USING UML FOR CONCEPTUAL MODELING: TOWARDS AN ONTOLOGICAL CORE

XUEMING LI

Using UML For Conceptual Modeling:

Towards An Ontological Core

by

©Xueming Li

A thesis submitted to the School of Graduate Studies

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Memorial University of Newfoundland

October 2007

St. John's Newfoundland

Using UML For Conceptual Modeling:

Towards An Ontological Core

Xueming Li

Memorial University of Newfoundland, School of Graduate Studies, 2007

ABSTRACT

Before developing an information system, the business and organizational domain in which the information system is to be used must be examined and understood. Creating conceptual models in the system analysis phase to faithfully represent the domain is critical for successful information system development. Although it is widely accepted that UML could be used both for modeling software, and for modeling the problem domain that is supported by a system, i.e. conceptual modeling, its suitability for the latter in the early development phase has been questioned. In fact, the semantics of its constructs (such as *object*, *class*, *attribute*, *link*, *association*, and *association class*) are clear with respect to software design and coding, but vague with respect to conceptual modeling.

To endow UML with semantics for conceptual modeling, in this thesis, an ontological framework of UML based on Bunge's ontology is proposed, focusing on static aspects (class/object diagrams and links in collaborations). The framework assigns precise ontological semantics for a core set of UML constructs (*object*, *attribute*, *class/type*, *link*, *association*, *state*, *state transition*, *operation*, and *role*) in class, object, and collaboration diagrams and is used to resolve a number of confusions in the UML literature.

Furthermore, in the thesis, a novel ontological metamodel of classifiers based on Bunge's ontology, OntoClean methodology, and Guizzardi et al.'s ontological profile is proposed, focusing on discussing the definition, properties, and representation of the notion of role in the object-oriented literature. The metamodel conforms to the fundamental role features identified in the literature and handles the counting problem and related role identity problem.

The study also compares conceptual models created using the metamodel to those created using ER approach with respect to conceptual database modeling and describes how to map a conceptual model based on the metamodel into relational database schema. Using examples, the study demonstrates that relational database schemata generated using our approach are more stable with respect to requirements change, and moreover a number of real world semantics and rules can be implemented as integrity constraints of a relational database schema.

Keywords: UML, Conceptual Modeling, Ontology, Role Modeling, ER, Conceptual Database Modeling

Acknowledgments

Doing a PhD is a very challenging work. It is much more challenging if one has to change supervisor during the course. There are several people that deserve my immense gratitude and appreciation for helping me in many possible ways in this difficult process.

First of all, I wish to express my highest gratitude to my supervisor Dr. Jeffrey Parsons for giving me the opportunity to develop the research reported in this PhD thesis when I felt despairing due to the change of supervisors. I am indebted to his patience, constant support, guidance, encouragement, and for the various fruitful discussions in his office. I have a great admiration for his seriousness towards research and his attitude with respect to understanding things in their essence.

Also I am honored to have Dr. Theodore Norvell and Dr. Miklos Bartha in my supervisory committee, who took great efforts in reviewing and providing valuable advice and comments regarding various aspects of this thesis. I am also grateful to Dr. Ulf Schünemann for vigorous and helpful discussions at nights in the department, to Dr. Andreas L. Opdahl, Dr. Dennis Peters, and Dr. Rodrigue Byrne for their comments on my thesis, and to Dr. Joerg Evermann and Dr. Carson C. Woo for their comments on my presentation in CAiSE conference. I have learned many things from them.

I also would like to thank the faculty, staff, and graduate students of the department of computer science, Mrs. Jane Foltz, Dr. Paul Gillard, Ms. Elaine Boone, Dr. Wolfgang Banzhaf, Mr. Nolan White, Dr. Manrique Mata-Montero, Dr. Todd Wareham,

Dr. Adrian Fiech, Ms. Sharon Deir, Mr. Jianmin Su, and many more, for their assistance and support throughout my program.

Last but not least, I am grateful to my brother Xuejun Li and my sister Qian Li for all their love, care, and everlasting support. There are no words I could possibly write to articulate my gratitude to my parents, Shaoyao Li and Yuanjing Zheng. During the period of my PhD program, I survived because of them. They always care about my happiness. This thesis is dedicated to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

The Unified Modeling Language (UML) [1] is a language for modeling object systems based on a unification of Booch, Rumbaugh and Jacobson's popular object-oriented modeling methods [75]. It has been rapidly adopted as the de facto standard for modeling such systems, primarily through the standardization efforts undertaken by the Object Management Group (OMG).

Conceptual modeling is an essential aspect of information system development [24]. It "is the activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication" [24]. It is a fundamental discipline in computer science, playing an essential role in areas such as database and information systems design, software and domain engineering, design of knowledge-based systems, requirements engineering, information integration, semantic interoperability, natural language processing, enterprise modeling, among many others.

Before developing an information system, the business and organizational domain in which the information system is to be used must be examined and understood. How to represent this domain is the focus of the system analysis phase of an information system development. Such a description is termed a *conceptual model*. Developing a conceptual model faithfully representing the domain it is intended to represent is critical for successful information system development. It is widely held that one important advantage of UML is that it could be used both for modeling software, and for modeling the problem domain that is supported by a system, i.e., conceptual modeling. However, since UML was developed based on ideas from the implementation domains such as databases and programming languages, the suitability of UML for modeling real world domains in the early development phases has been questioned [3]. More generally, UML lacks theoretical foundations for conceptual modeling. Indeed, the semantics of its constructs (such as *object*, *class*, *attribute*, *link*, and *association*) are clear with respect to software design and coding, but vague with respect to application domain modeling.

As indicated by various researchers [4][5][6][7], the UML suffers from some deficiencies such as inconsistency, ambiguity, inadequacy, and complexity. In view of UML's weaknesses, some researchers try to address these issues by formalizing the UML through various means [8][9][10]. Formalization may add mathematical rigour to the modeling language and software development process by specification, reasoning and refinement. However, as indicated by Evermann in [11][12], all these works relate to the internal consistency of the UML, not to its relationship to the real world domains. By not relating UML to real world entities, they fail to endow UML constructs with the real world meaning necessary for conceptual modeling of real-world domains.

An information system is an artificial representation of a real-world system as perceived by humans. Information systems development is a transformation from some perceptions of the real world into an implementation of a representation of these perceptions. Therefore, to develop a conceptual model that faithfully represents a real world domain, we must understand in advance what exists in the real world. In philosophy, ontology is the study of the nature and structure of the real world. It is a mature discipline that has been systematically developed since Aristotle. A number of pioneering researchers (e.g., Wand [14], Wand and Weber [17][18][19][20], Wand, et al. [15], Parsons and Wand [13], Wand, Storey, and Weber [16], Shanks, et al. [27], Evermann and Wand [11][12][2][25], Evermann [26], Weber and Zhang [21], Opdahl and Henderson-Sellers [28][3], Guizzardi, Herre, and Wagner [22][23]) have argued that ontology is an appropriate foundation for identifying the fundamental constructs and the relationships among them that need to be supported by a conceptual modeling language. Therefore, the question our research tries to address is: *Can we define an ontological static core of UML that has a formal ontological semantics and is suitable for conceptual modeling?* This question is answered throughout this thesis.

1.2 Objectives

In this thesis, we aim at developing an ontological core of UML for conceptual modeling based on Bunge's ontology [30][31], focusing on static aspects. We use Bunge's ontology because it provides a comprehensive framework for representing real-world phenomena as well as their relationships, and it has been successfully employed to evaluate a number

of modeling languages. By mapping ontological concepts to UML constructs and vice versa, we provide UML constructs with ontological semantics needed for modeling real world domains. Meanwhile, the mapping can be used to identify deficiencies in UML for conceptual modeling and moreover provide corresponding solutions. The mapping can also be used to transfer ontological assumptions about the real world to UML. Furthermore, we employ UML notations for our ontological core and create our own whenever necessary. However, Bunge's ontology lacks details with respect to some important real world phenomena. Accordingly, we investigate other ontological theories (e.g., OntoClean) and extend Bunge's ontology to incorporate additional considerations. We conclude by evaluating the impact of our ontological core on the design of relational database schemata in order to prescribe domain semantics.

Note that in this thesis, we do not consider all UML constructs (constructs considered in the thesis are *object*, *attribute*, *class/type*, *link*, *association*, *state*, *state transition*, *operation*, and *role*). In fact, many of them are implementation oriented constructs (e.g., *package* and *component*) which do not have counterpart in Bunge's ontology. Instead, the choice of the constructs in our ontological UML core is driven by Bunge's ontology.

1.3 Thesis Organization

The remainder of this thesis proceeds as follows. Chapter 2 describes UML and its weaknesses for the purpose of conceptual modeling. Chapter 3 presents our ontological core based on which the deficiencies of UML are identified and corresponding solutions

and modeling guidelines proposed. In Chapter 4, an ontological metamodel of classifiers based on OntoClean methodology and Guizzardi et al.'s ontological profile is proposed and incorporated into our ontological framework, Then we explore its implications for object-oriented and conceptual modeling as well as information system design and implementation, focusing on discussing the definition, properties, and representation of the notion of role in the literature. In Chapter 5, we compare conceptual models created using our metamodel to those created using ER approach with respect to conceptual database modeling in order to demonstrate its conceptual and practical usefulness. Finally, the thesis is concluded in Chapter 6 by summarizing the contributions of the thesis and listing some future works.

Chapter 2

UML and Its Weaknesses For The Purpose of Conceptual Modeling

2.1 UML

The Unified Modeling Language (UML) [1] is an object modeling and specification language officially defined by the Object Management Group (OMG). It includes a standardized graphical notation that may be used to create an abstract model of a system. While UML was designed to specify, visualize, construct, and document software-intensive systems, it is not restricted to modeling software, has been used for modeling hardware, and is commonly used for business process modeling, systems engineering modeling, and representing organizational structure, among many other domains.

Object-oriented modeling languages emerged sometime between the mid 1970s and the late 1980s for supporting analysis and design of increasingly complex application systems using object-oriented programming languages. Up to 1994, more than 50 object-

oriented methods were developed by methodologists such that many users had trouble finding an appropriate method that met their needs completely, leading to the so-called method wars [76]. Aiming at developing a unifying modeling language, the first version of UML (UML 1.1) was approved by OMG in 1997, based on a unification of Booch, Rumbaugh, and Jacobson's prominent object-oriented modeling methods. Since then UML has been revised with several releases (the major being UML 1.3, 1.5, and 2.0), fixing some problems and adding new notational capabilities. The current standard is UML 2.0, a major rewrite. However, since the development work on UML 2.0 was not complete when this research was done and it is very similar to UML 1.5 with respect to the parts of our concern, in this thesis, we focus on UML 1.5.

A system can be visualized from different perspectives by drawing different UML diagrams. There are nine diagram types in UML: Class diagram, Object diagram, Use Case diagram, Sequence diagram, Collaboration diagram, Statechart diagram, Activity diagram, Component diagram, and Deployment diagram. A class diagram shows a set of classifier[1] elements and their various static relationships, thus it is a graphic view of the static structural model of a system. An object diagram shows a set of objects and their relationships. It represents a static snapshot of instances of the classifiers in the corresponding class diagram at a point in time. In contrast, a collaboration diagram shows an interaction consisting of a set of objects and their relationships, thus it addresses the dynamic view of a system. In Chapter 3, we will focus on discussing these three UML diagrams.

[1] In UML, a classifier is a classification of instances describing a set of instances that have structural and behavioral features in common.

2.2 Weaknesses of UML

The interest in the object-oriented paradigm derives from some well-known claims: "the mapping established between real-world entities and modeled objects, the capacity to represent both entity structure and behavior, the possibility to classify them into a taxonomy: in short, a more *natural view* of the world" [32, p. 891]. However, as argued by Bonfatti and Pazzi, in spite of these expectations, only a generic idea of the object-oriented paradigm is available based on integrating heterogeneous assumptions made in the fields of programming languages, databases, and knowledge representation, intermingled with a number of implementation oriented considerations. A well-founded, complete model is still missing.

Recently, quite a few papers discussing UML deficiencies and corresponding possible remedies or extensions have been published (e.g., [4][6][7][5]). These papers address weaknesses of the UML from different points of view. Some of them are concerned with system design and implementation issues of UML, whereas others more conceptual aspects. According to them, the UML suffers from some deficiencies such as *inconsistency*, *ambiguity*, *inadequacy*, and *complexity*. For example, [4][6][7] are basically enumerations of the problems experienced by real developers in modeling real software systems using UML. In [6], Simons and Graham offer a catalogue of problems associated with using UML 1.1. Causes of these problems are various, including ambiguous semantics in the modeling notations, cognitive misdirection during the

development process, inadequate capture of salient system properties, lack of appropriate supporting tools and developer inexperience.

By analyzing these problems in detail, the authors claim that some of these problems can be addressed by increased guidance on the consistent interpretation of diagrams and the most helpful sequencing of modeling techniques; others are claimed to require a revision of the UML and its supporting tools. Moreover, they believe that most of these problems can be traced to the awkward transition between analysis and design.

Later in [7], Simons and Graham point out that in UML 1.3, although a number of semantic inconsistencies in the notation were fixed, the biggest problems by far is cognitive misdirection, or the apparent ease with which the rush to build UML models may distract the developer from important perspectives on a system. They believe that it is not a problem which can be fixed simply by trying to clarify the semantics of UML as it stands; instead, large chunks of UML need to be reconstructed to take into account the ways in which developers' minds operate.

In [5], Dori observes that the problems of UML can be classified into model multiplicity resulting from excess diagram types and symbols, confused behavior modeling, and the obscuring influence of programming languages. He argues that "Since UML has evolved bottom up from OO programming concepts, it lacks a system-theoretical ontological foundation encompassing observations about common features characterizing systems regardless of domain".

Without adding to the ever-expanding list of weaknesses of UML, we maintain in this thesis that some of UML's unsuitability to be used in the early stages of information systems development are rooted in its foundation. An ontological framework can be used

as the much-needed foundation of the object-oriented paradigm, providing principles and criteria for effective knowledge organization from domain analysis to system design.

2.3 Formalization of UML

In view of UML's weaknesses, some researchers try to address these issues by formalizing the UML through various means (e.g., [33][8][9][10]) in an attempt to base the UML on a formal, mathematical foundation, such that the syntax and semantics of UML constructs can be precisely defined. Though the UML standard provides a rich set of syntactically well-defined modeling notations, the semantics are described informally using a mixture of diagrams and natural language text. The informal descriptions can lead to confusion over the appropriate use and interpretation of the notations.

