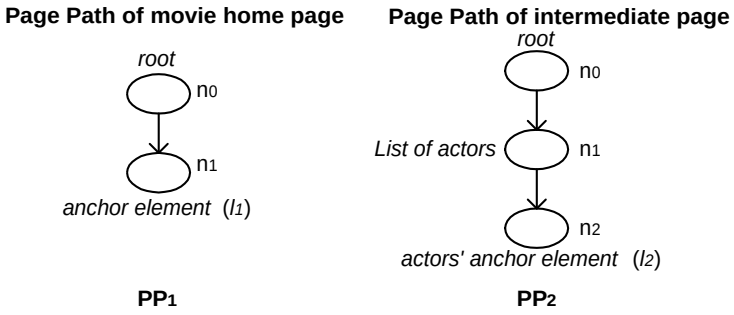



Figure 4.1: Page Path Pattern of *Acted-By(Movie,Actor)* Relationship

Web pages. In this subsection, we give the extraction rule definition and describe how extraction rules are assigned to the node(s) of a page path.

**Definition 4.5 (Extraction Rule)**
*An extraction rule is defined as a triple* $\langle sr, tr, er \rangle$*, where* $sr$*,* $tr$ *and* $er$ *denote the* **start**, **target** *and* **end rules** *respectively.*

Given a page path, $(n_0, n_1, \ldots, n_k)$ , we assign an extraction rule for each non-root node, i.e., $n_i$ where $i > 0$. The start, target and end rules of $n_i$ are denoted by $n_i.sr$, $n_i.tr$ and $n_i.er$ respectively. The definitions of start,end and target rules will be given in Section 4.3.2.

The purpose of a start (or end) rule is to skip the page content that appears before(or after) the page element modelled by a page path node, and returns the extracted page element. For example, consider $w_1$ in Figure 3.1. The "Cast and Credit" anchor element modelled by $n_1$ of $pp_1$ (see Figure 4.1) is to be located by a pair of start rule `Skipto(⟨font face=arial size=-1⟩)` and end rule `Skipto(⟨/font⟩ ⟨/td⟩⟨/tr⟩)`.

Target rules are only applicable to the leaf nodes of page paths. They are designed to further select the anchor elements to be used as chain elements based on their anchor text content. This is necessary because the start and end rules assigned to the leaf node may return some redundant anchor elements. For example, the chain element to be extracted from $w_1$ has anchor text "Cast and Credits" and this is

among the several irrelevant anchor elements (e.g., "DVD/Video Info", "Showtimes and Tickets", etc.) extracted by the start and end rules. Similarly, the chain element to be extracted from $w_2$ has actor (or actress) names as anchor text. A target rule that detects names will be able to extract only those relevant anchor elements.

### Start, End and Target Rules

**Definition 4.6 (Start/End Rule)**
*A* **start/end rule** *is defined as a* $Skipto(\langle landmark \rangle)$ *predicate or a disjunction of conjuncts of* $Skipto(\langle landmark \rangle)$ *predicates.*
*The conjunct of* $Skipto(\langle landmark \rangle)$ *predicates is known as a Skipto clause.*

Each $\langle landmark \rangle$ is a sequence of string tokens or class tokens each representing a set of string tokens sharing some common characteristics in a pre-established order. Our start and end rules currently support seven different class tokens, namely, `HTMLTag`, `Punctuation`, `Alphabetic`, `Alphanumeric`, `Numeric`, `AllCaps`, and `Symbol`.

When applying a start rule on a given page element, the *Skipto*() predicate skips the front portion of the page element till it consumes a sequence of string tokens matching its landmark. The *Skipto*() predicate returns True if a match is found, and False otherwise. The end rule evaluation will however skip the rear portion of the page element when it uses the *Skipto*() predicate.

The learning of start and end rules will be described in Section 4.4.

**Definition 4.7 (Target Rule)**
*A* **target rule** *is defined as a* $HasAnchorTextPattern(\langle regularExpression \rangle)$ *predicate.*

The target rule is applied on a given anchor element and its $HasAnchorTextPattern()$ predicate conducts matching between the anchor text and the regular expression, $\langle regularExpression \rangle$. The predicate returns `True` if the anchor text satisfies the regular expression, and `False` otherwise.

We define three types of regular expressions: *normal regular expressions(NRE)*, *meta-regular expressions(MRE)* and *mixed regular expressions(MixRE)*. The *normal regular expression* is more appropriate for anchor text strings that are largely similar in their string tokens or even identical. The *meta-regular expression*, on the other hand, is used when the anchor text strings demonstrate similarity at the meta-symbol level but not the string level. For example, the anchor text strings may be titles of articles written in uppercase words. A *meta-regular expression* matching uppercase and tokens is therefore appropriate. The *mixed regular expression* is designed to handle other remaining cases.

Each regular expression denotes a language consisting of a set of sequences of symbols. Before we defined the above three types of regular expressions, we first describe the following terms:

- *Symbol*: A symbol($s$) is a letter, character, digit or punctuation. For example: *"a"*, *"9"*, *"$"*, *"#"* are symbols, but "9$" is not.

- *Word Token*: A word token, $t = s_1 s_2 \ldots s_k$ is a sequence of symbols without any white space. For example, *"ab"*.

- *String*: A string, $st = t_1\ t_2\ \ldots\ t_n$ is a sequence of word tokens $t_i$. For example, *"ab cd"* is a string with two word tokens.

**Definition 4.8 Normal Regular Expression (NRE)**

*Let $\Sigma$ be a finite set of* word tokens*, the set of NREs over $\Sigma$ is defined as:*

- $\forall a \in \Sigma$ *: a is a NRE and denotes the set* $\{a\}$

- *If r and s are NREs denoting the languages R and S respectively, the concatenation (rs), union (r|s), Kleene Star Closure (r)*, Kleene Plus Closure $(r)^+$ and option (r)? are also NREs that denote the sets RS, $R \cup S$, $R^*$, $R^+$ and $\{\varnothing\} \cup R$ respectively.*

Table 4.1: Meta Symbols

| Metasymbols | Description | Example Instances | Precedence |
|---|---|---|---|
| $\phi$ | Html tag | ⟨html⟩ ⟨table⟩ | 1 |
| $\rho$ | Punctuation mark | # & ? : ; , | 2 |
| $\beta^F$ | Female first name† | Catherine | 3 |
| $\beta^M$ | Male first name† | Willion | 4 |
| $\beta^L$ | Last name of a person† | Johnson Smith | 5 |
| $\zeta$ | All capital | INFORMATION | 6 |
| $\tau$ | All small | information | 7 |
| $\mu$ | Alphanumeric | E564 | 8 |
| $\eta$ | All numeric | 123 | 9 |
| $\theta$ | Email address | mmnaing@pmail.ntu.edu.sg | 10 |
| $\alpha$ | Alphabet | Information | 11 |
| $\xi$ | All symbol | ∞ ¥ $ | 12 |
| $\lambda$ | Anything | None of the above | 13 |

†We create the name lexicon using name files downloaded from U.S. Census Bureau ($http://www.census.gov/genealogy/names/$) to recognize the first names of males and females and last names.

As the *NRE* involves word tokens only, it is good for those anchor text that share common word patterns but not for those that are different at word level. For the latter cases, we may represent the word tokens at the meta-level.

A ***meta symbol*** is a class of word tokens that share some common syntactic or semantic properties. Table 4.1 depicts the meta-symbols used in our proposed method. The precedence among the 13 meta symbols is also given to ensure that any given word token to be assigned a unique meta symbol. The lower the precedence number, the higher the meta-symbol has precedence over the other meta-symbols. The meta-symbol with the largest precedence number (i.e., $\lambda$) is most general and it can be assigned to any given word token.

**Definition 4.9 Meta-Regular Expression (MRE)**

*Let $\Omega$ be the set of meta symbols given in Table 4.1, the set of MREs over $\Omega$ is defined as:*

- $\forall g \in \Omega$ *: $g$ is a MRE and denotes the set of word tokens that are instances of $g$.*

- *If p and q are MREs denoting the languages P and Q, the concatenation (pq), union (p|q), Kleene Star Closure (p)*, Kleene Plus Closure (p)$^+$ and option (p)? are also MREs that denote the sets PQ, $P \cup Q$, P*, $P^+$ and $\{\varnothing\} \cup P$ respectively.*

**Definition 4.10 Mixed Regular Expression (MixRE)**

*Let $\Delta$ be a finite set of word tokens and meta symbols($\Delta = \Sigma \cup \Omega$), the set of MixREs over $\Delta$ is defined as:*

- *$\forall x \in \Delta$ : x is a MixRE and denotes the set $\{x\}$ if $x \in \Sigma$, or the set of word tokens that are instances of x.*

- *If y and z are MixREs denoting the languages Y and Z, then concatenation (yz), union (y|z), Kleene Star Closure (y)*, Kleene Plus Closure (y)$^+$ and option (y)? are also MixREs that denote the sets YZ, $Y \cup Z$, Y*, $Y^+$ and $\{\varnothing\} \cup Y$ respectively.*

**Example:**

Consider the Web page $w_2$ in Figure 3.1 and the page path for $w_2$, i.e. $pp_2$, in Figure 4.1. The HTML code of $w_2$ is given in Figure 4.2.

The possible start, end and target rules in $pp_2$ are:

$n_1.sr =$ `Skipto("Starring:") Skipto("⟨font⟩")`

$n_1.er =$ `Skipto("Director:") Skipto("⟨/table⟩") Skipto("⟨/font⟩⟨/td⟩")`

$n_2.sr =$ `Skipto("⟨font⟩")`

$n_2.er =$ `Skipto("⟨/font⟩⟨/td⟩")`

$n_2.tr =$ `HasAnchorTextPattern(`$((\beta^F|\beta^M)^*(\beta^L|\alpha))$

Note that the target rule $n_2.tr$ consists of a *meta-regular expression (MRE)* that requires the anchor text to be a first name (male or female) followed by a last name.


```
:
<table> <tr> <tr> <td><font>
<b>Starring:</b></font></td> </tr>
<tr> <td> <font>
<A HRef="/shop?d=hc&id=1802753883&cf=gen&intl=us">
Mandy Moore</a> </font></td> <td> </td>
 :
<td> <font>
<A HRef="/shop?d=hc&id=1800289052&cf=gen&intl=us">
 Connie Ray</a> </font></td> </tr>
</table>
:
```

Figure 4.2: Example intermediate Web page and its partial source

A given set of anchor text strings may satisfy different alternative regular expressions. This is illustrated by the following example.

**Example:**

Suppose that the set of strings {"Academic Staff No1", "Research Staff No2", "Visiting Staff No3"} is given. We can derive the following *normal*, *meta-* and *mixed regular expressions* satisfied by all the strings.

- *Normal regular expression* examples:

  (i) ( *"Academic Staff No1"*| *"Research Staff No2"*| *"Visiting Staff No3"*)

  (ii) ( *"Academic"*| *"Research"*| *"Visiting"*) ( *"Staff No1"*| *"Staff No2"*| *"Staff No3"*)

  (iii) ( *"Academic"*| *"Research"*| *"Visiting"*) *"Staff"* ( *"No1"*| *"No2"*| *"No3"*)

- *Meta-regular expression* examples:

  (i) $\alpha\alpha\mu$

(ii) $(\alpha)^*\mu$

- *Mixed regular expression* examples:

  (i) $\alpha$ *"Staff"* $\mu$

  (ii) $(\alpha)^*$ ( *"No1"*| *"No2"*| *"No3"*)

  (iii) ( *"Academic"*| *"Research"*| *"Visiting")* *"Staff"* $\mu$

  (iv) $\alpha$ *"Staff"* ( *"No1"*| *"No2"*| *"No3"*)

Since one can derive regular expressions of different types for the same set of strings, deciding the most appropriate regular expressions is therefore critical in the learning of target rules. We will describe our proposed target rule learning method in Section 4.4.2. Before that, we briefly describe the evaluation of extraction rules on a given set of Web pages representing source concept instances of a relationship.

## 4.4 Extraction Rule Learning

Similar to the evaluation of extraction rules, the extraction rule learning process can be divided into multiple steps, one for each page path in the given page path pattern. Our proposed learning method for extraction rules further consists of two parts, one for start and end rules, and another for the target rules due to the different rule definitions.

The following are information assumed to be given:

- A set of training link chains together with their Web pages; and
- The page path pattern of the link chains to be extracted such that the Web page elements corresponding to the (non-root) nodes of its page paths are annotated in the given Web pages.

Since each page path may have multiple nodes, we focus on the extraction rule learning for a single node in the page path. The part for learning start and end rules is given in Section 4.4.1, and that for target rules is given in Section 4.4.2.

### 4.4.1 Learning the Start and End Rules

The learning algorithm of the start rule (or end rule) for a non-root node of a page path is based on the sequential covering algorithm [Mit97] which is also adopted by STALKER [MMK01]. This algorithm, called **GetSERule()**, is shown as Algorithm 4.1. Since the learning of start rule and end rule are identical except in the training examples and order of rule clauses[1], we will describe only the learning of start rules using the algorithm.

Recall that each start rule is a disjunction of `Skipto` clauses. Suppose the given non-root node is $n_i$. The rule learning requires both positive and negative training examples. Each positive training example is an annotated substring tokens that is to be skipped in the Web page element of the parent of $n_i$, and that appears before the Web page element (or anchor element) to be extracted. Each negative training example is a randomly generated sub-string or super string of some positive examples.

The **GetSERule()** algorithm repeatedly finds the `Skipto` clauses that cover as many positive training examples as possible by calling **LearnRule()** function, and add it to the start rule represented by *SERule*. The covered positive training examples are removed in each iteration. The step is repeated until all positive training examples are covered. Finally, the start rule is returned as disjunction of `Skipto()` clauses.

In **LearnRule()**, the non-covered positive training example with the smallest number of tokens is used as a `seed` to generate candidate `Skipto` clauses. The function first generates the initial set of candidates by calling *GetInitialCandidates()* with the `seed`. The latter returns two `Skipto` predicates: one involving the last token of the `seed` and an other with the class of the last token[2]. For example, if the `seed` is "...⟨font⟩", *GetInitialCandidates()* returns `Skipto(`⟨`font`⟩`)` and `Skipto( HTMLTag)`. Then the returned predicates are used to select or refine the clauses repetitively by *GetBestRefiner()*, *GetBestSolution()* and *Refine()* until either the clause is perfect or

[1]In the case of learning end rules, the string of tokens in training examples and output rule clauses have their orders reversed.

[2]In the case of learning end rules, the first token will be adopted.

