A PROPERTY-BASED APPROACH TO INTEGRATING INFORMATION FROM MULTIPLE SOURCES

by

HAOHUA (HOWARD) ZHUANG

B.ECO., SHANGHAI JIAO TONG UNIVERSITY, 1995

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
(Faculty of Commerce and Business Administration)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

April 2003

Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Haohua (Howard) Zhuang
Name of Author *(please print)*

Dec. 10th, 2003
Date

Title of Thesis: A Property-Based Approach To Integrating Information From Multiple Sources

Degree: Master of Science(Business Administration) Year: 2003

ABSTRACT

The challenge of integrating information from multiple data sources automatically, via computers, has long been of great interest in the area of Databases and Information Systems. It has become more important with the growth of the Internet and proliferation of Semi-Structured information sources. The major obstacle to integrating information from multiple sources lies in reconciling the meaning of data, which is known as Semantic Reconciliation or Semantic Integration.

In the design of databases and information systems, the meaning of data that can be stored is usually described by a conceptual model. Thus, one needs to express data or transform data into a comprehensive and uniform conceptual model before approaching the issue of automatic (computer-based) integration.

This thesis proposes a conceptual model - the property-based model - as the basis for semantic data integration. The main premise underlying the model is that to identify the meaning of data requires identifying what the data represents in the world. The branch of philosophy dealing with what exists in the world is Ontology; hence, the proposed model is based on ontological foundations. Specifically, it is based on Bunge's ontology as adapted to information systems by Wand and Weber, and on a method for semantic reconciliation proposed by Parsons and Wand.

The thesis begins with an overview of research in the field of Information Integration. It discusses the most common type of conceptual models - the Class-Based Model, which includes the Entity-Relationship Model and Object-Oriented Models, and mentions some problems with this type of models. Then the thesis presents the property-based model. Based on the Property-Based model, the thesis proposes and develops a Property-Based Approach to integrating information from multiple sources. The approach uses two main tools, the Property-Precedence Schema (PPS) and the Instance Function (IF). After discussing briefly some practical issues, the thesis introduces a general integration procedure and demonstrates it on a case study to illustrate the usability of the approach.

TABLE OF CONTENTS

Abstract ii
Table of Contents iii
List of figures vi
List of Tables viii
Acknowledgments ix
Chapter 1 Introduction 1
1.1 Motivation 2
1.2 Thesis Objectives 2
1.3 Thesis Outline 3
Chapter 2 Context and Related Works 5
2.1 Heterogeneity and Multiple Schemas 6
2.2 Semantic Interoperability and Information Systems (IS) Ontology 7
2.2.1 Translating Queries by Ontologies 9
2.2.2 Global Schema Generated by Ontologies 11
2.2.3 OntoClean 14
2.2.4 ONIONS 15
2.3 Knowledge Representation and Knowledge Discovery 16
2.3.1 Ontological Model of Knowledge 18
2.4 Conceptual Modelling and Philosophy Ontology 19
2.4.1 The Class-Based Model and its Ontology 19
2.4.2 The BWW-Ontology and the Property-Based Model 21
2.5 Summary 22
Chapter 3 The BWW-Ontology and Its Implications 23
3.1 Some Fundamental Constructs of the BWW-Ontology 23
3.2 Precedence and its Implications 29
3.2.1 Class Definition 31
3.2.2 Property Value 33
3.2.3 Property Aggregation 36

3.3 Query and Information Integration 38
3.3.1 Query 38
3.3.2 Information Integration 40
3.4 Summary 43
Chapter 4 The Property-Based Model 44
4.1 Thing Representation 45
4.1.1 Identity 45
4.1.2 Property Representation 48
4.1.3 Representation of Thing and Property 50
4.2 Define the Semantics 52
4.2.1 Intrinsic Properties (Attributes) 53
4.2.2 Mutual Properties (Attributes) 55
4.2.3 The Whole Picture of Property and Precedence 56
4.3 Summary 58
Chapter 5 The Property-Based Approach To Information Integration 59
5.1 Represent Precedences 59
5.1.1 Property Precedence Schema (PPS) 60
5.1.2 Precedence Functions 61
5.1.3 Some Special Cases in PPSs 62
5.2 Generating Global PPS 65
5.2.1 Domain Problem 67
5.2.2 A Procedure to Create a Global PPS 69
5.3 Instance Function 70
5.3.1 Basic Instance Function 70
5.3.2 Expansion Feature and the Inference of Precedence 72
5.3.3 Conjugation Feature and the Inference of Property Aggregation 74
5.3.4 Complete Instance Function 77
5.4 Summary 78
Chapter 6 Practical Issues of the Property-Based Approach 79
6.1 The Property-Based View 79
6.1.1 Classes 80

6.1.2 High-order Attributes 81
6.2 Implement PPSs 82
6.3 Implement Instance Functions 84
6.3.1 Identities 84
6.3.2 Represent a Set of Things 85
6.3.3 The Hint Attribute 86
6.4 Summary 87
Chapter 7 The Integration Process and a Case Study 88
7.1 The Background of the Case 88
7.2. The Preparation for Integration 93
7.2.1 The Property-Based Views 93
7.2.2 Two local PPSs 94
7.2.3 The Global PPS 95
7.3 The Integration Process 97
7.3.1 Demonstrating the Integration Process with the Case 98
7.3.2 Discussion 108
7.4 Summary 109
Chapter 8 Conclusion and Future Research 110
8.1 Contribution 111
8.2 Limitations 112
8.3 Future Research 113
Bibliography 115
Appendix A: All Properties in the Case 118
Appendix B: Three Precedence Tables 119
Appendix C: An Instance Function in XML syntax 120

LIST OF FIGURES

Number *Page*

Figure 2-1: Ontology Interoperability in SHOE 10
Figure 2-2: Global Schema Generation by Integration of Ontologies 12
Figure 4-1: Representation Model I 45
Figure 4-2: Representation Model II 47
Figure 4-3: Representation Model III 47
Figure 4-5: Representation Model V 51
Figure 4-6: The Whole Picture of Property and Precedence 57
Figure 5-1: Four Kinds of Precedences 60
Figure 5-2: Two Examples of Precedence Functions 62
Figure 5-3: The First Special Case in PPSs 63
Figure 5-4: The Second Special Case in PPSs 64
Figure 5-5: Two Examples of Redundant Precedences 64
Figure 5-6: An Example of Co-Precedence 65
Figure 5-7: Local PPSs and Global PPS 68
Figure 5-8: The Basic Instance Function 71
Figure 5-9: Expansion Feature in the Instance Function 73
Figure 5-10: The Inference of Precedence 74
Figure 5-11: Conjugation Feature in the Instance Function 75
Figure 5-12. An Example of the Inference of Property Aggregation 76
Figure 5-13: An Example in the Instance Function 78
Figure 6-1: An Example of The Transformation Process 80
Figure 6-2: The Structure of XML File of Instance Functions 85
Figure 7-1: Two Local PPSs in Aeroplan and United 94
Figure 7-2: The Global PPS in Star Alliance 96
Figure 7-3: An Instance Function for Main.2 104
Figure 7-4: An Instance Function for Main.3 105
Figure 7-5: The First Middle Result of Main.4 106

Figure 7-6: The Second Middle Result of Main.4 106
Figure 7-7: The Third Middle Result of Main.4 107
Figure 7-8: The Fourth Middle Result of Main.4 107

LIST OF TABLES

Table 4-1: An Example of the Semantics of Intrinsic Property (Attribute) 55
Table 4-2: An Example of the Semantics of Mutual Property (Attribute) 56
Table 6-1: An Example of The Property-Based Views 81
Table 6-2: Precedence Table (I) 83
Table 6-3: Precedence Table (II) 83
Table 7-1: Determine FFP Status by FFP's Miles or Segments 89
Table 7-2: Abbreviations of Privilege Services 90
Table 7-3: Determine FFP Privilege by FFP Status 90
Table 7-4: Determine SA Status by FFP Status 91
Table 7-5: Determine Privilege Service by SA Status 91
Table 7-6: Determine FFP Privilege by SA Privilege 92
Table 7-7: Determine SA Status by SA_mileage or SA_segments 92
Table 7-10a: Data in the Local Sources (Aeroplan) 98
Table 7-10b: Data in the Local Sources (United) 100
Table 7-11: Main Procedure 101
Table 7-12: Sub-Procedures 104

ACKNOWLEDGMENTS

First, I am very grateful to my supervisor Professor Yair Wand. He provided me constructive opinions and guidelines during the process of this thesis. I am also indebted to Professor Jeffrey Parsons who, with Professor Yair Wand, conducted the research project that led to this thesis.

It is my pleasure to record my deep gratitude to the MIS division in the Faculty of Commerce in University of British Columbia for the opportunity and experience that I am enjoying now to develop my academic path. In particular, I want thank Professor Carson Woo, Professor Jacob Steif, William Tan, and Julie Nichols.

Finally, I want to give my appreciation to my parents, Mr. Yongfang Zhuang and Mrs. Cuiying Yu. Their constant support is the foundation for the success of this thesis. My last thanks will go to my lovely wife, Jingwei Xu, who brings me the luck, inspiration and support during the composition.

Chapter One

INTRODUCTION

Since the wide adoption of relational databases, more and more companies have stored their business information in computerized Information Systems. Along with the development of Information Systems, the information becomes critical to the improvement of efficiency and productivity, and even the survival of businesses. The demand for exchanging and integrating the information between Information Systems fulfilled automatically by computers is increasing higher and higher. This is particularly true when we look back at the changes in the Internet in the last decade, such as E-Commerce and E-Business, and the current development of the third generation Internet, the Semantic Web.

For this reason, Information Integration has been an active research area. The first step of Information Integration is Semantic Interoperability, which is defined as reconciling and integrating the meaning of the information. The difficulty arises from the problem of Semantic Heterogeneity of the local Schemas of multiple data sources. Semantic Heterogeneity is defined as differences or similarities in the meaning of local data, or in other words, the semantic of meta-data in the local schemas (Hakimpour and Geppert, 2001). Semantic Interoperability is also known as *Semantic Integration*, or *Semantic Reconciliation.*

1.1 Motivation

In the current field of Information Integration, there are several related research topics. These include Databases Design and Management, Knowledge Representation (KR) in the field of Artificial Intelligence (AI), Knowledge Discovery in Databases (KDD) or Data Mining, Structured and Semi-Structured Data, Semantic Heterogeneity, Semantic Interoperability, Schema Integration and most recently Ontology Languages on the Semantic Web.

However, Conceptual Modeling, as a fundamental research field of Information Systems that studies the way people view the world (or the domain of interest) and thus determines the structure and semantics of the information by which we describe the world, seems restricted in the current paradigm. We term the common type of conceptual model in the current use, Class-Based, because things or instances are represented through classes or types. Two class-based models commonly used in Information System are the Entity-Relation (ER) model and the Objected-Oriented Model. There are several problems (see details in section 2.4.1) inherent in such models (Parsons and Wand, 2000). We might wonder if these models are obstacles when we try to reconcile the semantics of the data.

1.2 Thesis Objectives

Parsons and Wand (2002) suggested a theoretical foundation for property reconciliation based on the notion of Property Precedence (see more in Chapter 3), and described a theoretical approach to property-based reconciliation. This work is one of the newest results in the line of research on the use of Bunge's Ontology (Wand and Weber, 1989, 1990, 1993, 1995; Wand et al., 1999; Parsons and Wand, 2000). Since this theoretical approach only concerns properties and precedence, it cannot be applied on the concepts of classes. In order to utilize this approach

effectively, we need a Property-Based view of data rather than the current paradigm – the Class-Based Model. In other words, we need a model that does not use the concept of class. Such a conceptual model is suggested by Parsons and Wand (2000), but has not yet been defined so it can support practical semantics reconciliation.[1] We term it the Property-Based Model, which represents things directly using properties before classification. The objectives of this thesis are:

- To propose and develop a practical Property-Based Approach to integrating information from multiple sources based on properties and their precedence in the following three steps:

    a. To develop and formalize the basic part of the Property-Based Model that is sufficient to support the Property-Based Approach;

    b. To formalize the theoretical part of the Property-Based Approach for semantics reconciliation;

    c. To introduce a general integration procedure by using the Property-Based Approach.

1.3 Thesis Outline

This thesis is organised into eight chapters.

Chapter 2 provides an overview of some main research topics and related research in the field of Information Integration and discussion of conceptual modelling. We also cover the basic assumptions of this thesis during the discussions.

---

[1] Rubin E. (2002) defined a data structure based on the theory of property and precedences. However, he did not address any issues of semantic reconciliation.

Chapter 3 presents the BWW-Ontology and its implications. It first discusses the postulates and the definitions of some critical constructs in the BWW-Ontology, including the core concept of this thesis, Property Precedence. Then it conducts a discussion to clarify some concepts in the field of databases in terms of the BWW-Ontology.

In Chapter 4, we develop and formalize the basic part of the Property-Based Model based on the BWW-Ontology and its theories. In addition, we formalize the definition of the semantics of property by the use of Property Precedence.

In Chapter 5, based on the Property-Based Model, we develop the Property-Based Approach to integrating information from multiple sources. We formalize the notations to express precedence and inference processes. Then we propose a method to express the semantics of properties, which is the Property & Precedence Schema (PPS). Finally, we formalize and define Instance Functions to facilitate the process of information integration.

In Chapter 6, we discuss three practical issues related to the Property-Based Approach. The first issue is to build a Property-Based view on the Class-Based Model so that we can transform the data in the Class-Based Model to the data in the Property-Based Model. The next two issues are to implement the Property & Precedence Schema (PPS) and to implement Instance Functions by XML.

In Chapter 7, we introduce the general integration procedure by using the Property-Based Approach, which is illustrated by a case study.

Finally, Chapter 8 concludes the thesis by reviewing the contributions, acknowledging the limitations, and suggesting some future research directions.

Chapter Two

CONTEXT AND RELATED WORKS

Before we present the Property-Based approach, we would like to provide an overview of the context and the related research works of Information Integration. During the overview, some relevant research works are discussed briefly in terms of conceptual modelling, thus we can see how conceptual modelling is relevant and important in the field of Information Integration.

First, the main obstacle in Information Integration is the Semantic Heterogeneity of the Schemas of multiple data sources, which refers to differences or similarities in the meaning of local data, or in other words, the semantic of meta-data in the local schemas (Hakimpour and Geppert, 2001). How to solve this kind of heterogeneity becomes the initiation of Information Integration.

To solve Semantic Heterogeneity is to achieve Semantic Interoperability, which is also called Semantic Integration and Semantic Reconciliation. Many methods have been proposed based on the use of ontologies. Ontologies are supposed to provide a uniform viewpoint on multiple data sources. Since the word "ontology" has a very different meaning from what we use in this thesis, we will clarify the difference between them first.

Next, we discuss two general terms: Knowledge Representation and Knowledge Discovery. We discuss the relationship between them and Information Integration, and explain Information Integration in terms of Knowledge Representation and Knowledge Discovery.

Finally, we examine the field of Conceptual Modelling and its relationship with (Philosophy) Ontology. We discuss the Class-Based Model, which includes two well-known models in

Information Systems: the Entity-Relation Model and Object-Oriented Model, and its potential problems.

2.1 Heterogeneity and Multiple Schemas

Information Integration, or Data Integration, refers to combining data in such a way that a homogeneous and uniform view is presented to users. The obstacles for computers to integrate information from multiple sources automatically are due to the heterogeneities of information. We can distinguish two kinds of heterogeneity: *Data Heterogeneity* and *Semantic Heterogeneity* (Hakimpour and Geppert, 2001). *Data Heterogeneity* refers to differences among local definitions, such as attribute types, format, or precision. For example, two data sources may use different date formats. These differences can be easily resolved (Hakimpour and Geppert, 2001).

Semantic Heterogeneity refers to differences or similarities in the meaning of local data, or in other words, the semantic meaning of meta-data in the local schemas (Hakimpour and Geppert, 2001). This is considered the difficult issue in the field of Information Integration (Parsons and Wand, 2000).

Schemas are definitions that specify the meaning and structure of data and are the result of a database design phase. According to Hakimpour and Geppert (2001), in data integration, each local database provides a description of the data and is prepared to export the local schema. The aim of the integration process is the development of a global schema, which integrates and subsumes the local schema in such a way that (global) users are provided with a uniform and correct view of the global database (Hakimpour and Geppert, 2001).

One of the solutions to this problem is to build an explicit definition of terms used in schema definitions. Researchers have applied *formal ontologies* (Guarino, 1998) to build such an explicit and formal definition of semantics of the terms as a potential solution of Semantic Heterogeneity. They defined some meta-properties, or properties of properties, to deal with the problem of Identity and Subsumption (Guarino and Welty, 2000, 2001). For example, a property can be an essential property if and only if it necessarily holds for an instance at every possible time in every possible world. Examples of essential properties for a human being would be PERSON and HAVING A BRAIN. One inherent problem of this approach is that a property that is essential within some domains may not essential any more within other domains.

2.2 Semantic Interoperability and Information Systems (IS) Ontology

Such an explicit and formal definition of semantics of the terms in a schema is also referred to as "ontology" in the field of Information Systems. Here, ontology has a distinct meaning from the ontology as a branch of philosophy, which is a branch of metaphysics that deals with the nature of being. To distinguish them, we sometimes refer to them, respectively as Information Systems (IS) ontology and Philosophy Ontology (Zuniga, 2001).

Information System ontology can be defined as an explicit specification of a conceptualization (Gruber, 1995). Philosophy Ontology studies "*the generic (non-specific) traits of every mode of being and becoming, as well as the peculiar features of the major genera of existents*" (Bunge, 1977, p. 5). IS ontology provides us with a set of terminologies, such as a dictionary, that we use to describe our world, while Philosophy Ontology provides us with a systematic way to view the world, such as a pair of glasses. An example of this metaphor is that a colour-blind person has a different view from a normal person -- they have different Philosophy Ontology. Both of them may talk about black,

white or even red -- they have the same terminologies. However, they must have different meaning of black, white and red.

From the above example, we can know that we must have the same pair of glasses before we can understand each other. We need both to describe the world, and the Philosophy Ontology goes before IS ontologies and determine the general structure of IS ontologies. In other words, each IS ontology has an implicit Philosophy Ontology behind it.

There are some discussions on the topic of these two kinds of ontology (Guarino, 1997; Zuniga, 2001). Guarino (1997) also defines an IS ontology as an explicit, partial account of a conceptualization, which is a formal language designed to represent a particular domain of knowledge. In the current field of Semantic Interoperability, ontology usually refers to Information Systems ontology.

Since IS ontologies are supposed to provide an integrated view of domain of interests, Semantic Interoperability can be achieved by the use of IS ontologies. There have been already some research efforts on an ontology-based approach to achieving Semantic Interoperability (Hakimpour and Geppert, 2001; Stuckenschmidt and Wache, 2000), so Semantic Interoperability is referred to as Ontology Interoperability sometimes. Please notice that in the current field of Information Systems, Ontology Interoperability is part of Semantic Interoperability because the former deals with the semantics of data (including meta-data), while the latter focuses on the semantics of meta-data. Since the current paradigm to deal with meta-data is to use ontology and, this thesis targets also meta-data (properties), we treat Semantic Interoperability and Ontology Interoperability as the same in this thesis.

One of the major methods to retrieve wanted information from local data sources is to run queries against them. Therefore, Semantic Interoperability includes reconciling and integrating the meaning of queries and their results. There are mainly two trends for using ontologies in resolving Semantic Heterogeneity in retrieving data from local data sources (Hakimpour and Geppert, 2001): translating queries by ontologies and generating global schema by ontologies.

Semantic Interoperability is not limited to data retrieving by queries. There are some other facets, such as interoperability between multiple applications, communications between two computers or intelligent agents, etc. Therefore, much research has been done on developing a general approach to achieving Semantic Interoperability with ontologies. Several methodologies have been developed, two important ones of which are OntoClean (Guarino and Welty, 2002) and ONIONS (ONtological Integration Of Naïve Sources) (Gangemi et al., 1999).

2.2.1 Translating Queries by Ontologies

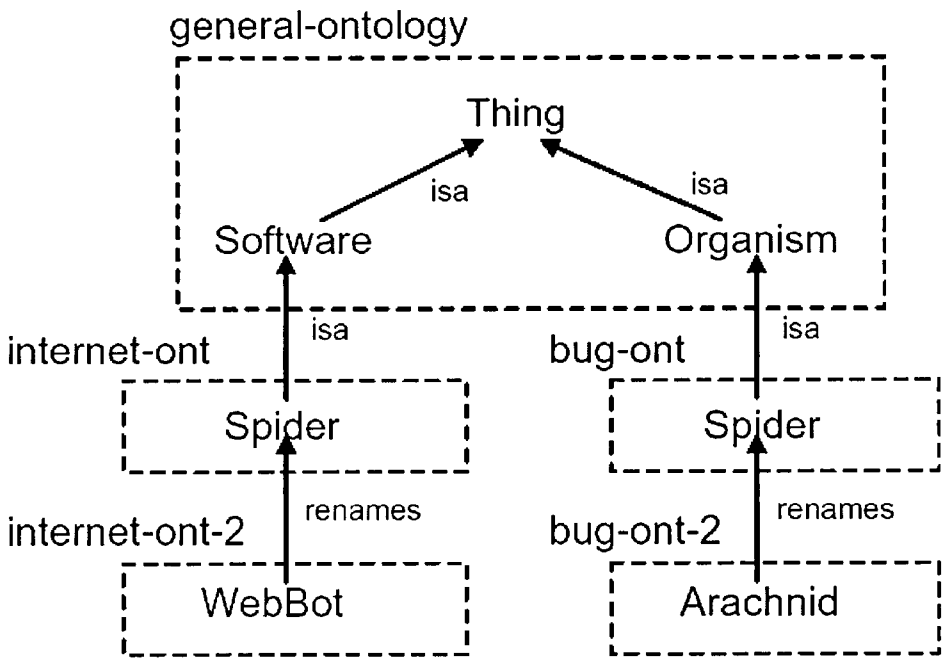
One uses ontologies for translating queries, or their results (as in SHOE, On2Broker or OBSERVER). This approach is suitable whenever schemas are subject to frequent changes, when many data sources are involved, or the number of involved data sources changes frequently (such as data sources on the Internet) (Hakimpour and Geppert, 2001).

There is a paper presented by Heflin and Hendler (2000) that discusses Ontological Interoperability by using SHOE, which is an ontology-based knowledge representation language designed for the Semantic Web. They claim (Heflin and Hendler, 2000) that SHOE uses knowledge-oriented elements, and associates meaning with content by making each web page commit to one or more ontologies. In SHOE ontologies, people can discover implicit knowledge

with taxonomies and inference rules, and thus achieve Semantic Interoperability through the sharing and reuse of these ontologies (Heflin and Hendler, 2000).

Heflin and Hendler (2000) claim that interoperability in SHOE is achieved mainly by the use of the ontology extension and renaming features, that is, two categories are similar to the extent that they share the same super-categories, and thus, as a result, ontologies are interoperable to the extent that they share the same ancestor ontologies.

Figure 2-1: Ontology Interoperability in SHOE



(Source: Heflin and Hendler, 2000)

They (Heflin and Hendler, 2000) provided the following example (in Figure 2.1). The term *Spider* means different things in "internet-ont" (here "ont" stands for "ontology")and "bug-ont" because the categories have different ancestors, whereas the term *WebBot* in "internet-ont2" means the

same thing as *Spider* in "internet-ont" because a rename feature indicates that it is an alias of the term. Interoperability is achieved in this manner through the correct construction of ontologies, although the nature of distributed environments may make ontologies not fully interoperable (Heflin and Hendler, 2000).

Hakimpour and Geppert (2001) points out three drawbacks of this approach: 1) one is the high processing cost because ontologies must be processed to derive required mappings for every query; 2) due to the need for immediate action, it is impossible to bring in human supervision to validate the results; and 3) lack of human supervision makes this approach less reliable.