Formalization may add mathematical rigour to the modeling language and software development process by specification, reasoning and refinement. However, as indicated by Evermann in [11][12], most of these works relate to the internal consistency of the UML, not to its relationship to real world domains. Their aim is to achieve meaning by translating the UML to more formal languages such as Z, rather than by determining their relations to real world entities. By not relating UML to real world entities, they fail to endow UML constructs with the real world meaning necessary for conceptual modeling of real-world domains.

2.4 Conceptual Modeling Using UML

Conceptual modeling is an essential aspect of information system development. According to Wand et al. [15], an information system is an artificial representation of a real-world system as perceived by humans. They then argue that information systems development is a transformation from some perceptions of the real world into an implementation of a representation of these perceptions. Conceptual modeling is especially important in the analysis stage of information system development, when abstract models of the represented system and its organizational environment are created. Such models are termed conceptual models. A conceptual model should reflect knowledge about the application domain rather than about the implementation of the information system.

Understanding and describing the real world is the first step in the information system analysis and design (ISAD) process. The result, a conceptual model, serves as a medium for communicating and reasoning about the real world domain as well as input for the next step in the ISAD process – system design. System design is the process of designing and describing the information system, as opposed to the real world. The resulting design models differ from conceptual models in reflecting implementation considerations.

It is widely held that one important advantage of UML is that it can be used both for modeling software, and for modeling the problem domain that is supported by a system, i.e., conceptual modeling. However, since OO techniques were originally motivated to address the issues of coding and software design [28], the suitability of the UML for conceptual modeling of real world domains in the early development phases has been called into question. For example, in [28], Opdahl and Henderson-Sellers indicate

that within requirements engineering (RE), OO constructs and their use appear less well understood and well defined and their value is controversial. In fact, in [38], Woodfield argues using eight examples that significant impedance mismatches [2] exist between conceptual modeling concepts and the concepts of object-oriented programming. For example, there are significant differences between the notions of class in conceptual modeling and class/type in object-oriented programming languages. In a conceptual model, to be considered members of a class, all objects of the class must share common properties. Besides, each object can have additional properties that are not common to others. Moreover, objects can migrate between classes, i.e., they may become and cease to be members of different classes. In a typical object-oriented programming language (such as C++) however, an object is an instance of a class. It has only those properties defined for the class, and objects cannot move from class to class.

These situations must be taken seriously since many authors have argued that RE is critical for successful information system development [34][35][36] – the cost of repairing requirements errors during maintenance may be two orders of magnitude greater than that of correcting them during RE [37].

Moreover, in [25], Evermann and Wand regard UML as an object-oriented information system design language whose meaning of constructs is well defined with respect to software concepts, as opposed to a conceptual modeling language whose meaning of constructs should be defined in terms of application domain concepts. As a result, one of the detrimental consequences of using UML for conceptual modeling would

[2] Originally developed for electrical power, impedance matching is the practice of attempting to make the output impedance of a source equal to the input impedance of the load to which it is ultimately connected, usually in order to maximize the power transfer and minimize reflections from the load. Here impedance mismatch is used to describe the conceptual mismatches between conceptual modeling concepts and object-oriented programming concepts.

be "IS development projects might begin without explicitly modelling the application domain and instead must rely on implicit assumptions of developers." [25, p. 147]

To sum up, although it is desirable for UML to be used as a conceptual modeling language, this expectation is problematic because of UML's implementation based origin. To enable the use of UML for conceptual modeling, at least a core set of its model constructs must be endowed with real world semantics, leading to an ontological core of UML.

Chapter 3

An Ontological Core of UML For Conceptual Modeling

3.1 Ontology and Ontological Evaluation of Conceptual Modeling Languages

"Any representation of knowledge and meaning inside a computer must embody some philosophical assumptions. Yet philosophers have been debating such issues for centuries without reaching final agreement. To avoid the controversy, many people working with computers try to ignore it. But to write a program without analyzing the issues is to make a blind choice instead of a reasoned commitment" [39, p. 1].

In philosophy, ontology is the most fundamental branch of metaphysics. It is a mature discipline that has been systematically developed since Aristotle. The purpose of ontology "is to study the most general features of reality" [44]. It aims to answer questions such as what kinds of entities exist in the world? What are the properties of a

thing and how they are related to the thing itself? Are there properties of properties? Can change occur without a changing thing? There are a number of ontological theories, each with specific assumptions about what is perceived to exist in reality.

Until recently, research in ontologies related to the computer science community was limited to Artificial Intelligence and Knowledge Engineering fields (e.g., [77]). Over the last few years, however, ontologies have become influential in the Information Systems discipline, transferring the research focus to topics related with system analysis and design, enterprise systems and web services. Such research is oriented to two well-differentiated lines: ontologies as technologies of information systems (covering the ontology-driven information systems that use domain, task and application ontologies) and the ontological models of information systems (ontologies as abstract models supporting the core of the information system discipline and contributing to the improvement of reality-modeling techniques). The research presented in this thesis follows the second line.

A number of researchers (e.g., Evermann and Wand [11][12][2][25], Parsons and Wand [13], Wand [14], Wand, et al. [15], Wand, Storey, and Weber [16], Wand and Weber [17][18][19][20], Weber and Zhang [21], Guizzardi, Herre, and Wagner [22][23], Guizzardi [45]) have argued that ontology is an appropriate theoretical foundation for identifying the fundamental constructs and the relationships among them that need to be supported by a conceptual modeling language. Many of them base their work on Wand and Weber's adaptation of Mario Bunge's ontology [30][31] (i.e., BWW model) to the Information Systems field. The idea of ontological evaluation of modeling languages developed by Wand and Weber is to find a mapping from ontological concepts into

language constructs and vice versa [19]. The first mapping shows how ontological concepts can be represented by the language and is therefore called *representation* mapping. The latter shows how language elements are interpreted ontologically and is called *interpretation* mapping. Based on these mappings, Wand and Weber identify four ontological discrepancies that may undermine the *ontological completeness* and *ontological clarity* of constructs in a conceptual modeling language:

— *Construct deficit*: an ontological concept is not represented by any modeling construct – usually a problematic situation.

— *Construct overload*: a modeling construct corresponds to several ontological concepts – usually a problematic situation.

— *Construct redundancy*: several modeling constructs represent the same ontological concept – not necessarily problematic.

— *Construct excess*: a modeling construct does not represent any ontological concept – only problematic if this construct represents phenomena in the real world.

The mappings between ontological concepts and conceptual modeling language constructs can be used to transfer principles and rules from ontology to the modeling language. If there are ontological rules or constraints that relate two or more concepts, then by virtue of the mapping, these same rules or constraints must also apply to the modeling language. Thus, the ontological mappings can lead to modeling rules and guidelines on how to use the conceptual modeling language to model real world domains [11][12].

Wand [14] and Parsons and Wand [13] apply the main concepts of Bunge's ontology to examine principles of object-oriented concepts. They distinguish representation related constructs from implementation related ones. Parsons and Wand derive guidelines for using them in systems analysis. They argue that "the key to applying object concepts in systems analysis successfully is viewing objects as representation, rather than implementation, constructs. While many object concepts may be common to representation and implementation, applying them with an implementation focus during analysis may have undesirable consequences" [13, p. 105].

In [16], Wand et al. employ Bunge's ontology to analyze the semantics of the relationship construct used in Entity Relationship (ER) diagrams. They derive rules for using relationships in real-world modeling and suggest ways in which to resolve ambiguities found in the ER language.

More recently, Opdahl and Henderson-Sellers analyze and evaluate OML (Open Modeling Language) and UML as languages for representing concrete problem domains using Bunge's ontology and BWW model [28][3]. Their analysis and evaluation show that "many of the central UML constructs are well matched with the BWW models" [3, p. 43], and at the same time, they also suggest several concrete improvements to the UML metamodel.

In [46], Rosemann and Green argue that although Bunge's ontology and BWW models have been applied successfully to evaluate a number of modeling techniques, the usefulness of BWW ontological models suffers from three shortcomings, i.e., *lack of understandability*, *lack of comparability*, and *lack of applicability*. They therefore

propose to develop a metamodel as a semi-formal description of the BWW constructs by selecting an extended Entity-Relationship model as the metalanguage.

Moreover, in [2], using UML as an example, Evermann and Wand propose a method based on Bunge's ontology to restrict the syntax of a modeling language such that impossible domain configurations cannot be modeled.

In [15], Wand et al. explain why they choose Bunge's ontological works as (one of) the base for conceptual modeling: "First, it is oriented towards systems. Second, it is intended to deal with a wide range of systems, from physical to social. Third, it is well formalized, both in defining the concepts and outlining the premises and in providing a consistent notation. Finally, it draws upon an extensive body of prior work related to ontology". We believe Bunge's ontology a good candidate for the purpose because it provides an ontological semantic framework for a conceptual modeling language, which can be used to represent static and dynamic aspects of real-world systems.

Besides Bunge's ontology, other researchers also employ other ontologies to analyze and evaluate UML. In [22], Guizzardi, Herre, and Wagner use the General Ontological Language (GOL) and its underlying upper level ontology to evaluate the ontological correctness of a UML conceptual model and develop guidelines for how UML constructs should be used in conceptual modeling. They focus on evaluating class, object, powertype, association, aggregation/composition, and suggest some proposals to extend UML for more satisfactory treatment of aggregation.

As indicated above, for conceptual modeling, Bunge's ontology is an appropriate theoretical framework for identifying the fundamental constructs, as well as their relationships, that need to be supported by a modeling language. It should be pointed out

that identifying relationships among modeling constructs is as important as identifying those constructs. However, most of the work above based on Bunge's ontology focuses on evaluating aspects of a modeling language by mapping only ontological constructs into those of the modeling language and vice versa, ignoring the equally important mapping of relationships among constructs.

In this Chapter, we set the foundation for our approach by adopting the ontology of Mario Bunge [30][31] to establish the semantic framework of our ontological UML core. Our research is based on the assumption that endowing core constructs of UML with precise ontological semantics and rules for modeling static and dynamic aspects of real world domains will mitigate to some extent the shortcomings of the UML's capability of real world modeling. The choice of the constructs in our ontological UML core is driven by Bunge's ontology, which is discussed in detail in the next section.

3.2 Bunge's Ontology

Bunge's *Treatise on Basic Philosophy* devotes two volumes to ontology, in which he articulates a set of high-level abstract constructs that are intended to be a means of representing all real-world phenomena [30][31][40]. In this thesis, we focus on discussing static issues of Bunge's ontology. In particular, we summarize and discuss in this section Bunge's notions of *thing* and *construct*, *property* and *attribute*, *kind* and *natural kind*, *law* and *law statement*, *functional schema*, *state* and *event*, *history* and *coupling*, as well as *aggregate* and *system*.

3.2.1 Thing and Construct

In Bunge's ontology, there are two kinds of objects: *concrete things* or simply *things*, and *conceptual things* or *constructs*. The world is viewed as composed of things. Constructs (e.g., mathematical concepts such as numbers, sets, and functions) are only creations of the human mind which take part in our representations of the real world. The totality of things is denoted Θ. Note that, no matter whether a thing perceived as a specific object by a modeler exists in reality or only in his/her mind, we say that it is a concrete thing. For example, we consider a person as a thing, as well as a unicorn.

All objects have properties. If the objects are concrete things, their properties are called *substantial properties* (or simply *properties*). If the objects are conceptual things, the words "attribute" and "property" are exchangeable because a conceptual thing has all the properties we consistently attribute to it. A substantial property is a feature that some things possess even if we are ignorant of this fact. On the other hand, an attribute is a feature we assign or attribute to a conceptual thing modeling some concrete things. An attribute may or may not represent a substantial property. It may also do it well or poorly. The totality of all the substantial properties is denoted $\mathbb{P}$, and that of all the attributes is denoted $\mathbb{A}$.

3.2.2 Property and Attribute

All things possess properties. Properties cannot have properties. Properties do not exist independent of things, i.e., every property is possessed by some thing or other. Some

properties, such as Height and Age of a person (which is a concrete thing), are inherent properties of things, called *intrinsic properties*. Other properties, such as Solubility and Being Enrolled In A University, are properties of pairs or, in general, n-tuples of things, called *mutual properties*. In essence, a mutual property of a thing is a property that has meaning only in terms of some other thing or things in the world. Note that, throughout this thesis, definitions and postulates taken directly from Bunge's ontology are indicated by "*".

Definition 3.1*: For a thing $x \in \Theta$, the *set of all the properties of thing* x is denoted as:

$$p(x) = \{P \mid P \in \mathbb{P} \ \wedge \ x \ \textit{possesses} \ P\}.$$

Humans conceive of things in terms of models of things (which Bunge calls *functional schemata*). Such models are conceptual things, thus constructs. Attributes are characteristics assigned to models of things according to human perceptions. We may use different models for the same thing, and therefore assign different sets of attributes to the same thing. Thus humans conceive of properties of things in terms of the attributes of their conceptual models, and properties are known to humans only as attributes. In a given model of a thing, usually not every property of the thing will be represented as an attribute. Therefore, every functional schema only reflects partial aspects of a thing. Likewise, an attribute in a given model may or may not reflect a substantial property. For example, the Height of a person (which is a model of a concrete thing) is an attribute that reflects a substantial property of the concrete thing. The Name of a person (which is a

model of a concrete thing) does not represent any specific substantial property of the concrete thing. It is an attribute that stands for the individual as a whole. Sometimes an attribute is used to represent one or more properties that we do not fully understand, e.g., Intelligence Quotient (IQ).

The representation of properties by attributes can be formalized as follows:

Postulate 3.1*: Let $\mathbb{P}$ be the set of all properties and $\mathbb{A}$ the set of all attributes. The representation of properties by attributes is via a function $\rho : \mathbb{P} \to 2^{\mathbb{A}}$ that for each $P \in \mathbb{P}$, $\rho(P)$ is a set of attributes $A \in 2^{\mathbb{A}}$ such that for any $a \in A$, a represents P.