**Algorithm 4.1 GetSERule(*examples*)**

1: $SERule \leftarrow$ empty
2: **while** $examples \neq$ empty **do**
3: $rule \leftarrow$ **LearnRule**($examples$)
4: $examples \leftarrow$($examples - examples$ covered by $rule$)
5: $SERule \leftarrow SERule + rule$
6: **end while**
7: Return $SERule$

**LearnRule(*examples*)**

1: $Seed \leftarrow example_i$ with the shortest length in $examples$
2: $Candidates \leftarrow GetInitialCandidates(Seed)$
3: **repeat**
4: BestRefiner $\leftarrow$ *GetBestRefiner*(Candidates, $examples$)
5: BestSolution $\leftarrow$ *GetBestSolution*(Candidates $\cup$ BestSolution, $examples$)
6: Candidates $\leftarrow$ *Refine*(BestRefiner, $Seed$)
7: **until** *IsPerfect*(BestSolution) **or** BestRefiner is *empty*
8: return *PostProcess*(BestSolution)

there is no more clauses to be refined. A perfect clause is the one which accepts at least one positive example and rejects all the others.

*GetBestRefiner()* and *GetBestSolution()* apply different selection criteria to select the current best candidate clauses. Given a candidate clause with one or more landmarks(s) in the form of (`Skipto(`$l_1$`)`... `Skipto(`$l_k$`)`) and a page element of the parent node, the clause matches the page element if all its landmarks matches with some string tokens in the page element according to the sequence of landmarks in the clause. For each positive example, there is a corresponding page element of the parent node where the example appears as a substring. A clause has a *correct match* on a positive example if all its landmarks matches with some string tokens in the positive example and its last landmark matches the last string token of the positive example. A clause has an *early match* on a positive example if all its landmarks matches with some string tokens in the positive example and its last landmark matches any string token before the last string token of the positive example. A clause has a *late match* on a positive example if all its landmarks matches with some strings tokens in the positive example and its last landmark matches any string token that appears after positive

example in the page element. A clause has a *failed match* on a positive example if it does not match the page element that embed the current positive example.

*GetBestRefiner()* prefers the candidate clause with the largest potential coverage. A clause is said to have a largest potential coverage if it has at least one correct match, and at least one early match, and the largest number of the sum of *early* and *correct matches* on a set of positive examples in the corresponding page elements of the parent node. If there are multiple clauses with the same largest potential coverage, a candidate clause with more *early matches* is preferred. In case there is still a tie, the clause(s) with more *failed matches* is preferred. If there are still multiple clauses left, the clause with fewer number of class tokens in its landmark is selected. If there is still a tie in the number of class tokens, the clause(s) with fewer unconsumed tokens[3] in the uncovered positive example(s) is preferred. Finally if all the above are still the same, the candidate clause(s) that has longer landmarks that match the last string token of positive examples will be selected.

*GetBestSolution()* selects the candidate clauses as follows. It starts by selecting the candidate clause with the largest number of *correct matches* in the set of positive examples. If there are multiple candidates with largest number of correct matches, the clause with largest number of *failed matches* on the remaining positive examples will be selected. In case if there is a tie, it selects the clause with fewer number of class tokens in its landmarks. If there are multiples clauses meeting the same criteria, the clause with the longest landmarks that match the end of the positive examples will be selected. Finally it selects the clause with fewer unconsumed tokens in the uncovered positive examples in the corresponding page elements of the parent node.

*Refine()* refines the candidate clauses in two ways, namely *landmark refinement* and *topology refinement*. Both use a refinement terminal `t` and its class token. The refinement terminal `t` is generated by using every token in the `seed` example. In *landmark refinement*, the candidate clause is refined by concatenating `t` and its class token

[3]The unconsumed tokens are the string tokens that comes after the match of last landmark in the clause.

either at the beginning or at the end of landmark to get more specific rule. For example, with the refinement terminal `t` =⟨td⟩, the candidate rule `Skipto(⟨font⟩)` is refined as `Skipto(⟨td⟩ ⟨font⟩)`, `Skipto(⟨font⟩ ⟨td⟩)`, and `Skipto(HTMLTag ⟨font⟩)`, `Skipto(⟨font⟩ HTMLTag)`. In *topology refinement*, the candidate clause is refined by adding a new disjunct with either `t` or it's corresponding class. For example, the candidate clause `Skipto(⟨font⟩)` is refined as `Skipto(⟨td⟩) Skipto(⟨font⟩)` or `Skipto(HTMLTag) Skipto(⟨font⟩)`.

The selected and refined clauses are post-processed to improve their qualities within the *PostProcess()* function. *PostProcess()* performs three post-processing operations: (i)replacing the classes with their corresponding string tokens, (ii)merging landmarks that always appear next to each other in the examples and (iii)adding more tokens to short landmarks to improve the accuracy of the clauses.

Finally, **GetSERule()** returns the disjunction of all candidate clauses as the start rule that accepts all the positive examples and rejects all the negative ones.

### 4.4.2 Learning of Target Rules

The learning of target rules is relatively more complex due to the regular expression for matching anchor text. Our proposed learning algorithm consists of the following steps:

(i) *Determining the type of regular expression*:

Since there are three possible types of regular expressions to be learnt for the target rules, we first determine the type of regular expression to be learnt as described in Section 4.4.2. The step requires a set of anchor text training examples to be given.

(ii) *Preprocessing of training anchor text examples*:

When the type of regular expression is determined to be either *MRE* or *MixRE*, the training anchor text examples are preprocessed as described in Section 4.4.2.

(iii) *Derivation of candidate regular expressions*:

The generation of candidate regular expressions of the designated type using preprocessed training examples is elaborated in Section 4.4.2.

(iv) *Selection of regular expression based on the minimum description length (MDL) principle*:

We employ the MDL principle to measure the goodness of candidate regular expressions with respect to the set of training anchor text examples. The best regular expression is selected for the target rule. This step is presented in Section 4.4.2.

### Determining the Regular Expression Type

The choice of regular expression type is very much dependent on the similarity among the given anchor text examples. Intuitively, the more similar the anchor text examples, the more likely word tokens are used in the regular expression to be learnt as opposed to meta symbols. Hence, we propose a similarity measure for a given set of anchor text examples by treating them as strings.

Firstly, *Levenshtein's edit distance* [Lev96] is adopted to compare two strings. Suppose $st_i$ and $st_j$ are two given strings. The edit distance denoted by $ldist(st_i,st_j)$ is defined by the number of character insertions, deletions, and substitutions required to transform $st_i$ into $st_j$. For example, $ldist$("Credit", "Credited") = 2, since two insertions are required to transform the former into latter.

To obtain a normalized similarity between two strings, we define the following **string similarity measure**.

$$sim(st_1, st_2) := max\left(0, \frac{min(|st_1|, |st_2|) - ldist(st_1, st_2)}{min(|st_1|, |st_2|)}\right) \qquad \text{(Eq. 4.1)}$$

The $sim(st_1, st_2)$ measure returns a value between 0 and 1. The higher the value, the more similar are $st_1$ and $st_2$. For example, $sim(\text{"}Credit\text{"}, \text{"}Credited\text{"}) = \frac{4}{6} = 0.67$.

Table 4.2: 3 Sets of Anchor Text Examples

| Set | $st_1$ | $st_2$ | $st_3$ |
|---|---|---|---|
| $ST_1$ | "Cast and Credit" | "Cast and Credit" | "Cast and Credit" |
| $ST_2$ | "Research Staff" | "Academic Staff" | "Visiting Staff" |
| $ST_3$ | "Rick Fox" | "Marisa Tomei" | "Joan Plowright" |

Table 4.3: Similarity Measures

| Set | $sim(st_1, st_2)$ | $sim(st_1, st_3)$ | $sim(st_2, st_3)$ | $\overline{sim}(ST_i)$ |
|---|---|---|---|---|
| $ST_1$ | 1.000 | 1.000 | 1.000 | 1.000 |
| $ST_2$ | 0.500 | 0.500 | 0.428 | 0.476 |
| $ST_3$ | 0.000 | 0.000 | 0.000 | 0.000 |

To compute the similarity among a given set of strings, $ST = \{st_1, st_2 \ldots st_k\}$, the following **average similarity measure** is defined.

$$\overline{sim}(ST) = \frac{2 \sum_{i=1}^{k} \sum_{j=i+1}^{k} sim(st_i, st_j)}{k(k-1)} \qquad \text{(Eq. 4.2)}$$

Again, the $\overline{sim}(ST)$ value is between 0 and 1. It is 1 if all the strings in $ST$ are identical.

To determine the type of regular expression to be derived for $ST$, we adopt two thresholds $r_u$ and $r_l$. If $\overline{sim}(ST) \geqslant r_u$, normal regular expression will be generated. If $r_l \leqslant \overline{sim}(ST) < r_u$, mixed regular expression will be generated instead. Finally, *meta-regular expression* will be generated for $\overline{sim}(ST) < r_l$.

In our experiment, $r_u$ and $r_l$ were set to 0.85 and 0.35 respectively. The intuition was that *normal regular expression* was used if there was at least 85% similarity among the strings. *Meta-regular expression* was used when the similarity was less than 35%. In other cases, the *mixed regular expression* was used.

**Example:**

Let $ST_1$, $ST_2$ and $ST_3$ be the given sets of anchor text examples shown in Table 4.2. Each set consists of three anchor text strings $st_1$, $st_2$ and $st_3$. The average similarities of $ST_1$, $ST_2$ and $ST_3$ are determined in Table 4.3.

### Preprocessing the Training Anchor Texts

Preprocessing of training anchor text examples is only applicable when *meta-* or *mixed regular expression* is to be learnt. In the preprocessing step, some or all word tokens in the training examples are replaced by the corresponding meta symbols.

To learn a *MRE*, every training example is converted into a meta string by replacing all word tokens with the corresponding meta symbols. For example, the training examples in the set $ST_3$ ("Rick Fox", "Marisa Tomei", and "Joan Plowright") are converted to "$\beta^M\beta^L$", "$\beta^F\beta^L$", and "$\beta^M\beta^L$" respectively. Recall that based on the precedence of meta symbols, exactly one unique meta symbol can be used to replace any given word token.

To learn a *MixRE*, each training example is converted into a mixed string as follows:

- Find the common word tokens that exist in every training example.
- Every word token in each training example except the common word token is replaced with the corresponding meta symbol.

For example, the training examples in the set $ST_2$ ("Research Staff", "Academic Staff", "Visiting Staff") are converted to "$\alpha Staff$", "$\alpha Staff$", and "$\alpha Staff$" respectively as "Staff" is a common word token.

### Derivation of Candidate Regular Expressions

The preprocessed training examples are used as candidate sequences to derive candidate regular expressions. The derivation method is based on the XTRACT approach [GGR+03] and it consists of two main steps:

- **Finding the Star and Or patterns within each candidate sequence:** Star and Or patterns are of the forms $(t_i)$* and $(t_1|t_2|\ldots|t_m)$* respectively where $t_1, \ldots, t_m$ are word tokens or meta symbols. In this step, we find subsequence(s) in a candidate sequence that can be replaced by such patterns using the following heuristics.

Star patterns are obtained by replacing all the subsequences of the form "$tt\ldots t$" with $(t)$* if $t$ repeats more than one time in the candidate sequence.

The Or patterns are obtained by performing two steps. Firstly, a candidate sequence $st$ is partitioned into smallest possible subsequences $st_1, st_2, \ldots, st_n$ such that for any word token or meta symbol $t$ in $st_i$, $t$ does not exist in some other $st_j$ within a predefined distance $d$. Then each subsequence $st_i$ containing some repeating symbol or token is replaced by the pattern $(t_1|t_2|\ldots|t_m)$* where $t_1, t_2, \ldots, t_m$ are the distinct meta symbols or word tokens in $st_i$.

For example, given a candidate sequence $st$="abcbac" and $d$=2, $st$ is partitioned into "a", "bcb" and "ac". Since only "bcb" contains a repeating token, it is replaced by $(b|c)$*. We therefore obtain the revised candidate sequence "a(b|c)*ac".

If $d$=3, $st$ is partitioned into "a" and "bcbac", and the latter is replaced by the pattern $(a|b|c)$*. We therefore obtain the revised candidate sequence "a(a|b|c)*".

In our implementation, we find the or patterns for three $d$ values. Each $d$ is determined based on the character length of the candidate sequence as follows: $\lceil |s| * 0.1 \rceil$, $\lceil |st| * 0.5 \rceil$ and $|st|$.

Given a set of example anchor text sequences, $ST$, we find the Star and Or patterns for some subset $G$ of $ST$. The resultant set of candidate sequences finally derived is $ST_G = ST \cup G$.

- **Generating candidate regular expressions with factor form within a set of candidate sequences:** The objective of finding regular expressions with factor form is to reduce the regular expression length. A candidate regular expression with factor form is generated from a subset $F$ of $ST_G$ in which all the elements of $F$ have the common prefixes or suffixes. For example, the 3 candidate sequences $t_1t_2$, $t_1t_3$, $t_1t_4$ have the common prefix $t_1$ and the regular expression with factor form $t_1(t_2|t_3|t_4)$ is generated.

There are two sub-steps included in this step. The first sub-step chooses the subset of candidate sequences having the common prefixes/suffixes and next sub-step generates factor form of the chosen candidate sequences. The sub-steps are described in Appendix B.

Assume that the set *F'* is generated from *F*. Then, the set of resultant candidate regular expression is $ST_F = ST_G \cup F'$. Note that, each candidate regular expression in $ST_F$ may cover only a subset of the anchor text examples in the given set *ST*.

**Selecting the Regular Expression with the Minimum Description Length Cost**

In this step, we choose a subset of candidate regular expressions *RE* from $ST_F$ and return the union of these regular expressions as the final regular expression that covers all anchor text examples in *ST* and has the minimum MDL cost.

The MDL cost of a regular expression $re_j$ with respect to a given set of anchor text examples *ST* includes two parts:

(a) $LB(re_j)$ : The number of bits required to represent $re_j$; and

(b) $EB(re_j, ST)$ : The number of bits required to encode *ST* using $re_j$.

*LB($re_j$)* is defined as:

$$LB(re_j) = n\lceil log(|\Omega| + |\Theta| + |\Lambda|)\rceil \quad \text{(Eq. 4.3)}$$

where

$\Omega$ is the set of distinct word tokens from $\Sigma$ that appear in *ST*,

$\Theta$ is the set of distinct meta symbols,

$\Lambda$ is the set of operators {|,*,+,?,(,)} and

$n$ is the character length of $re_j$.

and *EB($re_j$,ST)* is defined as:

$$EB(re_j, ST) = \sum_{st_i \in ST} EB(re_j, st_i) \quad \text{(Eq. 4.4)}$$

where

$$EB(re_j, st_i) = \begin{cases} f(Encode(re_j, st_i) & if\ st_i\ is\ covered\ by\ re_j \\ \infty & otherwise \end{cases} \quad \text{(Eq. 4.5)}$$

$Encode(re_j, st_i)$ returns a sequence of integral indices for an anchor text example $st_i$ when encoded with $re_j$. The function $f$ is applied on the sequences of indices to compute the total number of bits. Let $Encode(re_j, st_i)$= $a[0], \ldots, a[k]$, $f$ is defined as follows:

$$f(a[0], \ldots, a[k]) = \sum_{i=0}^{k} Numberofbits(binary(a[i])) \quad \text{(Eq. 4.6)}$$

In other words, $f$ is obtained by summing up the number of bits required to represent each index in binary representation i.e. $binary(a[i])$ .