2.2.2 Global Schema Generated by Ontologies

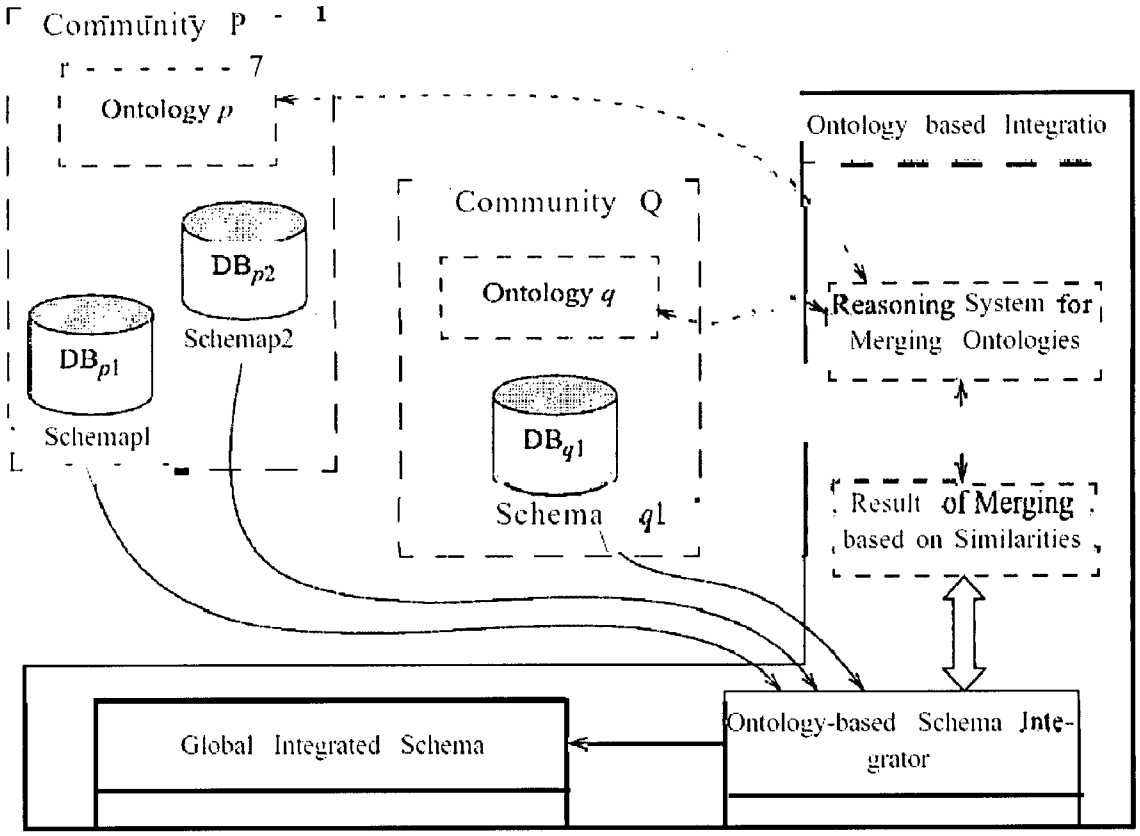
The second trend uses ontologies for the generation of global schemas. It is suitable whenever the schemas are not subject to frequent changes, as claimed by Hakimpour and Geppert (2001).

In this approach (Hakimpour and Geppert, 2001), database schemas commit to the ontology of a community, which is done by relating every term in the schema definitions to a definition in the ontology of the community. To demonstrate the general process of integration of ontologies, they (Hakimpour and Geppert, 2001) provide Figure 2-2, in which two databases DB_{p1} and DB_{p2} commit to an ontology p by referring to the terms defined in the ontology p. Such relation can be established either by hard links or by using the same terms as they are defined in the ontology.

Hakimpour and Geppert (2001) present an approach that uses formal ontologies to derive (global) schemas. In their approach (Hakimpour and Geppert, 2001), a global schema will be generated by the Schema Integrator that merges ontologies, which not only suggest a global schema, but also try to find all the possible meaningful mappings between the generated global schema and the

component schemas. They (Hakimpour and Geppert, 2001) claim that merging ontologies is based on finding similarities or differences between intensional definitions.

Figure 2-2: Global Schema Generation by Integration of Ontologies



(Source: Hakimpour and Geppert, 2001)

To achieve the merge of ontologies, they established similarity relations between terms defined in two ontologies and identified four levels of similarities between two coherent intensional definitions: 1) Disjoint definitions, 2) Overlapping definitions, 3) Specialized definitions, and 4) Equal definitions (Hakimpour and Geppert, 2001). However, their approach is not general enough because similarities or differences are relative to the domain. For example, they say that narrow-

street and highway were two disjoint definitions (Hakimpour and Geppert, 2001). However, from a higher point of view, they are all roads.

They also provided four rules of how to merge ontologies based on these levels (Hakimpour and Geppert, 2001).

1) If two definitions are equal, the result of merging is a unique intensional definition, which is referred to by both original terms, e.g. synonym terms such as "Person" and "Resident."

2) If an intentional definition "specializes" another definition, then the sub-concept or sub-relation similarity will be explicitly established between them, e.g., "Student" and "Person."

3) If a definition overlaps with another, then an additional new concept or relation will be declared as the conjunction of the two intentional definitions, e.g., concepts "Staff" and "Student" or "Lecturer" and "Graduate Student."

4) Nothing will be done about disjoint definitions them in this phase of merging.

The above rules are used in derivation of global classes and global class-attributes (attributes of instances in the classes). However, class-attributes are not general properties because they are confined within the definitions of classes. They said that if attributes in two classes (e.g. "paid by" in "resident" and "earns" in "staff") are referring to the same relation, one attribute will appear in the global class definition ("salary" in "person") representing both attributes. However, they did not address the situation of where there is no such global class or where we do not want to create such a global class.

2.2.3 OntoClean

They claim that "*OntoClean consists of principles for building and using upper ontologies for core and domain ontology analysis, revision, and development,*" and that "*the OntoClean methodology is based on highly general ontological notions drawn from philosophical ontology, especially from what is now called Analytic Metaphysics.*" Guarino and Welty (2002) believe that it is general enough to be used in any ontological effort, independently of a particular domain.

Guarino and Welty (2002) also claim that the OntoClean methodology uses these formal and ontological notions to define a set of meta-properties, which, in turn, are used to characterize relevant aspects of the intended meaning of the properties, classes and relations that make up an ontology. In addition, the meta-properties (property of property) impose several constraints on the taxonomic structure of an ontology (Guarino and Welty, 2002).

They (Guarino and Welty, 2000, 2001, 2002) elaborated four notions: essence, rigidity, identity and unity.

A property of an entity is essential to that entity if it must hold for it. For example, consider the property of being hard. We may say that it is an essential property of hammers. A special form of essence is rigidity: a property is rigid if it is essential to all its instances. For example, being a person is usually conceptualized as rigid. Based on rigidity, one of three meta-properties, rigid, non-rigid or anti-rigid, should be labelled to every property in an ontology.

In general, identity refers to the problem of being able to recognize individual entities in the world as being the same (or different). Unity refers to being able to recognize all the parts that form an individual entity. Unity can tell us something about the intended meaning of properties or classes

based on whether their instances are wholes. The corresponding meta-properties, including Identity Condition and Unity Condition, has been formalized (Guarino, 2001) based on these two notions. However, these meta-properties are too complicated to be elaborated here.

Guarino and Welty (2002) further claim that the OntoClean methodology, which is based on the above-mentioned formal notions, provides a formal, consistent and straightforward way to explain some of the most common misunderstandings in conceptual modeling regarding the taxonomic or subsumption relation. They (Guarino and Welty, 2002) suggest that one frequent mistake is confusing subsumption with instantiation (e.g. "John" is an instance of "Human," whereas the class "Humans" is subsumed by the class "Mammals"). They (Guarino and Welty, 2002) also suggest another frequent feature of bad taxonomies, which is the systematic confusion between "part_of" and "is_a" (subsumption) relationships (e.g. "engine" part_of "car"; "car" is_a "vehicle").

As we can see, their approach is focused on taxonomies, or classes, that is, they try to put instances into proper classes of taxonomy. We will point out later (Section 2.4.1) that this approach suffers some inherent problems. Furthermore, there is a debate on where there are properties of property. For example, in Bunge's Ontology (1977, p. 98), he denies the existence of meta-properties. However, this discussion is beyond the scope of this thesis.

2.2.4 ONIONS

ONIONS (ONtological Integration Of Naïve Sources) methodology for ontology integration has been developed since the early 1990s to account for the problem of conceptual heterogeneity, or Semantic Heterogeneity (Gangemi et al., 1999).

As Gangemi et al. (1999) state, ONIONS (ONtological Integration Of Naïve Sources) is a methodology for conceptual analysis and ontological integration, which exploits: 1) a set of formalisms; 2) a set of computational tools that implement and support the use of the formalisms; and 3) a set of generic ontologies, taken from the literature in either formal or informal status and translated or adapted to our formalisms.

Gangemi et al. (1999) call their approach *Principled Conceptual Integration* and claim that ontology integration is, generally speaking, the construction of an ontology that formally specifies the union of the vocabularies of two other ontologies. They identify (Gangemi, 1999) five relationships between two ontologies: 1) alternative ontologies, 2) truly overlapping ontologies, 3) equivalent ontologies with vocabulary mismatches, 4) Overlapping ontologies with disjoint domains, and 5) Homonymically overlapping ontologies.

Gangemi et al. (1999) also identify three levels of Integration: medium, alignment and unification. Based on different levels of integration, ONION uses the above five relationships between two ontologies to develop some rules of integration, and thus achieve Ontology Interoperability. We can see this approach is similar to the approach by using Hakimpour and Geppert's four levels of similarity (2001), discussed in section 2.2.2. Therefore, we do not elaborate it here.

2.3 Knowledge Representation and Knowledge Discovery

Our knowledge about the world, or the domain of interest, includes numerous observed *facts* accumulated throughout the history of human being, a multitude of empirical *laws* and *theories* summarized from these facts. An information system is a representation of some perceived reality, so we can claim that an Information System is a representation of our knowledge about the world, or the domain of interest, i.e. a knowledge representation indeed.

In Information Systems, a knowledge representation can be anything that possesses some information, such as a database, a website, a web page, an XML file and even a computer program (software). Most of these forms are about facts, especially in databases and software. This situation may be changed in the future, since one of the intentions of the next generation of the World Wide Web, the Semantic Web, is to represent empirical laws and theories (Berners-Lee and Fischetti, 1999).

From one knowledge representation or multiple knowledge representations, we can infer or induce new knowledge. The process is known as Knowledge Discovery. The following is a formalized explanation of Knowledge Discovery in Information Systems (Frawley et al., 1992, p. 58):

> *Given a set of facts (data) F, a language L, and some measure of certainty C, we define a pattern as a statement S in L that describes relationships among a subset FS of F with a certainty c, such that S is simpler (in some sense) than the enumeration of all facts in FS. A pattern that is interesting (according to a user-imposed interest measure) and certain enough (again according to the user's criteria) is called knowledge. The output of a program that monitors the set of facts in a database and produces patterns in this sense is discovered knowledge.*

A Pattern mentioned in the above quotation can be mapped into an empirical law or a theory, while data in a database can be mapped into a set of facts. While data sources can be viewed as Knowledge Representation, Information Integration can also be considered as a kind of Knowledge Discovery, which involves multiple data sources. These data sources usually use different data schemas. The result or output of Information Integration is discovered knowledge.

This explanation only mentions the discovery of empirical laws and theories from facts, but it fails to cover that we can also discover empirical laws and theories from empirical laws and theories themselves, and fails to cover that we can predicate facts from theories and empirical laws.

Therefore, this explanation of Knowledge Discovery is limited and incomplete. Fortunately, the vision of the Semantic Web includes the ambition of realizing all of them (Berners-Lee and Fischetti, 1999).

It seems reasonable that before we can integrate information or knowledge, we must have an integrated and uniform model for Knowledge Representation.

2.3.1 Ontological Model of Knowledge

Two of the roles that Knowledge Representation plays (Davis et al., 1993) are: 1) a knowledge representation is most fundamentally a surrogate, or a substitute for the thing itself; and 2) a knowledge representation is a set of ontological commitments. We make different commitments by selecting one or another ontology, which can produce a sharply different view of the reality. Here, a set of ontological commitments is similar to an IS ontology. As we argued (Section 2.2), each IS ontology, or a set of ontological commitments, has an implicit Philosophy Ontology behind it. Philosophy Ontology provides us a general structure of IS ontologies.

Therefore, one of premises of Information Integration is (Please note that the word "ontology" means Philosophy Ontology):

> *The information from multiple data sources is expressed in the same ontological model, or, has been transferred into the same ontological model.*

In order to make an integrated and uniform ontological model of knowledge and to facilitate the integration of information, we have first to make a choice among different Philosophy Ontologies and use this specific ontology as the fundamental model of Information Integration. This leads to the following discussion about Conceptual Modelling.

2.4 Conceptual Modelling and Philosophy Ontology

Conceptual modeling focuses on capturing and representing certain aspects of human perceptions of the real world so that these aspects can be incorporated into an Information System (Wand et al., 1999). The process of conceptual modelling is the way in which we observe the reality, address what happened in the world and predict what would happen. Ontology, as the branch of philosophy that deals with modelling reality, could play an important role in formalizing Information Systems concepts, and should be used to guide Conceptual Modelling.

2.4.1 The Class-Based Model and its Ontology

In the field of Information Systems, there are many models already, such as, Entity-Relation Model (CHEN, 1976); Object-Oriented Model (Rumbaugh et al., 1991; Coad and Yourdon, 1991); Object Role Modelling (Halpin, 2001), etc., behind which we can identity an underlying and implicit ontology. The ontology is based on classification, in which things are identified by their class. For example, we can say "this is a cat." The thing is identified as cat first, then one can infer other properties it may have. This approach is intuitive to human users because we use it in our daily life and classification is essential for human survival and adaptation (Lakoff, 1987).

Some researchers (Guarino, 1998) have tried to formalize this implicit ontology and the result is now called *Formal Ontology*, this underlying ontology is still far from a formal and systematic one that can be used and accepted in terms of its comprehensiveness. A research group of Ontology, Conceptual Modelling, and Knowledge Engineering at LADSEB-CNR (Institute for Systems Science and Biomedical Engineering of the Italian National Research Council) has achieved some progress in this direction (Guarino and Welty, 2000; Welty and Guarino, 2001). Since the

underlying ontology is based on classes of things. We term the models suggested by the Ontology the Class-Based Model.

In the Class-Based Model, everything must first belong to a class, and only through the definition of the class, a thing can have its own attributes (or properties). This feature has been termed the assumption of inherent classification (Parsons and Wand, 2000).

The Class-Based model has at least two implementation advantages. First, since the data used to represent this model is highly structured because everything that belongs to one class has the same set of attributes, it is easy to implement and is very efficient. This was especially important when the computational capacity was low and Information Technology was less advanced one decade ago. Second, the Class-Based Model is also intuitive to human users and enjoys cognitive advantages, such as cognitive efficiency and inference (Parsons and Wand, 1997).

However, the Class-Based Approach has some disadvantages. As Parsons and Wand (2000, pp. 231-235) point out, "*the extent and complexity of problems in schema integration, schema evolution, and interoperability are, to a large degree, consequences of inherent classification.*" They enumerated two kinds of problems: Schema Design Problems: 1) the multiple classifications problem; 2) the view integration problem; 3) the schema evolution problem and 4) the interoperability problem; and Database Operation Problems: 1) handling exceptional instances; 2) reclassifying instances; 3) adding and removing instances; 4) removing a class and 5) redefining a class.

Besides the above problems, the Class-Based Model causes several specific problems when we want to achieve Semantic Interoperability. The first problem is that the semantics of properties are restricted within the semantics of classes, which results in losing the general meaning of a property.

For example, birds can fly and aircrafts too. However, we define two kinds of flying in a Class-Based model because of the two classes, birds and aircrafts. In fact, flying is more general property than birds or aircrafts, in other words, being able to fly precedes being birds or aircrafts.

Due to the first problem, the approaches based on the Class-Based model have to deal with classes or categories first and then properties, such as the four approaches discussed in Section 2.2. They first put instances into classes or categories, compare categories to see if they have the similar meaning, and then determine the rules of integration. Therefore, two instances in two categories that have been considered as "distinct" have no chance to be integrated. To demonstrate, in the above example, it is reasonable that we treat birds and aircrafts as distinct categories because the former is animal and the latter is an artificial object. However, from another point of view, both of them are flying things. This practice significantly limits the ability and flexibility of these approaches. Furthermore, this leads to the third problem.

We can observe that the semantics of classes are dependent on the properties that we use to construct classes, or in other words, the similarity of two classes is relative to other properties or classes. In the above example, birds and aircrafts have different semantics relative to their natures, but have the same semantics relative to being able to fly. Because of these three reasons, the Class-Based Model seems to be becoming the obstacle of Semantic Interoperability.

2.4.2 The BWW-Ontology and the Property-Based Model

A specific ontology, the BWW-ontology, has been proposed and used in Conceptual Modelling in Information Systems for more than a decade (Wand, 1989; Wand and Weber, 1989, 1990a, 1990b, 1993, 1995; Wand et al., 1999). This ontology has been adapted from Mario Bunge's Ontology (1977; 1979). The reasons for choosing this particular ontology are (Wand and Weber, 1990a;

Evermann and Wand, 2002) listed in the following:

1) It is oriented towards systems and has been applied to different types of systems.

2) It is well formalized in terms of set theory.

3) It has not been developed specifically for use in Information Systems analysis and design.

4) It is comprehensive and part of a larger work including semantic.

5) It references extensively other philosophical work.

This ontology leads us to a simple conceptual model, the Property-Based Model, whose basic part we will develop and formalize in Chapter 4. The fundamental postulate of the ontology is that the world is made of *things* that possess *properties*. Therefore, in this model there are only two basic modelling primitives, *thing* and *property*. Of course, there are other constructs, such as *state, event, law, class, etc.* However, these constructs are defined by properties. This model can be used as the beginning to represent the world or the domain of interest in Information Systems. More details about the BWW-ontology are in Chapter 3.

2.5 Summary

In this chapter, we provided an overview of Information Integration. We selected some main research topics, including Semantic Heterogeneity, Semantic Interoperability, Information Systems Ontology, Knowledge Representation, Knowledge Discovery, Philosophy Ontology and Conceptual Modelling. Furthermore, we pointed out that the current main approaches of modelling are based on a similar conceptual model, the Class-Based Model and the problems inherited in this model. Finally, we suggested the use of the BWW-Ontology.

Chapter Three

THE BWW-ONTOLOGY AND ITS IMPLICATIONS

In this chapter, we discuss in detail the theoretical foundation, the BWW-Ontology, on which we build our approach to integrating information. First, we introduce in a formalized manner some fundamental constructs of the BWW-ontology, which give us an ontological view and model of the world.

In the second part, we examine thoroughly the concept of *Precedence* that is the core concept of this thesis and its implications, including class definition, property value and property aggregation. Therefore, we can get a deep and clear understanding of these concepts in terms of the BWW-Ontology.

Finally, we try to uncover the ontological meaning of queries and Information Integration in the context of the BWW-ontology.

3.1 Some Fundamental Constructs of the BWW-Ontology

The BWW-Ontology uses an ontological framework developed by Bunge (1977; 1979), whose model articulates a set of high-level, abstract constructs that are intended to be a means of representing all real-world phenomena. Therefore, to start our analysis, we select the specific ontological constructs that we require in this thesis, which are *thing, property, attribute, law, class and composition of things,* and provide an overview of these constructs in this section. In the next three chapters, we refer to the BWW-ontological model when we use the term "ontology."

In this thesis, we follow the formalization in two works (Wand et al., 1999; Parsons and Wand, 2000), in which some more discussions and examples can be found. In the following discussion in this section, the postulates, definitions and principles are extracted from these two works (Wand et al., 1999; Parsons and Wand, 2000) if not otherwise stated.

The Model of Things and Properties

Postulate 3.1: The world is made of *things* that possess *properties.*

Wand et al. (1999) claim that Ontology distinguishes between concrete things, called *substantial individuals* or *entities*, and conceptual things, e.g. mathematical concepts, such as sets and functions. They (Wand et al., 1999) first assume that any domain can be described in terms of concrete things and the linkages that exist among them, so we use the word "thing" to refer only to substantial individuals or concrete things in Information Systems. They (Wand et al., 1999) further assume that that domain modeling is based upon someone's view of existing or possible reality and thus the notion of a concrete thing applies to anything *perceived* as a *specific object* by someone, whether it exists in physical reality or only in someone's mind.

An ontological principle connects the existence of a thing with its properties:

Principle 3.1: There are no things without *properties*, and that *properties* are always attached to *things.*

Principle 3.2: No two *things* possess exactly the same (set of) *properties.*

A distinction is made between *intrinsic* properties — those that depend on one thing only, and *mutual* (or relational) properties — those that depend on two or more things (Wand et al., 1999).

Postulate 3.2: Every property in general can be represented by an attribute (propositional function); $A: T_1 \times \ldots \times T_n \times V_1 \times \ldots \times V_m \rightarrow$ Statement including A; and every specific property can be represented as an attribute of the form; $A(t_1, \ldots, t_n, v_1, \ldots, v_m)$ where $t_i \in T_i$, $v_j \in V_j$, T_k(k=*1*, . . . , *n*) represents a set of things and V_j(j=*1*, . . . , *m*) represents a set of *values.*

For example, the statement that a student studying in a university at a date is a property in general, which can be represented as STUDY: $T_1 \times T_2 \times V_1 \rightarrow$ P, where T_1 is a set of people, T_2 is a set of university, V_1 is a set of dates, and P is the statement. A specific property will be a statement that James studies in UBC at 2001-9-1, which can be represented as STUDY(James, UBC, 2001-9-1), where James $\in T_1$, UBC $\in T_2$, 2001-9-1 $\in V_1$.

Please note that an attribute is a predicate, which has only two values, *True* and *False.* Thus, in the above example, if STUDY(James, UBC, 2001-9-1) is true, James possesses the property. Otherwise, James fails to possess it.

An intrinsic property's attribute involves only one thing, that is n=1 in the function, while a mutual property involves more than one thing, or n>1. For example, the height of a person is an intrinsic property while study-in-a-university is mutual property because it at least involves a person and a university.

A property can be transformed between intrinsic and mutual property depending on the details that the viewers want to know. For example, the price of a stock is an intrinsic property with no doubt, but if the stock is traded on several security markets, the price becomes a mutual property between the markets and the stock. In fact, in the former case, what we do is just to disregard the market so we can treat the price as an intrinsic property.

An attribute can have multiple variables. For example, a price has two variables for a price in fact, currency and amount. In implementation, we always treat them as one variable by using various data types. This practice seems resolving the problem very well. Although it causes some problems in integration, these problems are included in data heterogeneity, which is considered as being able to resolve easily.

Definition 3.1: The *scope* of a property is the set of things that possess the property. That is, if Θ is the set of all things and P is the set of all properties, the scope function **S** is the mapping $\mathbf{S}: P \rightarrow 2^{\Theta}$.

Definition 3.2: A set of things T is a *class* if and only if there exists a finite set of properties P that are possessed by all members of T. In other words, each member of T is in the intersection of all scopes of all properties in P, that is, $T \in \mathbf{S}(x_1) \cap \ldots \cap \mathbf{S}(x_n)$.

Clearly, classes are defined by things and properties. The notion of a thing is fundamental and precedes any notion of classes. Things and their properties exist independently of any classification. Therefore, two representation principles and their corollaries are suggested and phrased by Parsons and Wand (2000).

Representation Principle 3.1: The world is viewed as made of things that possess properties. (Note that this representation principle reflects the Postulate 3.1.)

Representation Principle 3.2: Classes are abstractions created by humans in order to describe useful similarities among things.

Corollary 3.1: Recognizing the existence of things should precede classifying them.

Corollary 3.2: There is no single "correct" set of classes to model a given domain of instances and properties. The particular choice of classes (a view) depends on the application (Parsons and Wand, 1997).

We are interested in the relationships between properties, which are called laws in Bunge's Ontology. We do not introduce the formal definition of law in this thesis because it may involve another complicated concept, *Functional Schema*, which is not quite relevant to the topic of the thesis. Nevertheless, we introduce a specific law, Precedence, which could be the foundation of all kinds of laws.

Definition 3.3: Let P and Q be two properties. Q will be said to *precede* P if and only if the set of things possessing P (the *scope* of P) is a subset of the set of things possessing Q.

For example, a mammal must be an animal. Therefore, the property of being a mammal is preceded by the property of being an animal. The concept of Precedence is the core of this thesis and more discussion is in the section 3.3.

Postulate 3.3: Two things may associate to form another thing.

The notion of association can be formalized by the algebraic concept of concatenation (Bunge, 1977, p.27). The notion helps us to obtain "*the simplest, the most basic and most useful theory in scientific metaphysics*," called *Association Theory* (Bunge, 1977, p.27). Please notice that in Postulate 3.3, we are talking about bare individuals (things) and bare association, i.e. neither the kind of individual nor the manner of association are specified (Bunge, 1977, p.26). The association can be understand as the ability of bare individuals to form composite entities.