From postulate 3.1, it is clear that different attributes may be used to represent the same property. In Bunge's ontology, attributes can be formalized in a predicate form or in a functional form. The functional form is

$$A : T_1 \times \ldots \times T_n \times V_1 \times \ldots \times V_m \to V$$

where $T_i (i = 1, \ldots, n)$ represents a set of things similar in some respect and $V_j (j = 1, \ldots, m)$ as well as V represents a set of values such as time instances, currency, etc. Since an intrinsic property of a thing depends only on the thing itself whereas a mutual property depends on other thing or things, therefore if $n = 1$, then A represents an intrinsic property; otherwise, if $n \geq 2$, then A represents a mutual property. For example, the intrinsic property Age of persons can be represented as $Age : P \to I$, where P is the set of persons and I the set of positive real numbers. The mutual property StartDate held between employees and employers can be represented as $StartDate : E_1 \times E_2 \to D$, where

E_1 is the set of employees (or employers), E_2 the set of employers (or employees), and D the set of dates. Here, attribute function Age represents a *general* intrinsic property, or intrinsic property of a set of things, and each value of Age for a particular person represents an *individual* intrinsic property of that person. Similarly, attribute function StartDate represents a general mutual property shared by the sets of employees and employers, and each value of StartDate for a particular pair of employee and employer represents an individual mutual property shared by the employee and employer. In this thesis, we only consider functional form of attributes. Any general property must be representable as (at least) a functional form attribute.

We now define the notion of scope of a property:

Definition 3.2*: The *scope* of a property P denoted by $\varphi(P)$ is the collection of things possessing it:

$$\varphi(P) = \{x \mid x \in \Theta \ \wedge \ P \in p(x)\}.$$

3.2.3 Composition, Emergent Property, and Hereditary Property

A thing may be composed of other things. We denote $x \sqsubset z$ and $y \sqsubset z$ if things x and y are parts of thing z, z is thus a composite thing. Here, the part of relation $\sqsubset$ is a strict partial ordering, i.e., an asymmetric, irreflexive, and transitive relation, therefore nothing counts as part of itself.

$\forall x : \neg(x \sqsubset x)$
$\forall x, y : (x \sqsubset y) \rightarrow \neg(y \sqsubset x)$
$\forall x, y, z : (x \sqsubset y) \wedge (y \sqsubset z) \rightarrow (x \sqsubset z)$

Now we introduce the notion of composition:

Definition 3.3*: Let $x \in \Theta$ be a composite thing, then the composition of x is the set of its parts, i.e.,

$$C(x) = \{y \mid y \in \Theta \ \wedge \ y \sqsubset x\}.$$

Properties of a composite thing are of two kinds: hereditary properties that also belong to some component thing(s) of the composite thing, and emergent properties that characterize the composite thing as a whole. A fundamental ontological assumption is that the set of properties of a composite thing is not equal to that of all the properties of its parts, i.e., $p(z) \neq p(x) \cup p(y)$. Instead, a composite thing must have at least one emergent property that characterizes the composite thing as a whole. An emergent property can be either intrinsic or mutual.

Definition 3.4*: Let $x \in \Theta$ be a composite thing. $\forall P \in p(x)$, P is a *hereditary property* of x iff $\exists y : y \in C(x)$, such that $P \in p(y)$. Otherwise, P is an *emergent property* of x.

3.2.4 Kind

An arbitrary collection of things need not share a given set of properties. When they do, however, and no thing outside the collection has the properties of interest, the collection is called a kind.

Definition 3.5*: Let $k: 2^{\mathbb{P}} \to 2^{\Theta}$ be the function assigning to each nonempty set $\mathbb{R} \in 2^{\mathbb{P}}$ of properties the set $k(\mathbb{R}) = \bigcap_{P \in \mathbb{R}} \varphi(P)$ of things sharing the properties in $\mathbb{R}$. This value $k(\mathbb{R})$ is called the $\mathbb{R}$ – *kind* of things.

3.2.5 Law, Law Statement, and Natural Kind

A *law* is any condition or restriction on properties of a thing. A *law statement* is any restriction on the possible values of attributes of models of things and any relation among two or more attributes. The relation between laws and law statements is similar to that of properties and attributes discussed above: laws restrict and interrelate properties, whereas law statements restrict and interrelate attributes representing these properties, law statements of models of things represent laws of things. Laws are also properties in a broad sense. The totality of laws obeyed by things is denoted by $\mathbb{L}$, and we denote the totality of laws possessed by a thing x $\mathbb{L}(x)$. Since a law is also a property, according to definition 3.2, the scope of a law $L \in \mathbb{L}$ is the collection of things obeying it:

$$\varphi(L) = \{x \mid x \in \Theta \ \wedge \ L \in \mathbb{L}(x)\}.$$

A kind of things is determined by a set of properties. A natural kind, however, is determined by the laws the things obey. Indeed, the deepest method of grouping things is

by the laws they obey. As Bunge states, "When laws are made the fundamental divisions of a set of things, the resulting kinds are maximally natural – or, in Aristotelian jargon, accident is then unlikely to prevail over essence. The outcome is a set of natural kinds or species" [30, p. 145].

Definition 3.6*: Let $k_{\mathbb{L}} : 2^{\mathbb{L}} \to 2^{\Theta}$ be the function assigning to each $\mathbb{L}' \subset \mathbb{L}$ of laws the set $k_{\mathbb{L}}(\mathbb{L}') = \bigcap_{L \in \mathbb{L}'} \wp(L)$ of things sharing the laws in $\mathbb{L}'$. This value $k_{\mathbb{L}}(\mathbb{L}')$ is called the $\mathbb{L}'$*– natural kind.*

Note that, since a law restricts and interrelates a set of properties and a natural kind of things is determined by a set of laws, then every thing in the natural kind must also obey all the properties restricted and interrelated by the laws. In this sense, a natural kind K also determines a set of properties, denoted $P(K)$.

3.2.6 State, Conceivable State Space, and Lawful State Space

We do not handle directly concrete things but their models. The attributes of a model (or functional schema, see below) of things represent properties of the things. They are also called *state functions* because their values contribute to characterizing or identifying the *states* of the things of interest can be in. The number of attributes of every functional schema of things is finite.

Definition 3.7: For a natural kind $K \subset \Theta$, if the sequence of state functions of its functional schema is $\mathbb{F} =< F_1, F_2, \ldots, F_n >: K \times M \to V_1 \times V_2 \times \ldots \times V_n$, then the *state* of a thing $x \in K$ at $t \in M$ is $\mathbb{F}(x,t) = < F_1, F_2, \ldots, F_n > (x,t) = < F_1(x,t), F_2(x,t), \ldots, F_n(x,t) >$.

Here, set M and codomains V_i $(1 \le i \le n)$ are unspecified. In the simplest cases, M is the set of all possible time instances. $\mathbb{F}$ is called the *total state function* for K and each component of $\mathbb{F}$ (F_i, $1 \le i \le n$) is called the i th *state function* for K. Each component $F_i : K \times M \to V_i$ $(1 \le i \le n)$ of $\mathbb{F}$ is supposed (not necessarily) to represent a *general* property, or property of things of K. Each value of F_i at a particular point (x,t) ($x \in K$ and $t \in M$) is supposed to represent an *individual* property, or property of thing x at t.

For example, consider natural kind Person. A simple total state function for Person that has only one state function can be defined as $\mathbb{F} =< Status >: Person \times M \to \{Unborn, Alive, Dead, BecomeGod\}$, where M is the set of all possible time instances and $V_1 = \{Unborn, Alive, Dead, BecomeGod\}$. For any person $x \in Person$ and any time instance $t \in M$, the state of x at t is $\mathbb{F}(x,t) =< Status(x,t) >$. In another example, a total state function for natural kind Employee can be defined as $\mathbb{F} =< Salary, StartDate >: Employee \times Employer \times T \to R \times D$, where T the set of all possible time instances, $M = Employer \times T$, $V_1 = R$ the set of real numbers, $V_2 = D$ the set of dates. For any employee $x \in Employee$ and (one of) its employer(s) $y \in Employer$,

the state of x at any time instance $t \in T$ is $\mathbb{F}(x,y,t) = <Salary(x,y,t), StartDate(x,y,t)>$.

An ontological hypothesis is that, every thing is – at a given time – in some state or other. "Every theoretical model of a thing is concerned with representing the really possible (i.e., lawful) states, and perhaps also the really possible (lawful) changes of state, of the thing" [30, p. 131]. Therefore, we can identify two kinds of law statements: *state law statements*, which specify lawful states that a thing could actually stay in, and *transformation law statements*, which specify lawful transformations of a thing from a lawful state to another lawful state. While state law statements reflect the static characteristics of a thing, transformation law statements reflect its dynamic ones.

More formally, a state law statement can be viewed as a mapping L_S from the possible states of things in a given functional schema into the set {lawful, unlawful}; a transformation law statement can be viewed as a mapping L_T from a set of tuples $<s,s'>$, where s and s' are lawful states of the things in a given functional schema, into the set {lawful, unlawful}.

For a natural kind $K \subset \Theta$, the *conceivable state space* of things of K (denoted $S(K)$) in a given functional schema can be represented as $S(K) = \{<v_1,v_2,\ldots,v_n> \in V_1 \times V_2 \times \ldots \times V_n\}$. However, not every state in a conceivable state space is really possible. Let $\mathbb{L}_S$ be the set of state law statements shared by things of K, we can use $S_L(K)$ to denote the *lawful state space* of things of K determined by $\mathbb{L}_S$: $S_L(K) = \{<v_1,v_2,\ldots,v_n> \in S(K) \mid \forall L_S \in \mathbb{L}_S,\ L_S(<v_1,v_2,\ldots,v_n>) = lawful\}$. It is clear that

$S_{\mathrm{L}}(K) \subseteq S(K)$. Note that, given a functional schema of K, every thing in K has the same conceivable state space as $S(K)$ and the same lawful state space as $S_{\mathrm{L}}(K)$.

3.2.7 Functional Schemata

In Bunge's ontology, models of things are called functional schemata. Any natural kind of things can be modeled by some functional schema.

Definition 3.8: Let K be a natural kind. A *functional schema* X_K of K is a certain nonempty set M, together with a finite sequence $\mathbb{F}$ of state functions on $K \times M$ each of which is supposed to represent a property of $P(K)$, a finite set $\mathbb{L}_S$ of state law statements of K on $S(K)$, and a finite set $\mathbb{L}_T$ of transformation law statements of K on $S_{\mathrm{L}}(K) \times S_{\mathrm{L}}(K)$:

$$X_K =_{df} \lhd M, \mathbb{F}, \mathbb{L}_S, \mathbb{L}_T \rhd, \quad where$$

$$\mathbb{F} =< F_1, F_2, \ldots, F_n >: K \times M \to V_1 \times V_2 \times \ldots \times V_n$$

$$\mathbb{L}_S = \{L_S : S(K) \to \{lawful, unlawful\}\}$$

$$\mathbb{L}_T = \{L_T : S_{\mathrm{L}}(K) \times S_{\mathrm{L}}(K) \to \{lawful, unlawful\}\}.$$

As discussed in section 3.2.6, in a natural kind K with a given functional schema X_K, all things have the same conceivable state space and the same lawful state space. Moreover, they obey the same sets of state law statements and transformation law

statements. Nonetheless, every thing has its unique state history. $\forall x \in K$, its state history is determined by the function $\mathbb{F}(x) = <F_1, F_2, \ldots, F_n>(x) = <F_1(x), F_2(x), \ldots, F_n(x)>$ $: M \rightarrow V_1 \times V_2 \times \ldots \times V_n$.

A thing may have multiple functional schemata, reflecting different views of the same thing. Functional schema in Bunge's ontology is a crucial notion because the definitions of other notions of Bunge's ontology like state, state space, event, event space, etc. of thing(s) depend on the functional schema used to model the thing(s). Since a thing may have different functional schemata, it thus may have different sets of definitions of state, state space, event, event space, etc.

For example, consider natural kind Person in section 3.2.6 again. A simple functional schema X_{Person} of Person can be defined as $\lhd M, \mathbb{F}, \mathbb{L}_S, \mathbb{L}_T \rhd$ where M is the set of all possible time instances, $\mathbb{F} =< Status >: Person \times M \rightarrow \{Unborn, Alive, Dead, BecomeGod\}$, $\mathbb{L}_S = \{\{ (<Unborn>, lawful), (<Alive>, lawful), (<Dead>, lawful), (<BecomeGod>, unlawful) \}\}$, and $\mathbb{L}_T = \{\{ ((<Unborn>, <Alive>), lawful), ((<Unborn>, <Dead>), unlawful), ((<Alive>, <Dead>), lawful), ((<Alive>, <Unborn>), unlawful), ((<Dead>, <Unborn>), unlawful), ((<Dead>, <Alive>), unlawful) \}\}$. From $\mathbb{L}_S$, it can be determined that, for a person $x \in Person$ and a time instance $t \in M$, it may happen that $\mathbb{F}(x,t) =< Alive >$, but for any $x \in Person$ and $t \in M$, it cannot happen that x be in state $< BecomeGod >$ at t. Moreover, from $\mathbb{L}_T$, it can be determined that, for a person $x \in Person$ and a time instance $t \in M$, it may happen that x changes his/her state from $< Alive >$ to $< Dead >$ at t, but for any $x \in Person$ and

$t \in M$, it cannot happen that x changes his/her state from $< Dead >$ to $< Alive >$ at t in his/her state history.

3.2.8 Event and Transformation

So far we have considered static aspects of thing(s). In the real world, all things are changeable. Change may be qualitative (things acquiring or losing general properties) or quantitative (one or more individual properties of things are changed). Every qualitative change is accompanied by a quantitative change. We may conceive quantitative change of a thing simply as a transition from one state to another state. The *net change* of a thing from state s to state s' is representable by the ordered tuple $< s, s' >$ called event:

Definition 3.9*: Let K be a natural kind with functional schema X_K. An ordered pair $< s, s' >$ where s, $s' \in S(K)$ represents a *conceivable event* of things in K.