A subset $RE$ is chosen from $ST_F$ based on *Facility Location Problem (FLP)* [JV99] such that the sum of the number of bits required to represent the regular expressions in $RE$ and the cost of encoding $ST$ using $RE$ is minimized. That is,

$$RE = arg \min_{RE \subset ST_F} \{ \sum_{re_j \in RE} LB(re_j) + \sum_{st_i \in ST} \min_{re_j \in RE} EB(re_j, st_i) \} \quad \text{(Eq. 4.7)}$$

The time complexity for choosing $RE$ is $O(NlogN)$ where $N = |ST| * |ST_F|$ [JV99].

**Encoding Scheme**

$Encode(re, st)$ constructs a sequence of integral indices recursively based on the following recursive encoding formulas:

E1. $Encode(re,st) = \begin{cases} \epsilon\ (empty\ index) & if\ st = re \\ |st| & otherwise \end{cases}$
*where re does not contain any operator and st contains only one word token.*

E2. $Encode((re_1 \ldots re_k), st_1 st_2 \ldots st_k) = Encode(re_1, st_1) \ldots Encode(re_k, st_k)$
*where each $st_i$ matches the corresponding regular expression $re_i$.*

E3. $Encode((re_1| \ldots |re_m), st) = i\ Encode(re_i, st)$ *where st matches* $re_i$, $1 \leqslant i \leqslant m$

E4. $Encode((re)^*, st_1 \ldots st_k) = \begin{cases} k\ Encode(re, st_1) \ldots Encode(re, st_k) & if\ k > 0 \\ 0 & otherwise \end{cases}$

Table 4.4: Regular Expressions

| | **Regular Expressions** | **Type** |
|---|---|---|
| $re_1$ | $(Academic\ Staff\ No1 \mid Research\ Staff\ No2 \mid Visiting\ Staff\ No3)$ | *NRE* |
| $re_2$ | $(Academic \mid Research \mid Visiting)\ (Staff\ No1 \mid Staff\ No2 \mid Staff\ No3)$ | *NRE* |
| $re_3$ | $(Academic \mid Research \mid Visiting)\ Staff\ (No1 \mid No2 \mid No3)$ | *NRE* |
| $re_4$ | $\alpha\alpha\mu$ | *MRE* |
| $re_5$ | $(\alpha)^*\mu$ | *MRE* |
| $re_6$ | $\alpha\ Staff\ \mu$ | *MixRE* |
| $re_7$ | $(\alpha)^*\ (No1 \mid No2 \mid No3)$ | *MixRE* |
| $re_8$ | $(Academic \mid Research \mid Visiting\ Staff\ )\ \mu$ | *MixRE* |
| $re_9$ | $\alpha\ Staff\ (No1 \mid No2 \mid No3)$ | *MixRE* |

E5. $Encode((re)?, st) = 1\ Encode(re, st)$

**Example:**

Consider $ST$, a set of three anchor text examples, "Academic Staff No1", "Research Staff No2", and "Visiting Staff No3". We derive the MDL costs of the regular expressions in Table 4.4 with respect to the anchor text examples in $ST$ as shown in Table 4.5. In the table, $\lceil log(|\Omega| + |\Theta| + |\Lambda|) \rceil = \lceil log(7 + 13 + 6) \rceil = 5$ for each regular expression.

The encoding of "Academic Staff No1" using $re_1$, $re_5$ and $re_7$ is shown below.

Encode(*Academic Staff No1*|*Research Staff No2*|*Visiting Staff No3*, "Academic Staff No1")

= 1 Encode(*Academic Staff No1*,"Academic Staff No1") // Apply (E3)

= 1 Encode(*Academic*,"Academic")Encode(*Staff*,"Staff")Encode(*No1*,"No1") //Apply (E2)

= 1 $\epsilon$ $\epsilon$ $\epsilon$ //Apply (E1)

Therefore, EB($re_1$, "Academic Staff No1") = $f(1)$ = 1 bit

Encode($(\alpha) * \mu$, "Academic Staff No1")

= Encode($(\alpha)*$,"Academic Staff") Encode($\mu$,"No1") // Apply (E2)

= 2 Encode($\alpha$, "Academic") Encode($\alpha$, "Staff") Encode($\mu$,"No1") //Apply (E4)

= 2 8 5 3 //Apply (E1)

Table 4.5: MDL Costs

| Regular Expression | $n$ | LB($re_i$) | EB($re_i$,$ST$) | MDL Cost |
|---|---|---|---|---|
| $re_1$ | 52 | 260 | (1+2+2) | 265 |
| $re_2$ | 56 | 280 | (2+4+4) | 290 |
| $re_3$ | 46 | 230 | (2+4+4) | 240 |
| $re_4$ | 3 | 15 | (9+9+9) | 42 |
| $re_5$ | 5 | 25 | (11+11+11) | 58 |
| $re_6$ | 7 | 35 | (6+6+6) | 53 |
| $re_7$ | 17 | 85 | (10+11+11) | 117 |
| $re_8$ | 34 | 170 | (3+4+4) | 181 |
| $re_9$ | 19 | 95 | (5+6+6) | 112 |

EB($re_5$, "Academic Staff No1") = $f$(2 8 5 3) = 2+4+3+2 = 11 bits

Encode(($\alpha$)$*$(*No1*|*No2*|*No3*), "Academic Staff No1")
= Encode(($\alpha$)$*$,"Academic Staff") Encode((*No1*|*No2*|*No3*),"No1") // Apply (E2)
= 2 Encode($\alpha$, "Academic") Encode($\alpha$, "Staff") Encode((*No1*|*No2*|*No3*),"No1") //Apply (E4)
= 2 Encode($\alpha$, "Academic") Encode($\alpha$, "Staff") 1 Encode(*No1*,"No1") //Apply (E3)
= 2 8 5 1 $\epsilon$ //Apply (E1)

EB($re_7$, "Academic Staff No1") = $f$(2 8 5 1) = 2+4+3+1 = 10 bits.

**Discussions**

The MDL cost of a regular expression with respect to a given set of anchor text examples may sometimes be counter intuitive if the number of examples is small. For example, Table 4.5 shows that the MDL cost of $RE_4$ is the smallest. However, the MDL cost of $RE_6$ would be the smallest if there are more similar anchor text examples given. This explains why the type of regular expression is first determined using similarity measure before we use MDL cost to select the best candidate regular expression. In this example, the degree of average similarity measure of $ST$ is 0.537, which is within the medium range. Hence, *mixed regular expression* is preferred over *meta-regular expression*.

## 4.5 Link Chain Construction

In this section, we describe the extraction of link chains of a given relationship after its extraction rules are learnt. Again, we assume that the page path pattern of the relationship is given. Given a source page, we would like to use the page path pattern together with the extraction rules to extract and construct a set of link chain originating from the page. This is done using the **ConstructLinkChain()** algorithm (see Algorithm 4.2). Inputs are the *page path pattern* and a collection of Web pages which is initialized to contain the source page.

The **ConstructLinkChain()** algorithm first extracts the anchor elements from the source page and assigns them to *anchor-element-set* by using *EvaluateRule()* described in Section 4.5.1. To maintain the chain elements of the link chains, we keep a set of $i^{th}$ chain elements in $chain\text{-}elements_i$ using the *CreateChainElement()* function. The *CreateChainElement()* function takes the URL of a Web page and anchor elements extracted from the page as input and returns the set of chain elements. It further obtains the URL from the extracted anchor elements using the *GetURLs()* function.

The URLs are used to find the next set of Web pages to be extracted. This process repeats for all *page paths* in the *page path pattern*. Finally, the algorithm constructs the link chains using different combinations of chain elements where URL of the next page is contained in the anchor element of the previous page.

### 4.5.1 Extraction Rule Evaluation

Algorithm 4.3 depicts **EvaluateRule()** that extracts anchor elements from a given Web page using its corresponding *page path* and learnt extraction rules. We assume that the extraction rules have been learnt for every node in the *page path*. **EvaluateRule()** starts from the root element of a Web page $w$ denoted by *root-element* using *page path pp* containing $(n_0, \ldots, n_k)$ as input. *ExtractbyStartEndRules()* applies both the start and end rules of $n_i$ $(0<i<k)$ on the content of $n_{i-1}$ to extract a set of Web page elements if $n_i$ represents a list construct for page path containing more than

**Algorithm 4.2 ConstructLinkChain($W$, $(pp_1, pp_2, \ldots, pp_m)$)**

```
1: linkchains ← ∅
2: for each w_i in W do
3:    anchor-element-set ← EvaluateRule(w_i, pp_1)
4:    chain-elements_1 ← CreateChainElement(URL_{w_i},anchor-element-set)
5:    for (i=2 to m) do
6:       URL-Set ← GetURLs(anchor-element-set)
7:       anchor-element-set ← ∅
8:       for each URL in URL-Set do
9:          w ← GetPage(URL)
10:         anchor-elements ← EvaluateRule(w, pp_i)
11:         chain-elements_i ← CreateChainElement(URL,anchor-elements)
12:         Add anchor-elements to anchor-element-set
13:      end for
14:   end for
15:   for each combination of (⟨w_1, l_1⟩, ··· , ⟨w_m, l_m⟩) where (⟨w_i, l_i⟩ ∈ chain-
      elements_i & l_i.target = w_{i+1}.url) do
16:      Add (⟨w_1, l_1⟩, ··· , ⟨w_m, l_m⟩) to linkchains
17:   end for
18: end for
19: Return linkchains
```

2 nodes (i.e. $k \geqslant 2$). For each extracted Web page element, further evaluation of extraction rules of the successive node is carried out to get its sub-elements. To extract the anchor elements, the start, end and target rules of $n_k$ are applied on the content of $n_{k-1}$ by calling *ExtractbyStartEndTargetRules()*. The extracted *anchor-elements* are added to the *anchor-element-set* and return to the *ConstructLinkChain()*. *anchor-element-set* will be empty if $w$ does not follow the page path $pp$.

## 4.6 Performance Evaluation

In this section, we describe the experiments conducted to evaluate our proposed link chain extraction method. We measured the accuracies of the learnt extraction rules. We also studied the rule accuracies using a varying number of training examples.

**Algorithm 4.3 EvaluateRule(*root-element*,$(n_0, \ldots, n_k)$)**

```
1: Let anchor-element-set, page-element-set, temp-element-set be ∅
2: Add root-element to page-element-set
3: for i = 1 to (k-1) do
4:    for each element in page-element-set do
5:       element-set ← ExtractbyStartEndRules(element, n_i.sr,n_i.er)
6:       Add element-set to temp-element-set
7:    end for
8:    page-element-set ← temp-element-set
9: end for
10: for each element in page-element-set do
11:    anchor-elements ← ExtractbyStartEndTargetRules(element, n_k.sr,
       n_k.er,n_k.tr)
12:    Add anchor-elements to anchor-element-set
13: end for
14: Return anchor-element-set
```

### 4.6.1 Datasets

We evaluated our proposed link chain extraction method on two real-life well structured Web sites: Yahoo! movie Web site (MOVIE) and DBLP Web site. The statistics of these 2 datasets are given in Table 4.6.

Since the MOVIE Web site is very huge, we randomly collected 1000 movie home pages and other pages reachable by not more than three links as the MOVIE dataset for our experiment. The movie home pages are the source pages for four relationships: *Acted-By(Movie, Actor)*, *Directed-By(Movie, Director)*, *Produced-By(Movie, Producer)* and *Written-By(Movie, ScreenWriter)*. By manual inspection, we determined the page path patterns of these relationships as shown in Figure 4.1.

For the DBLP dataset, we collected all 210 computer science journal pages and the pages reachable by not more than 3 links. The journal pages are the source pages of the *Author-Of(Journal, Author)* relationship. The intermediate pages contain the list of authors and titles of the papers in each volume of the corresponding journal. The target pages contain the publications of different authors. The page path pattern of *Author-Of()* relationship was derived manually and is shown in Figure 4.3.

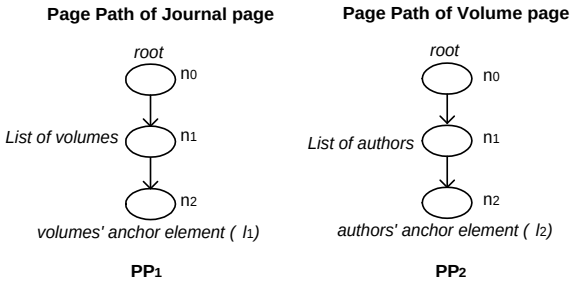



Figure 4.3: Page Path Pattern of *Author-of(Journal, Author)* Relationship

Table 4.6: MOVIE and DBLP Datasets

| Dataset | #source pages | #intermediate pages | Relationship($R$) | #target pages | $N_{lr}(R)$ |
|---|---|---|---|---|---|
| MOVIE | 1000 | 890 | Acted-By | 8061 | 8061 |
| | | | Directed-By | 979 | 979 |
| | | | Produced-By | 1463 | 1463 |
| | | | Written-By | 1310 | 1310 |
| DBLP | 210 | 3995 | Author-Of | 236178 | 236178 |

### 4.6.2 Performance Metrics

To evaluate the accuracy of extraction, we adopt the standard precision, recall and F1 measures. Let $R(C_s,C_t)$ denote a relationship, the recall, precision, and $F_1$ are defined as follows:

$$Recall(R) = \frac{N_{lc}(R)}{N_{lr}(R)} \quad \text{(Eq. 4.8)}$$

$$Precision(R) = \frac{N_{lc}(R)}{N_{le}(R)} \quad \text{(Eq. 4.9)}$$

$$F_1(R) = \frac{2 * Precision(R) * Recall(R)}{Precision(R) + Recall(R)} \quad \text{(Eq. 4.10)}$$

where

$N_{lr}(R)$ : number of link chains of $R$ in the dataset

$N_{le}(R)$ : number of extracted link chains of $R$; and

$N_{lc}(R)$ : number of correctly extracted link chains of $R$

To analyze the performance of learnt extraction rules for each chain element in the link chain, we define recall, precision and $F_1$ of the $i^{th}$ chain elements as follows:

$$Re^i(R) = \frac{N^i_{ac}(R)}{N^i_{ar}(R)} \quad \text{(Eq. 4.11)}$$

$$Pr^i(R) = \frac{N^i_{ac}(R)}{N^i_{ae}(R)} \quad \text{(Eq. 4.12)}$$

$$F^i_1(R) = \frac{2 * Pr^i(R) * Re^i(R)}{Pr^i(R) + Re^i(R)} \quad \text{(Eq. 4.13)}$$

where

$N^i_{ar}(R)$ : number of $i^{th}$ chain elements of the link chains of $R$ in the dataset

$N^i_{ae}(R)$ : number of extracted $i^{th}$ chain elements; and

$N^i_{ac}(R)$ : number of correctly extracted $i^{th}$ chain elements

To obtain the above performance metrics, we divided each dataset into training and test datasets. Each link chain of the training dataset was obtained by randomly choosing a source page and a link chain originating from the page. The pages in the selected link chains are then annotated. The extraction rules were learnt using the training data and tested on the unseen source pages.