Bunge also introduces another similar ontological theory, *Assembly Theory*, in which there are two specific concepts of association, juxtaposition and superposition (Bunge, 1977, p.39). Roughly speaking, whereas a superposition is a thing composed of the common part of two things, a juxtaposition is a composite thing composed additively of two things. The notion of juxtaposition is the same as the notion of association in Association Theory (Bunge, 1977, p.45). Bunge (1977, p. 113) defined the concept of the juxtaposition of things.

Definition 3.4: Let X and Y be two things and $X \neq Y$, then the juxtaposition of X and Y is the third thing (composite).

Definition 3.5: A property of a composite thing is *inherited* if and only if it is a property of any of its components; otherwise, it is *emergent.*

The third thing composed of two things X and Y possess at least one emergent property of being composed of X and Y. Therefore, we have the following postulate.

Postulate 3.4: Every composite thing possesses emergent properties.

Because of this postulate, Bunge claims that the association of bare individuals (things) is then a beginning of complexity and thus a step towards realism (Bunge, 1977, p.26).

Things can be viewed as simple things and composite things. Which one will be used depends on the details that the viewers want to know about the world. For example, we always refer to a stock as a simple item that has price, shares amount, P/E, etc. However, if every specific share has a share number and we want to know who owns which share, a stock becomes the composition of all the shares that the stock represents. Furthermore, a stock can also be viewed as the class of all

the shares since all the stock-shares, represented by the same stock, belong to the same company. Therefore, the set of instances in a class can be viewed as a thing if there are some emergent properties we want to know. In this example, price, shares amount and P/E are emergent properties of a stock, instead of a share.

We want to give another example to elaborate this thoroughly. Suppose in a database of a university, there is the data of its students and we want to know the average age of its female graduate students, what is the query about? We need to form a new class of female graduate students, add their ages and divide the sum by its number. Nevertheless, what possesses the property of the average age? (Please notice that here we use "property" to refer to concrete property.) Not female graduate students obviously, and not the new class since a class is a set and cannot have a property. (Certainly, a class can have a conceptual property, in which we are not interested for the current purpose.) It is the composite thing composed by all students in the class. We will give out a formal explanation of a query in Section 3.3.

3.2 Precedence and its Implications

As we have already known, Precedence, as one kind of law, deals with the subsumption relationship between two properties, in which one thing that possesses property A must also possesses B, i.e. B precedes A or A is preceded by B. For example, "having a student number in UBC" precedes "being a UBC student" if every student in UBC must have a student number. However, the latter does not precede the former in the situation that one will still possess his student number to access to his transcripts after he has graduated.

As Parsons and Wand (2002) point out, precedences enable us to view properties (in different sources) that seem different, to be considered "same" at a higher level, which facilitates Semantic

Interoperability. In order to infer new precedence from the known precedence, Parsons and Wand (2002) developed two lemmas, which establish some relationships between the observations that can be made about general properties and the observations that can be made about their manifestations.

Definition 3.6. A property G is termed *fully manifested* by a set of preceded properties S if and only if every thing that possesses G possesses at least one of the properties in S. (Parsons and Wand, 2002)

For example, the property of age is fully manifested by the set of properties from 1 year old to 150 years old if we assume that no one can live longer than 150 years.

Lemma 3.1: Let Si be a set of preceded properties of Gi (i=1, 2). If G2 is fully manifested by S2, and for each property in S2 there is a preceding property in S1, then G1 precedes G2. (Parsons and Wand, 2002)

For example, suppose that the property of age group (G1) is fully manifested by a set (S1) of defined age groups and the property of age (G2) is fully manifested by a set (S2) of ages. If for every age, we map at least one age group, then G1 precedes G2, that is, having an age must belong to an age group.

Lemma 3.2: Let Si be a set of preceded properties of Gi (i=1, 2). If G1 is fully manifested by S1, and G2 is preceded by G1, then each property in S2 is preceded by one or more properties in S1.

For example, suppose that participating in an academic program (G2) is preceded by being a student (G1). Suppose further that the manifestations of G1 are "undergraduate" and "graduate".

Lemma 2 enables us to infer that every manifestation of G1 is either "undergraduate" or "graduate" if we know the specific program in which a student participates.

Next, we will use the concepts of precedence and manifestation, and two lemmas to uncover the deep meaning of class definition and property value in the Class-Based Model. Thus, we can see that Precedence plays a pivotal role in the semantics of classes and properties and should be the key to achieving Semantics Interoperability. Later in the next chapter, we shall use precedence to define the semantics of properties in the Property-Based Model.

For the purpose of convenience, we select one representative implementation of the Class-Based Model, Relational DataBases (RDBs) as the subject of discussion. Since properties are represented by attributes in RDBs, we will treat properties and attributes as the same in the following discussion.

3.2.1 Class Definition

Please first recall Definition 3.2 about the definition of a class. We can see that classes are defined based on properties, not otherwise as we do in the Class-Based Model.

In RDBs, classes are represented by relations. For example, a relation called *Book* contains the information of things that are in the class *Book*. However, not every relation represents a class but a relation may represent a mutual property. For example, a relation called "order" that includes two foreign keys respectively from two relations *Customer* and *Book*. Classes, or entity types, are defined by some attributes possessed by these classes. For example, one possible definition of books is declared as book (ISBN, title, author, publisher, publish_year, price). There are two ways to interpret the word "book." One is using "book" as a type of a thing and the other is using "book"

to refer a composite property (We call it a *class-property* hereafter) that is commonly possessed by all books. If we use the second interpretation, we can see that the *class-property* of book is defined by a list of precedences between book and ISBN, title, author, publisher, publish_year, and price. For example, if the thing is a book, it must possess the property of author, that is, the *class-property* of book is preceded by the property of having an author. Therefore, at least we can say that the precedences related to one class-property are the main part of the definition of the class.

However, there are two potential problems in the practice in RDBs. First, there is no way to express that one class-property precedes one property. For example, if we assume that only books have ISBN, the *class-property* of book will precede the property of having an ISBN, that is, all things that have ISBN are books.

Second, sometimes, a class-property does not have to be preceded by the properties that are used in the declaration, as some instances in the class may not have some properties. For example, if we assume that not all the books have ISBN, book is not preceded by ISBN. However, in the Class-Based Model, even a book that does not have an ISBN will have an attribute of ISBN. This is one of the main shortcomings of fixed classification.

To solve the second problem, we have to use the value of Null. However, a value of Null has two meanings: inapplicable and unknown. In the example, the first means that the book does not have an ISBN. The second means that the book has an ISBN, but it is not available right now for some reason. This ambiguity causes many problems in the Class-Based Model. Wand et al. (1999) provide a detailed discussion on this.

We can use other properties and their relationships with a specific property to define it. These relationships are what we call laws in the BWW-ontology. Precedence between properties is one particular kind of such laws in the BWW-ontology.

Since listing the whole set is impossible theoretically, we have to identify the sub-sets of the entire set of laws, which are sufficient or acceptable. However, a complete discussion of this point is beyond this thesis. In this thesis, we shall only deal with the precedences in the property definition because it enables the properties to be viewed as having the same meaning on a higher level.

Another aspect of a property may be the value type and domain of the property. However, we will show that even this part is about precedence as well.

3.2.2 Property Value

For every attribute, we have to declare a data type for its value in RDBs. We need to know the real meaning of a property value. In the previous example, suppose there is a book the price of which is $50. The property of having a price of $50 is preceded by the property of having a price. In other words, the property of having a price has a manifestation of having a specific price of $50 in the instance. This is termed as *value manifestation* (Parsons and Wand, 2002), in which a generic property can be manifested by a specific value.

As Parsons and Wand (2002) pointed out, manifestation can be considered a *generalization of the notion of value* of a given (generic) property, or in other words, a preceded property can be viewed as a value of a preceding property. In the previous example, if we take book as a property, one set of possible value could be {novel, poem, drama and so on}. (This is done in RDBs by adding an attribute called Type.) Obviously, saying that a novel is also a book is semantically clearer than

saying that the book has an attribute of type, of which the value is novel. One practice that should be noticed is that a value called "others" is always put into the set of values in the similar situation of the example, which represents all the other manifestations except those already in the set of values. By doing so, the property can be fully manifested by the set of values.

Now, we can conclude the conditions that can make a set of properties be the value of another property: 1) the set of properties must be preceded by the property, 2) the property must be fully manifested by the set of properties, and 3) only one of the values manifested at a time in one thing.

Recalling Lemma 1 and Lemma 2, we can get the following two corollaries. Let **A**, **B** be two properties and let **X**, **Y** be their set of values, respectively,

Corollary 3.3: If each value in **Y** implies at least one value in **X**, **A** precedes **B**.

Corollary 3.4: If **A** precedes **B**, each value in **Y** implies one value or a set of values in **X**. (Note that we might not know which one.)

For example, let A and B be the properties of age group and age respectively, and let X be a set of defined age groups and Y be a set of ages. If for every age, define at least one age group, then A precedes B, that is, having an age implies belonging to an age group. If having an age implies belonging to an age group, an age implies one age group or multiple age groups if the scopes of age groups overlap each other.

These two corollaries are easier to use than the two lemmas. Corollary 3.3 enables us to find the relationships of precedence between properties. Corollary 4 tells us that the inferences exist between values of two properties.

It is possible that there are more than one set of properties that can be the values of another property. In practice, we can find that the same property may have different set of values from different data sources, and sometimes even within the same source. For example, we can have different definitions of age group. One database may have the values such as toddler, teenager, adult, and senior while another may have the values such as: less than twenty, twenties, thirties, forties, fifties and greater than sixty.

There are two solutions to the problem of values from multiple domains. One is to allow different value sets to co-exist. The result of this method is that there are different representations for the same property and we can just treat them as different properties. In the above example, we just have two properties, both called "age group" and having different set of values.

The other solution is to assign a Standard Set of Values to a property and map values from different domains to values in this Standard Set of Values by using some inference functions. The problem of this solution is that there is not necessarily a mapping between two value sets. Therefore, we have to make some compromise on the accuracy of inference. Since this problem is due to the deficiency of the original data, it is beyond the scope of Semantic Integration. In the example in the last paragraph, if the standard set of values of age group is the second one, but we have a value of 'adults' in the first set of values, we cannot infer it to a value in the Standard Set of Values.

Having such a Standard Set of Values for a property is also the key to solving another integration problem, which we term Property Aggregation.

3.2.3 Property Aggregation

Some properties may have multiple manifestations in one thing while some properties do not. For example, one can speak more than one language, but can only have one age. This problem is likely to occur in particular when we integrate information from different data sources because more than one manifestation of a property might exist. When we have two manifestations from two data sources, we have to know how to deal with them. For the first kind of properties in the above example, we can treat the manifestation as a set with two values. For the latter kind, we have to decide some kind of rules to tell us how to transmute two manifestations into one.

In the above example, if we learn from two data sources that one speaks English and Mandarin, we leave them separately. If we learn from one data source that one is 28, and from another that he is 29, how can we reconcile them? One possible solution, or a guess, is to set up some rules, such as to select the larger one under the assumption that the smaller one comes from the data source where information is more likely to be outdated. Although such a guess might be wrong, it is a trade-off that we have to make depending on the uses' demand.

There are four possible relationships between two manifestations. Let m_1 and m_2 be two manifestations of a property,

1) m_1 and m_2 come from the same domain, that is, they are the same type:

    a. m_1 and m_2 can co-exist. For example, English and Chinese are two manifestations of language skill from the same domain. A person can know both languages.

    b. only one manifestation is allowed. For example, 27 and 28 are two ages and come from the same domain. A person can only have one age at a time.

2) m_1 and m_2 come from different domains, that is, they are different types:

   a. m_1 and m_2 can co-exist. For example, driving and reading are two skills from two different domains and one person can drive and read.

   b. only one manifestation is allowed. For example, 'youth' and 'teenager' are two manifestations of age group from two different domains. We may only allow one manifestation for the interest of certainty.

For a specific property, there might be different rules in these four cases. For this reason, we might need a resolution process to apply suitable rules for various cases.

We term these rules Aggregation Rules. Please notice that no-rule is one specific kind of Aggregation Rule. No-rule means that the property can have multiple manifestations. For example, one can speak multiple languages. A property can have more than one aggregation rule for different situations. For example, the property "skill_level" may only allow one manifestation from the same domain because only one level is possible in one domain while it may allow multiple manifestations from different domains because one person can have a different level for a different skill in a different domain. Some simple aggregation rules are no-rule, the maximum, the minimum and the average. Of course, according to the users' demand, a property can have complicated rules.

One thing that needs to be emphasized is that property aggregation is different from the normal meaning of aggregation in the field of databases. The latter means the aggregation among multiple things while the former means multiple properties in one thing. We will elaborate upon this difference in the next section.

3.3 Query and Information Integration

In the previous section, we used the Property-Based Model and the theories of the BWW-Ontology to explain some concepts in the field of databases, including class definitions, property definitions and property values. In this section, we are going to use them to explain two processes: queries and integrations.

In databases, a query means a process to retrieve information from databases. However, it seems no clear theory exists to explain to what the retrieved information refers. In the words of BWW-Ontologies, which things possess the properties represented by the information? Information Integration can be considered the process of integrating the results of queries from multiple data sources. Therefore, we try to suggest a view of a query in the context of how we can integrate information in terms of the BWW-Ontology.

3.3.1 Query

According to the aims and the results of queries, we distinguish three basic types of queries:

1) For one thing, to retrieve information about this thing;

2) For a set of things:

    a. To make a new classification and retrieve information about things in the class;

    b. To retrieve information about composite things composed of things already represented in the data source.

Suppose that a university has a database that stores students' information and every student has his unique student number. A simple query to search for the contact information of a specific student is the first type of query. A query to search for the age information of all sophomores in the Commerce Department is the second type. This query actually makes a sub-class among all the students in the university and the students in the sub-class must have certain common properties.

A query to learn how many graduate students are female and are Canadian is the third one. Similar to the second query, this query makes a sub-class first. It does not return information of all the students in the sub-class, but looks for a composite thing that is composed by these students and computes out an emergent property about the composite thing, that is, the number of its components.

Normally, the third type of queries always involves an aggregation function. In the above example, the aggregation function is to count the number of its components. We term this *Composition Aggregation*. The differences between Property Aggregation and Composition Aggregation are:

1) The former happens in one thing, while the latter happens between a composite thing and its components;

2) The former does not generate a new generic property, while the latter generates an emergent generic property possessed by the composite thing; and

3) The former only happens to reconcile multiple manifestations when integrating information from multiple data sources, so there sometimes are no absolutely correct aggregation rules. The latter has a strict rule within the definition of the emergent property.

Besides the three basic types, a query can also be a combination of these types. For example, we may want to know the number of students from each country in the Computer Science Department. This query will be the combination of the second and third types. There are two classifications: department and country. By the second classification, we find some composite things and their emergent properties.

We can see that the second type is made of several queries of the first type plus one classification. The third type is made of the second type plus composition. Upon this analysis, we come up with the formal definition of query.

Definition 3.9: A query is to learn some information about things from data sources, re-classify them and learn some information of the composite things composed of these things.

3.3.2 Information Integration

From a certain point of view, Information Integration can also be regarded as one special kind of query, which targets on multiple data sources because it is also about things, re-classification and composition. However, due to multiple data sources and multiple data format, Information Integration involves its unique problems different from normal queries: 1) How to integrate information from multiple sources about the same things, and 2) How to classify things that come from multiple sources and have different sets of properties.

The problem of Identity

To address the first question, we have to solve first the problem of identity, that is, how we can know that two sets of information are actually about the same thing. In theory, there is no general way to learn a thing's identity from its properties because our representations in Information

Systems about things are usually limited and it is usually possible that two different things have the same properties. In practice, the problem is solved by assigning unique Identity, or primary key, to everything. This practice reflects ***Principle 3.2*** that no two things possess exactly the same set of properties.

However, in the environment of multiple databases, one thing can have different identities. Although we can guess according to some properties (for example, we can say two records belong to the same person if they include the same name and telephone number), there is no theoretically correct way to reconcile different identities unless we can learn it from other sources. For example, one company can ask a customer about his information in another company if we want to integrate customer information of the two companies.

Although the approach that we will discuss in Chapter 5 can support easily non-precise ways to decide the sameness of two things, we will assume that we can know from other sources that two instances are the same instance because this is not the emphasis of this thesis. In fact, this is a common, implicit assumption in the area of Information Integration.

The problem of identity also raises another issue: how can we identify composite things that are found dynamically by queries? The answer is that if two composite things have the same components, we can safely say that they are the same thing (Bunge, 1977). At the first glimpse, this might appear wrong. Someone may argue that two things can compose into multiple composite things. For example, one man and one woman can be a couple and at the same time, they are a team in a work. However, this sentence is expressed in the manner of the Class-Based Model. 'Couple' and 'Team' do not represent things themselves but represent properties. (In other words, they are different classes to which the same composite thing belongs.) Therefore, we know that the

same man and the woman are both a couple and a team. To speak it in a pure manner of the BWW-Ontology, one thing that is a man and one thing that is a woman compose a thing that is both a couple and a team.

The Problem of Semantic Heterogeneity

The second question is mainly about the problem of Semantic Heterogeneity, which is the main issue of this thesis. Precedence of properties can play a major role in reconciling the semantic meaning of different properties.

There are two main approaches to reconcile Semantic Heterogeneity: the bottom-up and upside-down procedures, which correspond respectively. to translating queries by ontologies and generating global schema by ontologies.

The former approach is looking for common general preceding properties on the side of integration of the properties in the data sources. This bottom-up procedure deals with constructing a common view for things from multiple data sources. For example, suppose that two schools in one community want to integrate their students' information to conduct a survey. In the survey, they want to classify all the students into different age groups. One school stores the birthdays of its students while the other stores their ages. Obviously, age group precedes age while age precedes birthday. Based on these two precedences, we can get a common property of age group about all the students in both schools and classify them by the common property.

The top-down procedure looks for preceding properties on the data-source side of the properties in the integration portion. The procedure deals with looking for things from multiple data sources that belong to the class defined on the integration side. We may not really want to know all the

specific properties; instead, we may be interested in some generic properties. In this situation, we can use precedences to look for properties preceded by one generic property, and perhaps, we can use property aggregation to get the value of the generic property. For example, one bank and one credit card company may want to integrate their customers' information to learn the total credit that one customer has used. The bank may use properties such as mortgage, personal loan, advance, etc., while the credit card company may just use the property of card balance. However, all of them are preceded by credit. Therefore, we can just add them together and get the total credit of one customer.

3.4 Summary

In this chapter, we have presented the Property-Based Model and the theories of the BWW-Ontology and discussed some main constructs, including things, properties, attributes function, law, class and composition of things. Next, two concepts in the Class-Based Model, class definition and property value, were clarified in the terms of the BWW-Ontology and one new concept, Property Integration, was presented. Finally, based on the previous discussion, we tried to analyze two processes in Information Integration: queries and integration, and to provide an ontological explanation of them in terms of the BWW-Ontology. In addition, two basic problems in Information Integration were identified and addressed: the problem of identity and the problem of Semantic Heterogeneity.

Chapter Four

THE PROPERTY-BASED MODEL

In this chapter, we formalize the basic part of the Property-Based Model based on the BWW-Ontology and its implications that we presented in the previous chapter. When we develop the model, we try to comply with the theoretical foundation while paying as little as possible attention to implementation issues. The reason for this is that we believe a general approach should be independent of most technical issues, which tend to change in the dynamic environment of Information Technology.

The Property-Based Model is the representation model of things and properties suggested in the BWW-Ontology. What we try to do is to express the Property-Based Model formally in the field of Information Systems. This model is the basis of our approach. Only after using a uniform model to represent things and properties in multiple data sources, we can start to consider the feasibility of integrating information from them.

We start with presenting the representation model of things, the representation model of properties, and then combine these two models. Next, we deal with the definitions of properties, or the semantics of properties. We use precedence as the basis of the definition model of properties. Finally, we provide the whole concept of properties and precedence to help readers understand the topics better.

4.1 Thing Representation

The first step of the Property-Based Approach is to represent one domain of interest in the Property-Based Model. According to Postulate 3.1 in the BWW-Ontology, the world is made of things that possess properties, which suggest the following representation model of thing. The model is adapted from the definition of things by Bunge (1977, p.111).

Representation Model I: Let t be a thing, p(t) be the collection of its properties, $p_1, \ldots, p_n \in p(t)$ then the thing together with its properties is represented as $<t, p(t)>$ or $<t, p_1, \ldots, p_n>$.

Figure 4-1: Representation Model I

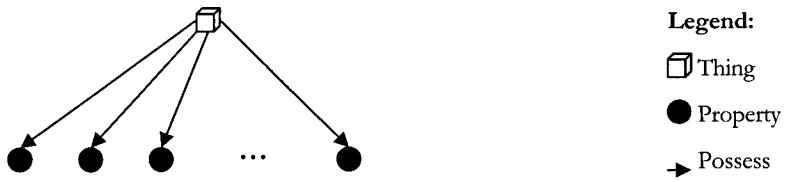


Figure 4-1 illustrates the representation model I of a thing. According to Principle 3.2, no two things possess exactly the same set of properties. However, since our representation of the world can never be complete and thorough, the basic model cannot ensure Principle 3.2. Therefore, we have to introduce another construct, Identity.

4.1.1 Identity

One of our daily practices is that we usually identify something by one unique combination of properties. For example, we may refer to a thing in a box (Here the box is viewed as our domain of interest) as "the red ball." Both "red" and "ball" are identities because there may be other colours such as "blue," or other objects, such as "cubes." If there is only one red ball in the box, we can identify the identity easily.

However, we can never guarantee that the scope of a property or a set of properties has only one thing, that is, only one thing possesses this property or this set of properties, especially when the environment of the domain is dynamic. For example, in the above example, the way to identify the red ball in the box can be totally fine if there is only one red ball. Nevertheless, if we put some red balls into the box, the original way of identification will not work anymore.

Therefore, we have to make an arbitrary solution for this. We assign a unique value to each thing's Identity Property. We can call such a value an identity, or global identifier (Parsons and Wand, 2000). This reflects the practice that in databases, we use primary key, tuple ID, etc. to identify records. An identity can be viewed as an attribute representing all properties not otherwise represented that can help us observe that a certain thing is distinct and make the set of its properties unique.

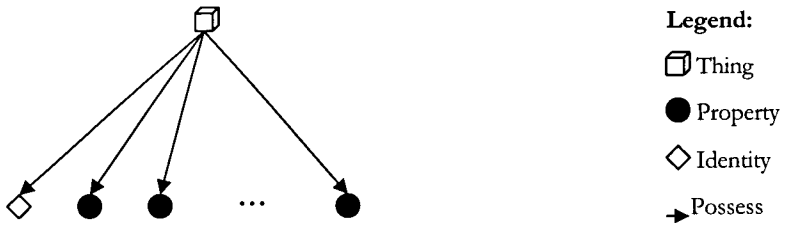
This solution is not contradictory to the argument of Wand et al. (1999, p.502) that Principle 3.2 "*implies that keys are unnecessary attributes in conceptual modeling. Indeed, it makes clear that keys are simply implementation mechanisms. Thus, key attributes have no informative role in a functional schema nor in a conceptual model of a domain.*" Contrarily, with the argument, we can see that this solution is simply an arbitrary mechanism to use our limited representation model to represent the unlimited world. By doing so, we can also achieve some advantages in implementation.

Now, we have a refined representation model of things and properties. Figure 4-2 illustrates the Representation Model II.

Representation Model II: Let t be a thing, ID be all identities that things can possess, id = ID(t) $\in$ ID be an identity possessed by t and that no other thing possesses, p(t) be the collection of its

properties, $p_1, \ldots, p_n \in p(t)$, then the thing together with its properties is represented as <t, ID(t), p(t)> or <t, id, $p_1, \ldots, p_n$>.

Figure 4-2: Representation Model II



Since one identity is possessed by only one thing according to the definition of an identity, it is safe to let the identity be a surrogate designating the existence of a corresponding thing (Parsons and Wand, 2000). So our representation model can be modified as follow.

Representation Model III: Let t be a thing, ID be all identities that things can possess, id = ID(t) $\in$ ID be an identity possessed by t and that no other thing possesses, p(id)=p(t) be the collection of its properties, $p_1, \ldots, p_n \in p(id)$, then the thing together with its properties is represented as < id, p(id)> or <id, $p_1, \ldots, p_n$>.

Figure 4-3: Representation Model III

In Figure 4-3, we can see that an identity is paired with other properties. This kind of pair relationship is in fact some kind of "implication" between identities and properties too, similar to

precedence between properties. That is if we know an identity, we can "imply" what properties the thing with the identity possesses based on the information stored in a database.