The *conceivable event space* of things in K (denoted $E(K)$) in functional schema X_K can be represented as $E(K) = \{< s, s' > | s, s' \in S(K)\}$. However, not all events in definition 3.9 are really possible (lawful) because (1) $S(K)$ is itself conceivable state space of K, and (2) even if s and s' are lawful, the transition from s to s' might not be lawful. In the example of section 3.2.7, both $< Dead >$ and $< Alive >$ are lawful states of natural kind Person, but the transition from $< Dead >$ to $< Alive >$ is not possible. To

define the notion of lawful event, we need the notion of a lawful transformation, i.e., transformation of the lawful state space $S_L(K)$ compatible with the law statements of K:

Definition 3.10: Let K be a natural kind with functional schema X_K. The *set of lawful transformations* of $S_L(K)$ into itself is the set of functions $G_L(K)$:

$$G_L(K) = \{g : S_L(K) \rightarrow S_L(K) \mid \forall s, s' \in S_L(K),\ \textit{if}\ \ s' = g(s),\ \ \textit{then}$$

$$\forall L_T \in \mathbb{L}_T\ \ \textit{of}\ \ X_K,\ \ L_T(s, s') = \textit{lawful}\}.$$

We can now define the notion of lawful event:

Definition 3.11*: Let $S_L(K)$ and $G_L(K)$ be the lawful state space and the set of lawful transformations of natural kind K with functional schema X_K respectively. A *lawful event* of things in K is represented by the ordered pair $< s, s' >$, where s, $s' \in S_L(K)$ and $\exists g \in G_L(K)$, such that $s' = g(s)$.

From definition 3.11, the *lawful event space* of things in K (denoted $E_L(K)$) in functional schema X_K can be represented as $E_L(K) = \bigcup_{g \in G_L(K)} \{< s, s' > \in S_L(K)^2 \mid s' = g(s)\}$.

3.2.9 History and Coupling

Every thing is changeable. Changes of states manifest a history of a thing:

Definition 3.12*: Let K be a natural kind with functional schema X_K. $\forall x \in K$, the *history* of x is the set of ordered pairs: $h(x) = \{< t, \mathbb{F}(x,t) > | t \in M\}$.

The notion of history allows us to determine when two things are interacting with each other thus bonded or coupled to each other. Intuitively, if two things interact, then at least one of them will not be traversing the same states as it would if the other did not exist. Let $h(y \mid x)$ denote the history of y acted on by x, we have:

Definition 3.13*: A thing x *acts on* a thing y, denoted $x \rhd y$, if $h(y \mid x) \neq h(y)$.

Postulate 3.2*: Every thing acts on, and is acted upon by other things.

Definition 3.14*: Two different things x and y are *coupled* (or *bonded*), denoted Bxy, iff $(x \rhd y)$ *or* $(y \rhd x)$. They *interact* iff each acts upon the other, i.e., $(x \rhd y)$ *and* $(y \rhd x)$.

For example, consider a husband and a wife who are married to each other, their histories are not independent. I.e., the state history of the husband is not the same as that he would traverse if he were single (and vice versa). As a result, they are coupled.

Couplings are also mutual properties. Different from the mutual properties we discussed in section 3.2.2 that do not affect their relata (e.g., "being greater than"), two things x, y are coupled iff some changes in x are accompanied or preceded or followed by some changes in y. We call couplings *binding mutual properties* and the mutual properties discussed in section 3.2.2 *nonbinding mutual properties*. Note that, *only a general nonbinding mutual property can be represented by an attribute function in a functional schema.*

The interaction between two or more things may give rise to a number of *semantically related* nonbinding mutual properties. For example, the interaction between an employee and an employer may incur a set of nonbinding mutual properties such as Salary, StartDate, and OfficePhone etc. Moreover, the effects of an interaction between two things may be lasting. Thus if two things interact for a while and then cease to do so, they are still coupled.

The collection of couplings among members of a set of things can be defined as:

Definition 3.15*: The *bondage* of a set T of things is the set $\mathbb{B}_T$ of all couplings among members of T.

3.2.10 Aggregate and System

A composite thing is a thing composed of component things. A composite thing can be either an aggregate, or a system:

Definition 3.16*: Let x be a thing composed of parts x_i, $1 \le i \le n$. Then x is an *aggregate* iff its history $h(x)$ equals the union of the partial histories $h(x_i)$. Otherwise x is a *system*.

Corollary 3.1*: Let x be a composite thing with composition $C(x)$. Then x is a *system* iff the bondage of $C(x)$ is not empty, i.e., $\mathbb{B}_{C(x)} \neq \Phi$.

3.3 Ontological Foundation of UML For Conceptual Modeling

In this section, we assign ontological semantics for conceptual modeling to UML by mapping ontological constructs discussed in section 3.2 to a core set of UML model elements and vice versa, focusing on static aspects. We refer to the OMG UML specification 1.5 through the mapping. We defend that, within current Bunge's ontological framework, our mapping is the most straightforward. If we want to achieve a more consistent mapping, then a lot of modifications to Bunge's ontology, especially the internal structure of Bunge's constructs, will be needed, which is out of the scope of the thesis.

Moreover, in UML, a classifier is a classification of instances describing a set of instances that have structural and behavioral features in common. There are several kinds of classifiers, including class, interface and data type, etc. In this thesis, we only consider classes.

3.3.1 UML Object

In UML, "An object represents a particular instance of a class. It has identity and attribute values." [1, p. 3-64] "An object is an instance that originates from a class, it is structured and behaves according to its class. All objects originating from the same class are structured in the same way, … An object may have multiple classes (i.e., it may originate from several classes). In this case, the object will have all the features declared in all of these classes, both the structural and the behavioral ones." [1, p. 2-109] Since real world things are represented in UML conceptual models as objects, we propose that Bunge-thing is modeled as UML-object and UML-object models only Bunge-thing. Note that, in software design, not every UML-object corresponds to a Bunge-thing. For example, in Figure 3.1, as argued in [25], instead of being a Bunge-thing, Job is a collection of mutual properties (Salary, StartDate etc.) shared by an employee and an employer. Therefore it should not be modeled as a UML-object.

Principle 1: In a UML conceptual model, every object models a Bunge-thing. Conversely, every Bunge-thing in the domain is modeled as a UML object.

It is clear from the UML specification quoted above that, in UML, class is a more fundamental notion than object: an object is only a particular instance that originates from a class, and all objects originating from the same class are structured and behave in the same way. This view reflects UML's implementation-oriented origin from object-oriented

programming languages. In a C++ program, we first have classes. Objects are then created from classes at run-time and behave exactly as specified by classes. Objects cannot move from class to class. In contrast, in Bunge's ontology, things are more fundamental than their classification: they exist in the real world. Things can be classified (into kinds and natural kinds, which are sets) according to some properties or laws they share. Therefore, in any classification of things, every thing has commonalities (the set of shared properties or laws) and idiosyncrasies (the set of unshared properties or laws). Moreover, things can migrate between natural kinds. These differences demonstrate the impedance mismatches [38] between UML and Bunge's ontology.

3.3.2 UML Attribute

In UML, "An attribute is a named slot within a classifier that describes a range of values that instances of the classifier may hold." [1, p. 2-24] Since in Bunge's ontology, general (intrinsic and nonbinding mutual) properties are represented by attribute functions, therefore we propose that Bunge-attribute function is modeled as UML-attribute and UML-attribute models Bunge-attribute function.

Principle 2: In a UML conceptual model, every attribute of a class/type models a Bunge-attribute function representing a general (intrinsic or nonbinding mutual) property. Conversely, every Bunge-attribute function representing a general (intrinsic or nonbinding mutual) property in the domain is modeled as an attribute of a class/type.

Note that, in Principle 2, if attributes of a class/type model Bunge-attribute functions which represent general intrinsic properties, then that class/type is an "ordinary" class/type, instances of which are objects which models things in the real world; otherwise, if the attributes model Bunge-attribute functions which represent general nonbinding mutual properties shared by things, then that class/type is an association class, instances of which are links connecting objects modeling those things.

3.3.3 UML Class/Type

In UML, "A class is a description of a set of objects that share the same attributes, operations, methods, relationships, and semantics." [1, p. 2-26] "A Type is used to specify a domain of objects together with operations applicable to the objects without defining the physical implementation of those objects. A Type may not include any methods, but it may provide behavioral specifications for its operations. It may also have attributes and associations that are defined solely for the purpose of specifying the behavior of the type's operations." [1, p. 3-49] Since a Bunge-functional schema models things of a natural kind using a set of attribute functions and law statements, therefore we propose that Bunge-functional schema is modeled as UML-class/type.

Principle 3: In a UML conceptual model, every Bunge-functional schema in the domain is modeled as a UML class/type. However, not every UML class/type models a Bunge-functional schema.

For example, Figure 3.1 is a UML conceptual model adapted from [42, p. 159]. In Figure 3.1, classes Company and Person model two functional schemata whose natural kinds are sets of companies and persons respectively. However, association class Job does not model any functional schema because its instances are links, not objects. In fact, in Chapter 4, we will see that the attributes of Job (Salary and StartDate) models two Bunge-attribute functions representing two general nonbinding mutual properties shared between employees and employers. Salary and StartDate are in fact the only attribute functions owned by functional schemata Employee and Employer which, in Figure 3.1, are represented as named places.

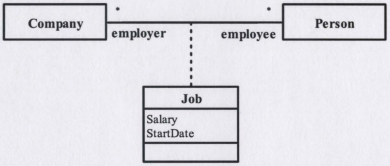

Fig. 3.1. Attributes of an association class

3.3.4 UML Link and Association

In UML, "A link is a connection between instances. Each link is an instance of an association, i.e., a link connects instances of (specializations of) the associated classifiers." [1, p. 2-110] "A link is a tuple (list) of object references. Most commonly, it is a pair of object references." [1, p. 3-84] "An association defines a semantic relationship

between classifiers. The instances of an association are a set of tuples relating instances of the classifiers. Each tuple value may appear at most once." [1, p. 2-19] "An association declares a connection (link) between instances of the associated classifiers (e.g., classes)." [1, p. 2-64]

Since Bunge-individual nonbinding mutual property connects things which are modeled as objects in UML, we have the following principle:

Principle 4: In a UML class or object diagram, every link between two or more objects models a tuple of values representing semantically related Bunge-individual nonbinding mutual properties shared between things modeled by these objects. Conversely, every tuple of values representing semantically related Bunge-individual nonbinding mutual properties shared between two or more things in the domain is modeled in a UML class or object diagram as a link between objects modeling these things.

As a result, instances of an association (i.e., links) in a UML class diagram are more than "a tuple (list) of object references" [1, p. 3-84]. Moreover, from Principle 2, each Bunge-attribute function representing a general nonbinding mutual property is modeled as an attribute of a UML association (or association class), thus we have Principle 5:

Principle 5: In a UML class diagram, every association between two or more classes/types models a tuple of Bunge-attribute functions representing semantically related Bunge-general nonbinding mutual properties shared by things that are modeled as

instances of these classes/types. Every association must have at least one attribute. Conversely, every tuple of Bunge-attribute functions representing semantically related Bunge-general nonbinding mutual properties shared by things in the domain is modeled in a UML class diagram as an association between two or more classes/types whose instances models these things.

In a UML class diagram, syntactically, attributes of an association can be illustrated using an association class attached to this association, thus in UML class diagrams, association and association class are semantically equivalent.

For example, in Figure 3.1, association (or association class) Job models a pair of Bunge-attribute functions $Salary: T_1 \times T_2 \rightarrow R$ and $StartDate: T_1 \times T_2 \rightarrow D$, each of which represents a semantically related Bunge-general nonbinding mutual property, where T_1 is the set of employees, T_2 the set of employers, R the set of real numbers, and D the set of dates. Therefore, given a specific pair of employee and employer (t_1, t_2), the instance (link) of association Job is a pair of values ($Salary(t_1, t_2)$, $StartDate(t_1, t_2)$) that model Bunge-individual nonbinding mutual properties of Salary and StartDate shared by the pair of employee and employer.

So far, the mutual properties we considered in a real world domain are nonbinding mutual properties as discussed in section 3.2.9. We suggest that, in a UML class diagram, every attribute of an association/association class models a Bunge-attribute function representing a Bunge-general nonbinding mutual property. Every association/association class in a UML class diagram must have at least one attribute shared by two or more

classes/types because, otherwise, this association/association class is not needed at all. On the other hand, links between two or more objects in a UML collaboration model only Bunge-couplings or Bunge-binding mutual properties holding between things modeled by these objects. Therefore, links and their corresponding associations in UML class diagrams and collaborations are of fundamentally different ontological nature in that, while links and associations in UML class diagrams reflect static characteristics of a real world domain, those in UML collaborations reflect dynamic ones. Consequently, we may call associations and their links in UML class and object diagrams *nonbinding associations* and *nonbinding links*, and associations and their links in UML collaborations *binding associations* and *binding links*.

In UML, "A collaboration describes how an operation or a classifier, like a use case, is realized by a set of classifiers and associations used in a specific way." [1, p. 2-117] "A collaboration defines an ensemble of participants that are needed for a given set of purposes. The participants define roles that instances and links play when interacting with each other. The roles to be played by the instances are modeled as classifier roles, and by the links as association roles. Classifier roles and association roles define a usage of instances and links, while the classifiers and associations specify all required properties of these instances and links. This means that the structure of an ensemble of interlinked instances conforms to the roles in a collaboration as they collaborate to achieve a given task. Reasoning about the behavior of an ensemble of instances can therefore be done in the context of the collaboration as well as in the context of the instances." [1, p. 2-124] From the UML specification, it seems that each link in a collaboration is supposed to be an instance of an association in the corresponding class diagram. However, we deem this

view an unpleasant consequence of UML's implementation oriented origin from object-oriented programming in which links in class diagrams are merely communication passages for sending messages to linked objects.

In fact, this view has given rise to a considerable amount of confusion in the UML literature when considering the relationship between class diagrams and collaborations. For example, in [41], in order to remedy the so-called *Baseless Link Problem*, namely a link in a UML collaboration may not have a corresponding association in class diagram, Stevens proposes to classify associations in UML into *static* associations and *dynamic* associations. An association is static if there is a *structural* relationship between the classifiers P and Q, i.e., one of the class definitions of P and Q includes an attribute that contains a reference to an object of the other class. An association is dynamic if there is a *behavioral* relationship between the classifiers, i.e., instances of P and Q may exchange a message. Although Stevens' classification of associations into static and dynamic associations is somewhat similar to our classification of associations into nonbinding and binding associations (both dynamic and binding associations imply interaction between instances of classifiers), instead of focusing on the correspondence between real world domain and UML conceptual model, her proposal focuses on the correspondence between UML conceptual model and program code, therefore is not suitable for conceptual modeling.

Based on the above discussion, we have Principle 6:

Principle 6: In a UML collaboration, every (binding) link between two or more objects models a Bunge-coupling or binding mutual property shared between things modeled by these objects.

As a result, in a UML collaboration, a (binding) link is completely determined by the objects it links. Note that, usually, not every coupling or binding mutual property shared between two or more things in the domain is modeled as a (binding) link in a UML collaboration. For example, consider a husband and a wife and the husband's employer. It is reasonable that the state history of the wife is not the same as that she would traverse if her husband's employer has not existed. As a result, the wife and the employer are coupled. However, in a UML collaboration, this (indirect) coupling is usually ignored and not modeled as a (binding) link.