We varied the number of training link chain examples from 5 to 15 to examine the effect of training size on the extraction performance[4]. Note that the training set was enlarged by adding randomly selected training examples one at a time to the original training example set. We intentionally keep the training examples small as users are expected to only annotate very few training examples. Our experiments also showed that our extraction method obtained high precision and good recall using less than 15 training examples and did not improve much beyond that.

[4]The reason for learning the rules starting with 5 training examples is to avoid generating over stringent regular expression for the target rules. If the regular expression in the target rule is not general enough, the recall will be extremely low for some Web sites with some variations of anchor text in the link chains.

Table 4.7: Extraction Results for MOVIE dataset

| Relationship | $N_{lr}(R)$ | $N_{le}$(R) | | | $N_{lc}$(R) | | |
|---|---|---|---|---|---|---|---|
| | | #tr(5) | #tr(10) | #tr(15) | #tr(5) | #tr(10) | #tr(15) |
| Acted-By | 8061 | 6262 | 6728 | 7853 | 6262 | 6728 | 7853 |
| Directed-By | 979 | 676 | 743 | 863 | 676 | 743 | 863 |
| Produced-By | 1463 | 1151 | 1228 | 1351 | 1151 | 1228 | 1351 |
| Written-By | 1310 | 1083 | 1152 | 1223 | 797 | 1031 | 1120 |

Table 4.8: Evaluation Results on MOVIE dataset

| Relationship | #tr(5) | | | #tr(10) | | | #tr(15) | | |
|---|---|---|---|---|---|---|---|---|---|
| | *Rec.* | *Prec.* | $F_1$ | *Rec.* | *Prec.* | $F_1$ | *Rec.* | *Prec.* | $F_1$ |
| Acted-By | 0.776 | 1 | 0.873 | 0.834 | 1 | 0.909 | 0.974 | 1 | 0.987 |
| Directed-By | 0.690 | 1 | 0.816 | 0.758 | 1 | 0.862 | 0.881 | 1 | 0.936 |
| Produced-By | 0.786 | 1 | 0.880 | 0.839 | 1 | 0.912 | 0.923 | 1 | 0.959 |
| Written-By | 0.608 | 0.735 | 0.665 | 0.787 | 0.894 | 0.837 | 0.854 | 0.915 | 0.883 |

### 4.6.3 Results and Discussions

#### MOVIE Dataset

Table 4.7 gives the extraction performance for the MOVIE dataset[5]. The corresponding recall, precision and $F_1$ results are shown in Table 4.8 and graphically shown in Figure 4.4.

We observed that the results were very encouraging as we obtained 100% precision for the link chains of *Acted-By*, *Directed-By* and *Produced-By* relationships. This indicates that extraction rules were able to filter away the false positive link chains. The precision of the *Written-By* relationship was lower with a small number of training data as we obtained wrong link chains that did not lead to the screen writer pages for the corresponding movie home page. They led to pages of composer, cameo, art directors, etc. instead. This was due to the fact that the format of the link chains for *Written-By* relationship was not well learnt from the training data set. By increasing the number of training examples, the learnt extraction rules were able to recognize the correct anchor elements to the screen writer pages, so the precision improved. The recall values were relatively low when using 5 training examples for all relationships.

[5]The detailed results for other number of training examples are omitted here to conserve space.

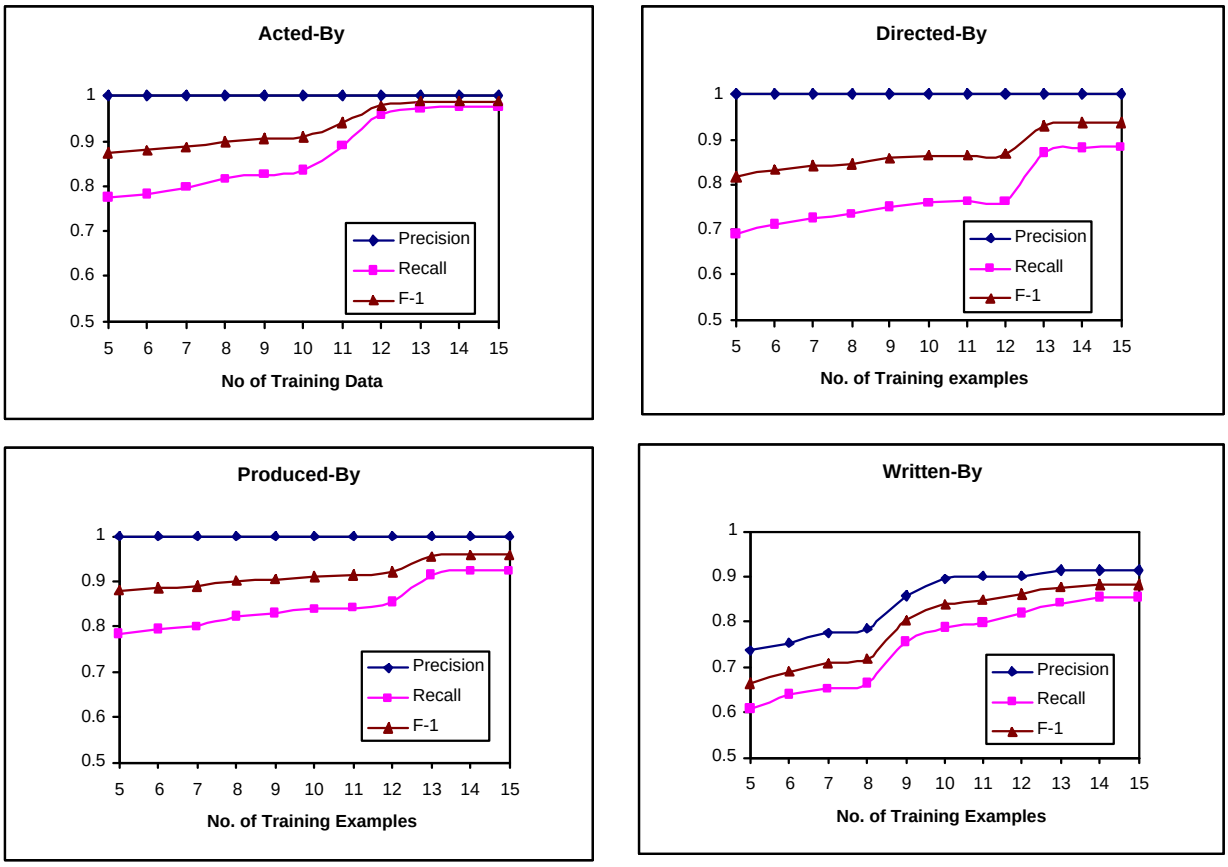



Figure 4.4: Evaluation Results of four Relationships in MOVIE Dataset

As more training examples were added, the recall also improved. We achieved up to 97% recall for *Acted-By*, 92% for *Produced-By*, 88% for *Directed-By* and 86% for *Written-By* relationship. The reason for less than 100% recall was due to the fact that some link chains were wrongly rejected by the learnt extraction rules since the latter do not cover all patterns of anchor texts in the test data set.

We also analyzed the performance of the learnt extraction rules for each chain

Table 4.9: Chain Element Evaluation on *Acted-By* Relationship

| chain-element | #tr(5) | | | #tr(10) | | | #tr(15) | | |
|---|---|---|---|---|---|---|---|---|---|
| | $Re^i$ | $Pr^i$ | $F_1^i$ | $Re^i$ | $Pr^i$ | $F_1^i$ | $Re^i$ | $Pr^i$ | $F_1^i$ |
| $i = 1$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $i = 2$ | 0.776 | 1 | 0.873 | 0.834 | 1 | 0.909 | 0.974 | 1 | 0.987 |

Table 4.10: Extraction Results for DBLP Dataset

| Relationship | $N_{lr}$(R) | $N_{le}$(R) | | | $N_{lc}$(R) | | |
|---|---|---|---|---|---|---|---|
| | | #tr(5) | #tr(10) | #tr(15) | #tr(5) | #tr(10) | #tr(15) |
| Author-Of | 236178 | 120873 | 183874 | 207584 | 120873 | 183874 | 207584 |

element in the link chain of *Acted-By* relationship in Table 4.9. The results in Table 4.9 confirmed that the $1^{st}$ chain elements were accurately extracted. This was due to the similar page layout and identical anchor text adopted by all source pages. In the intermediate pages, our learnt extraction rules were unable to pick up some of the $2^{nd}$ chain elements. Hence, the recall of extracting $l_2$ from $w_2$ did not reach 100% but all the extracted actor names were correct. We do not present the performance of the chain elements in the link chain of *Directed-By*, *Produced-By* and *Written-By* relationship as all of them were also accurately extracted $l_1$ in the source pages $w_1$. Hence, the results of extracting the $2^{nd}$ chain element is the same as that of extracting the whole link chains as described in Table 4.7.

**DBLP Dataset**

We repeated the experiment on the DBLP dataset to extract the link chains of *Author-Of(Journal, Author)* relationship. Table 4.10 gives the extracted link chain results for the DBLP dataset. For these results, we obtained the recall, precision and $F_1$ values as shown in Figure 4.5.

We also achieved 100% precision for DBLP dataset since it is also a well structured Web site with consistent page layout. However, the recall was low when using 5 training examples, because most of the anchor text patterns of the test pages were quite different from the training examples. Therefore the regular expressions in the extraction rules were not good enough discarding some true positive link chains. As more training examples that covered more different anchor text patterns were added, the generated regular expressions in the target rules were more general and more true positive values were obtained. Hence, the recall was improved to 88%.

We also analyzed the performance of the learnt extraction rules for each chain element in the link chain of *Author-Of* relationship in Table 4.11 and Table 4.12.

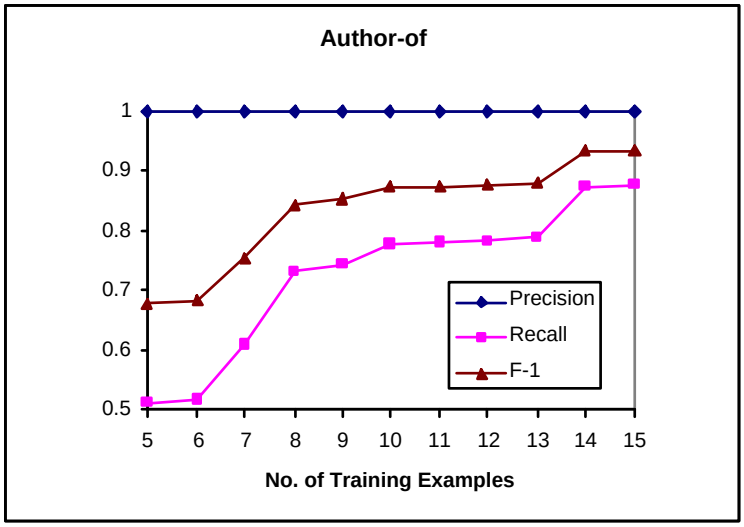



Figure 4.5: Evaluation Result of DBLP Dataset

Table 4.11: Chain Element Extraction Results of *Author-Of* Relationship in DBLP Dataset

| chain element | $N^i_{ar}$(R) | $N^i_{ae}(R)$ | | | $N^i_{ac}(R)$ | | |
|---|---|---|---|---|---|---|---|
| | | #tr(5) | #tr(10) | #tr(15) | #tr(5) | #tr(10) | #tr(15) |
| $i=1$ | 3995 | 2083 | 3777 | 3903 | 2083 | 3777 | 3903 |
| $i=2$ | 236178 | 150425 | 190911 | 214955 | 150425 | 190911 | 214955 |

The results indicated that the $1^{st}$ chain elements were accurately extracted since the source pages have similar layout. However, the recall of $l_1$ was low as the anchor text patterns were not identical in all the source pages. By increasing the number of training examples to 15, we achieved 98% recall for the $1^{st}$ chain element.

To evaluate the performance of extracting $2^{nd}$ chain elements from intermediate pages, we also conducted the experiment by giving all the correct 3995 intermediate pages and obtained the extraction results and performance shown in Tables 4.11

Table 4.12: Chain Element Evaluation of *Author-Of* Relationship in DBLP Dataset

| chain element | #tr(5) | | | #tr (10) | | | #tr(15) | | |
|---|---|---|---|---|---|---|---|---|---|
| | $Re^i$ | $Pr^i$ | $F^i_1$ | $Re^i$ | $Pr^i$ | $F^i_1$ | $Re^i$ | $Pr^i$ | $F^i_1$ |
| $i=1$ | 0.521 | 1 | 0.685 | 0.945 | 1 | 0.972 | 0.977 | 1 | 0.988 |
| $i=2$ | 0.637 | 1 | 0.778 | 0.808 | 1 | 0.894 | 0.910 | 1 | 0.953 |

and 4.12 respectively. The training anchor text examples for the target rules of $l_2$ contained author names. As we achieved low average similarity of the training anchor texts, *meta-regular expressions* were generated. Initially, using 5 training examples, our learnt extraction rules were not able to pick up most of the author names, although those extracted were correct. Hence, the recall of the $2^{nd}$ chain element was quite low with a few number of training examples but the precision was high. As the number of training examples increased, the generated extraction rules became more general and were able to extract the correct $2^{nd}$ chain elements up to 91% recall.

## 4.7 Summary

In this chapter, we propose to extract link chains, an important piece of information linking pairs of web pages with some relationships. The link chain information can be very useful in several web applications including ontology-based web annotation [NLG02]. By including link chain information in the annotation of relationship instance, users can use it to guide the web site browsing.

We describe our proposed link chain extraction method that consists of the learning methods for different types of extraction rules. In particular, we have introduced different types of regular expressions for the target extraction rules to generalize the tokens found in anchor text examples. The experimental results on two live Web sites have shown that our proposed method of extracting link chain information achieved high precision and recall for well structured web sites using not more than 15 training examples.

# Chapter 5

# Searching and Browsing Semantic Instances of Web Sites

The goal of SWA is to enhance the accessibility of the current Web by providing semantics information about Web pages so that applications and users can easily process the Web page content [LHL01]. In particular, we would expect better search engines and browsers to be developed to access Web pages that have been annotated with semantic concepts and relationships. Such tools should support more complex user query and navigation features. For example, instead of plain keyword queries, Web page queries can be augmented with concept labels. Web pages can also be browsed via their relationships instead of physical links.

In this chapter, we present a tool known as CORE to access a Web site that has been annotated. CORE provides an integrated search and browsing interface for COncept and RElationship instances. Its search module supports not only keyword queries on Web pages of user specified concepts, but also complex queries involving Web pages connected by some relationships. CORE's browser module also allows Web pages of user specified concepts to be browsed and generates virtual links for traversing other related Web pages.

## 5.1 Related Work

This section reviews three research projects closely related to CORE.

Magpie [DDM04, DBDEM03] is a semantic Web browsing tool that allows users to view concept and relationship instances embedded in the Web pages of a Web site. Each concept instance is represented by a name and a set of attribute values, and each relationship instance is represented by a pair of concept instances. Magpie assumes that the concept and relationship instances are pre-defined in a database. These concept and relationship instances will be highlighted in those Web pages containing their names. No search capability is provided in Magpie.