All of the three representation models of thing reflect the same idea and they can be used in different situations. The first model is the basis for the other two. The second one can be used when one thing has multiple identities, for example, when we integrate information from multiple sources. The third one can be used in the situation where there is only one identity for one thing, such as in a local database.

4.1.2 Property Representation

In the practice of Information Systems and Databases (e.g. Relational Databases (RDB)), the basic data item is comprised of an attribute name and a value. The former is what we call meta-data because it defines the meaning of the value. This data item can be described by a predicate (attribute), A(x), where A is the attribute name and x is a value. When A(x) is true, the data item attains the value x. We call this representation Basic Attribute (BA). Such a BA is useless until we assign it to something (or in other words, it refers to something). Suppose that t is a thing that a BA, A(x), refers to, we denote A(t, x). We may call this assigned BA.

A BA can represent a general property (Postulate 3.2), which we have not assigned to a certain thing. It is a predicate and when it is true, there is at least one thing that possesses the corresponding property. For example, if student(master) is true, there are master students. An assigned BA can represent a specific (or assigned) property (Postulate 3.2), which is possessed by a specific known thing. It is also a predicate and when it is true, the referred thing possesses the corresponding property. For example, student(James, master) is true only when James is a master

student. Please note that in the rest of the thesis, if we use A(x), we are talking about a general property, and if we use A(t, x), we are referring to an assigned property.

Our goal is to utilize the theory of properties and their precedences to reconcile information about properties of things of interest. Since the smallest unit of information in Information Systems is a data item, which is represented in the form of BA, our goal is to reconcile BAs. The key to our approach is that we assume BAs are attributes that represent properties.

Someone may argue that there are other attributes than BAs in Information Systems. For example, suppose that in a RDB we have a table *Sold* with three columns, *Product*, *Shop* and *Cost*. The table can be represented as a general attribute, Sold(Product, Shop, Cost), and a specific record in the table is a specific attribute, e.g. Sold(DVD-player, Futureshop, 150). The former attribute represents a general property that a product is sold at a price in a shop, while the latter a specific one that DVD-player is sold at 150 in Futureshop. However, this is not an individual data item because it is composed of three date items (BAs), which are Product(DVD-player), Shop(Futureshop) and Cost(150). The problem with this decomposition method is that the three BAs cannot express the original meaning until they are put together in the "sold" attribute.

Rubin (2002) used the theory of property and property precedences to develop a data structure that can represent all attributes (suggested in Postulate 3.2) by using BAs. He defined three kinds of property views: low-level, middle-level and high-level. The last two kinds are based on the first one, the low-level property view. The low-level property view includes an intrinsic property with one value, which is corresponding to a BA. However, when he developed this data structure, he did not consider any issues of information integration. Therefore, we cannot use this data structure

directly in this thesis. More research effort will be needed to incorporate Rubin's data structure into the Property-Based Approach proposed in this thesis.

We will also mention briefly in Section 6.1.2 a different method to transform other attributes into BAs, which is based on Bunge's notion of Unarization (Bunge, 1977, p.70). Another possible way is to use other theories in Bunge's Ontology, such as Association Theory and Assembly Theory (Bunge, 1977) to develop a representation model that needs BAs only.

A data item (the smallest information unit) is in the form of BA, and as mentioned above, there are potential ways to transform other attributes into BAs. Since our purpose in this thesis is to integrate information rather than to solve the problem of representing properties, we will concentrate on BAs in this thesis when developing the Property-Based Approach.

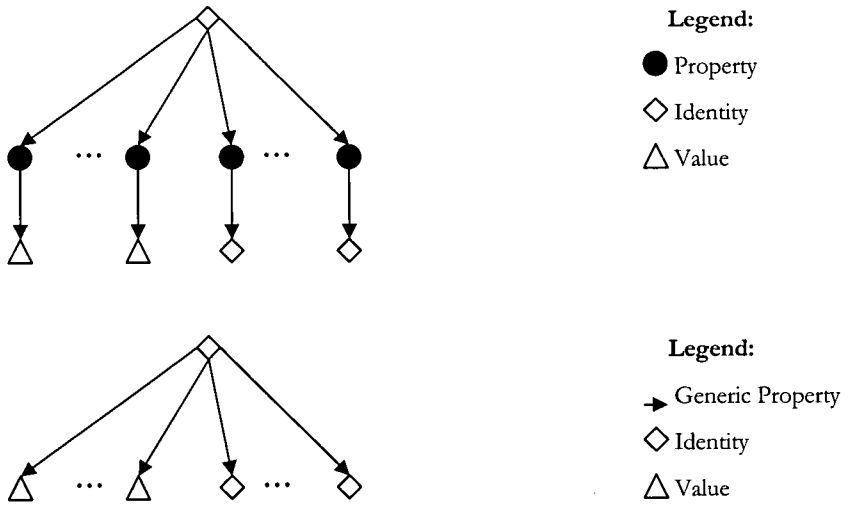
4.1.3 Representation of Thing and Property

From now on in the thesis, we only use BAs to represent properties in our model, that is, A(t, x) or A(x), where t is a thing and x can be a value or a thing. Again, A(t, x) represents an assigned property that we have associated with a certain thing, whereas A(x) represents a general property that we have not. We re-define the representation model III by putting the representation of properties.

Representation Model V: Let t be a thing, ID be all identities that things can possess, id = ID(t) $\in$ ID be an identity possessed by t and that no other thing possesses, p(id)=p(t) be the collection of its properties, $p_1(id, x_1), \ldots, p_n(id, x_n) \in p(id)$ where x_i (i=1, …, n) can be a value or an identity, then the thing together with its properties is represented as $<id, p_1(id, x_1), \ldots, p_n(id, x_n)>$, or $<id, p_1(x_1), \ldots, p_n(x_n)>$, or $<p_1(id, x_1), \ldots, p_n(id, x_n)>$.

We can illustrate the representation model V in two ways as shown in Figure 4-5. For example, we may represent a student as: <James, age(29), gender(male), major(MIS), course(Comm633> or, <age(James, 28), gender(James, male), major(James, MIS), course(Cpsc534)>.

Figure 4-5: Representation Model V



In fact, the example reflects two implementation strategies. In the first example, we may store an identity and properties together (or there are links between the identity and properties), which is just similar to a record in a table in RDBs. In the second example, we can pair an identity and a property together and store it as a unit. Therefore, we do not have to store these units at the same place and we can bind them by an identity dynamically. This is just as we join several tables together in RDBs.

The second diagram looks very similar to the RDF (Resource Description Framework) model (Lassila et al., 1999). In the RDF model, an identity is called Resources, a property Predicate, and a value Object. An object in the RDF model can be a data or another resource, while a value can

also be a data or another identity. The similarity between our model and the RDF model does give a strong technical support for the future use of the Property-Based model. Unfortunately, the RDF model is being used in the way to support the Class-Based model currently because the semantics of the model are misunderstood.

Up to now, we have discussed representation models to represent things and properties in the Property-Based Model. These models provide us with a foundation to achieve Semantic Interoperability because: 1) the models are simple, general and flexible enough to support any situations, while we do not lose their ontological sense; and 2) currently, we only use a simple and uniform format to represent properties, the Basic Attribute (BA). The next step towards Semantic Interoperability is to let computers "understand" the meaning of attributes.

4.2 Define the Semantics

For computers to "understand" the semantics of properties, they should be able to exchange the semantics of properties first and then compare these semantics to determine the similarities, differences, and relationships between properties.

Based on the contents of the previous chapter, we define formally the semantics of properties. As we discussed before, the semantics of a property can be defined by its relationships with other properties, or laws. We have discussed one particular kind of law, Precedence and pointed out the relevance of Precedence in the semantics of properties. In fact, precedence plays a specific and unique role in the semantics. It is *Precedence* that enables us to infer common semantics at a high level from two properties that are considered as different at a low level, which is the ultimate goal of Semantic Integration. Thus, we define the Property-Precedence Semantics as the following.

Definition 4.1: Let p be a property, $O(p)=\{o_1, \ldots, o_n\}$ be all preceding properties of p, and $Q(p)=\{q_1, \ldots, q_n\}$ be all preceded properties of p, $Q(p) \cap Q(p)$ is the *Property-Precedence Semantics* of p.

While we use attributes to represent properties, we use implications between attributes to represent precedences between properties. Thus, we can imply one attribute from other attributes because of precedences between properties. Based on the definition of property-precedence semantics, we define the semantics of attribute in the following sections.

In our representation models, we only have BAs, such as A(x). Corresponding to two kinds of properties; intrinsic and mutual properties, we have two kinds of BAs: namely intrinsic attribute A(x) where x is a non-identity value *v*, such as a number, a string, etc., and A(x) where x is an identity *t*. To distinguish them, from now on, we use A(v) as an intrinsic attribute and A(t) is a mutual attribute. We define them in two slightly different ways.

4.2.1 Intrinsic Properties (Attributes)

First, we notice that in our representation model IV of properties, there are two parts: the attribute's name and a value in the attribute. For example, age(29) has a function name of age and a value of 29. We term the property represented by the former Generic Property and the property represented by the latter Specific Property (Parsons and Wand, 2002). One thing that should be emphasized is that there is a special value called "*Null*." This happens when a thing possesses the generic property, but not a specific property. The generic property is represented as A(Null). Therefore, we still have a uniform format to represent properties, which is A(x). For example, we represent the property of having an age as age(Null) instead of age( ) or age.

The difference between generic properties and specific properties is relative. In theory, any specific property can be a generic property if we can give it a name and it has only one value "Null." We do not use this method because it is very inefficient in Information Systems. Furthermore, the arrangement of representing a property in a way of a property name and a value make a value as glue between a preceding property and its preceded properties (more in section 4.6.1). That is, a generic property at the lower level can be a manifestation, or a value, of another generic property at the higher level. For example, being a mammal can be a generic property with values, such as horse, dog, etc., while it can be a value of the generic property of animal.

Therefore, according to the property-precedence semantics of an intrinsic property, we can define the semantics of the corresponding BA by using its implications with other attributes. Please notice that a BA A(x) or A(Null) is a predicate, with only two values, True or False.

Definition 4.2: Let A(v) be an intrinsic attribute (a BA), where A is a the name of the attribute, v $\in$ {null, V} is a value, and V is the Value Set (the domain) of the attribute, the semantics of the attribute A(v) is defined as three kinds of implication with other attributes:

1) A(v) implies $B_1(x_1), \ldots, B_n(x_n)$ where B_i is an attribute name, x_i can be a thing or a value and i=1, …, n;

2) A(v) is implied by $C_1(x_1), \ldots, C_n(x_n)$ where C_i is an attribute name, x_i can be a thing or a value and i=1, …, n;

3) A(v) implies A(Null).

For example, being a master's student is represented as student (master's), whose semantics is defined at least by the following three implications, corresponding to the three kinds of implication

in the definition 4.2. Please notice that on the left side of the table, an underlined statement is a property, while on the right side, "student" is A, "tuition" is B_1, and "program" is C_1 in the Definition 4.2.

	Properties and Precedences	Attributes and Implications
1	If one is a master student, he will pay a tuition 1000.	Student(master) → tuition(1000)
2	If one is in the program of MSc in MIS, he is a master student.	program(MSc_in_MIS) → student(master)
3	If one is a master student, he is a student.	Student(master) → student(null)

Table 4-1: An Example of the Semantics of Intrinsic Property (Attribute)

4.2.2 Mutual Properties (Attributes)

For a mutual attribute, the value cannot be Null because Null is not a thing. For example, marry(Null) has no meaning. We can compare it with an intrinsic attribute marital_status(Null), which means one has a marital_status, but the status is not yet known. Thus, we can define the semantics of a mutual attribute as follows, according to the property-semantics of the corresponding mutual property.

Definition 4.3: Let A(t') be a mutual attribute (a BA), where $t' \in T'$ is a thing, T' is a set of things, A is the name of the attribute, the semantics of the attribute A(t') is defined as two kinds of implication with other attributes:

1) A(t') implies $B_1(x_1), \ldots, B_n(x_n)$ where B_i is an attribute name, x_i can be a thing or a value and $i=1, \ldots, n$;

2) A(t') is implied by $C_1(x_1), \ldots, C_n(x_n)$ where C_i is an attribute name, x_i can be a thing or a value and $i=1, \ldots, n$.

For example, one person marrying James (the property p) is represented as marry (James), whose semantics is defined at least by the following three implications, corresponding to the two kinds of implication in the definition 4.3. Please notice that on the left side of the table, an underlined statement is a property, while on the right side, "marry" is A, "relative" and "status" are B_i, and "husband" is C_i in the Definition 4.3. The second example shows us a situation when an intrinsic property precedes a mutual property.

	Properties and Precedences	Attributes and Implications
1	If one marries James, she is a relative of James.	marry (James) → relative(James),
2	If one marries James, her status is married.	marry(James) → status(married),
3	If one has a husband named James, she marries James	husband(James) → marry(James),

Table 4-2: An Example of the Semantics of Mutual Property (Attribute)

4.2.3 The Whole Picture of Property and Precedence

We summarize what we have discussed about precedence in the following statements:

1) Precedences enable us to infer from a property to another property, that is, from a more specific meaning to a more general meaning, thus enable us to integrate information from multiple sources by looking for the common meaning of properties.

2) We use Precedence as the foundation to define the semantics of properties.

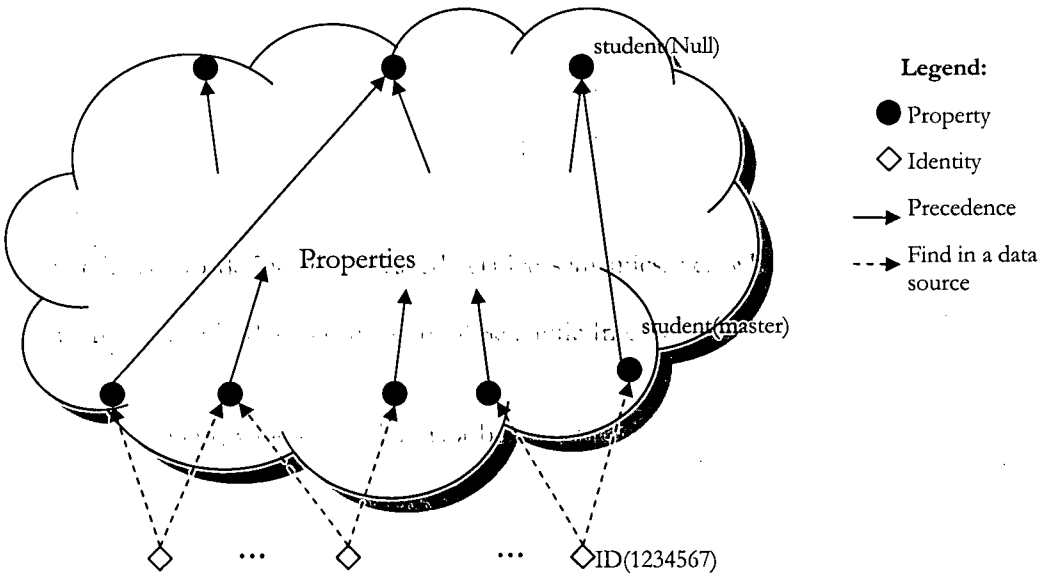
3) Precedences relate properties together.

The first two statements are directly related to Semantic Interoperability. The first one enables us to view properties (in different sources) that seem different, to be considered "same" at a higher level, while the second provides us a way to define the semantics of properties. Actually, we are

using Precedences to define the so-called relative semantics, i.e., what one property means with respect to another. Indeed, this is the core of Semantic Interoperability.

For the last statement, which can give us a better understanding of precedences, we present the whole picture of how all the properties are related together by precedences in Figure 4-6. The explanation and some simple examples follow the figure.

Figure 4-6: The Whole Picture of Property and Precedence



At the bottom of the picture, there are identities that surrogate things. These things can exist either in physical reality or only in someone's mind as long as we can perceive and identify them as specific objects (Wand et al., 1999). Given an identitiy, we can find what properties are possessed by the thing surrogated by the identity based on the information that we store in databases or other data sources. For example, if an identity ID(1234567), which is a surrogate of a thing, is paired with a property represented as student(master) in a database, we know ID(1234567) is a master student.

In the big cloud, we have all the properties, which are linked by precedences. For example, student(master) may be preceded by enrolling in a program. Each property has at least one down route to an identity. This feature indicates that properties cannot exist by themselves and must be possessed by things. We can learn at least part of the semantics of a property by knowing its "position" in this "cloud." We integrate information about two properties by looking for the common part of semantics they may have, i.e., they have the same preceding property.

Thus, to enumerate the precedences in the cloud is the first step to define the semantics of properties. Although in the practice of databases implies some precedences implicitly, especially when declaring classes and their attributes, it still lacks an explicit way to declare them. The reason is that common database approach did not treat classes and attributes as the same notion. Therefore, in the next chapter, we will deal with this problem first.

4.3 Summary

In this chapter, we have developed and formalized in detail the basic part of the Property-Based Model to represent things and their properties. We first presented the representation models of things. We then discussed how to represent properties in Information Systems. We suggested the Basic Attribute (BA) as the major method to represent properties in current Information Systems. Therefore, for the purpose of this thesis, all the properties are assumed represented by BAs in the Property-Based Model. In addition, we defined formally the semantics of properties and the corresponding attributes (BAs). Finally, we presented the whole picture of properties and precedences that link properties together.

Chapter Five

THE PROPERTY-BASED APPROACH TO INFORMATION INTEGRATION

Based on the Property-Based Model, the BWW-Ontology, and their implications, we develop the Property-Based Approach to integrating information from multiple sources in this chapter.

First, we develop a tool to describe precedences between properties, and the result is called the Property Precedence Schema (PPS). We can use PPS to define the semantics of properties. Next, we will discuss some issues of creating a global PPS, by which we can achieve Semantic Integration, and suggest a procedure to create a "global PPS."

Finally, we develop another tool, the Instance Function, to facilitate integration process. We add two features in the Instance Function, Expansion Feature and Conjunction Feature, which facilitate the inference of Property Precedence and Property Aggregation respectively.

5.1 Represent Precedences

Properties are represented by Basic Attributes (BAs) in our representation model. Whereas A(t, x) represents an assigned property, A(x) a general property (section 4.1.2), where t represent a thing, and x represents a thing or a manifestation. We can describe precedence using the following notation.

Let A and B be two BAs representing two properties, and x_1 and x_2 represents a thing or a value.

Notation 5.1: If the property represented as $A(x_1)$ precedes the property represented as $B(x_2)$, we denote the precedence as: $B(x_2) \rightarrow A(x_1)$.

Example 5.1: birthday (1973-01-30) → age (29).

Please notice again that this is only a notation. However, there is an implicit function, which is from x_2 to x_1. We term it Precedence Function, which we will define later (Section 5.1.2).

5.1.1 Property Precedence Schema (PPS)

In this section, we discuss the way to build a schema using the above notations. We call such a schema Property Precedence Schema (PPS). In our representation model of properties, there are two kinds of properties represented in two ways: generic properties (the name of attribute) and specific properties (values in a value set) (Section 4.3.1). Therefore, we have to describe them correspondingly. Between generic properties and specific properties, there can be four situations:

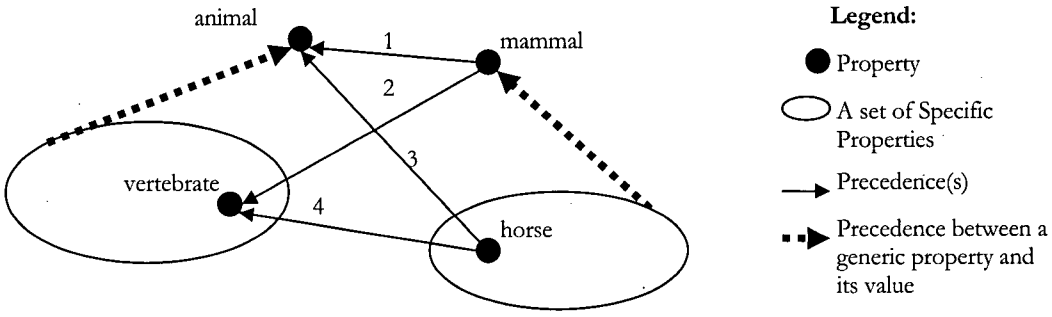
1) Between two generic properties;

2) From a generic property to a specific property;

3) From a specific property to a generic property;

4) Between two specific properties.

Figure 5-1: Four Kinds of Precedences



Example 5.2: 1) mammal (Null) → animal (Null), 2) mammal (cat) → animal (Null), 3) mammal (Null) → animal (vertebrate) and 4) mammal (horse) → animal (vertebrate).

In Figure 5-1, we illustrate the ways we represent precedences in those four situations respectively, which are the four arrows in the middle. We list four examples here to illustrate these four situations, respectively in Notation 5.2.

5.1.2 Precedence Functions

In the third situation, however, there are usually a set of specific properties, which are preceded by one generic property. For example, all birthdays in 1973 are preceded by the age of 29. Similarly, in the fourth situation, there is usually a group of precedences between two sets of specific properties. If we can define a mapping function to express those precedences For example, all possible birthdays are preceded by one age.

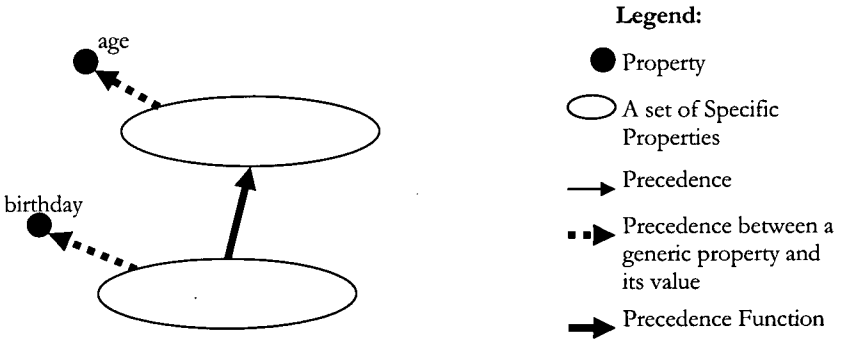
These mapping functions are a straight and efficient way to express precedences, so it seems necessary and useful to add it into our formal representation of precedence. We call these mapping functions Precedence Functions. One thing that should be mentioned is that precedence functions only happen in intrinsic properties, where x is a value.

Definition 5.1: A Precedence Function is a function, such as F: X_2 → X_1, where $A(X_1)$ and $B(X_2)$ are two attributes, X_1 and X_2 are two subsets of the value sets of A and B, respectively.

Example 5.3: F: D → N, D∈{All dates by today} and A∈{all integers}. This function computes a date to a number and is the Precedence Function between two attribute, birthday(D) and age(N).

Notation 5.2: If there is a Precedence Function F: $X_2 \rightarrow X_1$ between $A(X_1)$ and $B(X_2)$, we denote these precedences as: $B(X_2) \rightarrow A(X_1)$, where $A(X_1)$ and $B(X_2)$ are two attributes representing two sets of properties, X_1 and X_2 are two subsets of the value sets of A and B, respectively.

Figure 5-2: Two Examples of Precedence Functions



Example 5.4: birthday(D) → age(N), D∈{all birthdays}, N∈{all ages} (shown in Figure 5-2.). The Precedence Function is shown in Example 5.3

Figure 5-2 illustrates two examples of Precedence by using Precedence Function. The left one is from a set of properties to one property, while the right one is from a set of properties to another set of properties.

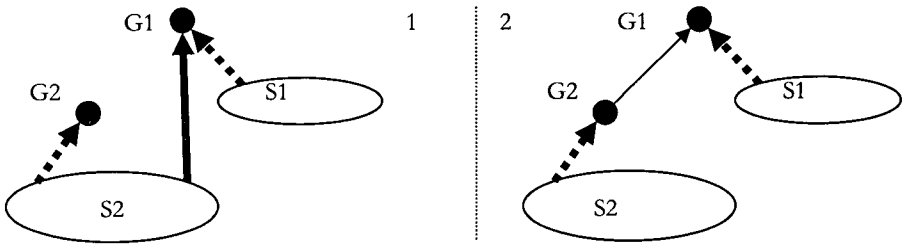
5.1.3 Some Special Cases in PPSs

In this section, we want to analyze some special cases that could happen in a PPS.