Similarly we have Principle 7:

Principle 7: In a UML collaboration, every (binding) association between two or more classes/types models a set of Bunge-couplings or binding mutual properties shared between things modeled by instances of these classes/types.

Also note that, as we discussed in section 3.2.9, the interaction between two or more things (thus they are coupled) will most likely give rise to a number of nonbinding mutual properties. Consequently, in a UML collaboration, whenever a binding link (representing a Bunge-coupling or binding mutual property) exists between two or more objects, there will be a corresponding nonbinding link (representing a tuple of Bunge-

individual nonbinding mutual properties) between these objects in the corresponding class or object diagram. Furthermore, in a UML collaboration, whenever a binding association exists between two or more classes/types, there will be a corresponding nonbinding association between these classes/types in the corresponding class diagram. For example, the interaction between employees and employers may incur a binding association between classes Employee and Employer in a UML collaboration, as well as a nonbinding association (with a tuple of attributes Salary, StartDate, and OfficePhone etc.) between classes Employee and Employer in the corresponding class diagram.

On the other hand, although rare, not every nonbinding association/link in a UML class or object diagram corresponds to a binding association/link in the corresponding collaboration. Examples are all spatiotemporal relations like "Thing A is five kilometers away from thing B". Here, thing A does not act on or is acted upon by thing B, thus they are not coupled.

3.3.5 UML Association Class

In UML, "An association class is an association that is also a class. It not only connects a set of classifiers but also defines a set of features that belong to the relationship itself and not any of the classifiers" [1, p. 2-21]. "An association class is useful when each link must have its own attribute values, operations, or references to objects" [42, p. 157].

From Principles 2 and 5, we know that an attribute of an association/association class models a Bunge-attribute function representing a Bunge-general nonbinding mutual property. Furthermore, we argue in section 3.3.3 that an association class does not model

any functional schema because its instances are links, not objects that model things in the real world domains. Therefore, as suggested by Evermann and Wand in [25], an association class cannot have operations or methods. Instead, operations and methods that change attribute values of an association/association class must be placed in participating *role* classes/types of the association. Moreover, an association class cannot be a composite class.

For example, in Figure 3.2 (adapted from [25, p. 152]), operations RaiseSalary and Terminate must be placed in either Employee or Employer, both are role types participating in the association Job. Note that, attributes of Employee and Employer (Salary and StartDate) model only Bunge-attribute functions representing Bunge-general nonbinding mutual properties shared by employees and employers, thus all of these attributes (at least one) are placed in association/association class Job.

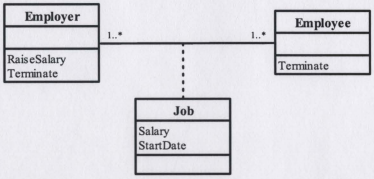

Fig. 3.2. Operations and methods of an association class (adapted from [25])

Moreover, in Figure 3.3 [42, p. 159], association class Job is associated with itself via an association Manages. However, as argued by Evermann and Wand in [25], since instances of Job are links that model values representing Bunge-individual nonbinding

mutual property Salary, and recall that in Bunge's ontology, a property cannot further have properties, Figure 3.3 is not an ontologically correct conceptual model. In fact, Figure 3.3 suggests that instances of Job which are not objects modeling persons in a domain can play role types Boss and Worker, which does not comply with our intuition.

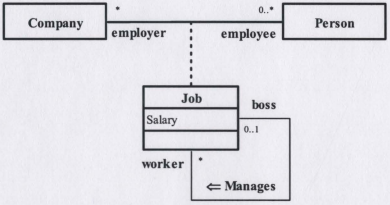

Fig. 3.3. Ontologically incorrect conceptual model using UML [42, p. 159]

Based on the above discussion, we have the following principle:

Principle 8: In a UML class diagram, an association class cannot have operations or methods, cannot be a composite class, and cannot be associated with other class(es).

3.3.6 UML Composition and Aggregation

In UML, "An association may represent an aggregation (i.e., a whole/part relationship). In this case, the association-end attached to the whole element is designated, and the other association-end of the association represents the parts of the aggregation. Only binary

associations may be aggregations. Composite aggregation is a strong form of aggregation, which requires that a part instance be included in at most one composite at a time and that the composite object has sole responsibility for the disposition of its parts. This means that the composite object is responsible for the creation and destruction of the parts. In implementation terms, it is responsible for their memory allocation. If a composite object is destroyed, it must destroy all of its parts." [1, p. 2-66] "A shareable aggregation denotes weak ownership (i.e., the part may be included in several aggregates) and its owner may also change over time. However, the semantics of a shareable aggregation does not imply deletion of the parts when an aggregate referencing it is deleted. Both kinds of aggregations define a transitive, antisymmetric relationship (i.e., the instances form a directed, non-cyclic graph). Composition instances form a strict tree (or rather a forest)." [1, p. 2-66]

It is clear that the distinction between UML-composition/aggregation and between Bunge-aggregate/system is along different dimensions. For UML-aggregation, components are existentially independent of the composite and moreover, they are sharable by other composites. For UML-composition, components are existentially dependent of the composite and they are owned by the composite exclusively. In contrast, for Bunge-aggregate, no couplings among components of the composite, which is not the case for Bunge-system. There are also various kinds of mereology, or formal ontological theory of part, whole, and related concepts [43]. One of our major future works is to analyze characteristics of different whole-part relationships and further investigate how to incorporate them into our ontological core.

In section 3.2.3, we indicate that a composite thing must have at least one emergent property that characterizes the composite thing as a whole. Therefore we have the following principle:

Principle 9: In a UML conceptual model, every composite class/type must own at least one attribute that models a Bunge-attribute function representing a Bunge-emergent property.

3.3.7 UML State

In UML, "A state is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event. … Conceptually, an object remains in a state for an interval of time. However, the semantics allow for modeling "flow-through" states that are instantaneous, as well as transitions that are not instantaneous. A state may be used to model an ongoing activity. Such an activity is specified either by a nested state machine or by a computational expression." [1, p. 3-137] "A state describes a period of time during the life of an object of a class. It can be characterized in three complementary ways: as a set of object values that are qualitatively similar in some respect; as a period of time during which an object waits for some event or events to occur; or as a period of time during which an object performs some ongoing activity." [42, p. 70] Therefore, in UML, there is no precise definition even for such fundamental notion as state. We propose that a UML-state models a Bunge-lawful state and a Bunge-lawful state is modeled as a UML-state.

Principle 10: In a UML conceptual model, every state of an object models a Bunge-lawful state of a thing modeled by the object. Conversely, a Bunge-lawful state of a thing in the domain is modeled as a state of an object modeling this thing.

From Principles 3 and 9, we may conclude that a state of an object in a UML conceptual model is a tuple of values of the attributes of its class/type.

3.3.8 UML State Transition

In UML, "A transition is a directed relationship between a source state vertex and a target state vertex. It may be part of a compound transition, which takes the state machine from one state configuration to another, representing the complete response of the state machine to a particular event instance." [1, p. 149] "A simple transition is a relationship between two states indicating that an instance in the first state will enter the second state and perform specific actions when a specified event occurs provided that certain specified conditions are satisfied. On such a change of state, the transition is said to "fire"." [1, p. 3-145] We propose that a UML-simple state transition models a Bunge-lawful event and a Bunge-lawful event is modeled as a UML-simple state transition.

Principle 11: In a UML conceptual model, every simple state transition of an object models a Bunge-lawful event of a thing modeled by the object. Conversely, a Bunge-

lawful event of a thing in the domain is modeled as a simple state transition of an object modeling this thing.

In UML, "An event is a specification of a type of observable occurrence. The occurrence that generates an event instance is assumed to take place at an instant in time with no duration." [1, p. 2-144] "Event instances are generated as a result of some action either within the system or in the environment surrounding the system. An event is then conveyed to one or more targets." [1, p. 2-155] "An event is *received* when it is placed on the event queue of its target. An event is *dispatched* when it is dequeued from the event queue and delivered to the state machine for processing. At this point, it is referred to as the *current event*. Finally, it is *consumed* when event processing is completed." [1, p. 2-155] It is obvious that the notion of UML-event is quite different from that of Bunge-event which is simply a state transition. One of our future works is to investigate and develop ontological foundation for UML dynamic aspects.

3.3.9 UML Operation and Method

In UML, "An operation is a service that can be requested from an object to effect behavior." [1, p. 2-44] "Operation is a conceptual construct, while Method is the implementation construct." [1, p. 2-71] "A method is the implementation of an operation. It specifies the algorithm or procedure that effects the results of an operation." [1, p. 2-40] Since UML-method is merely an implementation construct of UML-operation, it is clear that UML-method does not have any counterpart in Bunge's ontology. On the other hand,

since UML-operation can be requested to effect object behavior thus change its state, we propose that a UML-operation models a Bunge-lawful transformation and a Bunge-lawful transformation is modeled as a UML- operation.

Principle 12: In a UML conceptual model, every operation of a class/type models a Bunge-lawful transformation of a functional schema modeled by the class/type. Conversely, a Bunge-lawful transformation of a functional schema in the domain is modeled as an operation of a class/type modeling this functional schema.

3.3.10 Comparisons of Ontological Matches

The following table compares the proposed ontological semantics with those of Evermann & Wand [25] and Opdahl & Henderson-Sellers [3].

From table 3.1, it is clear that, compared with our approach, Evermann and Wand did not distinguish Bunge's individual property from general property. Thus they could not distinguish ontological counterparts of UML-link and association in Bunge's ontology. Similarly, they also did not distinguish binding and nonbinding mutual properties. In the approach of Opdahl and Henderson-Sellers, they map Bunge-mutual property directly to UML-link. This is problematic because a UML-link may actually represent a bundle of Bunge-mutual properties.

Table 3.1. Comparisons of ontological matches between UML and Bunge's ontology

UML Element	Proposed Approach	Evermann & Wand	Opdahl & Henderson-Sellers
Object	Bunge-thing	Bunge-thing	BWW-thing
Attributes	Bunge-attribute function	Bunge-property	BWW-intrinsic property
Class/Type	Bunge-functional schema	Bunge-functional schema	BWW-natural kind
Link in Class/Object diagram	A tuple of values representing semantically related Bunge-individual nonbinding mutual properties	A bundle of mutual properties	BWW-mutual property that is not a law or whole-part reation
Association	A tuple of Bunge-attribute functions representing semantically related Bunge-general nonbinding mutual properties		BWW-characteristic mutual property that is not a law or whole-part reation
Link in collaboration diagram	Bunge-coupling or binding mutual property		BWW-coupling or binding mutual property
State	Bunge-lawful state	Bunge-lawful state	BWW-state
Simple state transition	Bunge-lawful event		BWW-transformation law
Operation	Bunge-lawful transformation		BWW-transformation

3.4 Conclusion

In this chapter, we assign ontological semantics based on Bunge's ontology to a core set of UML constructs, namely UML object, attribute, class/type, association, link, association class, state, state transition, and operation. The choice of these UML constructs is driven by Bunge's ontology (In the future, more UML constructs will be considered). We also analyze consequences for conceptual modeling using UML based on this semantic mapping. In particular, we have focused on UML association and link and indicate that links and their corresponding associations in UML class diagrams are of

fundamentally different ontological nature from those in UML collaborations. As a result, the so-called *Baseless Link Problem* disappears naturally. Consequently, our result suggests that Bunge's ontology is an appropriate foundation for identifying the fundamental constructs and the relationships among them that need to be supported by a conceptual modeling language.

Chapter 4

An Ontological Metamodel of Classifiers For Conceptual Modeling

4.1 Introduction

In Chapter 3 Principle 3, we proposed that Bunge-functional schema is modeled by UML-class/type. Moreover, in Bunge's ontology, a thing may have multiple functional schemata, reflecting different views of the same thing. However, Bunge does not give guidelines as to how to differentiate different "kinds" of functional schemata. In this Chapter, we employ OntoClean methodology [47][48][49][50] and Guizzardi et al.'s ontological profile [51][52] to further distinguish different "kinds" of classes/types. We first introduce OntoClean methodology and Guizzardi et al.'s ontological profile, based on which we propose an ontological metamodel of classifiers and incorporate it into our ontological framework built on Bunge's ontology. Then, we explore the implications of this ontological metamodel for object-oriented and conceptual modeling as well as

information system design and implementation focusing on discussing the notion of role and its representation in object-oriented and conceptual modeling literature.

4.2 Related Work

4.2.1 OntoClean Methodology

OntoClean is a methodology proposed by Guarino and Welty in [47][48][49][50] with the purpose of validating taxonomies by exposing inappropriate and inconsistent modeling choices. It introduces a set of highly general ontological notions (*rigidity*, *identity*, *unity*, and *dependence*) to analyze ontological semantics of various types as well as their relationship. These meta-properties impose several constraints on taxonomic relationships which help to reveal modeling problems in taxonomies. "Properly structured taxonomies help bring substantial order to elements of a model, are particularly useful in presenting limited views of a model for human interpretation, and play a critical role in reuse and integration tasks. Improperly structured taxonomies have the opposite effect, making models confusing and difficult to reuse or integrate" [48]. Therefore, insights into how to properly construct a taxonomy are very useful for conceptual modeling and information system development.

In an ontology, every property (type) can be labeled using these metaproperties or their variants. Given a particular possible world, a property is associated with a set of entities that exhibit that property in that particular world (i.e., its extension) in OntoClean terminology. These entities are instances of that property. The meaning of *subsumption* is

defined as "A property p subsumes q if and only if, for every possible state of affairs, all instances of q are also instances of p" [47, p. 2]. It should be emphasized that instead of discussing the ontological nature of properties, the purpose of OntoClean is to make clear the logical consequences of the modeling choice made by the model designer.

4.2.1.1 Rigidity

In order to define rigidity, Guarino and Welty first define essentiality: "A property of an entity is *essential* to that entity if it *must* be true of it in every possible world, i.e. it *necessarily holds* for that entity" [47, p. 3]. For example, *being a person* is essential to persons, whereas *being heavy* is not essential to laptops. Then "a property is *rigid* if it is essential to all its possible instances". An instance of a rigid property (labeled $+R$) cannot stop being an instance of that property in a different world. For example, *being a person* is rigid, whereas *being a customer* is not. Of *non-rigid* properties, Guarino and Welty further distinguish properties that are not essential to some of their instances and essential to others (*semi-rigid*, labeled $-R$) from properties that are not essential to all of their instances (*anti-rigid*, labeled $\sim R$). For example, *being a customer* is anti-rigid, whereas *being seatable* is semi-rigid (be essential to Chair but accidental to Paper Box).