MnM project [VVMD+02b], similar to Core, is a tool for browsing and annotating Web pages. It adopts a knowledge base definition that consists of classes (similar to concepts in Core) and each class has one or more attribute slots. Relationships between Web pages however are not captured. MnM can learn extraction patterns from annotated attribute slots from a Web page and apply them to annotate other Web pages. Both the manually and automatically annotated attribute slots can be viewed in the MnM browser. Since MnM deals with mainly text based Web pages, the extraction patterns involves text matching only. There is no search module provided by MnM.

CREAM [Ste01] supports manual annotation of Web pages (e.g., home page of a PhD student) as concept instances, text fragments (e.g., name, postal address, email address) in these pages as concept attributes, and pairs of Web pages as relationship instances, in order to facilitate navigation and queries on the Web pages. When browsing Web pages, the attribute instances and anchor text of links to related concept instances will be highlighted.

Compared to the above three projects, Core provides both browsing and search facilities, and allows users to browse any search results. The concept instances in Core are Web pages, and relationship instances are pairs of Web pages such that the pages representing source concept instances are connected to the pages representing target concept instances. The target concept instances of currently browsed Web pages can be reached by mouse hovering the highlighted anchor texts.

## 5.2 System Architecture

This section presents the basic design principles and architecture of Core.

### 5.2.1 Design Principles

The main design goal of Core is to create an integrated environment for searching and browsing Web pages of a given Web site. Both search and browsing functions should be easy to use and one should be able to switch from one function to another in a flexible manner. In order to fulfill the main design goal, the following design principles have been adopted:

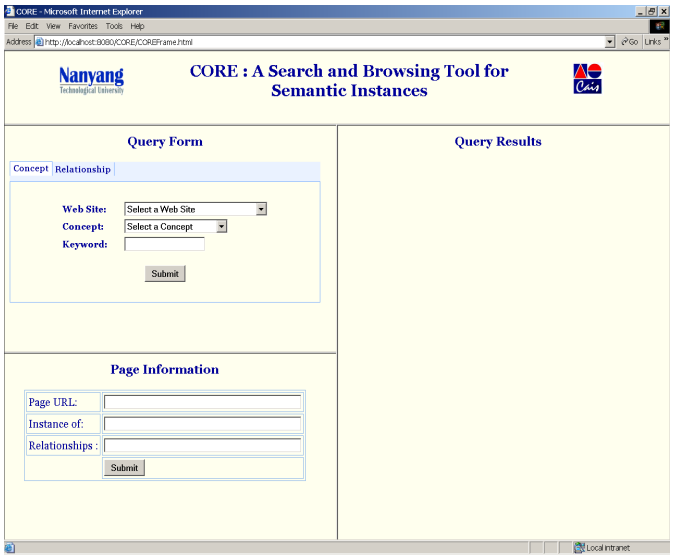



Figure 5.1: Screen design of Core

- Core should allow users to query both concept and relationship instances by keywords. With a search query model extended with concept and relationship semantics, users can express queries on Web pages assigned with some concept labels, or Web pages that are semantically related.

- Users should be able to visualize query results consisting of Web pages that are concept instances or related to one another by some relationships.

- The appearance of a Web page should also be preserved as much as possible when browsed even if new information are embedded. As part of the Core design, Web page browsing should be augmented with knowledge of the Web pages' concept labels and the relationship labels. These features however should appear non-intrusive compared to normal Web browsing.

Figure 5.1 shows the screen design of Core. It consists of 3 frames and an additional window for browsing Web pages. The query form consists of two tab panes for querying concept or relationship instances. The query form in Core allows users to directly choose the desired concepts or relationships to be queried. The query result frame contains a listing of result tuples each consisting of one or more Web pages satisfying the query conditions. Core displays for each result tuple the title and the links to its Web pages. The page information frame displays the information about the Web page currently browsed. These include the URL of the browsed Web page, its concept labels, and the relationships with the browsed Web page as the source. The browser window will appear when users directly enter the URL of the Web page to be browsed or select a link from a query result.

### 5.2.2 Architecture

Figure 5.2 depicts the system architecture of Core. It mainly consists of the search and a browsing server application, Web page classification module, link chain extraction module and knowledge repository.

#### Server Application

There are two main application components, namely:

- **Search Component** The search component accepts user queries and communicates with the knowledge repository in order to derive the query results. Both *concept queries* and *relationship queries* are supported as defined below:

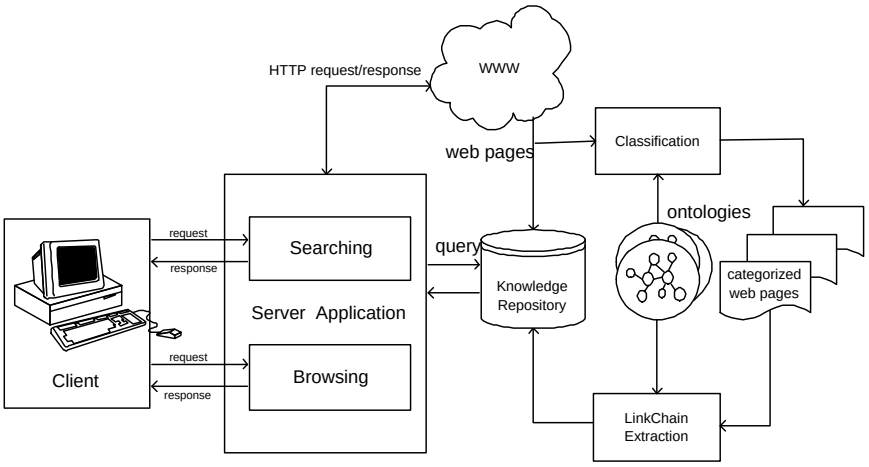



Figure 5.2: Overall architecture of Core

**Definition 5.1** ***Concept Query***

*Let site, $C$, and $K$ be a Website, a concept label and a set of keywords respectively.*

*CQ(site, $C$, $K$) = $\{w \mid w \in site$, $w$ is an instance of $C$ and $w$ contains $k$ $\forall$ $k \in K$ $\}$*

For example:

*CQ*("http://movies.yahoo.com", "*Actor*", {"*Harry*"}) returns Web pages that are *Actor* instances and that contain "Harry".

**Definition 5.2** ***Relationship Query***

*Let site, $R(C_1, C_2)$, $K_1$ and $K_2$ be a Website, a relationship from $C_1$ to $C_2$, two sets of keywords respectively.*

*RQ(site, $R(C_1, C_2)$,$K_1$,$K_2$) = $\{$ $(w_1, w_2)$ $\mid$ $w_1, w_2 \in$ site and $(w_1, w_2)$ is an instance of $R(C_1, C_2)$ where $w_1$ contains $k_1$ $\forall k_1 \in K_1$ and $w_2$ contains $k_2$ $\forall k_2 \in K_2$ $\}$*

For example:

*RQ*("http://movies.yahoo.com", *Acted-By(Movie,Actor)*,{"big"},{"Harry"}) returns pairs of Web pages each consists of a *Movie* instance that contains "big", and an *Actor* instance that contains "Harry".

The search component requires Web pages from each Website to be classified and their relationships to be extracted using the Web page classification module and link chain extraction module respectively.

We use Jakarta Lucene[1] text search tool as the core indexing and search engine in our search component. Since Lucene only handles term indexing and simple keyword search, additional query evaluation procedures have been developed to examine the concept and relationship labels of Web pages.

- **Browsing Component**

  The browsing component allows users to browse any instance (Web page) in the search results or by simply entering the URL of a Web page. If the Web page to be browsed has been classified, its concept label(s) and associated relationships will be displayed in the page information frame. Note that more than one concept label may be assigned to a Web page. For example, the director of a movie may also be a producer of the other movie. Similarly, a Web page can be the source concept instance of more than one relationship.

  The browsing component also allows users to view Web pages augmented with highlighted anchor text strings that lead to related target pages in terms of a particular relationship. The highlighted anchor texts are parts of link chains originating from the currently viewed Web pages.

### Web Page Classification Module

The Web page classification module assigns concept labels to Web pages. It assumes that each Web site can be associated with some ontology and ontology concepts will be used to label the Web pages. For example, in the movies domain, the relevant concepts are *Actor*, *Movie*, *Director*, etc. It is not necessary for Core to classify every Web page as some of them could not be assigned any concept labels and others will be labelled during the link chain extraction process as they are identified as target concept instances of some relationship instances. The current version of Core

[1] http://jakarta.apache.org/lucene/docs/index.html

uses URL features to classify Web pages as the two experimented Web sites are well structured. Web pages from the two Web sites, Yahoo! Movies Web site[2] and DBLP Web site[3], were categorized under seven concepts. They are *Movie*, *Actor*, *Director*, *Producer* and *Writer* for the Yahoo! movie Web site and *Journal* and *Author* for the DBLP Web site.

### Link Chain Extraction Module

Link chain extraction module extracts *link chains* from some given Web pages that constitute the source concept instances. A link chain is a series of links from a source Web page to a target Web page such that a relationship exists between the two Web pages. The link chain extraction module is designed to learn the extraction patterns of link chains from some training examples and apply them to other source Web pages. The formal definition of *link chain* and extraction of link chain information have been described in Chapters 3 and 4 respectively. In the current implementation of link chain extraction, we assume that page path patterns are specified by expert users. To learn extraction rules associated with the page segment nodes, a method combining sequential covering [Mit97], generalization and factoring algorithms [Law64] has been developed. Detailed description of these algorithms have been reported in Chapter 4. Based on our experiments that involved over 1000 movie source pages from Yahoo! movie Web site and 210 journal source pages from DBLP Web site, our link chain extraction method was able to extract link chains of a few pre-defined relationships with high precision and good recall using less than 15 training examples.

### Knowledge Repository

The knowledge repository maintains information about the ontology concepts and relationships, their instances, the link chain information of relationships, and Web pages. The ER diagram describing the content of knowledge repository is shown in Figure 5.3. At the top of Figure 5.3, the ontology concepts and relationships are

[2]http://movies.yahoo.com/
[3]http://www.informatik.uni-trier.de/~ley/db/journals/

modelled. The lower portion of the figure depicts the instance-level knowledge about the Web site. We apply one-to-one association between link chains and relationship instances.

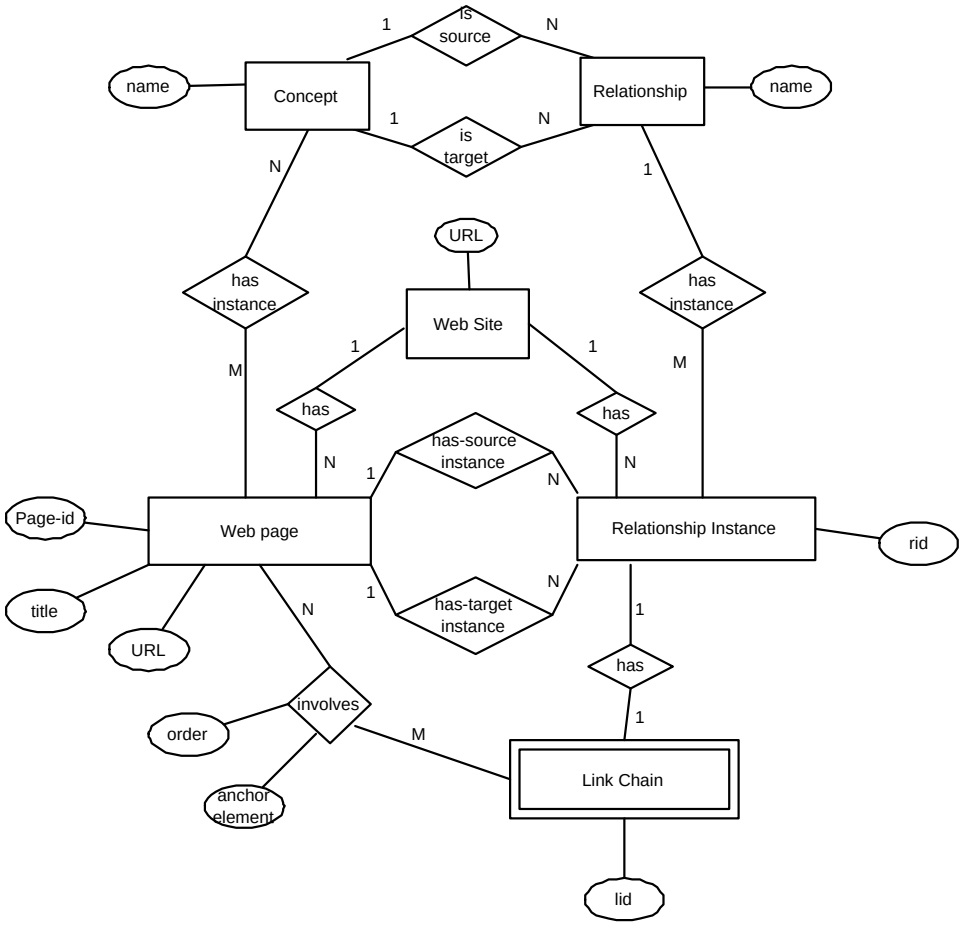



Figure 5.3: ER diagram of Knowledge Repository

Core has been fully implemented in Java running on a Tomcat application server. It uses Microsoft Access as its underlying database system. The graphical user interface is implemented using Java Server Page(JSP) in J2EE platform.

## 5.3 Scenario of Searching and Browsing the Semantic Instances

In this section, we describe some usage scenarios of Core and highlight its unique features.

### 5.3.1 Searching Scenarios

We describe the usage scenarios for querying concept and relationship instances as follows.

**Concept Query**:

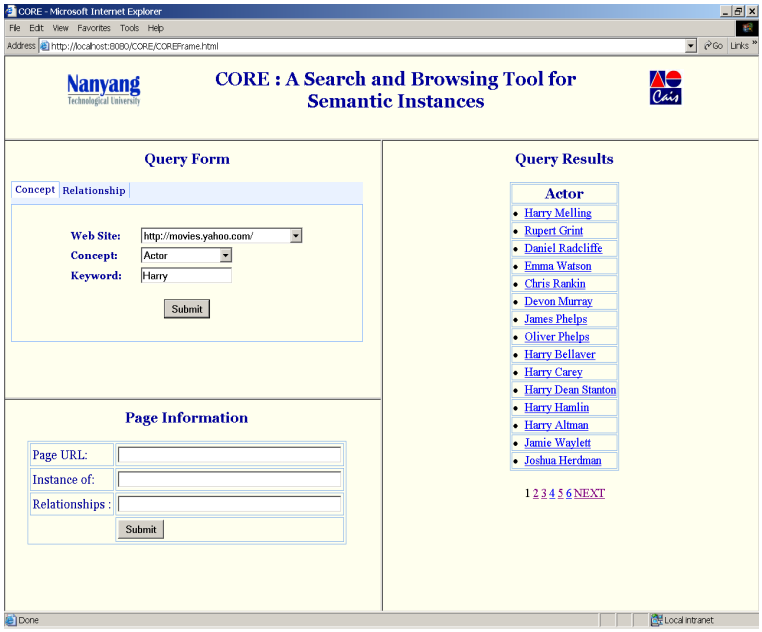
Consider a movie enthusiast gathering information about movies and related information such as actors, directors, etc.. Suppose he/she wants to search his/her favourite actor at Yahoo! Movie Web site. He/she however does not recall the full name of the actor except the first name "Harry". When Google[4] search engine is given the query string "Harry site:.movies.yahoo.com" which specifies "Harry" as the search term and "movies.yahoo.com" as the Web site to be searched, the results returned consist of mainly the home pages of movies containing "Harry" in their titles, e.g., "Harry Potter and the Prisoner of Azkaban", "Harry Potter and the Chamber of Secrets", etc.. In this example, Google actually fails to return any actor page in the top 10.