The first case happens when all values (manifestations) of a generic property are preceded by another generic property, which also means that the first generic property itself is preceded by the second one (recall Lemma 3.1 or Lemma 3.3 in Chapter 3). Therefore, there are two ways to

express this in PPSs (Example 5.5 and Figure 5-3). Obviously, the second way is more efficient and enjoys clearer semantics.

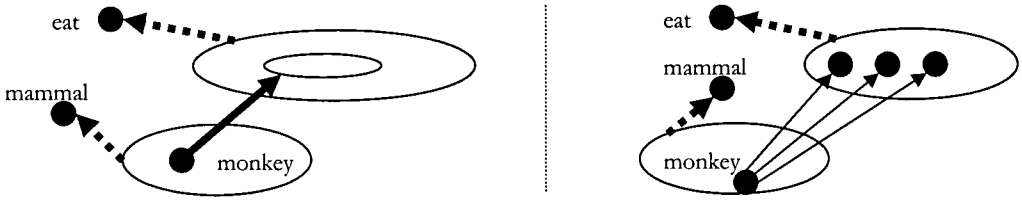
Figure 5-3: The First Special Case in PPSs



Example 5.5: 1) mammal(X) → animal(Null), X∈{all mammals}, and 2) mammal(Null) → animal(Null). (G1: animal, G2: mammal, and S1 and S2 are their values, respectively)

The second case is also about two equivalent expressions, as illustrated in Example 5.6 and Figure 5-4. One rough explanation of the first example can be that a property is preceded by a set of properties, a new user of PPSs may be confused about its meaning. Does this mean that the property can be preceded by any property in the set, or that all properties precede the property? After more consideration, we will find that the former is indeed not a precedence because we are not sure which one property in the set precedes the property. The answer is the latter, that is, if one thing possesses the property, it must possess all the properties in the set. In fact, the number of properties in the set is usually limited. For this reason, if convenient, we suggest the use of a second way to represent this by enumerating all the precedences between them, as shown on the right in Figure 5-4.

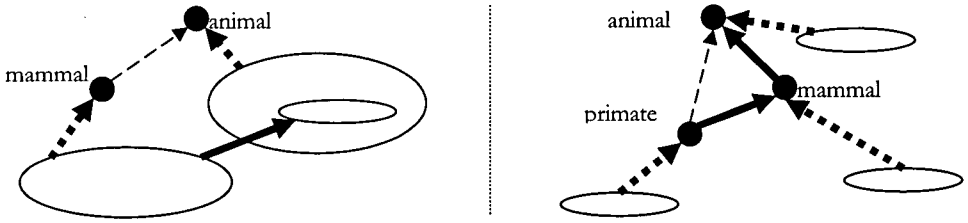
Figure 5-4: The Second Special Case in PPSs



Example 5.6: 1) mammal(monkey) → eat(X), X∈{fruit, vegetable, leaf}, and 2) mammal (monkey) → eat(fruit), mammal(monkey) → eat(vegetable), mammal(monkey)→ eat(leaf).

In a PPS, there might be many redundant precedences, such as the two examples in Example 5.7. The two examples are illustrated in Figure 5-5. (The dashed arrows are redundant precedences.)

Figure 5-5: Two Examples of Redundant Precedences

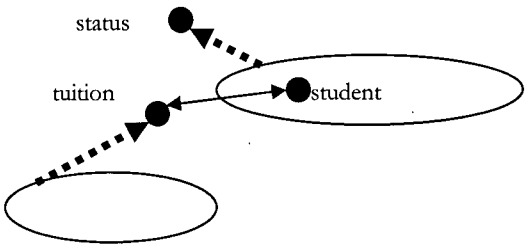


Example 5.7: 1) mammal (X) → animal (vertebrate), X∈{all mammals}, so mammal (Null) → animal (Null) is a redundant precedence (Lemma 3.1 or Lemma 3.3). 2)primate(Null) → mammal(Null) and mammal(Null) → animal(Null), so primate(Null) → animal(Null) is a redundant precedence.

Redundant precedences usually give us shortcuts between properties, which could mean more efficient in implementation and more importantly, could capture more meaningful semantics. Therefore, my argument is that as long as users think they need a shortcut, we can always add it.

The fourth case happens when two properties can precedes each other, which means that their scope is the same, or that every thing that possesses one property of them must possess another one. We term this precedence Co-Precedence, which is represented by a two-headed arrow. Such two properties are concomitant properties (Bunge, 1977) in the BWW-Ontology.

Figure 5-6: an Example of Co-Precedence



Example 5.8: tuition (Null) → status (student) and status (student) → tuition (Null). This means all students pay tuition while all who pay tuition are students

We have introduced all basic patterns and some special cases possibly occurring in PPSs. We can express complicated situations by using these patterns. In the next chapter, we will provide an application to implement PPSs.

5.2 Generating Global PPS

Until now, we have discussed the representation models to represent things and properties, the method to define the semantics of properties, and a tool to describe precedences between properties. We can use them to model a domain of interest. These issues are mainly on the local side, i.e., the side of data source. However, before we can integrate information, we should achieve Semantic Integration first, that is, to know the meaning of the information from the global point of view.

As we pointed out in the last section, a PPS of a domain is actually an (at least partial) Information Systems ontology, describing the schema of one domain. One way to achieve Semantic Interoperability is to create a global schema generated by an integrated ontology (Hakimpour and Geppert, 2001). In our approach, all we have to do is to integrate local PPSs from multiple data sources to create a global PPS.

Although a global PPS will not differ from local PPSs in the syntax and expressions, there are indeed different issues and considerations between them. The way to create a local PPS is straightforward, mainly including finding out what information is needed according to the users' demand, determining necessary properties to model the domain and describing the precedences between these properties. In contrast, a global PPS has its unique concerns.

We identify three main differences between a global PPS and a local PPS:

1. Global users might have different points of view from local users' points of view. For example, from the viewpoint of a faculty in a university, someone is a professor while someone is a lecturer. However, from the university's point of view, all of them are staff.

2. The global side has to deal with multiple local PPSs, between which there might be conflicts, so the global side has to reconcile local PPSs. For example, two local PPSs could use the same attribute name to represent two different properties.

3. The global side does not have direct control on data so they have to make some compromises. For example, a local side may use a categorization that cannot be mapped directly into the categorization used in the global side.

The first two differences cause a modelling problem, which we term Domain Problem. After discussing the Domain Problem, we will suggest a general procedure to create a global PPD to cope with the differences.

5.2.1 Domain Problem

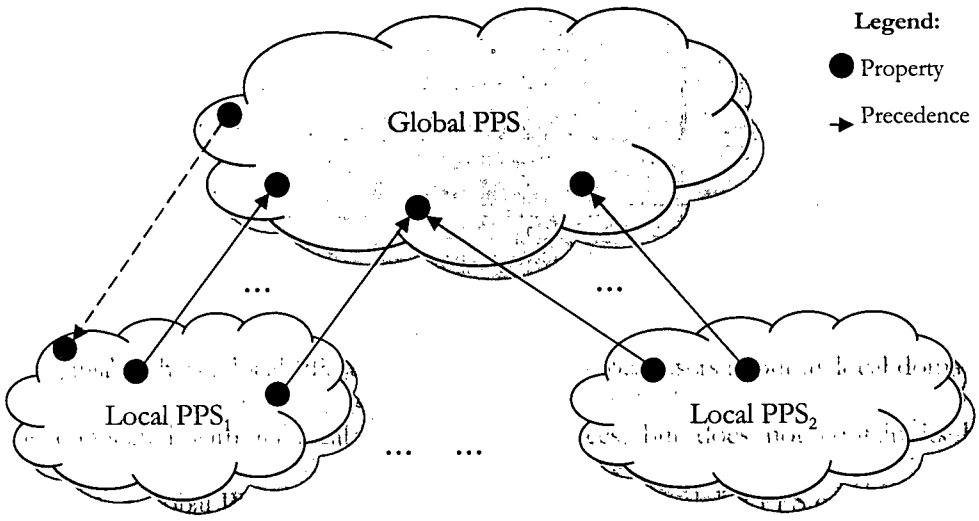
In the field of Information Integration, we usually refer to a domain that collects and integrates information from other domains as a super-domain, while these other domains are referred to as sub-domains. A super-domain does not necessarily include all properties in its sub-domains, while it can include properties other than those in its sub-domains. Therefore, a super-domain is not simply an addition of its sub-domains. Furthermore, from the viewpoint of the users in a super-domain, it may assign a different attribute to a property from that to the same property in its sub-domains. For instance, in the domain of loan department in a bank, they say something is a loan. However, from the viewpoint of the accounting department, which is the domain of all the departments in the bank, a loan is considered an asset.

Therefore, my argument is that every domain exists independent of other domains and can stand by its own. That means that the existence of a super-domain is not due to the existence of its sub-domains. For instance, the accounting department is independent of other departments in a company. The argument implies that we do not create properties in one super-domain by using properties from its sub-domains; instead, properties in one domain exist by themselves, while preceding or being preceded by properties from other domains.

Since a PPS is a schema of a domain, a global PPS refers to a schema of a domain corresponding to some local PPSs of its sub-domains. A global PPS is not just a simple aggregation of local PPSs. It is in fact a reflection of the view of global users about the entire domain containing several small

domains, just such as a local PPS reflecting the view of local users about its local domain. A global PPS is connected with its local PPSs by precedences, but does not contain its local PPSs. Furthermore, a global PPS can also be a local PPS of another global PPS on a higher level. Figure 5-7 illustrates the relationship between local PPSs and a Global PPS.

Figure 5-7: Local PPSs and Global PPS



One thing that should be emphasized is that it is quite possible for a property in a local PPS to precede a property in its global PPS, as shown by the dashed arrow in Figure 5-7. For example, in the case study in Chapter 7, if a flyer enjoys some privileges in Star Alliance (the global side), he will also enjoy the corresponding privileges in a specific airline (local side), which is a member of Star Alliance.

The global side might not be sure whether two properties from two local sources are the same (the same semantics and the same value domains) or not, and the global side needs to know where properties come from and determine different Aggregation Rules for them. Therefore, we use an

arbitrary solution, which is to create new properties in a global PPS as the preceding properties of properties from local PPSs. By doing so, we can also trace back easily from the global side to the original data sources. We can use a naming method to differentiate properties for local PPS: add source name before the name of the properties from local sources. (This is only a technical issue.) For example, we can rename a property "name" from UBC as "ubc.name" and rename a property "last name" from SFU. In the global PPS, we may add a property called "name," which precedes "ubc.name" or "sfu.last_name."

5.2.2 A Procedure to Create a Global PPS

As we pointed out above, a global PPS does not contain local PPSs but connects with them. We do not always need every property of local PPSs in a global PPS. A global PPS should reflect global users' needs first and then consider how to connect to local PPSs. The connections between a global PPS and a local PPS define the way that we can retrieve information from the local sources using a global view. Therefore, in the following, we introduce a general procedure to develop a global PPS upon multiple local PPSs according to the demand of global users.

1. Find out what information is needed according to the needs of users on the global side, determine necessary properties to model the entire domain and describe the precedences between these properties.
2. Go through local PPSs one by one and for each local PPS
    a. Assign a source name to the local PPS (This is only a technical point);
    b. Look for properties that can be preceded by the properties in the global PPS;
    c. Add relevant precedences between them in the global PPS, using the naming method of adding source names before properties from the local PPS;
    d. Check if there is any property in the global PPS that does not have at least one route of precedences to the local PPS. If there is one, make sure that there is

indeed no way to infer from any properties of the local PPS to it. (This means that such a property is of no interest in the integration, or in other words, we cannot learn the information of the property from the available local sources.)

e. For each property in the global PPS, determine the Aggregation Rules for it, based on what properties it precedes and its semantics.

Please note that this procedure can only be done manually by users or experts in the global domain, who know the semantics of both a global PPS and local PPSs. More research is needed on the methodologies to generate an integrated global PPS from local PPSs automatically.

5.3 Instance Function

In this section, we develop and formalize the Instance Function to facilitate the integration process. An Instance Function is a tree-like structure, representing an instance by putting all information from multiple sources together, which can facilitate two inference procedures: one is for Property Precedence and the other for Property Aggregation.

5.3.1 Basic Instance Function

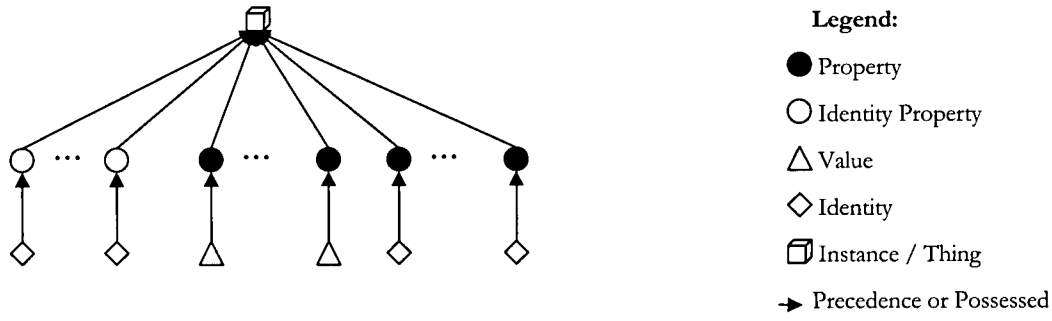
The Instance Function is actually developed based on the representation model of thing and property. Recall that we have two representation models of thing: Representation Model II and III (section 4.1.1). The difference between them is that in the Representation Model III, identity is a surrogate of a thing, which means, only one identity is allowed in one thing. Therefore, this model is very suitable and efficient within one data source because there is only one identity needed in one data source.

However, this is no longer true when we integrate information from multiple data sources, that is, we have a multiple-identities problem. Therefore, we use the Representation Model II, which is

general enough to cope with this problem. We can formalize the Basic Instance Function according to the Representation Model II as the following.

Definition 5.2: A Basic Instance Function is defined in a predicate function named INSTANCE, such as INSTANCE{ID(id_1), …, ID(id_m), $P_1(x_1)$, …, $P_n(x_n)$}, where m≥1, n≥1, ID( ) represents Identity Property, id_1, …, id_m are identities, P_1, … P_n are BAs representing properties, and x_1, …, x_n are values or identities that are surrogates of things.

Figure 5-8: The Basic Instance Function



Example 5.9: INSTANCE{ID(s1.student.number.12345), ID(s2.resident.SIN.1234567), givenname(James), surname(Bond), gender(male), live_in(Canada)} is such a Basic Instance Function representing a student.

Note that in this model we can consider Identity Property as a generic property named "ID" while an identity is a value of the generic property ID. Besides, in our approach, we will combine two sets of information from two data sources that are about the same thing so we have to put different identities under one thing, as shown in Figure 5-8. Please note, we assume that we have a way to know this is the same thing (Section 3.3.2).

Furthermore, we need to differentiate these identities by their sources. Therefore, in the example 5.9, we can see a solution for this. We put the source name (e.g. s1, s2), table name (e.g. student, resident), attribute name (e.g. number, SIN) before each identity, to make the unique on the global side and to keep the sources' information. Please also see more about this in the application of the Property-Based Approach in the next chapter (Section 6.3.1).

Upon this model, we add into Basic Instance Function two features, Expansion Feature and Conjunction Feature, to support precedence and property aggregation.

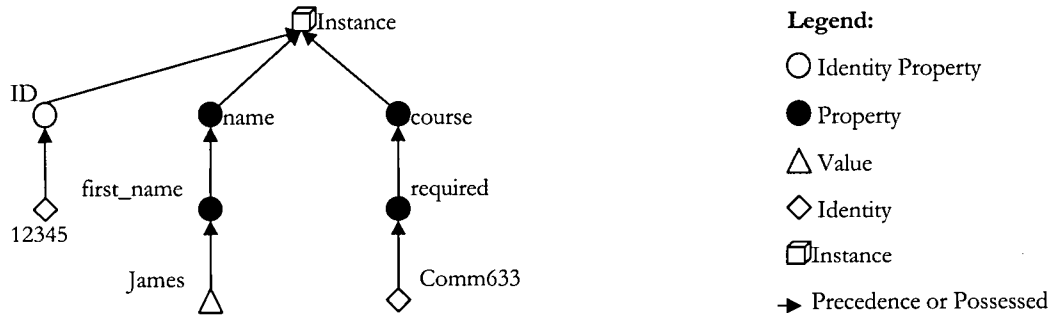
5.3.2 Expansion Feature and the Inference of Precedence

The first feature, Expansion Feature, allows us to link together two properties, between which there is a precedence. In the following Definition 5.3, the reason that x is no long needed is because there is an implicit Precedence Function that will help us compute x from x'. Usually, a Precedence Function is defined in a PPS. More discussion about this is after the following example.

Definition 5.3: If the property A(x) precedes the property B(x'), that is B(x') → A(x), denote this by A*B(x') in an Instance Function. We call this Expansion Feature.

Example 5.10: INSTANCE{ID(12345), Name*first_name(James), course*required(Comm633)} (shown in Figure 5-9). ("Name" and "Course" are A, "first_name" and "required" are B, "James" and "Comm633" are x' in the Definition 5.3)

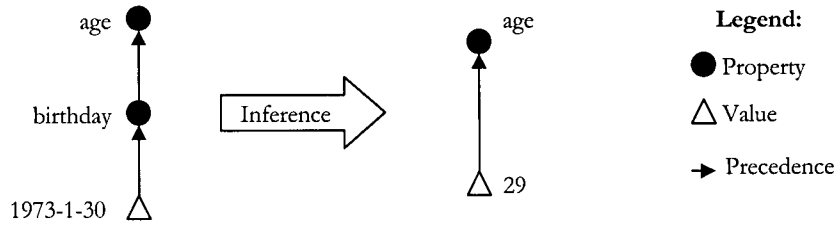
Figure 5-9: Expansion Feature in the Instance Function



Since this feature puts two properties into the order of the precedence between them, it enables us to facilitate the inference procedure of precedence directly in an Instance Function. For example, if we have some thing such as age_group*age(16) in an Instance Function, we will know the property age(16) must be preceded by some property age_group(x). The problem is now how to compute out x. It is simple. We just go back to the relevant PPS, where we store the precedence information between age and age group. Suppose in the relevant PPS, there is a Precedence Function, from which we know F: 16 → teenage, we will infer from age_group(age16) to age_group(teenage). Therefore, we can use the inference of precedence to collapse all the nodes under one property into its value.

Definition 5.4: In an Instance Function, if a property represented as A*B(x') (Expansion Feature) and a Precedence Function F: x' → x, we can infer A(x) from B(x'). We call this the Inference of Precedence and denote it as $A*B(x') \Rightarrow A(F(x')) \Rightarrow A(x)$.

Figure 5-10: The Inference of Precedence



Example 5.11: age*birthday(1973-1-30) ⇒ age(birthdayToAge(1973-1-30)) ⇒ age(29). (where A is age, B is birthday, and birthdayToAge( ) is a Precedence Function from birthday to age.)

Note that Expansion Feature can be used between two intrinsic properties, between two mutual properties, and between one intrinsic property and one mutual property as well because precedences can exist in these situations.

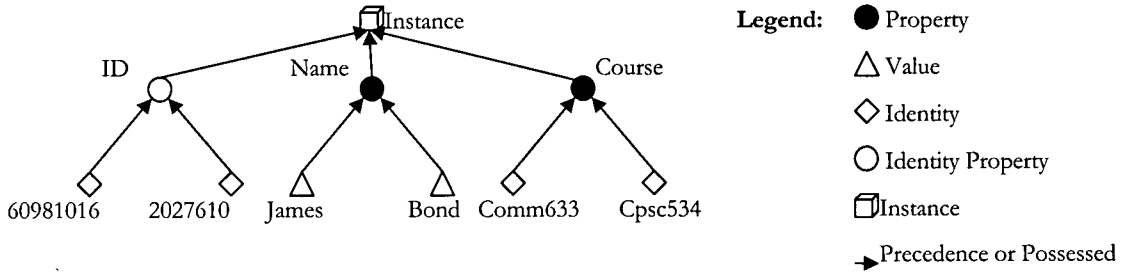
Example 5.11 is between two intrinsic properties. An example between two mutual properties is: relative*brother(James) if brother(James) → relative(James), or, program*course(comm633) if course(Comm633) → program(MIS). An example between one intrinsic property and one mutual property is: marital_status*husband(James) if husband(James) → marital_status (married), or spouse*marital_status(married) if marital_status(married) → spouse(Null), where spouse(Null) means one has a spouse, who is not known now.

5.3.3 Conjugation Feature and the Inference of Property Aggregation

The second feature, Conjugation Feature, allows us to join two property nodes together if both of them have the same attribute name, and are just under the same node in an Instance Function.

Definition 5.5: In an Instance Function, if $A(x_1)$, …, $A(x_n)$ are under the same node, denote this by $A[x_1, \ldots, x_n]$. We call this Conjugation Feature.

Figure 5-11: Conjugation Feature in the Instance Function



Example 5.12: INSTANCE{ID(60981016), ID(2027610), Name(James), Name(Bond), Course(Comm633), Course(Cpsc534)}, which means that James Bond takes two courses, Comm633 and Cpsc534. Using the Conjugation Feature, the Instance Function is, INSTANCE(ID[60981016, 2027610], Name[James, Bond], Course[Comm633, Cpsc534]}. The last property represented by Course is mutual property, while comm633 and cpsc534 are identities of two courses (Figure 5-11).

By Conjunction Feature, multiple values have been put together, thus we can facilitate the inference of aggregation directly in an Instance Function.

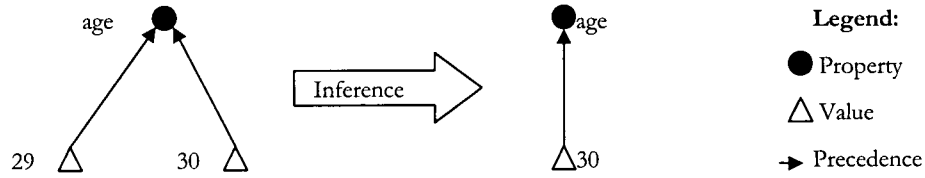
Recall that a property aggregation (section 3.2.4). We use the aggregation rule of a property to aggregate multiple values into one value. For example, we may want to know the entire income of a person from two sources. His salary for his main job is 80,000 while he has a second job with a salary of 5,000. Therefore, his whole salary is 85,000. Another example, suppose we learn from one source that one's age is 29 and from another source, he is 30, we will take 30 as his age if the aggregation rule is to get the maximum value.

Definition 5.6: An Aggregation Function is a function, such as F: $\{x_1, \ldots, x_n\} \rightarrow x$ where $n>1$, $A(x_1), \ldots, A(x_n)$, $A(x)$ are attributes (predicates), and $x_1, \ldots, x_n$, x are values of A.

Example 13: Max: {S} → N, S is a set of numbers and N is the maximum number in the set S. It is an Aggregation Function of age(N) if we take the maximum age as one's age from several different ages.

Definition 5.7: In an Instance Function, if there are multiple attributes $A(x_1), \ldots, A(x_n)$ that we put into $A[x_1, \ldots, x_n]$, we can infer A(x) from them based on an Aggregation Function F: $x_1, \ldots, x_n \rightarrow x$ (n>1). We call this the Inference of Property Aggregation and denote it as $A[x_1, \ldots, x_n] \Rightarrow A(x)$.

Figure 5-12. An Example of the Inference of Property Aggregation



Example 14: age[29, 30] ⇒ age(30), if we choose the Aggregation Function shown in Example 13.

Please notice that the value can be a set of values. For example, if we have skill[write, drive] in an Instance Function, and the aggregation rule is to form the values into a set, we will have the property skill({write, drive}).

We can also aggregate mutual properties together. For example, if we have brother[James, Bill] in an Instance Function, we will have the property brother({James, Bill}). However, this requires us to compose a new composite thing of things related by these mutual properties. In the example, James and Bill compose a new thing that may be called brothers. Further discussion on the implications of composition is beyond the purpose of this thesis.

5.3.4 Complete Instance Function

In a Basic Instance Function, properties are in the format P(x), which is not enough to represent the two features, Expansion Feature and Conjugation Feature. Therefore, we need a flexible format of representation, which is Property Function.

Definition 5.8: A Property Function is defined in the format of an n-ary function, such as $P[x_1, \ldots, x_k, *x_{k+1}, \ldots, *x_n]$, where $n \geq 1$, $x_1, \ldots, x_k$ are values, or identities that are surrogates of things, and $x_{k+1}, \ldots, x_n$ are other Property Functions.

Example 5.15: age_group[adults, *age(40), *birthday(1963-1-1)] is such a Property Function.