4.2.1.2 Identity

"Identity refers to the problem of being able to recognize individual entities in the world as being the same (or different)" [47, p. 4]. Identity criteria are conditions used to

determine equality (sufficient conditions) and that are *entailed* by equality (necessary conditions). If a property carries an identity criterion (labeled $+I$), then it is called a *sortal*; otherwise, it is labeled with $-I$. Moreover, identity criteria are inherited along property subsumption hierarchies. If instead of inheriting from the subsuming properties, a property supplies its own identity criteria, then instead of $+I$, it is labeled $+O$. For example, *being a person* supplies its identity criteria whereas *being a student* only carries (or inherits) it.

4.2.1.3 Unity

"Unity refers to the problem of describing the parts and boundaries of objects, such that we know in general what is the part of the object, what is not, and under what conditions the object is *whole*" [47, p. 5]. For some properties, all their instances are wholes (e.g., *being an ocean*), whereas for other properties, their instances are not (e.g., *being an amount of water*). If all the instances of a property carry a common unity criterion – a specific condition that must hold among the parts of each instance of the property in order to consider the instance a whole, then this property is labeled with metaproperty $+U$ (e.g., *being an ocean*); if all the instances of the property are wholes, but with different unity criteria, then this property is labeled $-U$ (*semi-unity*, e.g., *being legal agent*, with both people and companies with different unity criteria among its instances); otherwise, if all the instances of the property are not necessarily wholes, the property is labeled $\sim U$ (*anti-unity*, e.g., *being an amount of water*).

4.2.1.4 Dependence

The final metaproperty is *dependence*. A property p is *externally dependent* on a property q if for any of its instances x, necessarily some instance of q exists, which is not a part nor a constituent of x [48]. If p is externally dependent, then it is labeled with $+D$, otherwise $-D$. For example, *being a student* is externally dependent on *being a university*, whereas *being a person* is independent.

4.2.1.5 Ontological Constraints on Taxonomic Relationships

These four meta-properties impose several constraints on taxonomic relationships which help to reveal modeling problems in taxonomies. Given two properties p and q, q subsumes p, then the constraints below hold [47][48]:

1. If q is anti-rigid, then p must be anti-rigid;
2. If q carries an identity criterion, then p must carry the same criterion;
3. If q carries a unity criterion, then p must carry the same criterion;
4. If q has anti-unity, then p must also have anti-unity;
5. If q is externally dependent, then p must be externally dependent.

Guarino and Welty further make two assumptions (*Sortal Individuation* and *Sortal Expandability*) regarding identity, which imply that every domain element must instantiate a *unique* most general property carrying a criterion for its identity [47, p. 6].

The above constraints can be used to analyze and evaluate a taxonomy after assigning each property in it corresponding metaproperties discussed above. When a violation is encountered, the assigned metaproperties and/or the subsumption link should be reconsidered and some corrective modification to the taxonomy made. For example, the property *being a customer* ($\sim R$) cannot subsume property *being a person* ($+R$) according to constraint 1. Intuitively, not every person is a customer. Another example is the property *being an amount of water* ($\sim U$) cannot subsume property *being an ocean* ($+U$). Oceans are not an amount of water, they are *composed of* water.

Moreover, Guarino and Welty distinguish eight different kinds of properties based on different combinations of these metaproperties, which they think valid and most useful (table 4.1). In [50], a more detailed discussion of these property kinds as well as where they should appear in a taxonomy is given.

Table 4.1. Different combinations of the metaproperties [48]

<table>
<tr><td rowspan="2">+O</td><td rowspan="2">+I</td><td rowspan="2">+R</td><td>+D</td><td rowspan="2">Type</td><td rowspan="11">Sortal</td></tr>
<tr><td>-D</td></tr>
<tr><td rowspan="2">-O</td><td rowspan="2">+I</td><td rowspan="2">+R</td><td>+D</td><td rowspan="2">Quasi-type</td></tr>
<tr><td>-D</td></tr>
<tr><td>-O</td><td>+I</td><td>~R</td><td>+D</td><td>Material role</td></tr>
<tr><td>-O</td><td>+I</td><td>~R</td><td>-D</td><td>Phased sortal</td></tr>
<tr><td rowspan="2">-O</td><td rowspan="2">+I</td><td rowspan="2">¬R</td><td>+D</td><td rowspan="2">Mixin</td></tr>
<tr><td>-D</td></tr>
<tr><td rowspan="2">-O</td><td rowspan="2">-I</td><td rowspan="2">+R</td><td>+D</td><td rowspan="2">Category</td><td rowspan="5">Non-sortal</td></tr>
<tr><td>-D</td></tr>
<tr><td>-O</td><td>-I</td><td>~R</td><td>+D</td><td>Formal role</td></tr>
<tr><td rowspan="3">-O</td><td rowspan="3">-I</td><td>~R</td><td>-D</td><td rowspan="3">Attribution</td></tr>
<tr><td rowspan="2">¬R</td><td>+D</td></tr>
<tr><td>-D</td></tr>
<tr><td rowspan="3">+O</td><td>-I</td><td></td><td></td><td rowspan="3">incoherent</td><td></td></tr>
<tr><td rowspan="2">+I</td><td>~R</td><td></td><td></td></tr>
<tr><td>-R</td><td></td><td></td></tr>
</table>

Guarino and Welty then propose an idealized view of how ontologies should be structured taxonomically as shown in Figure 4.1, "While strict adherence to this idealized structure may not always be possible, we believe that following it to the degree possible will grow to be an important design principle for conceptual modeling, with payoffs in understandability and ease of integration" [48].

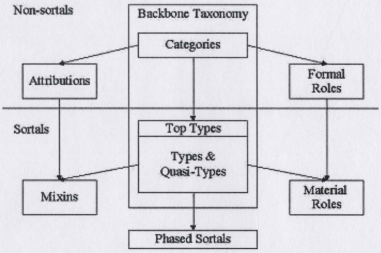

Fig. 4.1. Ideal taxonomy structure [48]

4.2.2 Representing Roles With Multiple Disjoint Allowed Types

In object-oriented and conceptual modeling, role is a powerful modeling concept. However, a lot of confusion exists on the definition, properties, and representation of role. Among them, the problem of representing roles with multiple disjoint allowed natural

types[3] is discussed in a series of papers by different authors. In [54][55][56], Steimann argues using an example illustrated in Figure 4.2 that "viewing role types as subtypes of natural types is a consequence of an inadmissible intermingling of the dynamic nature of the role concept with the static properties of type hierarchies". In Figure 4.2, Person and Organization are natural types; role types Customer and Supplier are their subtypes (Figure 4.2(a)). However, since the intersection of the extensions of Person and Organization is empty, the extensions of Customer and Supplier have to be empty, which is clearly not the case. As a result, Steimann argues to separate natural types and role types into different type hierarchy and use role-filler relationship to relate them, as demonstrated in Figure 4.2(b).

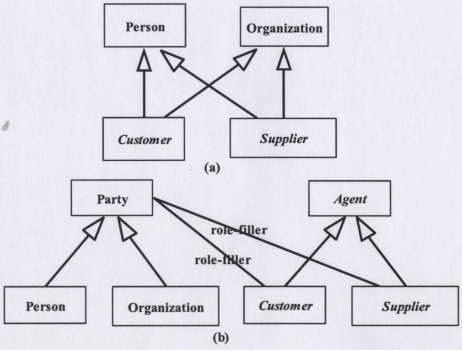

Fig. 4.2. Relating role types and natural types through role-filler relationship [48]

[3] In [56], Sowa distinguishes *natural types* that relate to the essence of its instances from *role types* whose instances depend on an accidental relationship to some other entity. Here, role types include formal role and material role in table 4.1 and natural types include type, quasi-type, and category.

For a deeper analysis on this issue, we can now employ the ontological constraints of OntoClean on taxonomic relationships to analyze the admissible relationship between natural types and role types. According to OntoClean, role types (e.g., Customer and Supplier) cannot be supertypes of natural types (e.g., Person and Organization) in a type hierarchy because a concept which is anti-rigid cannot subsume another which is rigid. For example, usually, not all persons are customers. The type hierarchy in Figure 4.2(a) (role types are italicized) does not have this problem. On the other hand, can role types be subtypes of natural types? Employing OntoClean, we can see that the taxonomy in Figure 4.2(a) is not ontologically correct because natural type Person, which has a common identity criterion ($+I$), cannot subsume role type Customer or Supplier which does not have a common identity criterion ($-I$) (because a customer or supplier can be a person or an organization). Therefore, it seems that Steimann's solution of separating natural type hierarchy from role type hierarchy is correct. However, in [51], Guizzardi et al. argue using an ontological profile for UML conceptual models that Steimann's solution is not warranted.

4.2.3 Guizzardi et al.'s Ontological Profile For UML Conceptual Models

In [51], Guizzardi et al. propose a UML profile for ontology representation and conceptual modeling based on a theory of classifiers. Basically, their proposal is similar to OntoClean, but also with some important differences detailed in [52]. In [51],

Guizzardi et al. postulate four ontological principles for ontology representation and conceptual modeling.

First, *in a conceptual model of a domain, every object must be an instance of a class representing a sortal*. This is because only sortal universals provide a principle of individuation and identity to their instances. As a result, we can only make identity and quantification statements in relation to a sortal. Moreover, if an individual is an instance of two sortals in the course of its history, then there must be exactly one ultimate sortal of which both sortals are specializations. A sortal F is ultimate if there is no other sortal F' distinct from F which F specializes. For example, Person is the ultimate sortal of sortals Child and Adult. This amounts to the *Restriction* and *Uniqueness* principles [51, p. 115]: if an individual is an instance of two distinct sortals F and F' in the course of its history, then a) there is at least one sortal of which F and F' are both specializations; b) there is only one ultimate sortal of which F and F' are both specializations.

Second, *in a conceptual model of a domain, an object cannot instantiate more than one class representing an ultimate substance sortal* (A substance sortal is the unique ultimate sortal that supplies the principle of identity for its instances). For example, sortal Person is the ultimate substance sortal of sortals Child and Adult.

The third principle postulated by Guizzardi et al. is, *in a conceptual model of a domain, a class representing a rigid classifier cannot be a subclass of a class representing an anti-rigid classifier*. It is actually a constraint of OntoClean. Note that, the notion of phased-sortal in [51] is different from that in [48] in that it also includes material role type in [48], i.e., instead of $+I \sim R - D$, the metaproperties of phased-sortal in [51] are $+I \sim R$. Guizzardi et al. distinguish two important types of phased-sortals

according to specialization conditions φ: phase types (φ is a condition that depends solely on intrinsic properties) and role types (φ depends on extrinsic properties). It is the phase type in Guizzardi et al.'s ontological profile that corresponds to phased-sortal in OntoClean. Classes representing phase types constitute a partition of the substance sortal they specialize. Examples of phase types include Caterpillar and Butterfly of Lepidopteron, Town and Metropolis of City.

Furthermore, Guizzardi et al. distinguish sortals from dispersive types (such as Thing, Customer, Supplier) which cover many concepts with different principles of identity. Based on this, the fourth principle is, *in a conceptual model of a domain, a class representing a dispersive universal cannot be a subclass of a class representing a sortal.* In fact, a dispersive type has no common identity criterion for its instances ($-I$) whereas a sortal has ($+I$), so this principle is actually a constraint in OntoClean.

Based on these principles, Guizzardi et al. propose a UML profile for ontology representation and conceptual modeling. In the profile, a UML class stereotyped as a ≪kind≫ represents a substance sortal. Subtypes that specialize kinds and inherit their principle of identity are stereotyped as ≪subkind≫. For example, Person is a kind and its subkinds could be Man and Woman. Moreover, stereotypes ≪phase≫ and ≪role≫ represent phase type and role type respectively. Guizzardi et al. further distinguish dispersive types into category (which is rigid, stereotyped as ≪category≫), rolemixin (which is anti-rigid, stereotyped as ≪rolemixin≫), and mixin (which is semi-rigid, stereotyped as ≪mixin≫). A summary of different types of classifiers and the

restrictions on their specialization relationships can be illustrated in table 4.2 (taken from [51, p. 122]).

Table 4.2. Different types of classifiers and the restrictions on their specialization relationships [51, p. 122]

Stereotype	Constraints
≪kind≫	Supertype is not a member of {≪subkind≫, ≪phase≫, ≪role≫, ≪rolemixin≫}
≪subkind≫	Supertype is not a member of {≪phase≫, ≪role≫, ≪rolemixin≫}
≪phase≫	Always defined as part of partition
≪role≫	Let X be a class stereotyped as ≪role≫ and r be an association representing X's restriction condition, then $\#X.r \geq 1$
≪category≫	Supertype is not a member of {≪kind≫, ≪subkind≫, ≪phase≫, ≪role≫, ≪rolemixin≫}
≪rolemixin≫	Supertype is not a member of {≪kind≫, ≪subkind≫, ≪phase≫, ≪role≫}. Let X be a class stereotyped as ≪rolemixin≫ and r be an association representing X's restriction condition, then $\#X.r \geq 1$
≪mixin≫	Supertype is not a member of {≪kind≫, ≪subkind≫, ≪phase≫, ≪role≫, ≪rolemixin≫}

4.2.4 Representing Roles With Multiple Disjoint Allowed Types Revisited

Based on the profile described in section 4.2.3, Guizzardi et al. propose a design pattern to represent the problem of role modeling discussed in section 4.2.2, i.e., role type Customer cannot be represented as subtype of natural types Person and Organization. They argue that Steimann's claim that the solution to this problem lies in the separation of role type and natural type hierarchies (which leads to a radical revision of the UML metamodel) is not warranted. According to their theory, the difficulty arises because

Customer (with metaproperties $\sim R+D$) is a non-sortal (or a dispersive type, $-I$) that has in its extension individuals that belong to different kinds Person and Organization that obey different principles of identities. Therefore, they propose that an ontologically correct solution to this problem is to, for example, define the sortals PrivateCustomer and CorporateCustomer as subtypes of Customer, and these sortals in turn carry the (incompatible) principles of identity inherited from kind Person and subkind Organization, respectively. This could be illustrated in Figure 4.3.