Unlike Google, Core allows the user to use the concept query facility and select *Actor* as the concept to be queried. The search term "Harry" can now be specified against the Web pages that are instances of the *Actor* concept. As expected, the results returned will consists of only the actor pages that contains "Harry" in their contents as shown in Figure 5.4(a).

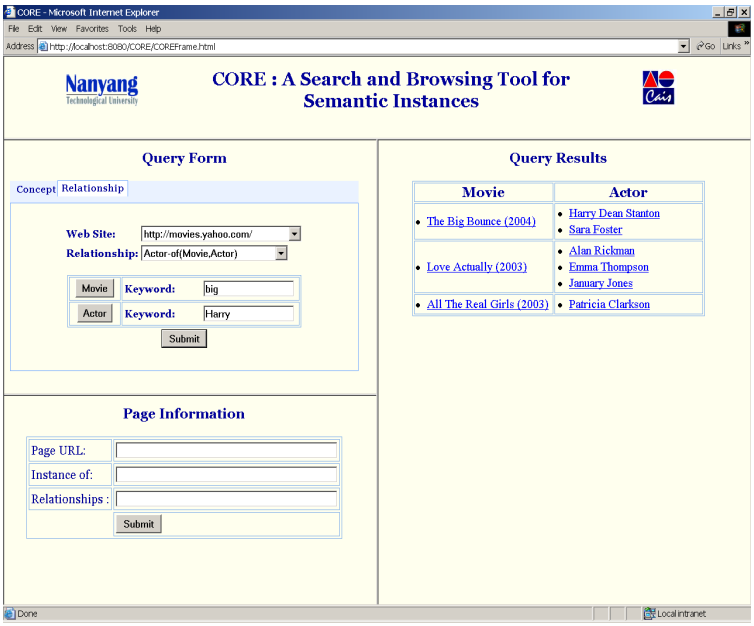
**Relationship Query**:

Suppose now the user wants to find a specific movie which involves his/her favourite actor. Unfortunately, he/she only remembers that the movie title includes the word "big" and the actor's name includes "Harry".

[4]http://www.google.com




(a)




(b)

Figure 5.4: (a) Results of querying *Actor* concept instances using search term "Harry" (b) Results of querying *Acted-By* relationship instances using search term "big" for the *Movie* concept instances and search term "Harry" for the *Actor* concept instances

Again, when such a query is posed to Google in a query string "big Harry site:.movies.yahoo.com" where the search terms "big" and "Harry" are included, the

results fail to match the user requirement. The query results will even be more unexpected when the query string "movie + give + actor + harry site:.movies.yahoo.com" is used.

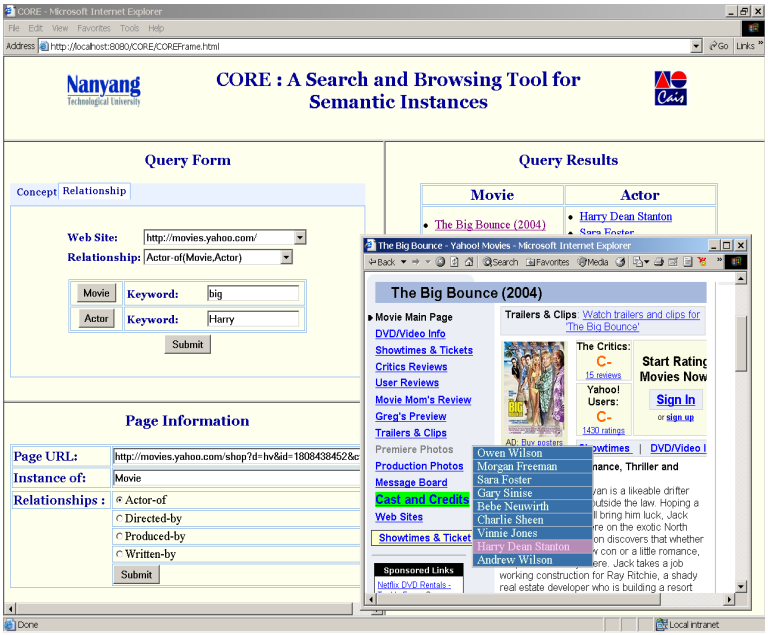
Using Core, the user can formulate a relationship query that involves *Acted-By(Movie, Actor)* relationship. Appropriate search terms can be specified against the source and target concepts. In this example scenario, the user can input "big" as the search term for the *Movie* concept and "Harry" as the search term for the *Actor* concept. Core will return pairs of instances (Web pages) of the *Acted-By* relationship that satisfy the relationship query. The query results are shown in Figure 5.4(b).

The left column of the result table displays the *Movie* concept instances and the right column displays the related *Actor* concept instances. Note that when the same source Web page have multiple related target Web pages that satisfy the relationship query, the results with the same source page will be merged together showing the source page only once.

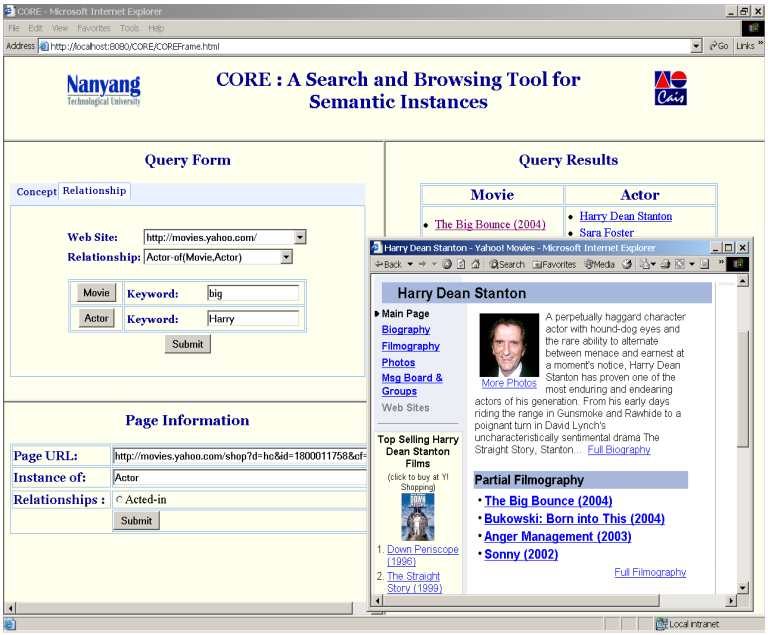
### 5.3.2 Browsing Scenario

The traditional Web browsers are not aware of the semantics of Web pages. They are not equipped with the ability to display semantic information and to support browsing based on the concept and relationship labels. Users therefore have to decide which Web pages to visit by examining the anchor elements linking to the pages. Decision of related target page can be made only after retrieving and analyzing the target page. Due to the limited information provided by the anchor text and intermediate Web pages to be traversed before reaching the wanted Web pages, the users are likely to commit errors in the choice of Web pages to be browsed next and waste some of their time and efforts.

For example, starting from a movie home page, a user may want to navigate to its actor home page. The user can directly reach the actor home pages only if there are some anchor elements (i.e, with the names of the actors as anchor text) available within the movie home page. If there is no direct link from the movie page to the




(a)




(b)

Figure 5.5: (a)Browsing Movie home page with highlighted anchor texts for Acted-By relationship (b)Browsing Actor's home page by directly clicking the Acted-By relationship instance listed in the Movie source page

actor page, the user will not know which anchor element should be traversed in order to reach the desired actor page. Even if there exists a direct link from the movie page

to the actor page, the decision of actual related actor page can be made only after the actor page has been browsed or analyzed.

To rectify the above problem, Core allows a semantic-based selection of Web pages for browsing, and the navigation of relationship instances. Users do not need to spend time on blindly navigating or analyzing the unseen Web pages.

Let us assume that a user has browsed the movie page "The Big Bounce(2004)" from the relationship query result as shown in Figure 5.5. In the figure, the URL, concept label and relevant relationships of the selected movie Web page are automatically displayed in the page information frame while the page is displayed in another browser window.

Suppose the user wishes to see the Web page of some actor who appeared in "The Big Bounce(2004)", he/she can select the *Acted-By* relationship in the page information frame. Upon selection, the anchor element (i.e., "Cast and Credits") that leads to actor Web pages will immediately be highlighted. Moreover, as the user moves the mouse over the highlighted anchor text, virtual links that lead to actors' Web pages will appear in the browser window. These virtual links display the names of actors in the movie as shown in Figure 5.5(a). The user can now choose any of the actor Web pages to browse without going astray.

Suppose the user chooses to visit the Web page of actor "Harry Dean Stanton" as shown in Figure 5.5(b). The Web page is loaded into the browser window and the page information panel is updated accordingly including a new set of relationships relevant to actors. In this way, user can minimize errors in Web site navigation.

## 5.4 Summary

Having knowledge of sematic information about the Web pages and related Web pages in the Web Sites is crucial in realizing the goal of the Semantic Web. In particular, finding the semantically related Web pages in the Web Sites is an unexplored and hard task. In this chapter, we present a tool Core for efficient searching of semantic instances and browsing of semantically related Web pages that are related under some

ontological relationships. The relationships between Web pages are determined by exploiting the link chain information among them. CORE is implemented by combining search and browsing services, classification and link chain extraction components.

# Chapter 6

# Conclusion and Future Work

Adding a layer of semantic information on the existing Web can bring about much benefits to the users accessing and navigating Web. To realize this additional semantic layer, annotation of Web objects must be carried out. Nevertheless, Web annotations without well defined structures can only bring about additional chaos, not order. Semantic Web Annotation (SWA) minimizes these chaos by annotating Web with well defined conceptual knowledge known as ontologies. These ontologies mainly consist of concepts, relationships and their attributes.

## 6.1 Summary of Contributions

In our SWA research, three main tasks have been identified. They include the assignment of concept labels to Web objects, assignment of relationship labels to pairs of Web objects and assignment of attribute values to Web objects labelled as concept instances and pairs of Web objects labelled as relationship instances. While many SWA languages and techniques have been proposed, we have found very little research in the annotation of link structures between related Web objects. In this thesis, we model these link structures as link chains and contribute to the research on link chain annotation as follows:

**Annotation of Link Chains for Relationship Instances**

Most of the previous work focused on the annotation of concept instances and their

attributes. In the annotation of relationship instances, most existing research does not consider the annotation of link structure information about relationship instances. Our research therefore gives a formal definition of link chain and develops a normalization approach to identify common shared-paths among link chains so as to reduce redundancy and updating problem in link chain annotation. We also introduce an annotation scheme to define an annotation language that supports the annotation of link chains as part of relationship instance in addition to the annotation of concept and relationship instances.

**Semi-automatic Link Chain Extraction Method**

To reduce the annotation efforts, we have proposed to semi-automatically annotate link chains among related Web pages. We formulate link chain extraction as a multi-page multi-slot IE problem. In our proposed extraction method, we define link chain extraction patterns to be a set of extraction rules on a series of HTML segments locating the chain elements of link chains. Given a set of training examples, the extraction patterns are learnt using a combination of sequential covering, generalization and factoring algorithms. Our link chain extraction method was evaluated on two real life well-structured Web sites for pre-defined relationships. High precision and recall have been observed on our experiments using not more than 15 training examples.

**Exploiting Link Chain Information for Web Browsing and Searching**

Web sites semantically annotated with concept, relationship and link chain information can be browsed and queried very differently. Queries can be augmented with concept and relationship labels. Web pages can also be navigated via their relationships instead of physical links. To allow users to access a Web site using this semantic approach, we developed the Core system for searching and browsing annotated Web site. It mainly consists of the search and browsing server applications, Web page classification module, link chain extraction module and a knowledge repository. We have indexed a Web site using Core and illustrate the various possible ontology based query and browsing scenarios using Core.

## 6.2 Future Work

Annotation and extraction of link chains for the relationship instances is a non-trivial task. Although some foundation work has been carried out in this thesis research, there are several directions of future research that need to be addressed. Some of them are described as follows.

- **Page Path Pattern Discovery:**

  Our link chain extraction method requires link chains sharing the same page path pattern for a given relationship and the page path pattern is assumed to be given. It will be interesting to investigate the discovery of page path patterns automatically assuming that only the source and target Web pages of some link chain training examples are given.

  Automatic discovery of page path pattern includes two sub challenges to be tackled. The first challenge is how to select the Web pages that sharing a common structure of each page path in a particular page path pattern of a Web site. This is related to the problem of finding automatic Web site structure discovery [CMM03]. In automatic Web site structure discovery, a Web site is represented as a graph consisting of nodes(classes) and edges. The pages sharing a common structure are grouped under the same node and their hyperlink structures represent the edges in a graph. In our future work, we may configure the page path pattern as such web site graph. Then one can automatically group the collection of Web pages sharing a common structure for each page path by mapping the nodes in this Web site graph with corresponding page paths contained in a page path pattern.

  Next challenge is to find the page path(schema) of each common group of Web pages automatically. To find a page path representing a collection of Web pages, one can only look for the links in the contents of Web pages. This is closely related to the problem of learning the structure of a single Web page in information extraction [Coh03]. Once we can tackle these challenges in automatic

discovery of page path pattern, it will further save annotation efforts for the link chain examples.

- **Dynamic Learning Methods for Extraction Rules:**

  In our proposed link chain extraction method, extraction rules are assumed to be static once learnt. This assumption can be relaxed if our rule learning method accepts user feedback during link chain extraction to revise extraction rules for evolving Web sites. In dynamic learning approach, user is provided with the intermediate results of extraction rules for each time they are generated. By judging the intermediate results, user can enhance the training data set and iteratively generate extraction rules until he/she is satisfied with the extraction results.

  Dynamic learning methods however require a lot of human efforts and the challenge here is how to enhance the training data set with minimum human intervention. One can further investigate active learning strategies for enhancing the training data set where the document selection algorithm only selects the informative documents for human annotation [FK03]. The selected documents will maximize the future performance of the learnt extraction rules. This dynamic approach to extraction rule learning will be especially useful for Web sites with more complex site structures in which more precise extraction rules are required.

- **Extraction of Relationship Attributes using Link Chains:**

  A relationship instance when exist in the form of Web page pair, not only can have link chain(s) associated with it, but also other attribute values. With much of the research focus on assigning Web pages with concept labels and extracting concept attributes from these Web pages [Bri98, VVMD+01], there has been little or no progress on extracting attributes from related Web pages. The challenge in relationship attribute extraction is to locate the correct attribute values in pairs of related Web pages.