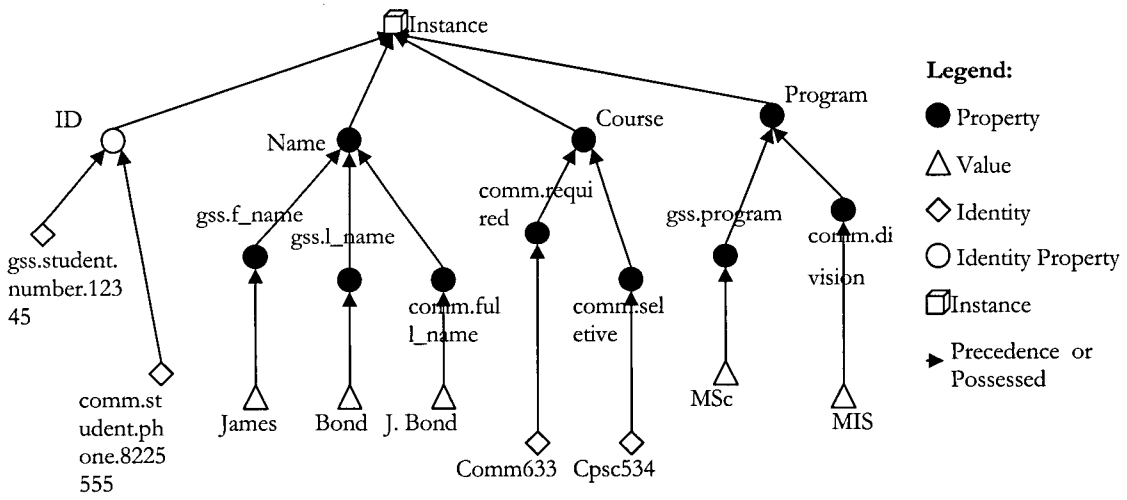
A Property Function contains the Expansion and Conjugation Features and thus can facilitate the Inference of Precedence and Property Aggregation. By using Property Functions, we can formalize the Complete Instance Function according to the full-featured Instance Function as the following.

Definition 5.9: A Complete Instance Function is defined in a predicate function named INSTANCE, such as $INSTANCE\{ID[id_1, \ldots, id_m], PF_1, \ldots, PF_n\}$, where $m \geq 1$, $n \geq 1$, ID represents Identity Property, $id_1, \ldots, id_m$ are identities, and $PF_1, \ldots PF_n$ are Property Functions.

We give one richer example to illustrate the whole idea of the Instance Function thoroughly.

Example 5.16: INSTANCE{ID[gss.student.number.12345, comm.student.phone.8225555], Name[*gss.first_name(James), *gss.l_name(Bond), *comm.full_name(J.Bond)], Course [*comm.required(Comm633), *comm.selective(CPSC534)], program[*ubc.program(MSc), *comm.division(MIS)]}.

Figure 5-13: An Example in the Instance Function



In the example, the data is coming from two data sources, which are 'gss'(graduate student society) and 'comm'(commerce). The first identity is from a table called student and an attribute called number in the source 'gss', while the second is from a table also called student and an attribute called phone in the source 'comm'. The source information is embedded in the identities after the integration. The properties coming from local sources have the prefixes indicating their sources.

5.4 Summary

The primary idea for achieving Semantic Interoperability in the Property-Based Approach is to look for a common preceding property of two properties so that we can view these two properties as having the same generic meaning. In this chapter, we developed two tools, PPS and Instance Function, to fulfill the idea. They are two main components of the Property-Based Approach to integration information. PPS helps us to define the semantics of property, whereas Instance Function help us integration information by the inference of Property Precedence and by Property Aggregation.

Chapter Six

PRACTICAL ISSUES OF THE PROPERTY-BASED APPROACH

In this chapter, we discuss briefly three practical issues that are related to the Property-Based Approach. The discussions about them are rather suggestive and should not be taken directly into implementation. The purpose is to show feasibility of the Property-Based Approach and to support the case study illustrated in next chapter.

The first issue concerning us is the way to transform the data in the Class-Based Model into the data in the Property-Based Model, so that we can utilize the vast amount of information stored in popular databases, such as Relational Databases (RDBs) and Object-Oriented Databases (OODBs). The second issue is regarding a possible way to implement Property Precedence Schema (PPS). The third issue focuses on implementing Instance Functions to facilitate the integration process by using XML.

6.1 The Property-Based View

To transform the data in the Class-Based Model to the data in the Property-Based Model we need to build a mapping between attributes of both models, which we call the Property-Based views.

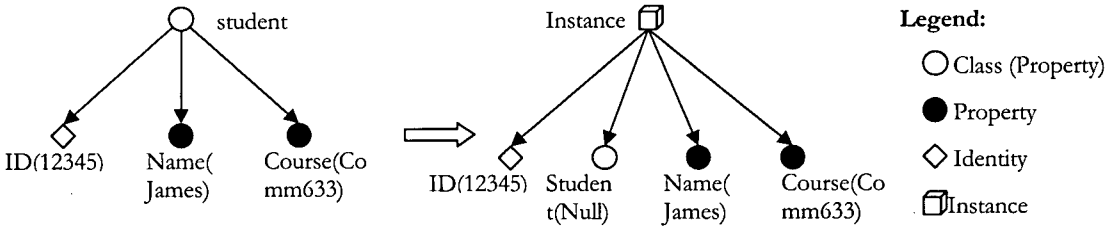
Most of the attributes that are used in the Class-Based Model are basic attributes, i.e., in the format of A(v). Therefore, we can use them directly in the Property-Based Model. For example, name(James), age(30), and course(Comm633) are such kind of attributes. They represent basic intrinsic properties or binary mutual properties. However, we have to worry about at least two problems. First, whereas every instance in the Class-Based Model belongs to a class, we do not

have an equivalent counterpart in the Property-Based Model. Second, they can represent some binary mutual properties with values (one kind of high-order properties) in the Class-Based Model. This is usually represented in RDB by a table with two foreign keys and some other columns.

6.1.1 Classes

As we have discussed before (Definition 3.2), a class is a set of things possessing the same property. Therefore, we can just replace a class with a property in the Property-Based Model. That is to assign an attribute to every instance in a class. In practice, we can just take the class name as the attribute name usually with a value Null. Figure 6-1 shows an example of transforming the data from the Class-Based Model to the Property-Based Model.

Figure 6-1: An Example of The Transformation Process



1. In a RDB, we have a record, student(ID(12345), name(James), course(Comm633));

2. In the Property-Based Model, we add one more attribute student(Null) to it. Its Instance Function will be, INSTANCE{ID(12345), student(Null), name(James), takingCourse(Comm633)}.

Therefore, we can list the mapping between them in the Table 6-1. Since the mapping changes our view on the data, we call the mapping the Property-Based View. As we can see, by using the Property-Based view, we can have a renaming feature. Thus, we can rename the attribute to a more

meaningful one, such as takingCourse in the example. We will actually use it in the case study in the next chapter.

The Property-Based Model	The Class-Based Model
student(Null)	The class of Student
name( )	name( )
takingCourse( )	course( )

Table 6-1: An Example of The Property-Based views

6.1.2 High-order Attributes

In RDB, we sometimes have a binary mutual attribute with a value. For example, a customer orders a product at some date, which is possibly represented in a RDB by a table, such as order(customerID, productID, date). There is at least one way to transform it into several basic attributes.

The method is to use the same technique used by Bunge, which is called Unarization (Bunge, 1977, p.70). The basic idea is to freeze some arguments in the attribute and move them to the part of attribute name. For example, suppose we have a record in the table order, such as order(James, DVD, 2003-1-1). We may transform it into two basic attributes: $order\_date_{DVD}$(James, 2003-1-1) and $order\_date_{james}$(DVD, 2003-1-1). Obviously, the first attribute represents a property possessed by James, while the second represents one possessed by the product DVD. Although it might be unwieldy to use $order\_date_{DVD}$ and $order\_date_{james}$ as attribute names, it is legitimate because the choices of attribute names are quite arbitrary in our daily life. For example, we use both Canadian and Citizenship of Canada. To put them in attributes, it will be such as Canadian(James), $Citizenship_{canada}$(James).

We admit that this method will become "uglier" when we deal with higher order attributes. However, since we are dealing with Semantics Integration, we are satisfied by the ability of the method to transform every kind of attribute into basic attributes. Certainly, how to make it feasible will be another interesting research topic. In addition, we may have better methods later to do this. Therefore, we leave this problem for further research.

6.2 Implement PPSs

In this section, we discuss a general method to implement PPSs. From the viewpoint of implementation, we can divide precedences into three types:

1. Between a generic property and its values (which is actually the domain of an attribute, e.g. a generic property Student has two values: undergraduate and graduate);

2. A single precedence (e.g. from being a student to having a student number);

3. A mapping function (a bunch of precedences, e.g. from birthday to age).

The first type is defined in the definition of generic properties and is done automatically after we set up a Property-Based Database (PBDB) or other databases.

Generally, the second type can be enumerated in a binary table with two columns: one is preceded property and the other preceding property. However, we use a property name and a value to represent a property, so we need four columns to store them. For example, in a university, a student must have a student number while others can have a student number too because they were students before. The generic property, Student, has the values of Undergraduate, Master,

Ph.D. and Null. The precedence is expressed as student(Null) → student_number (Null). (Please notice that these are two generic properties, whose values are Null.) The table will be:

Preceded Generic Property	*Preceded Value*	*Preceding Generic Property*	*Preceding Value*
Student	Null	Student Number	Null

Table 6-2: Precedence Table (I)

The third one, which is about a mapping function, is more complicated. It involves four factors at most, which are preceded generic properties, preceded value set, preceding generic properties, and preceding value set. The value set can be the entire set of values or a subset of values. The mapping function has only one variable, which can be any one in a preceded value set and returns one in preceded value set of another property. An inverse function is also needed. The inverse function may return a set of values. Since these functions can be independent of properties, these functions are reusable. Based on this analysis, we can use a table with six columns to enumerate these mapping functions.

For example, we take two generic properties: birthday and age. Obviously, age precedes birthday and between their values, there is a mapping function. We can express the mapping function as the following:

Preceded Generic Property	*Preceded Value*	*Preceding Generic Property*	*Preceding Value*	*Preceding Function*	*Preceded Function*
Birthday	the entire value set	Student Number	the entire value set	age(B)	birthday(A)

Table 6-3: Precedence Table (II)

The preceding function, age(B), returns an age according to a birthday B, while the preceded function, birthday(A), returns a set of dates of birth according to an age A. There are two ways to express a set. One is a list of all values, while the other is to give a range, such as between Jan 1st, 1973 and Dec 31st, 1973.

Obviously, we can express the first four-column table in the second six-column table by ignoring the last two columns of mapping functions. We can call such a six-column table to implement a PPD a Precedence Table.

6.3 Implement Instance Functions

Since XML is becoming a standard format of information exchange in Information Systems, one benefit of implementing Instance Functions by using XML is that this application of the Property-Based Approach will be easily adapted into the current technology.

As we can see in the previous chapter, an Instance Function appears in the form of a tree. Therefore, it is straightforward to use XML to express an Instance Function, that is, we can use "Instance" as the root element of an Instance Function, use all the property names as the tag name of its relevant elements and use values as the contents of their above elements. Certainly, because the root name makes no difference in the semantics of an Instance Function, we can use other names, (e.g. thing, entity and whatever is reasonable for users), as the root name.

6.3.1 Identities

In the last chapter, we mentioned that we need to differentiate identities by their sources. Furthermore, as discussed in Section 5.2, we may use primary keys as identities. However, in RDBs, it is possible that two primary keys from two tables can have the same value and even the

same attribute name. Therefore, we may need source name, table name and attribute name to make an identity unique when we integrate information from multiple sources.

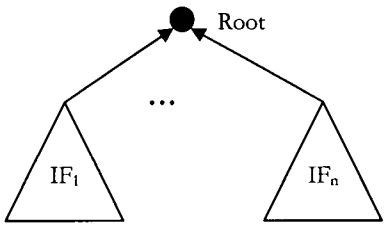
A possible solution is to add these three names before the identity by using the following format: *SourceName.TableName.AttributeName.AttributeValue*. For example, suppose that UBC stores student information in a table of Student and its primary key is Student_Number, a student with a student number of "60981016" will be identified as "UBC.Student.Student_Number.60981016".

Please note that this solution does not intend to solve the issue of a universal ID, but to make an identity unique and keep the information of its source.

6.3.2 Represent a Set of Things

First, an Instance Function (IF) expresses only one thing and its properties. However, we usually want to include a set of things in one XML file instead of only one thing. This can be achieved easily, by adding a root element to hold all the things in the set, as shown in Figure 6-2.

Figure 6-2: The Structure of XML File of Instance Functions



There are several ways to name a root element. One way is to use the source name. This is especially suitable when a data source return the results to the global side so users can know where the information comes from. For example, when we retrieve information about people in UBC, we can use UBC as the root name. Another way is to select a class name for the set of things. This is

suitable when global users collect information from multiple data sources. The name may be given by the users. For example, we may use "Students" when we retrieve information about students from some schools. Of course, we can use other naming methods and add attributes in the root element to provide more information about this set of things.

To avoid potential ambiguities, we suggest to keep the format strictly through the whole process of integration, namely that the nodes at the second level will be the roots of Instance Functions. Technically, we can always add elements above and claim that the father element is a super-set of several sub-sets represented by the son elements. For example, we may want two sets of people, Undergraduates and Graduates, under a super class Students. However, this practice might be problematic if we want to know exactly where the root of an Instance Function is. Furthermore, it seems pointless to express super-sets and sub-sets in the integration process.

6.3.3 The Hint Attribute

The second issue occurs in the situation where users on the global side may not know the mapping functions between properties from the local sides and those from the global side. Therefore, we must have a mechanism to transfer the information from the local side to the global side. One possible solution is to add an attribute called "Hint" in a son element and the attribute's content is about the mapping function from the son element to its father element. The content could be a pointer to a mapping function or just a "hint" to tell the users about the inference process.

For example, suppose a university assigns an odd number to a male student and an even number to a female student. However, on the global side, the users might not know this. Therefore, the data source in the university can use a Hint attribute to indicate the mapping function. Suppose

that the Property Function in the Instance Function is gender*number(123456), which can be inferred to gender(female), we can put it in the XML format in the following:

<gender>
    <number Hint= "odd: male, even: female">123456</ number>
</gender>

The above example is about Precedence Function. Certainly, we can also add other HINT attributes to express useful description about properties, including Property Aggregation, to facilitate information exchange and integration. For example, we can put into one HINT attribute the statement that if somebody has different academic degrees the highest one prevails.

Before finishing this section, we want to conclude the usage of XML in expressing Instance Functions into five expression rules:

1. The root element represents a set of things or instances.
2. An element at the second level represents an instance.
3. The lower elements represent generic properties (including Identity Property).
4. The content of an element is the value of the generic property that the element represents.
5. Attributes provide additional information about the corresponding elements to indicate the rules of integration.

6.4 Summary

In this chapter, we discussed three issues of the Property-Based Approach. First, we presented a method to build a Property-Based view on a Classed-Based data sources. Second, we presented the way to express a Property Precedence Schema (PPS). Finally, we used XML to express Instance Functions and suggested some expression rules.

Chapter Seven

THE INTEGRATION PROCESS AND A CASE STUDY

Up to now, we have already discussed the representation models, with which we can model a domain into the Property-Based Model. We have formalized and presented the notation to build an IS ontology, or data schemas (PPS), upon both local data sources and the global side. We have formalized and defined Instance Function to facilitate the information integration from multiple sources. We also have discussed some application issues.

In this chapter, we provide a general idea of the integration process in the Property-Based Approach by using all we have discussed, including PPSs and Instance Function. Generally, the integration process in the Property-Based Approach is to build Instance Functions according to both global and local PPSs.

Along with the introduction of the integration process, we conduct a case study, which can help readers to understand the integration process. Therefore, we first introduce the background of the case and then introduce and demonstrate the integration process.

7.1 The Background of the Case

The case is extracted from a real business situation, where fourteen independent airlines try to cooperate with each other under the Star Alliance (SA) by combining their Frequent Flyer Programs (FFPs). For the purpose of this thesis, we made some modifications on their requirements of cooperation, which make the case more interesting. To simplify the case, we only select two FFPs, Aeroplan of Air Canada (AC) and Mileage Plus of United Airlines (United), as the

subjects of the case study. According to their business information, we create one mini-RDB (where there is only one relation) for each of them, which we believe is enough to demonstrate the Property-Based Approach.

AC and United have their own FFPs. A flyer's status in a FFP is determined by two factors: mileage and the number of segments (one-way). For example, in AC's Aeroplan, if a flyer has more than 100,000 miles or more than 150 segments, his status will be Super Elite. In United's Mileage Plus, if a flyer has more than 100,000 miles or more than 100 segments, his status will be Premier Executive 1K. Table 7-1 shows the details.

FFP's Miles or Segments (one-way)	FFP Status
>=100,000 miles or >=150 segments	Aeroplan Super Elite
(<100,000 and >=35,000 miles) or (<150 and >=60 segments)	Aeroplan Elite
(<35,000 and >=18,000 miles) or (<60 and >=30 segments	Aeroplan Prestige
all the rest	Aeroplan Normal
>=100,000 miles or >=100segments	United's Premier Executive 1K
(<100,000 and >=50,000 miles) or (<100 and >=60 segments)	United's Premier Executive
(<50,000 and >=25,000 miles) or (<60 and >=30 segments)	United's Premier
all the rest	United's Normal

Table 7-1: Determine FFP Status by FFP's Miles or Segments

A flyer that holds a status in a FFP can enjoy some privilege services. For example, a flyer with Aeroplan Super Elite can access the lounge. Table 7-3 shows the details. Before it, Table 7-2 shows the abbreviation we use for these privilege services.

Privilege Services	Abbreviations
Lounge Access	LA
Extra Baggage Allowance	EBA
Priority Airport Check-in	PAC
Priority Baggage Handling	PBH
Priority Boarding	PB
Priority Reservation Waitlist	PRW
Priority Airport Standby	PAS

Table 7-2: Abbreviations of Privilege Services

FFP Status	FFP Privilege
Aeroplan Super Elite	LA, EBA, PAC, PBH, PB, PRW, PAS
Aeroplan Elite	LA, EBA, PB, PRW, PAS
Aeroplan Prestige	PB, PRW, PAS
Aeroplan Normal	none
United's Premier Executive 1K	LA, EBA, PAC, PBH, PB, PRW, PAS
United's Premier Executive	LA, EBA, PB, PRW, PAS
United's Premier	PB, PRW, PAS
United's Normal	none

Table 7-3: Determine FFP Privilege by FFP Status

The airlines cooperate with each other in the SA network by first converting a flyer's FFP status to a Star Alliance Status, which can be Gold or Silver. For example, both Aeroplan's Super Elite and United's Premier Executive 1K will be converted to Star Alliance Gold. Table 7-4 shows the details.

FFP Status	SA Status
Aeroplan Super Elite	Gold
Aeroplan Elite	Gold
Aeroplan Prestige	Silver
United's Premier Executive 1K	Gold
United's Premier Executive	Gold
United's Premier	Silver

Table 7-4: Determine SA status by FFP Status

After determining the Star Alliance Status of a flyer from another FFP, we can decide which kind of privilege services should be provided when the flyer flies within the Star Alliance. For example, a flyer with SA Gold will have the privileges of Lounge, while a flyer with SA Silver cannot. Table 7-5 shows the details. (We modify a little in what privileges Star Alliance provides.)

SA Status	SA Privilege
Gold	LA, EBA, PAC, PRW, PAS
Silver	PRW, PAS

Table 7-5: Determine Privilege Service by SA Status

After determining the privilege of the Star Alliance, one airline can decide which kind of privilege services should be provided when the flyer flies with the airline. Usually, the airline should provide the same privilege services, if a flyer enjoys the privileges of the Star Alliance. For example, a flyer with the privileges of Lounge Access of the Star Alliance will also have the privilege to access the lounge of United Aairlines. Table 7-6 show the details.

SA Privilege	FFP Privilege
Lounge Access	Lounge Access
Extra Baggage Allowance	Extra Baggage Allowance
Priority Airport Check-in	Priority Airport Check-in
Priority Reservation Waitlist	Priority Reservation Waitlist
Priority Airport Standby	Priority Airport Standby

Table 7-6: Determine FFP Privilege by SA Privilege

We now assume that a flyer who takes part into multiple FFPs can combine all the miles or segments of these FFPs to be able to get a higher status of Star Alliance than he could get through any of his FFPs. In this case, we assume that 1 mile in Aeroplan is equal to 1 mile in SA while 1.2 mile in United is equal to 1 mile in SA. To facilitate this feature, two concepts have to be added: SA_mileage and SA_segment, which become the connection between Star Alliance Status and mileage or segment of a FFP. Table 7-7 shows the details. (Please notice that this feature is not supported in the current Star Alliance. We changed it to make the case more interesting.)

SA Status	SA_mileage or SA_segments
Gold	>=35,000 miles or >=60 segments
Silver	>=18,000 miles or >=30 segments

Table 7-7: Determine SA Status by SA_mileage or SA_segments

Both AC's Aeroplan and United Mileage Club maintain a RDB to store information about their flyers. Let us suppose further that their RDBs would be in the following:

AC's Aeroplan:

Member(mem_no, mem_name, stat_mil, stat_seg);

United's Mileage Plus:

Customer(num, name, mile, segment).

The definitions of the attributes in the two tables are quite straightforward, so we do not list them here in the details. Nevertheless, we list all of the properties in the case in Appendix A, which includes the properties that these attributes represent.

7.2. The Preparation for Integration

To support the Property-Based Approach to integrating information, we need: 1) two property-based views on the two data sources; 2) one local PPS for each airline and 3) a global PPS for the Star Alliance.

7.2.1 The Property-Based Views

The following two tables are the two possible Property-Based views. We will use the attributes on the left side in the case study.

The Property-Based Model	The Class-Based Model
member(Null)	the class of Member
identity( )	num( )
name( )	mem_name( )
mileage( )	status_mil( )
segment( )	status_seg( )

Table 7-8: The Property-Based View for Aeroplan

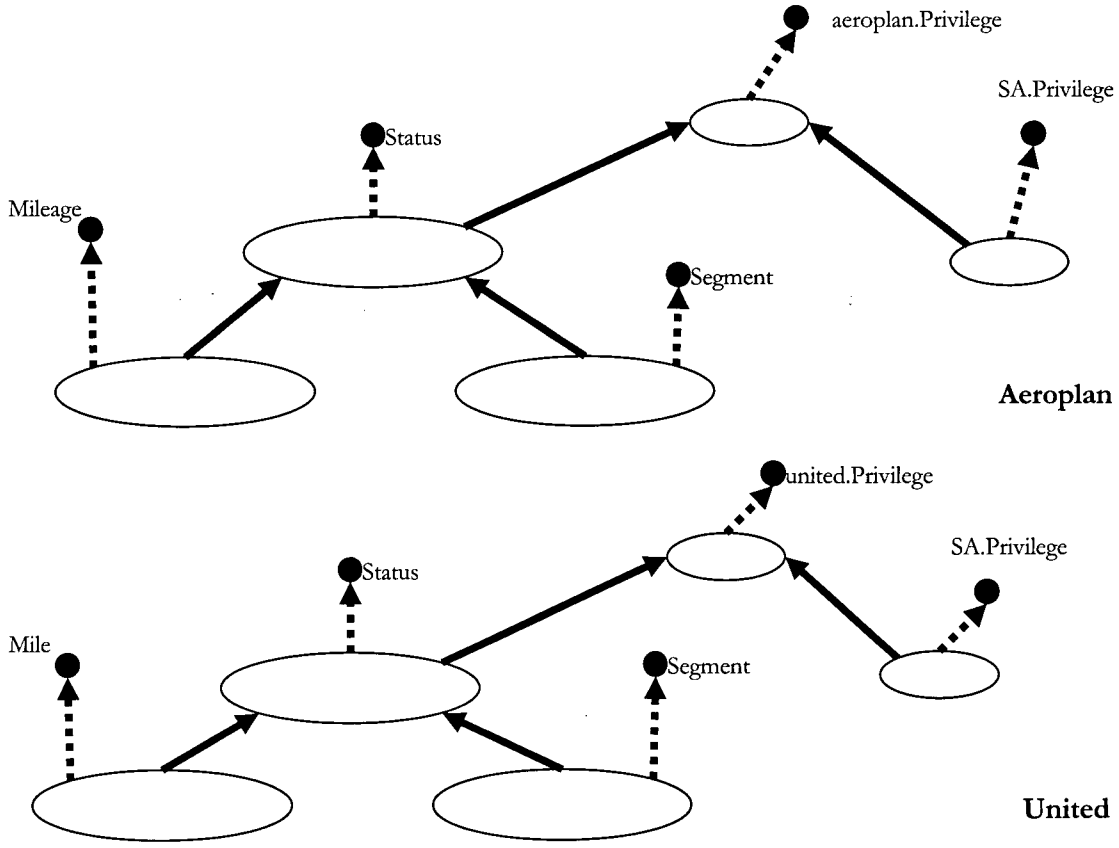
The Property-Based Model	The Class-Based Model
customer(Null)	the class of customer
identity( )	mem_no( )
name( )	name( )
mile( )	mile( )
segment	segment( )

Table 7-9: The Property-Based View for United

7.2.2 Two local PPSs

According to the information provided in Section 7.1, we draw two PPSs for Aeroplan and United respectively. Appendix B shows the corresponding Precedence Tables. In the interest of brevity, we do not indicate the value sets in the PPSs, which are self-evident. In Appendix A, we can find the definition of properties used in the case. Please refer to Section 5.1.1 for the meaning of the legends used in the following diagram (Figure 7-1).