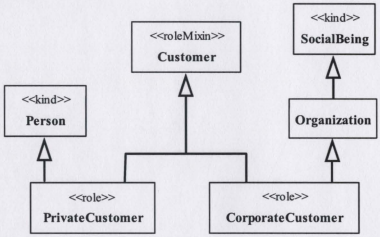

Fig. 4.3. A design pattern for the problem of representing role types with multiple disjoint allowed natural types [51, p. 123]

4.2.5 Roles and Their Representation in Object-Oriented and Conceptual Modeling

Roles represent a fundamental notion for our conceptualization of reality. As manifested by the recent AAAI symposium on role [60], the notion of role is ubiquitous in many areas of computer science ranging from conceptual modeling, artificial intelligence, programming languages, software engineering, database, coordination, multiagent

systems, computational linguistics, and also in other scientific fields like formal ontology, sociology, cognitive science, organizational science, and linguistics.

In object-oriented and conceptual modeling, role is a powerful modeling concept. However, a lot of confusion exists on the definition, properties, and representation of role. Up to now, most role models proposed have been primarily based on implementation considerations. In contrast, in a comprehensive study on this topic [54][55][56], Steimann argues that the role concept naturally complements those of object and relationship, standing on the same level of importance. However, he also recognizes that "although there appears to be a general awareness that roles are an important modeling concept, until now no consensus has been reached as to how roles should be represented or integrated into the established modeling frameworks" [54, p. 84].

In this section, we first present the notion of roles and some of its fundamental features in the literature, then we discuss three different ways of representing roles, highlighting existing difficulties with each approach.

4.2.5.1 On The Notion of Role in Object-Oriented and Conceptual Modeling

Two main definitions of role given by Merriam-Webster online dictionary are: (1) a character assigned or assumed; (2) a socially expected behavior pattern usually determined by an individual's status in a particular society. In sociology, a role or social role is a set of connected behaviors, as conceptualized by actors in a social situation. It is mostly defined as an expected behavior in a given individual social status.

In [39] and [61], Sowa introduces the notion of role as capturing *a particular pattern of relationships*: "Subtypes of Entity are of two kinds: *natural types*, which have no required set of linguistic associations; and *role types*, which are subtypes of natural types in some particular pattern of relationships". An example he uses to illustrate his idea is: Person is a natural type, and Teacher is a subtype of Person in the role of teaching. Sowa further proposes a test for distinguishing role types from natural types: (1) τ is a natural type if something can be identified as type τ in isolation; (2) τ is a role type if something can only be identified as type τ by considering some other entity, action, or state.

Although intuitively appealing, in [63], Guarino argues against Sowa's view towards role by indicating that Sowa's test for distinguishing role types from natural types is "too vague to capture intended meaning". For example, Car is a natural type since it is essentially independent. However, it is also a role type according to (2) above because the existence of a car implies the existence of its engine, which is a part of the car. Guarino therefore proposes a criterion for distinguishing role types from natural types. To do that, he first introduces the notion of *foundation* (*dependence* in OntoClean) among concepts: in order for concept α to be founded on another concept β, any instance x of α has to be necessarily associated to an instance y of β *which is not related to* x *by a part-of relation*. For example, Son is founded since sons are associated to their parents. On the other hand, Car is essentially independent since although the existence of a car implies the existence of its engine, this engine is part of the car. The second notion he introduces is *rigidity*, which gives a more adequate characterization of the meaning of

roles: a concept α is rigid if it contributes to the very identity of its instances in such a way that, if x is an instance of α in a particular world, it must be an instance of α in any possible world in order to keep its identity. For example, a person can cease to be a customer or supplier while still being a person: Person is rigid, while Customer and Supplier are not. Based on these two notions, Guarino proposes that, a concept α is called a *role* concept if it is founded but not rigid; it is called a *natural* concept if it is essentially independent and rigid.

Moreover, by characterizing different approaches on the basis of the ontological nature of the contexts that determine roles, Loebe differentiates three kinds of role: *relational* roles, *processual* roles, and *social* roles [64]. And more recently, Masolo and his colleagues propose a general formal framework for developing a foundational ontology of socially constructed entities [65], maintaining four key features of social roles: (1) Roles are 'properties'; (2) Roles are anti-rigid; (3) Roles have a relational nature; and (4) Roles are linked to contexts.

In object-oriented and conceptual modeling, the first role model (*role data model*) is introduced by Bachman and Daya in [62] as an extension of the network model in the late 1970s. Up to now, most role models proposed in the literature have primarily focused on implementation considerations. In a comprehensive study by Steimann in [54][55][56], he individuates 15 fundamental features of roles identified in the literature, some conflict with others. The following is a subset of the list of features of particular interest to us (the rest are mostly implementation-oriented) [54, p. 86]:

1. *A role comes with its own properties and behavior*. For example, a student has his/her role specific properties (such as StartDate, StudentID, CoursesTaken, etc.) and behavior (such as Register, AttendClass, etc.). This feature suggests that role is type.
2. *Roles depend on relationships*. This indicates that a role is meaningful only in the context of a relationship. For example, a person is a student only when he is enrolled in a school. After he graduates from the school (thus the enrollment relationship between this person and the school no longer exists), he ceases to be a student.
3. *An object may play different roles simultaneously*. For example, a person can be a student and an employee at the same time.
4. *An object may play the same role several times, simultaneously*. For example, a person can be a student in several universities at the same time, each with different StudentID, StartDate, etc. This is a difficult issue in the literature of role modeling as will be discussed later in detail.
5. *An object may acquire and abandon roles dynamically*.
6. *Objects of unrelated types can play the same role*. For example, both a person and an organization can be a customer. This feature amounts to the polymorphism inherent to roles.
7. *Roles can play roles*.
8. *Different roles may share structure and behavior*. This implies that role types may be organized in a generalization/specialization hierarchy. For example, role

type TeamLeader may inherit structure and behavior from its super role type Employee.

9. *An object and its roles share identity*. I.e., an object and its roles are the same.
10. *An object and its roles have different identities*. The motivation for this view lies in the so-called *Counting Problem*. It refers to the situations in which instances counted in their roles yield a greater number than the same instances counted by the objects playing the roles. The counting may be done at the same time or in a certain period of time. The former situation is described in feature 4. The latter can be shown by the example below: if we count the number of persons served by Air Canada in 2005, we may count 1000, but if we count passengers, we may count 3000.

Clearly, feature 9 above conflicts with feature 10, so there is not one ideal approach of defining the role concept.

4.2.5.2 Three Different Ways of Representing Roles

As indicated by Steimann [54], there are three different viewpoints on the representation of roles, i.e., roles as named places in relationships, roles as a form of generalization/specialization, and roles as separate instances adjoined to the entities playing the roles.

4.2.5.2.1 Roles as Named Places in Relationships

This view is taken by ER (and many of its extensions) and UML by assigning role names to the entity types participating in relationships. This practice is useful when, in a conceptual model, more than one place of a relationship is played by the same entity type. In this case, role names could be used to differentiate different places of the same entity type.

However, Steimann argues that the main problem with viewing roles as named places in relationships is that, since roles are not modeled as explicit types, "it fails to account for the fact that roles come with their own properties and behavior" [54, p. 88]. Furthermore, since roles are mere labels of types, it is thus impossible to construct role type generalization/specialization hierarchies leading to better-organized conceptual models.

4.2.5.2.2 Roles as a Form of Generalization/Specialization

As discussed in section 4.2.5.1, Sowa views role types as subtypes of natural types in some particular pattern of relationships [39][61]. This view is quite nature considering, for example, the fact that a student is also a person who is enrolled in a university. However, in [66], Al-Jadir and Leonard argue using a number of implementation level examples that inheritance is not flexible enough with respect to object dynamics and schema evolution. Additionally, Dahchour et al. acknowledge in [57] that it is "heavy" and "impractical" to deal with object dynamics using generalization/specialization.

More interestingly, as discussed in section 4.2.2 and 4.2.4, a more inherent conceptual obstacle with this view is the difficulty of representing roles allowing multiple disjoint types. As a consequence, in [54], Steimann argues to separate natural types and role types into different type hierarchy and use role-filler relationship to relate them. However, Guizzardi et al. further indicate using their ontological profile that this problem can be resolved by classifying dispersive role types into a number of role subtypes which are sortals. Therefore, Steimann's claim which leads to a *radical* revision of the UML metamodel is not warranted. Although Guizzardi et al.'s solution provides some useful insight into the problem, we still believe that using generalization/specialization hierarchy alone is cumbersome thus not suitable for dealing with role modeling.

4.2.5.2.3 Roles as Separate Instances Adjoined to The Entities Playing The Roles

In this view, role types are treated as independent types whose instances are existentially dependent on its players (instances of natural types), have role specific state and behavior with separate identity different from their players. A player and its roles are related by a *played-by* relation, thus role instances act as bridges between relationships and its related players. Examples of the approaches that adopt this view are [57][58][59] and [64].

As argued by Steimann in [67] and Masolo et al. in [68], this view is mostly motivated to model some real world situations such as *a person plays exactly three employee roles simultaneously, with different salary and office number*, i.e., the counting problem described in section 4.2.5.1. In this case, each employee role instance of a person

object is a quasi-object in that it describes a state of the corresponding object playing the role in a particular context.

Although it looks appealing, this approach is problematic because, as we proposed in Chapter 3, an object in a conceptual model should correspond to a distinct thing in the real world. Moreover, the requirement that each role instance has a unique identity (e.g., EmployeeID) different from its player is quite artificial which may lead to some problems in information systems development such as database modeling. Detailed discussion on this issue will be given in Chapter 5.

Indeed, as stated by Steimann [67], "as far as I can see there is no practical need to do this, nor do good theoretical arguments exist".

4.3 An Ontological Metamodel of Classifiers For Object-Oriented and Conceptual Modeling

In this section, we propose an ontological metamodel of classifiers based on OntoClean methodology and Guizzardi et al.'s ontological profile and incorporate it into our ontological framework built on Bunge's ontology. Then we explore the implications of this ontological metamodel for object-oriented conceptual modeling as well as information system design and implementation focusing on discussing the notion of role and its representation in object-oriented and conceptual modeling literature.

4.3.1 Conceptual Modeling vs. Ontological Modeling

In [51], Guizzardi et al. indicate that "Conceptual modeling is concerned with identifying, analyzing and describing the essential concepts and constraints of a domain with the help of a (diagrammatic) modeling language that is based on a small set of basic meta-concepts (forming a metamodel). Ontological modeling, on the other hand, is concerned with capturing the relevant entities of a domain in an ontology of that domain using an ontology specification language that is based on a small set of basic, domain-independent ontological categories (forming an upper level ontology)". They then indicate that "While conceptual modeling languages are evaluated on the basis of their successful use in (the early phases of) information systems development, ontology specification languages and their underlying upper level ontologies have to be rooted in principled philosophical theories about what kinds of things exist and what their basic relationships with each other are".

It is clear that Guizzardi et al. try to clarify or emphasize the distinction between conceptual modeling and ontological modeling. It seems from [51] that the most important difference between conceptual modeling and ontological modeling is their purposes and thus their evaluation criteria: the purpose of conceptual modeling languages is to support (the early phases of) information systems development, therefore their evaluation criterion is whether they could be successfully useful in information system development; the purpose of ontology specification languages, on the other hand, is to model reality, therefore their evaluation criterion is whether or not they conform to philosophical theories.

One of our main research objectives is to use ontology to improve UML's ability of conceptual modeling, i.e., we focus on the similarity between ontological modeling and

conceptual modeling. Indeed, as argued by Marcos and Cavero in [53], one of the most important objectives in conceptual modeling consists in narrowing the gap between the real world and its representation. To make this possible, conceptual models have improved their expressiveness through new primitives that are closer to the real world. Therefore, ontological theories are useful sources of inspiration for conceptual modeling language designers.

In contrast with our objective, Guizzardi et al. focus on using UML as an ontology representation language. As a result, they use their theory to propose a UML profile for ontological representation which requires no changes to be made to the UML metamodel.

However, we believe that although their profile is good for constructing conceptual models organized in taxonomies conforming to some ontological commitments, using generalization/specialization hierarchy alone is not suitable for dealing with information system development because usually it does not seem reasonable to have to differentiate what kind of Customer (PrivateCustomer or CorporateCustomer) the information system is dealing with. Modelers should have the freedom of not having to specify this clearly in a conceptual model. In fact, one of the most important advantages of introducing the notion of role into conceptual modeling languages is that instances of different natural types can play the same role type (in this case, this role type is always dispersive. Please refer to feature 6 in section 4.2.5.1), which amounts to the polymorphism inherent to roles. Due to this reason, we should avoid modeling roles as generalization/specialization.

4.3.2 An Ontological Metamodel of Classifiers

In Figure 4.4, the ontological metamodel based on OntoClean and Guizzardi et al.'s profile (using the terms of the latter) is shown. Note that we do not consider mixin in Guizzardi et al.'s profile (e.g., *being seatable* or *being red*) because in OntoClean, it is actually attribution and according to Guarino and Welty, "in general, it is not useful to represent attributions explicitly in a taxonomy, and that the proper way to model attributions is with a simple attribute" [47, p.18].

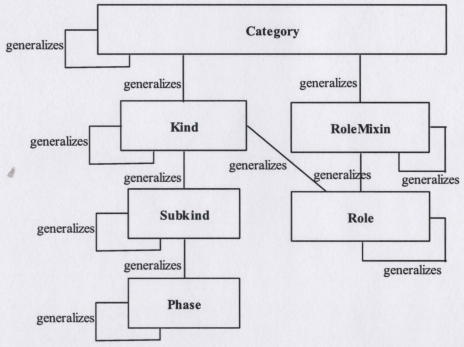

Fig. 4.4. The ontological metamodel based on OntoClean and Guizzardi et al.'s profile

Moreover, in contrast to rigidity and dependence, we assume that in most cases, object modelers do not need to differentiate whether a type in a conceptual model supplies or carries a common principle of identity or not. Then based on this assumption,

we may combine rolemixin (which includes formal role in OntoClean) and role (material role in OntoClean) (their only difference is $-I/+I$) into one type called *role* type. Similarly, we combine category, kind, and subkind (their only difference is $-O-I/+O+I/-O+I$) into one type called *natural* type. Thus in Figure 4.4, there are only three metatypes: natural type ($+R\ -D$), phase type ($\sim R\ -D$), and role type ($\sim R\ +D$). Based on the discussion in section 4.3.1, we still propose to separate natural/phase type hierarchy from role type hierarchy and connect them using a relationship called *plays*. In this way, our ontological metamodel complies with the proposed ontological principles in [51][47] and at the same time is suited for information system development. The simplified metamodel is illustrated in Figure 4.5.