Once link chains of relationship instances are found, we believe that the attribute values of these relationship instances can be derived from information nearby the link chains. For example, *Work-In(Student, Project)* relationship instance in a University Web site is the student homepage and project homepage pair. The *Work-In* relationship attribute (e.g. startdate, enddate) values can be found near the chain elements in the link chain of the corresponding instances. To extract these attribute values, new extraction rules will have to be learnt. An integrated extraction of link chains and relationship attribute values can also be explored.

In conclusion, we have analyzed several existing SWA approaches, proposed the annotation of link chains for relationship instances, proposed a new semi-automatic link chain extraction method and evaluated the proposed method on two real life well-structured Web site. We also have developed a search and browsing tool for concept and relationship instances in SWA by exploiting the link chain information. Finally, we have also indicated several future directions to be investigated. We therefore believe that the works done in this thesis will be useful for future researchers in the area of Semantic Web and Information Extraction.

# Appendix A

# Generating Regular Expressions with Factor Form

In this appendix, we illustrate how to generate regular expressions with factor form for a given set of candidate sequences. As described in Section 4.4.2, generating regular expression with factor form consists of two steps:
(1) Determining the subsets of candidate sequences with factor components and
(2) Generating the regular expression with factor form for each subset. The technique used in this step is based on multilevel boolean minimization [Law64] and multi-level logic optimization [Wan91].

**Step (1)**

Each subset $F$ of candidate sequences with factor components is obtained based on the following two properties [GGR+03]:

- Every sequence in $F$ has a common prefix or suffix with other sequences in $F$. A *score(C,F)* function is used to measure how well a sequence $C$ would factor with other sequences in $F$. Given a candidate sequence $C$, let *pref(C)* and *suf(C)* denote the set of prefixes and suffixes of $C$ respectively. Let *psup(p, F)* and *ssup(s, F)* denote the support of the number of candidate sequences in $F$ having $p$ and $s$ appear as prefix and suffix respectively.

  Then, *score(C,F)* is defined as follows:

$$score(C, F) = max(\{|p| \cdot psup(p, F) : p \in pref(C)\} \cup \{|s| * psup(s, F) : s \in suf(C)\}) \quad \text{(Eq. A.1)}$$

The sequence with a high *score* with respect to $F$ is a good candidate sequence to be factored with other candidate sequences in $F$.

- The *overlap* between every pair of sequences ($C$, $C'$) in $F$ is minimal. Let *cover(C)* denote the set of original training sequences that contain $C$. The *overlap* between $C$ and $C'$ is defined as:

$$overlap(C, C') = \frac{|cover(C) \cap cover(C')|}{|cover(C) \cup cover(C')|} \qquad \text{(Eq. A.2)}$$

The *overlap*($C$, $C'$) is said to be minimal if it is less than a pre-specified parameter $\delta$.

**Step (2)**

We first describe this step using an example. Suppose $F$ is $\{\rho\beta^M, \rho\beta^F, \beta^M, \beta^F, \eta\xi, \eta\zeta, \tau\xi, \tau\zeta\}$. We generate the regular expression with factor form from $F$ as follows:

(i) **Compute the set of possible divisors for $F$**. A divisor is a set of prefixes of sequences from $F$ that have a common suffix containing at least two elements. For the above example, the possible divisors are $\{\eta, \tau\}$ and$\{\rho, 1\}$. The symbol 1 denotes the identity symbol with respect to the concatenation operation, i.e, $1t = t1 = t$ for every sequence $t$.

(ii) **Choose divisor $V$ that gives the minimum total length of divisor, quotient and remainder**. Given a divisor $V$, the quotient $Q$ and remainder $R$ can be determined. For example, let $V$ be $\{\eta, \tau\}$, $Q$ and $R$ will be $\{\zeta, \xi\}$ and $\{\rho\beta^M, \rho\beta^F, \beta^M, \beta^F\}$) respectively. The total length of $V$, $Q$ and $R$ is the sum of their character lengths. In the given example, $V$= $\{\eta, \tau\}$ gives a smaller total length of divisor, quotient and remainder and is thus chosen.

(iii) **Recursively factor $V$,$Q$ and $R$**. Given that the resultant factor form is `FACTOR(V)FACTOR(Q) |FACTOR(R)`, we continue to factor $R$ until there is no divisor for $R$. In the given example, $V = \{\eta, \tau\}, Q = \{\zeta, \xi\}$ and $R = \{\rho\beta^M, \rho\beta^F, \beta^M, \beta^F\}$. $V$ and $Q$ cannot be factored further since they have no divisors. Hence, `FACTOR(V)`

$= \{\eta, \tau\}$ and `FACTOR(Q)` $= \{\zeta, \xi\}$. $R$ can be further factored with $V$= $\{\rho$,1$\}$. Thus, we obtain `FACTOR(R)` $= (\rho|1)(\beta^F|\beta^M)$. There is no further divisors in $R$ and we obtain the final factor form as $((\eta|\tau)(\zeta|\xi)|(\rho|1)(\beta^F|\beta^M))$.

(iv) **Simplify final expression by eliminating "1"**. The term $(\rho|1)$ in the final expression can be further simplified to $(\rho)$? and finally we obtain the desired factor form of the given set as $((\eta|\tau)(\zeta|\xi)|(\rho)?(\beta^F|\beta^M))$ .

# Appendix B

# List of Publications

The author has contributed a total of 5 international conference publications and one awaiting review result for the journal publication. The list is given below. The publication that result indirectly from the work done in this thesis is marked with a '⋆'.

- Myo-Myo Naing, Ee-Peng Lim and Dion Hoe-Lian Goh. Ontology-based Web Annotation Framework for HyperLink Structures. *In Proceedings of WISE'02 Workshop on Data Semantics in Web Information Systems*. pp. 184-193, Singapore, Dec. 2002.

- Myo-Myo Naing, Ee-Peng Lim and Dion Hoe-Lian Goh. A Survey of Ontology-based Web Annotation. *In Proceedings of the 1st International Conference on Computer Applications*. pp. 113-123, Yangon, Myanmar, Jan. 2003.

- Aixin Sun, Myo-Myo Naing and Ee-Peng Lim and Wai Lam. Using Support Vector Machines for Terrorism Information Extraction. *In Proceedings of 1st NSF/NIJ Symposium on Intelligence and Security Informatics*. pp. 1–12, Tucson, Arizona, USA, Jun. 2003.⋆

- Myo-Myo Naing, Ee-Peng Lim and Dion Hoe-Lian Goh. On Extracting Link Information of Relationship Instances from a Web Site. *In Proceedings of the International Conference on Web Services-Europe 2003*. pp. 213-226, Erfurt, Germany, Sept. 2003.

- Myo-Myo Naing, Ee-Peng Lim and Roger H.L. Chiang. Extracting Link Chains of Relationship Instances from a Web Site. Submitted to *Journal of the American Society for Information Science and Technology*. Oct. 2004.

- Myo-Myo Naing, Ee-Peng Lim and Roger H.L. Chiang. Core: A Search and Browsing Tool for Semantic Instances of Web Sites. *In Proceedings of the 7th Asia Pacific Web Conference (APWeb'05)*. pp. 429-440, Shanghai, China, April. 2005.

# References

[AG00] E. Agichtein and L. Gravano. Snowball: Extracting Relations from Large Plain-Text Collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*, pages 85–94, San Antonio, Texas, USA, June 2000.

[AGP+01] E. Agichtein, L. Gravano, J. Pavel, V. Sokolova, and A. Voskoboynik. Snowball: A Prototype System for Extracting Relations from Large Text Collections. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, page 612. ACM Press, 2001.

[AKM+03] H. Alani, S. Kim, D. E. Millard, M. J. Weal, W. Hall, P. H. Lewis, and N. R. Shadbolt. Automatic Ontology-Based Knowledge Extraction from Web Documents. *IEEE Intelligent Systems*, 18(1):14–21, 2003.

[BBC+98] J. Bosak, T. Bray, D. Connolly, E. Maler, G. Nicol, C. M. S-McQueen, L. Wood, and J. Clark. Guide to the W3C XML Specification ("XML-Spec") DTD, 1998. http://www.w3.org/XML/1998/06/xmlspec-report-v21.htm.

[BG01] M. Bremer and M. Gertz. Web data indexing through external semantic-carrying annotations. In *Proceedings of the 11th International Workshop on Research Issues in Data Engineering: Document Management for Data Intensive Business and Scientific Applications*, pages 69–76, Heidelberg, Germany, April 2001.

[BG02] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, April 2002. http://www.w3.org/TR/rdf-schema/#ch_appendix_rdfs.

[BM98] A. Blum and T. Mitchell. Combining Labeled and Unlabeled Data with Co-training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pages 92–100, Madison, Wisconsin, United States, 1998. Morgan Kaufmann Publishers.

[BPSM00] T. Bray, J. Paoli, C. M. Sperberg, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition), October 2000. http://www.w3.org/TR/REC-xml.

[Bri98] S. Brin. Extracting Patterns and Relations from the World Wide Web. In *Proceedings of International Workshop on the Web and Databases at 6th International Conference on Extending Database Technology EDBT'98*, pages 172–183, Valencia, Spain, 1998.

[CDF+00] M. Craven, D. DiPasquo, D. Freitag, A. K. McCallum, T. M. Mitchell, K. N., and Seán Slattery. Learning to Construct Knowledge Bases from the World Wide Web. *Artificial Intelligence*, 118(1/2):69–113, 2000.

[CDH+00] B. Czejdo, J. Dinsmore, C. Hwang, R. Miller, and M. Rusinkiewicz. Automatic Generation of Ontology based Annotations in XML and Their Use in Retrieval Systems. In *Proceedings of 1st International Conference on Web Information Systems Engineering*, pages 296–300, Hong Kong, China, June 2000. IEEE Computer Society.

[CDI98] S. Chakrabarti, B. E. Dom, and P. Indyk. Enhanced Hypertext Categorization using Hyperlinks. In Laura M. Haas and Ashutosh Tiwary, editors, *Proceedings of SIGMOD-98, ACM International Conference on Management of Data*, pages 307–318, Seattle, US, 1998. ACM Press, New York, US.

[CGP00] O. Corcho and A. G.-Perez. A Roadmap to Ontology Specification Languages. In *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management*, pages 80–96, Juan-Les-Pins, France, October 2000.

[Cir01] F. Ciravegna. Adaptive Information Extraction from Text by Rule Induction and Generalisation. In *Proceedings of the 17th International Conference on Artificial Intelligence (IJ CAI-01)*, pages 1251–1256, San Francisco, Seattle, USA, August 2001.

[CM98] M. E. Califf and R. J. Mooney. Relational Learning of Pattern-Match Rules for Information Extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.

[CMBT02] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, Philadelphia, US, July 2002.

[CMM01] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118, Roma, Italy, 2001.

[CMM02] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Automatic Data Extraction from Data-Intensive Web Sites. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, page 624. ACM Press, 2002.

[CMM03] V. Crescenzi, P. Merialdo, and P. Missier. Fine-Grain Web Site Structure Discovery. In *Proceedings of the 5th ACM International Workshop*

*on Web Information and Data Management*, pages 15–22. ACM Press, 2003.

[Coh03] W. W. Cohen. Learning and discovering structure in web pages. *IEEE Data Engineering Bulletin*, 26(3):3–10, 2003.

[CS99] W. W. Cohen and Y. Singer. Context-sensitive learning methods for text categorization. *ACM Transactions on Information Systems (TOIS)*, 17(2):141–173, 1999.

[CS00] B. D. Czejdo and C. Sobaniec. Using a Semantic Model and XML for Document Annotation. In *Proceedings of 13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 236–241, New Orleans, Louisiana, USA, June 2000.

[CS01] M. Craven and S. Slattery. Relational Learning with Statistical Predicate Invention: Better Models for Hypertext. *Machine Learning*, 43(1/2):97–119, 2001.

[CS04] W. W. Cohen and S. Sarawaji. Exploiting Dictionaries in Named Entity Extraction:Combining Semi Markov Extraction Processes and Data Integration Methods. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 89–98, Seattle, Washington, USA, August 2004.

[CSN98] M. Craven, S. Slattery, and K. Nigam. First-Order Learning for Web Mining. In *European Conference on Machine Learning*, pages 250–255, Chemnitz, Germany, 1998.

[CW03] F. Ciravegna and Y. Wilks. *Annotation for the Semantic Web*, volume 96, chapter Designing Adaptive Information Extraction for the Semantic Web in Amilcare, pages 112–127. IOS Press, August 2003.

[DBDEM03] M. Dzbor, J. B.-Domingue, and E.-Motta. Magpie - Towards a Semantic Web Browser. In *Proceedings of the 2nd International Semantic Web Conference*, Florida, USA, October 2003.

[DDM04] J. Domingue, M. Dzbor, and E. Motta. Magpie: Supporting Browsing and Navigation on the Semantic Web. In *Proceedings of the 9th International Conference on Intelligent User Interface (IUI'2004)*, pages 191–197, Funchal, Madeira, Portugal, January 2004. ACM Press.

[DEFS99] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology based Access to Distributed and Semi-Structured Information. In *Proceedings of the Semantic Issues in Multimedia Systems Conference*, pages 351–369, Boston, MA, USA, 1999.

[DH95] J. R. Davis and D. P. Huttenlocher. Shared annotation for cooperative learning. In *Proceedings of the 1st International Conference on Computer Support for Collaborative Learning*, pages 84–88, Indiana Univ., Bloomington, Indiana, United States, 1995. Lawrence Erlbaum Associates, Inc.

[ECJ+99] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, D. W. Lonsdale, Y. K. Ng, and R. D. Smith. Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages. *Data and Knowledge Engineering*, 31(3):227–251, November 1999.

[EKS02] M. Ester, H.-Peter Kriegel, and M. Schubert. Web Site Mining : A new way to spot Competitors, Customers And Suppliers in the World Wide Web. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD'02)*, pages 249–258, Edmonton,CA, 2002.

[EMSS01] M. ErdMann, A. Maedche, H. P. Schnurr, and S. Staab. From Manual to Semi-automatic Semantic Annotation: About Ontology-based Text Annotation Tools. *Computer and Information Science*, 6, 2001.

[EN02] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems(Third Edition)*. Addison-Wesley, 2002.

[FAD+99] D. Fensel, J. Angele, S. Decker, M. Erdmann, H.-P. Schnurr, S. Staab, R. Studer, and A. Witt. On2broker: Semantic-based access to information sources at the WWW. In *Proceedings of the World Conference on the WWW Internet(WebNet99)*, pages 366–371, Honolulu, Hawaii, 1999.

[FDES98] Dieter Fensel, Stefen Decker, Michael Erdmann, and Rudi Studer. Ontobroker: Or how to enable intelligent access to the WWW. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System workshop*, Banff, Canada, April 1998. URL: http://www.aifb.uni-karlsruhe.de/WBS/broker.