Figure 7-1: Two Local PPSs in Aeroplan and United



On the left of each PPS, there are three uncontained nodes, which represent three generic properties while there are three ovals, which represent three Standard Sets of Value of these three generic properties. The two arrows between ovals represent two mapping functions between two

Standard Sets of Value or Precedence Functions. Therefore, there is no need to express single precedence in the situation where we have Precedence Functions, such as Aeroplan Super Elite preceding more than 100,000 miles.

Similarly, the right of each PPS expresses a Precedence Function from Star Alliance privileges to privileges of local airlines, Aeroplan and United. Please notice this is the situation where properties in a local PPS (United) precede properties in a global PPS (Star Alliance) (also see the discussion in Section 5.2.1).

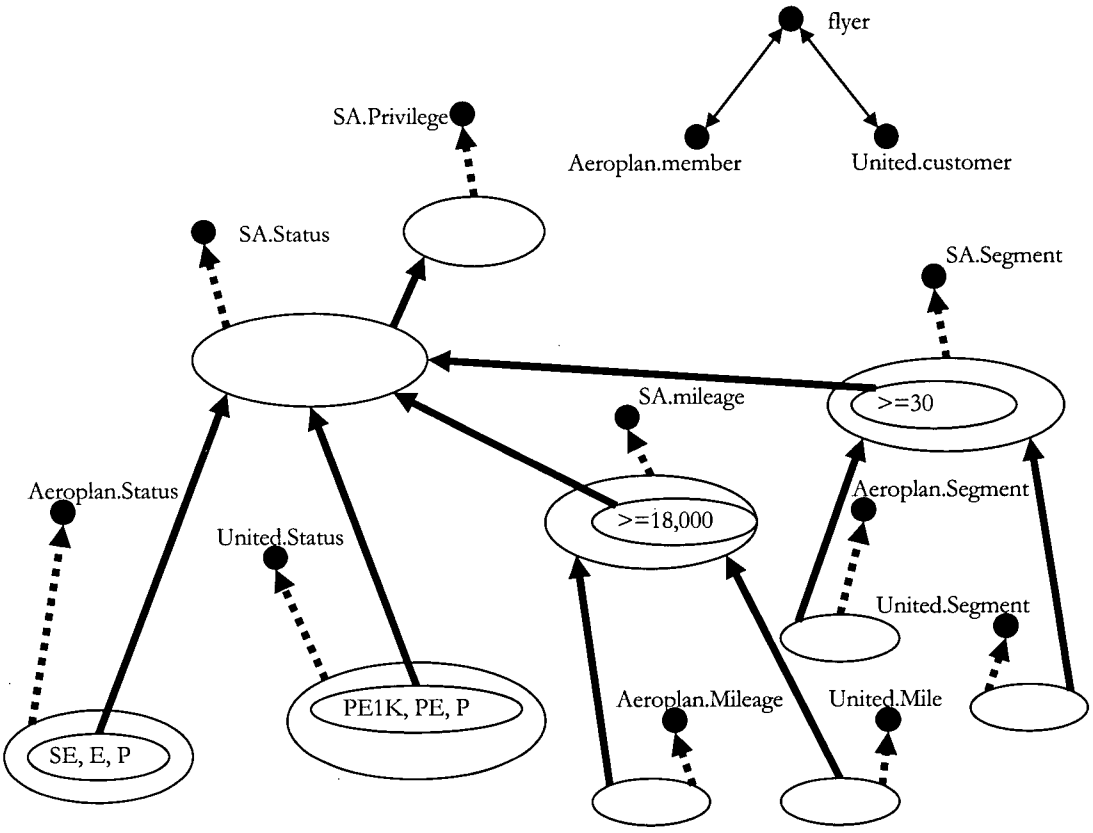
7.2.3 The Global PPS

We draw the Global PPS for the case in Figure 7-2 according the information provided in the section 7.1 with the information provided in Appendix A. The corresponding Precedence Tables are shown in Appendix B.

At the top of the Global PPS, there are three nodes connected by two-headed arrows. As we discussed in Section 5.1.3, they are concomitant properties and the precedences are co-precedences. For example, a member in Aeroplan must be a flyer in Star Alliance and vice versa.

One thing that should be emphasized is that some details are not shown in both the local PPSs and the global PPS. They are compressed into Precedence Functions, which are listed in Precedence Tables in Appendix C. These Precedence Functions are illustrated in bold arrows. Although we do not list all precedences in detail in the diagram, we can still get the idea about the way the properties in the domain precede others.

Figure 7-2: The Global PPS in Star Alliance



The above PPS shows how a PPS illustrates the relevant semantics of properties clearly and effectively. Taking *United Mile* for an example, we know that miles in United also mean miles in Star Alliance, which could imply some status of Star Alliance. Then a SAstatus determines the privilege services that should be provided. Thus, we learn the semantics that some miles in United may help a flyer earn some kinds of privilege. If we put all such kinds of semantics into words, we could have several pages and eventually the semantics would be buried into words.

7.3 The Integration Process

We first list the main steps in the integration process in the Property-Based Approach. There are two procedures:

1. The main procedure, which controls the whole process of query on the query side and build up an Instance Function for each relevant instance using the data retrieved from data sources, and

2. The sub-procedure, which is a recursive function to build up a Property Tree for a property according to the global PPS and/or the local PPSs.

The sub-procedure is a recursive sub-procedure that is called by both the main procedure and the sub-procedure itself. These two procedures are very general and quite inefficient when implementing them as is. More optimization problems in implementation should be researched before we can use this general algorithm in the Property-Based Approach. The demonstration of the following two procedures will be in the next section with a case study. Therefore, readers may wish to read these two sections together.

Main Procedure of Integration:

1. A user express a query in the form of a basic Instance Function: PATTERN$\{P_1(x_1), \ldots, P_n(x_n)\}$, where $n \geq 1$, $P_1, \ldots, P_n$ are BAs, $x_1, \ldots, x_n$ are values or identities. We call it a Pattern Function, which is in fact to request a class (a set) of instances that possess all these properties.
2. For each $P_i(x_i)$, $i=1, \ldots, n$, run the sub-procedure of building Property Trees for it; (Please notice that during the procedure, Property Trees are always bound together

with identities: INSTANCE{ID(id), PT_{ij}}, where i=1, …, n, j=0, …, m, m is the number of things possess the property represented by $P_i(x_i)$ in all data sources.)

3. According to identities, put properties under one INSTANCE{} if they are about the same instance. For each an instance, put together property trees if they are about the same $P_i(x_i)$ by using the Conjugation Feature of Instance Function.
4. For each instance, use the inference model of precedence and property aggregation to computer x_i.

Sub-procedure of building Property Functions

1. After receiving P, and according to both the global and the local PPSs, get all the preceded properties $P_1(x_1)$, …, $P_n(x_n)$ the property P(Null) might have, and put them in the form of $P^*[P_1(x_1), \ldots, P_n(x_n)]$;
2. for each $P_i(x_i)$:
   a. if $P_i(x_i)$ does not have preceded properties, retrieve all instances that satisfy $P_i(x_i)$ from the corresponding data source;
   b. run the sub-procedure of building Property Trees for $P_i(x_i)$;
3. For each instance retrieved from data sources, return an instance function with identities and properties retrieved, INSTANCE {ID(id), $P^*[P_1(x_1), \ldots, P_n(x_n)]$}.

7.3.1 Demonstrating the Integration Process with the Case

Suppose that we want to know what kind of privileges a flyer could enjoy in United Airline. Some sample data in the two FFPs is listed in the Table 7-10a and 7-10b.

mem_no	mem_name	stat_mil	stat_seg
1234	B. Gates	18000	20
5678	J. Bond	10000	5

Table 7-10a: Data in the local sources (Aeroplan)

num	name	mile	segment
1111	Bill Gates	21000	15
2222	James Bond	24000	8

Table 7-10b: Data in the local sources (United)

We demonstrate the integration process in the following tables. In the tables, main.1 means the first step of the main procedure.

Step#	In the Case	Comments
main.1	Pattern{customer(Null), united.privilege(Null)}	The pattern function means we want to know anyone who is a customer in United Airline and his privileges.
main.2	Instance{ID(united.customer.num.1111), United.privilege*SA.privilege*SA.status*SA.mileage* united.mile (21000)}. ***(illustrated in Figure 7-3)***	For united.privilege(Null), we call the sub procedure, i.e. sub1, which is demonstrated in Table 7-12 and get the results on the left . All the results are returned by sub1 while only the first two are demonstrated in the Table 7-12. We only list some of the results, which are about one person. We will use these results to demonstrate the following steps.
	Instance{ID(united.customer.num.1111), United.privilege*SA.privilege*SA.status*united.status *united.mile (21000)}.	
	Instance{ID(united.customer.num.1111), United.privilege*SA.privilege*SA.status*SA.segment* united.segment (15)}.	
	Instance{ID(united.customer.num.1111), United.privilege*SA.privilege*SA.status*United.status *united.segment (15)}.	
	Instance{ID(united.customer.num.1111), united.privilege*united.status*united.mile(21000)}.	
	Instance{ID(united.customer.num.1111), united.privilege*united.status*united.segment(5)}.	

Table 7-11: Main Procedure (continued)

	Instance{ID(Aeroplan.member.member_no.1234), United.privilege*SA.privilege*SA.status*SA.mileage* Aeroplan.mileage (18000)}.	
	Instance{ID(Aeroplan.member.member_no.1234), United.privilege*SA.privilege*SA.status*Aeroplan.status*Aeroplan.mileage (18000)}.	
	Instance{ID(Aeroplan.member.member_no.1234), United.privilege*SA.privilege*SA.status*SA.segment* Aeroplan.segment (20)}.	
	Instance{ID(Aeroplan.member.member_no.1234), United.privilege*SA.privilege*SA.status*Aeroplan.status*Aeroplan.segment (20)}.	
	Instance{ID(united.customer.num.1111), customer(Null)}.	For customer(Null), we call another sub-procedure and get the results on the left.
	Instance{ID(Aeroplan.member.member_no.1234), customer*SA.flyer*Aeroplan.member(Null)}.	
main.3	Instance{ID[united.customer.num.1111, Aeroplan.member.member_no.1234], customer[Null, *SA.flyer*Aeroplan.member(Null)], United.privilege[*united.status[*united.mile(21000), *united.segment(5)], *SA.privilege*SA.status[*SA.mileage[*united.mile(21000), *Aeroplan.mileage(18000)], *united.status[*united.mile(21000), *united.segment(15)], *SA.segment[*united.segment(15), *Aeroplan.segment(20)], *Aeroplan.status[*Aeroplan.mileage(18000), *Aeroplan.segment(20)]]]}.	Suppose we know united.customer.num.1111 and Aeroplan.member.member_no.1234 represent the same person, while the other two are about different persons. There are other two instance functions, which we do not list here for brevity. ***(illustrated in Figure 7-4)***

Table 7-11: Main Procedure (continued)

main.4	Instance{ID[united.customer.num.1111, Aeroplan.member.member_no.1234], customer (Null), United.privilege[*United.status[normal, normal], SA.privilege*SA.status[*SA.mileage[17500, 18000], *united.status[normal, normal], *SA.segment[15, 20], *Aeroplan.status[Presitge, Normal]]]}. ***(illustrated in Figure 7-5)***	By using the result in main.3, We list four middle steps along with the inference of precedence and property aggregation.
	Instance{ID[united.customer.num.1111, Aeroplan.member.member_no.1234], customer(Null), United.privilege[*SA.privilege*SA.status[*SA.mileage (35000), *united.status (normal), *SA.segment (35)), *Aeroplan.status(Presitge)]]}. ***(illustrated in Figure 7-6)***	
	Instance{ID[united.customer.num.1111, Aeroplan.member.member_no.1234], customer(Null), United.privilege*SA.privilege*SA.status[Gold, Silver, Silver]}. ***(illustrated in Figure 7-7)***	
	Instance{ID[united.customer.num.1111, Aeroplan.member.member_no.1234], customer(Null), United.privilege[LA, EBA, PAC, PRW, PAS]}. ***(illustrated in Figure 7-8)***	

Table 7-11: Main Procedure

The following table include four sub-procedures, which are four invocations of the same procedure, but are called by different procedures. They are in fact recursively called from main to the first invocation, from the first to the second, from the second to the third, and from the third to the last one in one recursion.

First invocation of Sub procedure: (called by main)		
Step#	**In the Case**	**Comments**
1	united.privilege*SA.privilege(Null)	SA.privilege → united.privilege
2	Instance{ID(united.customer.num.1111), SA.privilege*SA.status*SA.mileage*united.mile (21000)}.	Because SA.privilege is preceded by other properties, call a new sub-procedure for SA.privilege(Null), i.e. ***the second***, and get the result shown on the left side
	Instance{ID(united.customer.num.2222), SA.privilege*SA.status*SA.mileage*united.mile (20000)}.	
3	Instance{ID(united.customer.num.1111), United.privilege*SA.privilege*SA.status*SA.mileage *united.mile (21000)}.	Return the results on the left side to main. The first Instance Function is also ***illustrated in Figure 7-3***.
	Instance{ID(united.customer.num.2222), United.privilege*SA.privilege*SA.status*SA.mileage *united.mile(20000)}.	
Second Invocation of Sub procedure: (called by the first)		
Step#	**In the Case**	**Comments**
1	SA.privilege*SA.status(Null))	SA.status → SA.privilege
2	Instance{ID(united.customer.num.1111), SA.status*SA.mileage*united.mile(21000)}.	Because SA.status is preceded by other properties, call a new sub-procedure for SA.status (Null), i.e. ***the third***, and get the result on the left side.
	Instance{ID(united.customer.num.2222), SA.status*SA.mileage*united.mile(20000)}.	
Table 7-12: Sub-Procedures (continued)		

3	Instance{ID(united.customer.num.1111), SA.privilege *SA.status*SA.mileage*united.mile(21000)}.	Return the results on the left side to the first.
	Instance{ID(united.customer.num.2222), SA. priviledge*SA.status*SA.mileage*united.mile(20000)}.	

Third Invocation of Sub procedure: (called by the second)

Step#	In the Case	Comments
1	SA.status[*aeroplan.status(Null), united.status(Null), *SA.mileage(Null), *SA.segment(Null)]	aeroplan.status → SA.status united.status → SA.status SA.mileage → SA.status SA.segment → SA.status
2	Instance{ID(united.customer.num.1111), SA.mileage*united.mile (21000)}.	We select SA.mileage as the example. Because SA.mileage is preceded by other properties, call a new sub-procedure for SA.mileage(Null), i.e. the fourth, and get the result on the left side.
	Instance{ID(united.customer.num.2222), SA.mileage*united.mile(20000)}.	
3	Instance{ID(united.customer.num.1111), SA.status*SA.mileage*united.mile (21000)}.	return the results on the left side to the second.
	Instance{ID(united.customer.num.2222), SA.status*SA.mileage*united.mile(20000)}.	

Fourth Invocation of Sub procedure: (called by the third)

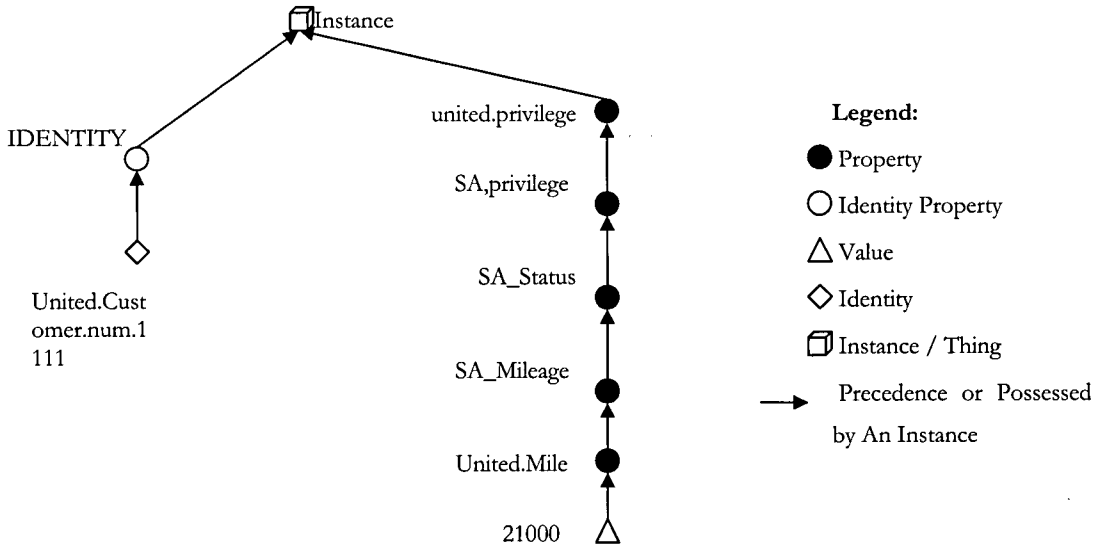
Step#	In the Case	Comments
1	SA.mileage[*aeroplan.mileage(Null), *united.mile(Null)]	aeroplan.mileage → SA.mileage united.mile → SA.mileage
2	united.mile(21000)) with the identity, united.customer.num.1111.	We select united.mile as an example. Because united.mile does not have preceded properties, retrieve the result on the left from the RDB in United.
	united.mile(20000)) with the identity, united.customer.num.2222.	

Table 7-12: Sub-Procedures (continued)

3	Instance{ID(united.customer.num.1111), SA.mileage*united.mile (21000)};	Put the above data into Instance Function and with identities. Return the results on the left side to the third.
	Instance{ID(united.customer.num.2222), SA.mileage*united.mile(20000)}	

Table 7-12: Sub-Procedures

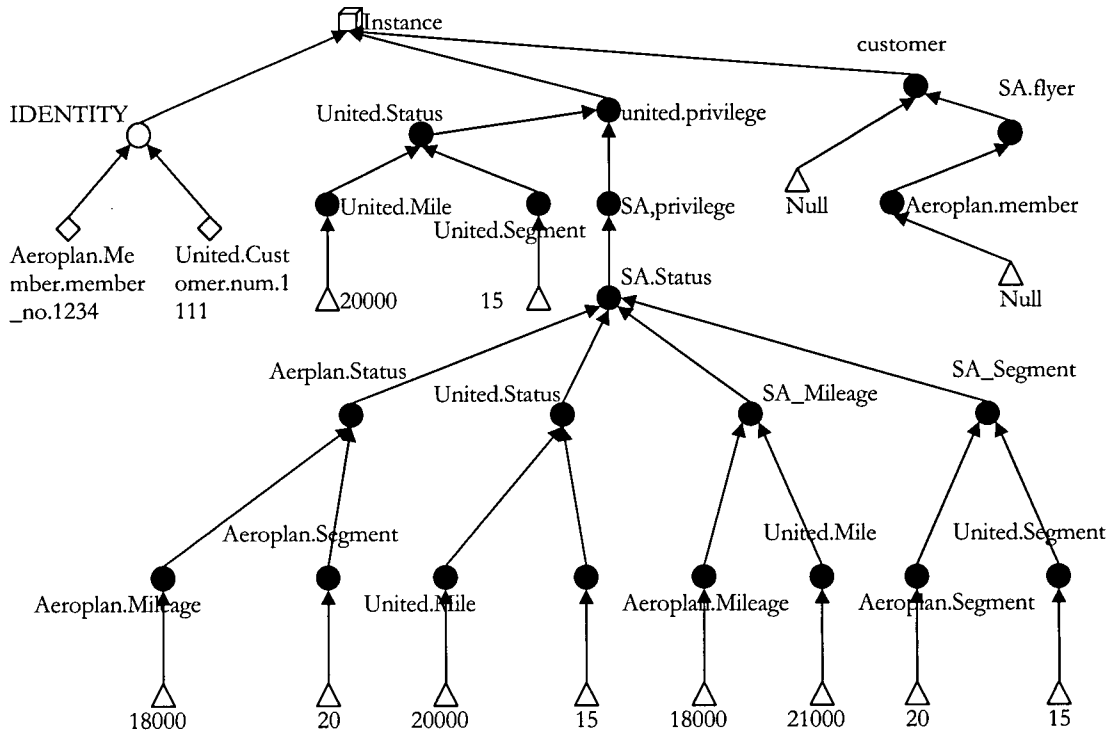
Figure 7-3: An Instance Function for main.2



After we retrieve all the information in two data sources, and put the information about the same thing together, we will have completed Instance Functions about the potential qualified instances, one of whose Instance Function is shown in Figure 7-4. We also express the Instance Function in XML syntax in Appendix C.

Based on these completed Instance Functions, we use the inference of precedence and property aggregation (Section 5.3) to compute the results. We provide some examples to show the process.

Figure 7-4: An Instance Function for main.3



For example, the person has 18,000 Aeroplan mileages, which means his Aeroplan Status is Prestige according to the local PPS of Aeroplan. He has 21,000 United Mile, which means that he has 17,500 SA.Mileage (According to the Precedence Function, 21000/1.2=17500). We can also infer other properties by the same method. The results are shown in Figure 7-5.

Figure 7-5: The First Middle Result of main.4

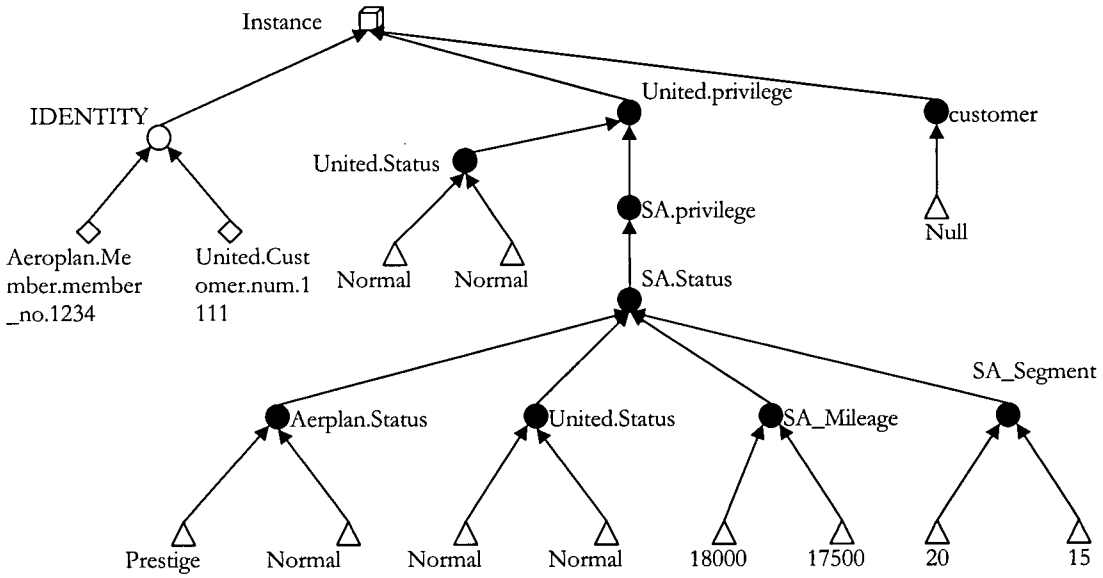
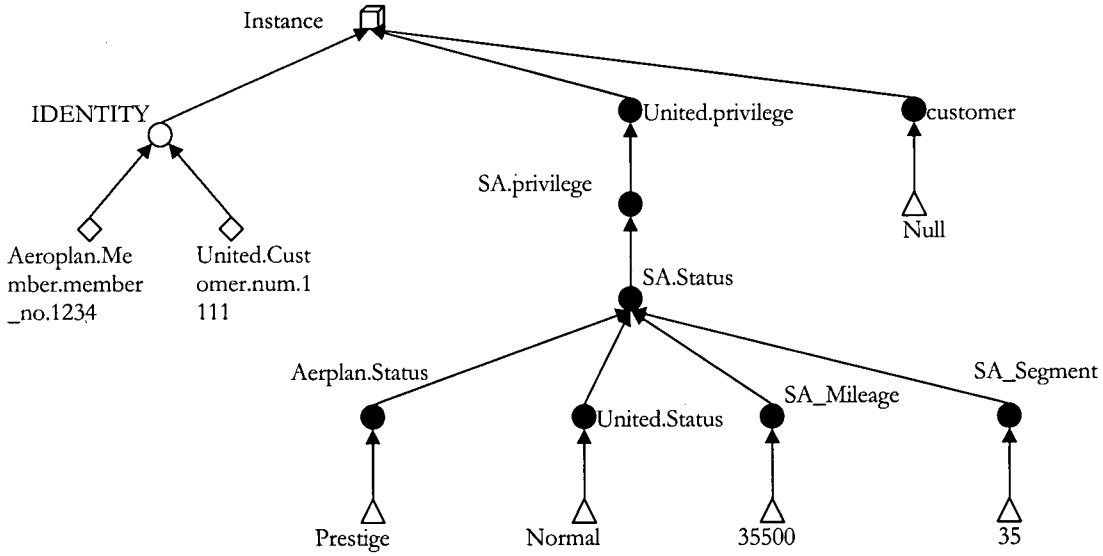


Figure 7-6: The Second Middle Result of main.4



In the Figure 7-5, we can see that the person has two manifestations of Aeroplan Status, Prestige and Normal. According the aggregation rule of Aeroplan.Status, we know his Aeroplan is Prestige,

which means that he holds a SA.Status of Silver. He also two values of SA.Mileage, 18,000 and 17500. According the aggregation rule of SA.mileage, we add them up and get 35,500 SA.Mileage, which in turn means that the person's SA.Status is Gold. The same is with other properties. Therefore, we have the result shown in Figure 7-6 and Figure 7-7.