[FK03] A. Finn and N. Kushmerick. Active learning selection strategies for information extraction. In *Proceedings of the Workshop on Adaptive Text Extraction and Mining (ECML-03)*, Croatia, September 2003.

[FNR03] Jürgen Franke, Gholamreza Nakhaeizadeh, and Ingrid Renz, editors. *Text Mining, Theoretical Aspects and Applications*. Physica-Verlag, 2003.

[Fre98] D. Freitag. Information Extraction from HTML: Application of a General Machine Learning Approach. In *Proceedings of the 15th International Confonference on Artificial Intelligence (AAAI-98)*, pages 517–523, 1998.

[Fur99] J. Furnkranz. Exploiting Structural Information for Text Classification on the WWW. In *Proceedings of 3rd Symposium on Intelligent Data Analysis*, pages 487–498, Amsterdam, Netherlands, August 1999.

[Gai97] B. Gaines. Editorial: Using Explicit Ontologies in Knowledge-based System Development. *International Journal of Human-Computer Systems*, 46(181), 1997.

[GGR+03] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: Learning Document Type Descriptors from XML Document Collections. *Data Mining and Knowledge Discovery*, 7(1):23–56, January 2003.

[GTL+02] E.J. Glover, K. Tsioutsiouliklis, S. Lawrence, D.M. Pennock, and G.W. Flake. Using Web Structure for Classifying and Describing Web Pages. In *Proceedings of the 11th International Conference on the World Wide Web*, pages 562–569, Honolulu, Hawaii, USA, May 2002.

[Hef01] J. Heflin. *Towards the Semantic WEB: Knowledge Representation in a Dynamic, Distributed Environment*. PhD thesis, University of Maryland, College Park, MD 20742, USA 301.405.1000, 2001.

[HLO99] R. M. Heck, S. M. Luebke, and C. H. Obermark. A Survey of Web Annotation Systems, 1999. http://www.math.grin.edu/ rebelsky/-Research/Summer1999/Papers/survey_paper.html.

[Hol96] K. Holtman. The futplex system. In *Proceedings of the International ERCIM Workshop on Community Support for Colleboration on the Web (CSCW)*, Sankt Augustin,Germany, February 1996.

[HSC02] S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM–Semi-automatic CREAtion of Metadata. In *Proceedings of 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, pages 358–372, Sigüenza, Spain, October 2002.

[JV99] K. Jain and V. V. Vazirani. Primal-Dual Approximation Algorithms for Metric Facility Location and k-Median Problems. In *Proceedings*

*of the 40th Annual Symposium on Foundations of Computer Science*, page 2. IEEE Computer Society, 1999.

[KAH+02] S. Kim, H. Alani, W. Hall, P. Lewis, D. Millard, N. Shadbolt, and M. Weal. Artequakt: Generating Tailored Biographies from Automatically Annotated Fragments from the Web. In *Proceedings of Workshop on Semantic Authoring, Annotation & Knowledge Markup (SAAKM02), the 15th European Conference on Artificial Intelligence, (ECAI02)*, pages 1–6, Lyon, France, 2002.

[KC04] G. Klyne and J.J. Carroll. Resource Description Framework (RDF):Concepts and Abstract Syntax, February 2004. http://www.w3.org/TR/rdf-concepts/.

[KK01] J. Kahan and M.-R. Koivunen. Annotea: an open RDF infrastructure for shared Web annotations. In *Proceedings of the 10th International World Wide Web Conference on World Wide Web*, Hong Kong, May 2001.

[KPT+04] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff. Semantic Annotation, Indexing, and Retrieval . *Elsevier's Journal of Web Sematics*, 1(2), 2004.

[KS97] D. Koller and M. Sahami. Hierarchically Classifying Documents Using Very Few Words. In *Proceedings of the 14th International Conference on Machine Learning*, pages 170–178, Nashville, Tennessee, 1997.

[Kus00] N. Kushmerick. Wrapper Induction: Efficiency and Epressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[Law64] E. L. Lawler. An Approach to Multilevel Boolean Minimization. *Journal of ACM*, 11(3):283–295, 1964.

[Lev96] V. I. Levenshetin. Binary Codes Capable of Correcting Deletions,Insertions and Reversals. *Cybernetics and Control Theory*, 10(4):707–710, 1996.

[LG03] Q. Lu and L. Getoor. Link-based Text Classification. In *Proceedings of the 20th International Conference on Machine Learning*, pages 496–503, Washington, DC USA, August 2003.

[LHL01] T.-B. Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):35–43, 2001.

[LNL04] Z. Liu, W.-K. Ng, and E.-P. Lim. An Automated Algorithm for Extracting Website Skeleton. In *Proceedings of the 9th International Conference on Database Systems for Advanced Applications (DASFAA 2004)*, pages 799–811, Jeju Island, KOREA, March 2004.

[LNLL04] Z. Liu, W.-K. Ng, E.-P. Lim, and F. Li. Towards Building Logical Views of Websites. *Data and Knowledge Engineering*, 49(2):197–222, May 2004.

[LRN02] A. H. F. Laender and B. A. R.-Neto. A Brief Survey of Web Data Extraction Tools. *SIGMOD Record*, 31(2):84–93, June 2002.

[Mac90] MacGregor. LOOM Users Manual. ISI/WP-22, 1990. http://www.isi.edu:80/isd/LOOM/.

[McG98] D. McGuinness. Ontological Issues for Knowledge-Enhanced Search. In *Proceedings of the 1st International Conference on Formal Ontology in Information Systems*, pages 302–316, Trento,Italy, 1998.

[ME00] P. Martin and P. Eklund. Knowledge Retrieval and the World Wide Web. *IEEE Intelligent Systems Special issue on Knowledge Management and the Internet*, 15(3):18–25, May-June 2000.

[MI01] E. Mena and A. Illarramendi. *Ontology-based Query Processing for Global Information Systems*, page 215. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[Mil98] E. Miller. An Introduction to the Resource Description Framework, May 1998. http://www.dlib.org/dlib/may98/miller/05miller.html.

[Mit97] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[MKSA00] E. Mena, V. Kashyap, A. Sheth, and A.Illarramendi. OBSERVER: An Approach for Query Processing in Global Information Systems Based on Interoperation Across Pre-Existing Ontologies. *Distributed and Parallel Databases - An International Journal*, 8(2):223–271, 2000.

[MKSI98] E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi. Domain Specific Ontologies for Semantic Information Brokering on the Global Information Infrastructure. In *Proc. of the 1st Int. Conf. on Formal Ontologies in Information Systems*, Trento, Italy, June 1998.

[MMK01] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical Wrapper Induction for Semistructured Information Sources. *Autonomous Agents and Multi-Agent Systems*, 4(1-2):93–114, 2001.

[MNS03] A. Maedche, G. Neumann, and S. Staab. Bootstrapping an Ontology-Based Information Extraction System. *Studies in Fuzziness and Soft Computing, Intelligent Exploration of the Web*, pages 345–359, 2003.

[Mot98] E. Motta. An Overview of the OCML Modelling Language. In *Proceedings of KEML'98: 8th Workshop on Knowledge Engineering Methods & Languages*, pages 21–22, Karlsruhe, Germany, January 1998.

[MTC+02] D. Maynard, V. Tablan, H. Cunningham, C. Ursu, H. Saggion, K. Bontcheva, and Y. Wilks. Architectural Elements of Language Engineering Robustness. *Journal of Natural Language Engineering*

– *Special Issue on Robust Methods in Analysis of Natural Language Data*, 2002.

[Mus99] I. Muslea. Extraction Patterns for Information Extraction Tasks: A Survey. In *Proceedings of AAAI-99 Workshop on Machine Learning for Information Extraction*, pages 1–6, Orlando, Florida, July 1999.

[NGL97] H. T. Ng, W. B. Goh, and K. L. Low. Feature selection, perceptron learning, and a usability case study for text categorization. In Nicholas J. Belkin and A. Desai Narasimhalu and Peter Willett, editor, *Proceedings of 20th ACM International Conference on Research and Development in Information Retrieval*, pages 67–73, Philadelphia, US, 1997. ACM Press, New York, US.

[NLC05] M.-M. Naing, E.-P. Lim, and R. H. L. Chiang. CORE: A Search and Browsing Tool for Semantic Instances of Web Sites. In *Proceedings of the 7th Asia Pacific Web Conference, APWeb'05*, Shanghai, China, 2005. To appear.

[NLG02] M. M. Naing, E.-P. Lim, and D. H.-L. Goh. Ontology-based Web Annotation Framework for HyperLink Structures. In *Proceedings of WISE'02 Workshop on Data Semantics in Web Information Systems*, pages 184–193, Singapore, December 2002.

[NLG03] M. M. Naing, E.-P. Lim, and D. H.-L. Goh. On Extracting Link Information of Relationship Instances from a Web Site. In *Proceedings of the International Conference on Web Services-Europe 2003*, pages 213–226, Erfurt, Germany, September 2003.

[OML00] H.-J. Oh, S. H. Myaeng, and M.-H. Lee. A Practical Hypertext Catergorization method using Links and Incrementally Available Class Information. In *Proceedings of the 23rd Annual International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 264–271. ACM Press, 2000.

[Pie01] J. M. Pierre. On Automated Classification of Web Sites. *Linköping Electronic Articles in Computer and Information Science*, 6(001), March 2001.

[PKO+03] B. Popov, A. Kiryakov, D. Ognyanoff, D. Manov, A. Kirilov, and M. Goranov. Towards Semantic Web Information Extraction. In *Proceedings of the Human Language Technologies Workshop at the 2nd International Semantic Web Conference (ISWC2003)*, Florida, USA, October 2003.

[QCJ93] J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667, pages 3–20. Springer-Verlag, 1993.

[QK04] D. A. Quan and R. Karger. How to make a Semantic Web Browser. In *Proceedings of the 13th International Conference on World Wide Web*, pages 255–265, New York, USA, 2004. ACM Press.

[RM92] B. L. Richards and R. J. Mooney. Learning relations by pathfindings. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 723–728, 1992.

[RMW95] M. Röscheisen, C. Mogensen, and T. Winograd. Shared Web Annotations As A Platform for Third-Party Value-Added Information Providers: Architecture, Protocols, and Usage Examples. Technical report, Computer Science Department, Stanford University, Stanford, CA 94305, U.S.A., February 1995. http://www-diglib.stanford.edu/diglib/pub/reports/commentor.html.

[SB88] G. Salton and C. Buckley. Term-weighting approaches in Automatic Text Retrieval. *Information Processing and Management*, 25(5):513–523, 1988.

[SCD+97] Veda C. Storey, Roger H. L. Chiang, Debabrata Dey, Robert C. Goldstein, and Shankar Sudaresan. Database design with common sense business reasoning and learning. *ACM Trans. Database Syst.*, 22(4):471–512, 1997.

[Seb02] F. Sebastiani. Machine Learning in Automated Text Categorization. *ACM Computing Survey*, 34(1):1–47, March 2002.

[SEMD00] S. Staab, M. Erdmann, A. Maedche, and S. Decker. An extensible approach for modeling Ontologies in RDF(S), 2000.

[SG95] S. Sekine and R. Grishman. A Corpus-based Probablistic Grammar with only Two Non-Terminals. In *Proceedings of 4th International Workshop on Parsing Technologies*, pages 216–223, Prague, Czech Republic, 1995.

[SL01] A. Sun and E.-P. Lim. Hierarchical Text Classification and Evaluation. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 521–528, San Jose, California, USA, 2001. IEEE Computer Society.

[SL03] A. Sun and E.-P. Lim. Web Unit Mining: Finding and Classifying Subgraphs of Web Pages. In *Proceedings of the 12th International Conference on Information and Knowledge Management (CIKM'2003)*, pages 108–115, New Orleans, USA, 2003.

[SLN02] A. Sun, E.-P. Lim, and W.-K. Ng. Web classification using Support Vector Machine. In *Proceedings of the 4th International Workshop on Web Information and Data Management (WIDM 2002)*, pages 96–99, Virginia, USA, November 2002.

[SLN03] A. Sun, E.-P. Lim, and W.-K. Ng. Performance Measurement Framework for Hierarchical Text Classification. *Journal of the American Society for Information Science and Technology*, 54(11):10141028, September 2003.

[SMH01] S. Staab, A. Maedche, and S. Handschuh. An Annotation Framework for the Semantic Web. In *Proceedings of the 1st Workshop on Multimedia Annotation*, Tokyo,Japan, 2001.

[Sod99] S. Soderland. Learning Information Extraction Rules for Semi-structured and Free Text. *Journal of Machine Learning*, 34(1-3):233–272, 1999.

[SS00] R. E. Schapire and Y. Singer. BoosTexter: A Boosting-based System for Text Categorization. *Machine Learning*, 39(2-3):135–168, May-June 2000.

[Ste01] S. H. Steffen. CREAM - Creating Relational Metadata with a Component-based, Ontology-driven annotation Framework. In *Proceedings of 1st International Conference on Knowledge Capture*, pages 76–83, Victoria, BC, Canada, 2001.

[Sul02] D. Sullivan. How To Use HTML Meta Tags, December 2002. http://searchenginewatch.com/webmasters/article.php/2167931.

[TBMM01] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part1: Structures, May 2001. http://www.w3.org/TR/xmlschema-1/.

[UG96] M. Uschold and M. Grüninger. Ontologies: Principles, Methods, and Applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[VVDK+01] M. V.-Vera, J. Domingue, Y. Kalfoglou, E. Motta, and S. B. Shum. Template-Driven Information Extraction for Populating Ontologies. In *Proceedings of the IJCAI01 Workshop on Ontologies Learning*, Seattle, WA, USA, August 2001.

[VVMD+01] M. V.-Vera, E. Motta, J. Domingue, S. Shum, and M. Lanzoni. Knowledge Extraction by using an Ontology-based Annotation Tool. In *Proceedings of the Workshop Knowledge Markup and Semantic Annotation, K-CAP'01*, Victoria, Canada, October 2001.

[VVMD+02a] M. V.-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology Driven Tool for Semantic Markup. In *Proceedings of the Workshop Semantic Authoring, Annotation & Knowledge Markup (SAAKM 2002)*, Lyon France, July 2002.

[VVMD+02b] M. V.-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology Driven Semi-Automatic and Automatic Support for Semantic Markup. In *Proceedings of the 13th International Conference on Knowledge Engineering and Management(ECAW 02)*, pages 379–391, Siguenza, Spain, October, 2002.

[Wan91] A. R. R. Wang. *Algorithms for Multi-Level Logic Optimization*. PhD thesis, The University of California, Berkeley, 1991.

[Wel98] C. Welty. The Ontological Nature of Subject Taxonomies. In *Proceedings of the International Conference on Formal Ontology in Information Systems*, June 1998.

[YSG02] Y. Yang, S. Slattery, and R. Ghani. A Study of Approaches to Hypertext Categorization. *Journal of Intelligent Information Systems*, 18(2-3):219–241, 2002.

[ZS02] G. Zhou and J. Su. Named Entity Recognition using an HMM-based Chunk Tagger. In *Proceedings of Association for Computational Linguistics'02*, pages 473–480, Philadelphia, USA, 2002.