Figure 7-7: The Third Middle Result of main.4

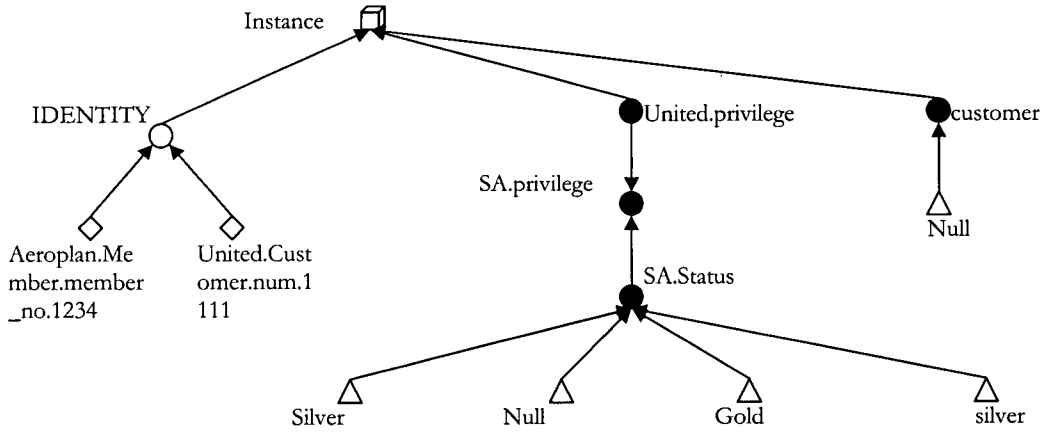
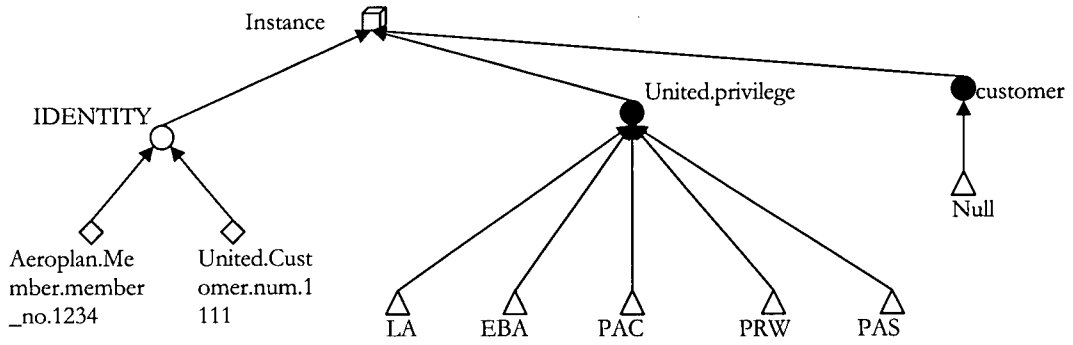


Figure 7-8: The Fourth Middle Result of main.4



Finally, the person has three manifestations of SA.Status. According to the aggregation rule of SA.Status, we choose the highest one, which is Gold. Since the person's SA.status is Gold, he can enjoy all of the privilege available in Star Alliance. However, when he flies with United Airline, he

cannot enjoy the privilege Priority Boarding because United Airline does not provide the privilege. The result is shown in Figure 7-8.

Similarly, we also get the other two Instance Functions, one of which is discarded because it does not satisfy the requirement specified in the Pattern Function.

7.3.2 Discussion

In the case, the query actually starts from a local side (United), then goes to the global side (Star Alliance) and, through the global side, goes to another local side (Aeroplan). After reaching relevant information, the results are returned along the way back. We may think that a normal integrating query should start from the global side to the local sides if we want to integrate the information from the local sides, so we may say the query in the case actually contains a normal query. In fact, no matter where a query starts, it will follow the precedences expressed in both global PPS and local PPSs to look for relevant information.

This leads to another observation. The difference between a global side and a local side is not absolute but relative. Every side could be a global side of other sides if some properties on the side precede some properties from others, while every side could be a local side of other side too if some properties on the side are preceded some properties from others.

Thus, other approaches that treat global sides and local sides strictly may become complicated in such situations. They are just too rigid and not as flexible as our approach.

In the case, we want to know the privileges that a specific flyer can enjoy. Much information, some of which is even not from the local domain, can influence the property, or in other words, has the

semantics of having privileges. We can surely retrieve much information from some data sources, but the problem is how we know the information has the semantics that we want.

This is where precedences help us out. Thus, we know the mileage of one flyer in Aeroplan also have some semantics in the privilege of United. In fact, one piece of information has several different semantics according to the same property. For example, mileage in United has three semantics through three routes in Figure 7.5. Furthermore, several semantics can be aggregated into a new semantics. For example, mileage in SA comes from mileages in both United and Aeroplan.

We can see that our concept of semantics has a broader sense than that of most of other researchers in Information Systems. We understand semantics as what other properties a property implies (precedence) while others understand it as its superficial meaning. It is their limited sense of Semantics that makes their approach limited and rigid. We adopt the broad sense of semantics and develop a much more comprehensive and flexible approach to achieving Semantic Interoperability.

7.4 Summary

In this chapter, we introduced a general process of integration in the Property-Based Approach along with a case study. The case illustrated the entire process of integrating information from two data sources. We hope that the integration process and this case have given readers a comprehensive impression of the methodology and soundness of the Property-Based Approach.

Chapter Eight

CONCLUSION AND FUTURE RESEARCH

This thesis aims at providing a practical approach to integrating information from multiple data sources. To achieve this objective, we started the research from theoretical foundations rather than from the practical and implementation aspects.

The theoretical foundation is Conceptual Modelling, that is, a set of theories about a proper conceptual model of information. Since the information in one domain is our knowledge about the domain, we are looking for a proper conceptual model of knowledge, which belongs to the arena of Ontology. A specific ontology, the BWW-Ontology, has been adapted in the field of Information Systems for more than a decade. In this thesis, we used the simple conceptual model suggested in the BWW-Ontology and some research results developed on the BWW-Ontology to formalize such a general approach, which we term the Property-Based Approach.

We first reviewed the relevant research areas in the fields of Information Integration, such as Conceptual Modelling, Knowledge Representation, Knowledge Discovery, Semantic Heterogeneity and Semantic Interoperability. In the review, two conceptual models were discussed. One is the current dominant model, which we call the Class-Based Model. The other is suggested in the BWW-Ontology, the Property-Based Model.

Then, we introduced the BWW-Ontology and some of its constructs. Most importantly, we presented the core concept of the Property-Based Approach: Precedences between Properties. In

addition, we discussed the meaning of two concepts, queries and information integration, and explained their relevance in terms of the BWW-Ontology.

In Chapter 4, we presented the basic part of Property-Based Model, which includes representation models of things and properties. For the purpose of this thesis, we chose Basic Attribute to represent properties. We also defined formally the semantics of properties in the Property-Based Model.

In Chapter 5, we presented the Property-Based Approach to integrating information from multiple sources. The approach included two major parts. One is how to express precedences and the other is how to facilitate the integration process. For the first part, we provided formal notations to describe precedences in the Property Precedence Schema (PPS) based on these notations. For the second part, we formalized and defined Instance Functions to facilitate the integration process.

Next, some practical issues have been considered. One issue is how to transfer information in the Class-Based Model into information in the Property-Based Model. The next two issues were implementing PPSs and Instance Functions.

Finally, a general integration procedure was introduced by using the Property-Based Approach. In addition, a case study was provided to demonstrate the procedure.

8.1 Contribution

This thesis has made the following contributions:

1. We formalized the basic part of a conceptual model called the Property-Based Model in Information Systems based on in the BWW-Ontology. The model is different from the

traditional model, the Class-Based Model. The main difference is that things are represented before classification in the Property-Based Model while things are represented through classes in the Class-Based Model. The main idea was suggested by Parsons and Wand (2000) initially, and we elaborated on it in detail. We also suggested a way to transform the data in the Class-Based Model to the data in the Property-Based Model.

2. The major contribution of this thesis is the development of a practical approach to integrating information from multiple sources, based on *Property Precedence* originally proposed by Parsons and Wand (2002). We also introduced another concept, *Property Aggregation*. *Property Precedence* helps us to find the same general semantics of two properties while *Property Aggregation* to aggregate two semantics of two properties into one.

The approach includes two main parts. The first part is to provide a model to define semantics of properties, which can be used in other areas, such as Semantic Web. The second part related to the integration process. We formalized the process by defining Instance Functions, which connect related properties together under the things possessing them, to facilitate the integration process. An advantage of Instance Functions is that its structure is the same as the Semi-Structured Data, therefore it can be implemented easily. An Instance Function is composed of Property Functions, which utilizes the ideas of *Property Precedence* and *Property Aggregation* to achieve Semantic Interoperability.

8.2 Limitations

We think there are several limitations in this thesis.

1. We have not developed completely the Property-Based Approach to integrating information from multiple sources. We only provide a basic framework, such as Representation Models, Property Precedence Schema and Instance Functions. We only present the general procedure to integrating information about things. We did not mention the procedures of associating things and composing new things. More research on the features of association and composition of things is needed to formalize these procedures.

2. The general procedure to use the Property-Based Approach to integrating information from multiple sources is very inefficient right now. More optimization algorithms should be developed before we can apply the Property-Based Approach in real usage.

8.3 Future Research

Some future research can be done to continue the development of the Property-Based Approach and address some of the above-mentioned limitations. First, the features of association and composition must be analyzed and the results would be a basis for formalizing our related concepts and procedures. These concepts and procedures should also be consistent with those we have developed in this thesis. Moreover, most importantly, they should conform to the BWW-Ontology and the Property-Based Model.

Second, while the optimization issues are mostly within the scope of Computer Science, we can actually conduct some more case studies to facilitate the development of optimization algorithms. These cases should be from real situations, rich enough and complicated enough to uncover some potential inherent problems in the Property-Based Approach.

Furthermore, we believe that this thesis plus other research results (Wand et al., 1999; Parson and Wand, 2000; Parson and Wand, 2002) is just the beginning in adapting the BWW-Ontology into the field of Databases, Knowledge Representation, and Knowledge Discovery and further into the whole field of Information Systems.

Three research directions are identified. The first research direction is about the implementation of Property-Based Databases (PBDBs). Research topics can include storage design, developing a query language, query optimization, and other issues related to databases. The implementation of a PBDB can also contain the implementation of a PPS. These two compose an Instance and Property Base, which facilitate the Class Base (Parsons and Wand, 2000). These topics are worthy of cooperative research projects between MIS and Computer Science.

The second research direction is about the representation of properties to facilitate Semantic Integration. We have shown that the semantics of Basic Attributes (BAs) can be easily integrated in this thesis. However, it is not clear whether BAs are sufficient for us to represent mutual properties directly and indirectly. There are at least two possible ways to make the Property-Based Approach complete. One is to use only BAs and look for an ontologically correct method to transform between BAs and other kinds of attributes. Such a method should be able to disassemble other attributes into BAs and retain their semantics when reassembling BAs into the original attribute. The other way is to represent properties directly using all types of attributes, such as unary, binary, and high-order attributes, and then we need further research on how to incorporate all attributes other than BAs into the Property-Based Approach.

BIBLIOGRAPHY

Berners-Lee, T. & Fischetti, M. 1999. Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor, Harper, San Francisco.

BUNGE, M. 1977. *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World.* D. Reidel Publishing Co., Inc., New York, NY.

BUNGE, M. 1979. *Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems.* D. Reidel Publishing Co., Inc., New York, NY.

CHEN, P. P. 1976. The entity-relationship model: Toward a unified view of data. *ACM Trans. Database Syst.* 1, 1, 9–36.

COAD, P. AND YOURDON, E. 1991. *Object-Oriented Analysis.* 2nd ed. Yourdon Press Computing Series. Yourdon Press, Upper Saddle River, NJ.

Davis, R., Shrobe, H., and Szolovits, P. 1993. What is a Knowledge Representation? *AI Magazine*, 14(1):17-33, 1993.

Evermann, Joerg and Wand, Yair. 2002. Towards Ontologically Based Semantics for UML Constructs. Conceptual Modeling - ER 2001. Lecture Notes in Computer Science, LNCS 2224, Springer 2002, pp. 254-367.

Frawley, W., Piatetsky-Shapiro, G. and Matheus,C. 1992. Knowledge Discovery in Databases - An Overview. In *KDD collection*, pp. 1--30. Reprinted in AI Magazine, Fall 1992.

Gangemi, A. Pisanelli, DM. and Steve, G. 1999. An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies. *Data and Knowledge Engineering*, 1999, vol.31, pp. 183-220.

Gruber, T. R. 1995. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. Int. J. *Human–Computer Studies*, 43, 907-928.

Guarino, N. 1997. Understanding, Building, and Using Ontologies: A Commentary to *Using Explicit Ontologies in KBS Development.* by van Heijst, Schreiber, and Wielinga. International Journal of Human and Computer Studies, 46: 293-310.

Guarino, Nicola. 1998. Formal Ontology and Information Systems. In Nicola Guarino, editor, *Formal Ontology in Information Systems, Proceedings* of *FOIS'98,* pp. 3--17, Trento, Italy, June 1998. IOS Press, Amsterdam.

Guarino, N. and Welty, C. 2000. Ontological Analysis of Taxonomic Relationships. In, Laender, A. and Storey, V., eds, *Proceedings of ER-2000: The 19th International Conference on Conceptual Modeling.* Springer-Verlag. October, 2000.

Guarino, N. and Welty, C. 2001. Identity and Subsumption. LADSEB-CNR Internal Report.

Guarino, N and Welty, C. 2002. Evaluating Ontological Decisions with Ontoclean. *Communications of the ACM, 2002*, vol.45 (2): 61-65.

Hakimpour, Farshad. & Geppert, Andreas. 2001. Resolving semantic heterogeneity in schema integration: an Ontology Based Approach. In *Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001.*

Halpin, T.A. 2001. *Information Modeling and Relational Databases.* Morgan Kaufmann Publishers.

Heflin, J. and Hendler, J. 2000. Semantic Interoperability on the Web. In *Proceedings of Extreme Markup Languages 2000.* Graphic Communications Association.

LAKOFF, G. 1987. *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind.* University of Chicago Press, Chicago, IL.

Lassila, Ora. Swick, Ralph R. 1999. Resource Description Framework (RDF) Model and Syntax Specification. online. *World Wide Web Consortium.* http://www.w3.org/TR /1999/REC-rdfsyntax- 19990222

Parsons, J. and Wand, Y. 1997. Choosing Classes in Conceptual Modelling. *Communications of the ACM*, 40(6), 63-69.

Parsons, J. and Wand, Y. 2000. Emancipating Instances from the Tyranny of Classes in information Modeling. *ACM Transactions on Database Systems*, 25(2), 228-268.

Parsons, J. and Wand Y. 2002. Property-based Semantic Reconciliation of Heterogeneous Information Sources. To be appeared in *ER2002*, Tampere, Finland.

Pottinger, Rachel and Levy, Alon. 2000. A Scalable Algorithm for Answering Queries Using Views. *Proceedings of the 26th VLDB Conference,* Cairo, Egypt, 2000.

Rubin, E. 2002. *A Data Model.* Master Thesis. UBC.

RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F., AND LORENSEN, W. 1991. *Object-Oriented Modeling and Design.* Prentice-Hall, Inc., Upper Saddle River, NJ.

Singh, Lisa. Scheuermann, Peter. & Chen, Bin. 1997. Generating association rules from semistructured documents using an extended concept hierarchy. In *Proceedings of the sixth international conference on Information and knowledge management.*

Stuckenschmidt, H. and Wache, H. 2000. Context Modeling and Transformation for Semantic Interoperability In: *Knowledge Representation meets Databases - Proceedings of the Workshop at ECAI 2000.*

Stuckenschmidt, H. et al. 2000. Catalogue Integration: A Case Study in Ontology-Based Semantic Translation. Online. *On-to-Knowledge.* http://www.ontoknowledge.org/oil/downl/CatIntegr.pdf

Wand, Y . 1989. A Proposal for a Formal Model of Objects. In W. Kim and F. Lchovsky,edit ors, *Object-Oriented Concepts, Languages, Applications and Databases*, pages 537 - 559. ACM Press. Addison-Wesley.

Wand, Y. and Weber, R . 1989. An Ontological Evaluation of Systems Analysis and Design Methods. In E. Falkenberg and P. Lingreen,editors, *Information System Concepts: An In-Depth Analysis.* Elsevier Science Publishers B.V., North-Holland.

Wand, Y. and Weber, R. 1990a. Mario Bunge's Ontology as a Formal Foundation for Information Systems Concepts", in: *Studies in Bunge's Treatise on Basic Philosophy*, G. Dorn and P. Weingartner (eds.), the Poznan Studies in the Philosophy of the Sciences and the Humanities, Rodopi, Amsterdam, 1990, pp. 123-150.

Wand, Y. and Weber, R. 1990b. An Ontological Model of an Information System. *IEEE Transactions on Software Engineering*, Vol. 16, No. 11, November 1990, pp. 1282-1292.

Wand, Y. and Weber, R. 1993. On the Ontological Expressiveness of Information Systems Analysis and Design Grammars. *Journal of Information Systems*, 1993, No. 3.

Wand, Y. and Weber, R. 1995. Towards a Theory of Deep Structure of Information Systems. *Journal of Information Systems.*

Wand, Y. and Wang, R. 1996. Anchoring Data Quality Dimensions in Ontological Foundations. *Communications of the ACM*, 39 (11) November 1996.

Wand, Y., Storey, V. and Weber, R. 1999. An Ontological Analysis of the relationship Construct in Conceptual Modelling. *ACM Transactions on Database Systems*, Vol. 24, No. 4, December 1999, pp. 494-528.

Welty, C. and Guarino, N. 2001. Supporting Ontological Analysis of Taxonomic Relationships. *Data and Knowledge Engineering*39(1), pp. 51-74.

Zuniga, L. Gloria. 2001. Ontology: Its Transformation from philosophy to Information Systems. *FOIS'01*, October 2001, Ogunquit, Maine USA.

APPENDIX A: ALL PROPERTIES IN THE CASE

Property Name	Value Set	Aggregation Rule
Aeroplan.Name	String[1]	No rule[2]
Aeroplan.StatusMileage	positive integer[3]	add together[4]
Aeroplan.StatusSegment	positive integer[3]	add together
United.Name	String	No rule
United.Mile	positive integer	add together
United.Segment	positive integer	add together
Aeroplan.Status	{SE, E, P, N}[5]	the highest
United.Status	{PE1K, PE, P, N}[6]	the highest
SA.Status	{Gold, Silver}	the highest
SA.Mileage	positive integer	add together
SA.Segments	positive integer	add together
SA.privilege	{LA, EBA, PAC, PRW, PAS}	No rule
United.privilege	{LA, EBA, PAC, PBH, PB, PRW, PAS}[7]	No rule
Aeroplan.privilege	{LA, EBA, PAC, PBH, PB, PRW, PAS}	No rule

Notes:

1. In theory, a name can be any string;
2. It means that this property can have multiple values;
3. In theory, a mileage can be any positive integer. In practice, we may add a limit, e.g. 10,000,000 and The same is for segments;
4. It means that we have to add multiple values together to get one value for the property;
5. SE: Super Elite, E: Elite, P: Prestige, N: Normal;
6. PE1K: Premier Executive 1K, PE: Premier Executive, P: Premier, N: Normal;
7. LA: Lounge Access, EBA: Extra Baggage Allowance, PAC: Priority Airport Check-in, PBH: Priority Baggage Handling, PB: Priority Boarding, PRW: Priority Reservation Waitlist, PAS: Priority Airport Standby.

APPENDIX B: THREE PRECEDENCE TABLES

Aeroplan

Preceded Property	Preceded Value	Preceding Property	Preceding Value	Preceding function	Preceded function
Status	entire domain	StatusMileage	entire domain	MileToSta(a)	StaToMile(a)
Status	entire domain	StatusSegment	entire domain	SegToSta(a)	StaToSeg(b)

United

Preceded Property	Preceded Value	Preceding Property	Preceding Value	Preceding function	Preceded function
Status	entire set	Mile	entire set	MileToSta(a)	StaToMile(a)
Status	entire set	Segment	entire set	SegToSta(a)	StaToSeg(b)

Star Alliance

Preceded Property	Preceded Value	Preceding Property	Preceding Value	Preceding function	Preceded function
Aeroplan.Status	{SE, E, P}	SA_Status	entire set	AtoSA(a)	SAtoA(a)
United.Status	{PE1K, PE, P}	SA_Status	entire set	UtoSA(a)	SAtoU(a)
Aeroplan.Status Mileage	entire set	SA_Mileage	entire set	the same	the same
United.Mile	entire set	SA_Mileage	entire set	÷1.2	×1.2
Aeroplan.Status Segment	entire set	SA_Segment	entire set	the same	the same
United.Segment	entire set	SA_Segment	entire set	the same	the same
SA.Mileage	{>=18,000}	SA_Status	entire set	MileToSta(a)	StaToMile(a)
SA.Segments	{>=30}	SA_Status	entire set	SegToSta(a)	StaToSeg(b)
SA.Status	Gold	SA_Privilege	entire set	NA	NA
SA.Status	Silver	SA_Privilege	two privileges	NA	NA

APPENDIX C: AN INSTANCE FUNCTION IN XML SYNTAX

```
<Flyers>
  <Instance>
    <IDENTITY>Aeroplan.Member.member_no.1234</IDENTITY>
    <IDENTITY>United.Customer.num.1111</IDENTITY>
    <United.privilege>
      <SA.privilege>
        <United.Status>
          <United.Mile>21000</United.Mile>
          <United.Segment>15</United.Segment>
        </United.Status>
        <SA.Status>
          <SA.Mileage>
            <Aeroplan.Mileage>18000</Aeroplan.Mileage>
            <United.Mile>21000</United.Mile>
          </SA.Mileage>
          <United.Status>
            <United.Mile>21000</United.Mile>
            <United.Segment>15</United.Segment>
          </United.Status>
          <SA.Segment>
            <Aeroplan.Segment>20</Aeroplan.Segment>
            <United.Segment>15</United.Segment>
          </SA.Segment>
          <Aeroplan.Status>
            <Aeroplan.Mileage>18000</Aeroplan.Mileage>
            <Aeroplan.Segment>20</Aeroplan.Segment>
          </Aeroplan.Status>
        </SA.Status>
      </SA.privilege>
    </United.privilege>
  </Instance>
</Flyers>
```