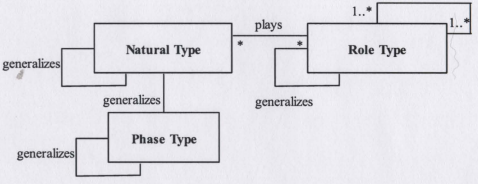

Fig. 4.5. The simplified metamodel of Figure 4.4

It should be emphasized that in a phase type partition of a natural type or phase type, the subtypes should be constructed such that they are mutually disjoint and constitute a *total* partition of this supertype, i.e.,, any instance of the supertype should have a corresponding instance in exactly one phase subtype.

Also note that, different from natural types and phase types, there are relationships among role types (because role types are externally dependent). Each role type must associate to at least one other role type (could be the same role type), therefore has the cardinality constraint 1..*. For example, role type Student must be related to at least a role type UniversityEnrolled. In OntoClean, natural types may be externally dependent, which is disallowed in Guizzardi et al.'s profile. In our model, we adopt the latter view. That is to say, we consider that instances of natural types and phase types exist in isolation from any external entities. For an instance of a natural type or phase type to be able to interact with other entity(-ies) in a particular context, it must become an instance of a role type. On the other hand, at implementation level, in order for instances of role types to access features of natural/phase types, *delegation* from roles to their players can be used [71].

The cardinality constraints on *plays* relationship in the metamodel indicate that a natural type or phase type can play zero to many role types and a role type can be played by zero to many natural types or phase types simultaneously. For clarity, we call instances of a natural type *objects*, instances of a role type *roles*, and instances of a phase type *phases*.

Note that, like Steimann's role model in [54][55][56], in our model, instances of role types are directly recruited from natural/phase types. In fact, natural type, phase type, and role type are just different kinds of functional schemata of the same real world things viewed from different perspectives. In contrast, in some role models such as [57][58][59], a role type has its own instances with separate global unique identity different from other roles as well as from instances of natural types and phase types, i.e., roles as adjunct

instances representation. Although intuitively appealing, as we discussed in section 4.2.5.2.3, this approach is conceptually problematic. In Chapter 5, we will compare this approach with ours with respect to conceptual database modeling in order to highlight its drawbacks.

Also note that, different from [54][55][56], in our model, since instances of role types are also instances of natural/phase types, roles can also play roles. Although a more technical consideration, this is desirable in some cases such as we want to model that Instructor is a subset of the union of Faculty and GraduateStudent. Here, Faculty and GraduateStudent are disjoint. Since not every graduate student is an instructor, we cannot model Instructor as supertype of Faculty and GraduateStudent, nor can it be subtype of Faculty and GraduateStudent (because they are disjoint). Thus the only way to model it is to let Faculty and GraduateStudent play role type Instructor.

As to the combination of generalization/specialization and role relationships, informally, we say that a natural/phase type N can play role type R if there is a supertype N' of N that can play R; On the other hand, we say that a natural/phase type N can play role type R if there is a supertype R' of R that can be played by N.

4.3.3 Incorporating The Metamodel Into Our Ontological Framework Based On Bunge's Ontology

In this section, we incorporate the metamodel into our ontological framework based on Bunge's ontology. In Bunge's ontology, a general intrinsic or nonbinding mutual property

is represented by an attribute function. An attribute function can be formalized in a functional form as follows:

$$A: T_1 \times \ldots \times T_n \times V_1 \times \ldots \times V_m \rightarrow V$$

where $T_i (i = 1, \ldots, n)$ represents a set of things similar in some respect and $V_j (j = 1, \ldots, m)$ as well as V represents a set of values (may be multivalued). If $n = 1$, then A represents a general intrinsic property. Otherwise, if $n \geq 2$, then A represents a general nonbinding mutual property. For example, the mutual property *Salary* held between an employee and an employer in year 2005 can be represented as $Salary: T_1 \times T_2 \times D \rightarrow R$, where T_1 is the set of employees, T_2 the set of employers, D the set of dates, and R the set of real numbers. Moreover, in Bunge's ontology, a natural kind can be modeled by a functional schema. Recall that in Chapter 3, we propose that Bunge-functional schema is modeled by UML-class/type, and it is defined as follows:

Definition 3.8: Let K be a natural kind. A *functional schema* X_K of K is a certain nonempty set M together with a finite sequence $\mathbb{F}$ of state functions on $K \times M$, each of which is supposed to represent a property of $P(K)$, a finite set $\mathbb{L}_S$ of state law statements of K on $S(K)$, and a finite set $\mathbb{L}_T$ of transformation law statements of K on $S_{\mathbb{L}}(K) \times S_{\mathbb{L}}(K)$:

$$X_K =_{df} \triangleleft M, \mathbb{F}, \mathbb{L}_S, \mathbb{L}_T \triangleright, \quad where$$

$$\mathbb{F} =< F_1, F_2, \ldots, F_n >: K \times M \rightarrow V_1 \times V_2 \times \ldots \times V_n$$

$$\mathbb{L}_S = \{L_S : S(K) \rightarrow \{lawful, unlawful\}\}$$

$$\mathbb{L}_T = \{L_T : S_{\mathrm{L}}(K) \times S_{\mathrm{L}}(K) \to \{lawful, unlawful\}\}$$

Here, domain M and codomains V_i $(1 \le i \le n)$ are unspecified. In the simplest cases, M is the set of time instances T. If X_K is the functional schema of natural kind K, then for any thing $x \in K$, $X_K(x)$ is a functional schema of x, in which each F_i $(1 \le i \le n)$ is evaluated at a fixed thing x.

Since natural types and phase types are independent $(-D)$, while role types are dependent $(+D)$, we propose that all the properties represented by functional schemata modeled by natural types and phase types are general intrinsic properties of things, and all the properties represented by functional schemata modeled by role types are semantically related general nonbinding mutual properties of things.

As a result, in the definition of the functional schema corresponding to natural type and phase type, M does not contain any component which is a set of things. For example, the $\mathbb{F}$ component of functional schema X_{Person} (which is modeled as a natural type) can be defined as:

$$\mathbb{F} = < SSN, Name, BirthDate >: P \times T \to N \times S \times D$$

where P is the set of persons, T the set of time instances, N the set of numbers, S the set of strings, and D the set of dates. It is clear that in this simple example, $M = T$, i.e., the set of time instances.

On the other hand, since functional schemata that correspond to role types represent only general nonbinding mutual properties between two or more sets of things, in its definition, M contains one (for binary nonbinding mutual properties) or more (for

higher-order nonbinding mutual properties) components which are a set of things. Note that M may also contain K as its component. In this case, a thing in K has a nonbinding mutual property with other thing(s) in K.

For example, the $\mathbb{F}$ component of functional schema $X_{Employee}$ (which is modeled as a role type) can be defined as:

$$\mathbb{F}=< Salary, StartDate, OfficePhone >: E \times E' \times T \rightarrow R \times D \times PN$$

where E is the set of employees, E' the set of employers, T the set of time instances, R the set of real numbers, D the set of dates, and PN the set of phone numbers. In this example, $M = E' \times T$ where E' is a set of things. State functions Salary, StartDate and OfficePhone represent three general nonbinding mutual properties shared by employees and employers.

Moreover, the extension of a concept such as Person is the natural kind including all persons and its intension is the functional schema representing this natural kind. Since natural type is rigid ($+R$), any instance (thing) of the functional schema corresponding to a natural type cannot cease to be so without ceasing to exist. In contrast, since phase type and role type are anti-rigid ($\sim R$), any instance (thing) of the functional schema corresponding to a phase type or a role type is the instance of the functional schema only during a certain phase of its existence. We may provide formal definitions on rigidity based on [51]:

Let W be the set of all possible worlds and let $w \in W$ be a specific world. Let K be a natural kind defined by a set of laws. K is the set of things satisfying these laws in all possible worlds. Let K_w denotes the set of things satisfying these laws in world w,

thus $K = \bigcup_{w \in W} K_w$. We say that the functional schema X_K of K is rigid if and only if K is world invariant:

Definition 4.1: A functional schema X_K of natural kind K is rigid iff, for any $w \in W$, $K_w = K$.

If functional schema X_K is rigid, then we have, for any $w \in W$, $K = K_w$. In contrast, we say that a functional schema X_K is anti-rigid if and only if it applies to its instances contingently:

Definition 4.2: A functional schema X_K of natural kind K is anti-rigid iff, for any $w \in W$ and $x \in K_w$, there exists a $w' \in W$ such that $x \notin K_{w'}$.

In Figure 4.5, all natural types are rigid and independent ($+R-D$), all phase types are anti-rigid and independent ($\sim R-D$), and all role types are anti-rigid and dependent ($\sim R+D$). These different kinds of types can be defined formally in our ontological framework based on Bunge's ontology as follows:

Definition 4.3: A functional schema X_K of natural kind K corresponds to a natural type iff it is rigid and its attribute functions represent only general intrinsic properties.

Definition 4.4: A functional schema X_K of natural kind K corresponds to a phase type iff it is anti-rigid and its attribute functions represent only general intrinsic properties.

Definition 4.5: A functional schema X_K of natural kind K corresponds to a role type iff it is anti-rigid and its attribute functions represent a set of general nonbinding mutual properties.

Moreover, for two types T and T' such that T is a specialization of T', if functional schema X_K of natural kind K corresponds to T and functional schema $X_{K'}$ of natural kind K' corresponds to T', then for all $w \in W$, $K_w \subseteq K'_w$.

Definition 4.6: A functional schema X_K of natural kind K is a sub-functional schema of functional schema $X_{K'}$ of natural kind K' iff for all $w \in W$, $K_w \subseteq K'_w$.

4.3.4 Implications of The Ontological Metamodel For Object-Oriented and Conceptual Modeling

4.3.4.1 Representing Intrinsic and Mutual Property in UML Class Diagrams

In Bunge's ontology, a value property of a thing is either an intrinsic property, or a nonbinding mutual property. Usually intrinsic properties are relatively stable properties of things over their lifetime. In contrast, nonbinding mutual properties are less stable

properties in that they can be acquired/dropped by things entering/leaving a particular context. In [26] and [25], Evermann proposes to use attributes of association classes to represent mutual properties. Following his idea, since an individual intrinsic property is owned by a thing exclusively, in a UML class diagram, Bunge-attribute functions representing general intrinsic properties are modeled as attributes of an *ordinary* class/type placed in the attribute compartment of the class/type. That is to say, the attribute compartment of an ordinary class/type in a UML class diagram contains a list of attributes each of which models a Bunge-attribute function representing a general intrinsic property of the things modeled by the functional schemata corresponding to this class/type. We may call these UML attributes *intrinsic attributes*.

On the other hand, by definition, an individual nonbinding mutual property is shared by multiple things, say, for example, A and B. I.e., neither A nor B owns this individual nonbinding mutual property exclusively. As a result, in a UML class diagram, Bunge-attribute functions representing general nonbinding mutual properties shared between things are modeled as attributes placed in the attribute compartment of an association class connecting classes/types whose instances model these things. That is to say, the attribute compartment of an association class between two or more classes/types in a UML class diagram contains a list of attributes each of which models a Bunge-attribute function representing a general nonbinding mutual property shared by things modeled by the functional schemata corresponding to these interconnecting classes/types. We may call these UML attributes *mutual attributes*.

4.3.4.2 Representing Natural, Phase, and Role Types in UML Class Diagrams

Furthermore, we already know that natural/phase type has only intrinsic attributes owned by itself exclusively whereas role type has only mutual attributes shared with other role type(s) (Definitions 4.3, 4.4, and 4.5). Thus the attribute compartment of a role type in a UML class diagram should be empty. All the (mutual) attributes of the role type are placed in the association class of the association connecting this role type to other role type(s) with which it shares these attributes. Consequently, in a UML conceptual model, a role type cannot occur without being related to other role type(s). Actually this rule conforms to the observation in the literature that *roles imply patterns of relationships*, i.e., the existence of roles depends on additional external entity(-ies) [39][61].

An example UML diagram illustrating natural types and their role types is shown in Figure 4.6 (rectangle for natural/phase type and oval for role type). In this figure, Person, Company, and Team are natural types, Employee, Employer, TeamLeader, and SupervisedTeam are role types. A team leader which is a person is also an employee working for an employer which is a company, thus in addition to his/her own mutual attribute (Goal of association class Supervision), he/she also inherits mutual attributes Salary, StartDate and OfficePhone of association class Employment from Employee.

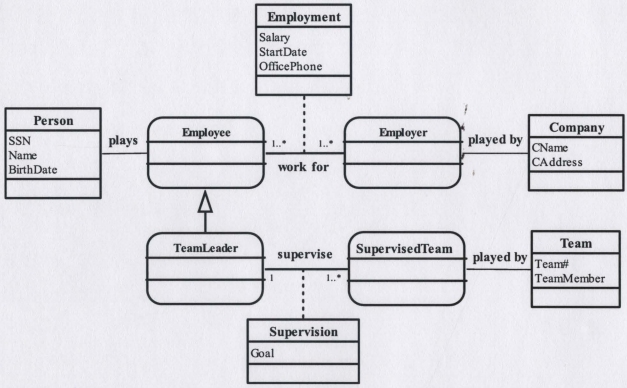

Fig. 4.6. An UML diagram illustrating natural types and their role types

Note that as argued by Wand, Storey, and Weber in [16], in our approach, it is not possible for a role type to have optional associations with other role type(s). This is because for an instance of a natural/phase type to be an instance of a role type, it must have some mutual attributes shared with other entities. For example, a person cannot be an employee without working for an employer. Moreover, an optional association between two role types means that a equivalent conceptual model exists that may better express the same real world semantics. For example, an optional association Manage between role type Employee and itself could be better expressed by a mandatory association Manage between role types Manager and ManagedEmployee.

In Figure 4.6, let P be the set of persons, C the set of companies, T the set of time instances, T' the set of teams, E the set of employees, E' the set of employers, TL the set of team leaders, ST the set of supervised teams, R the set of real numbers, D the set of dates, I the set of integers, and S the set of strings. In our ontological framework, the $\mathbb{F}$ components of functional schemata corresponding to natural types Person, Company, and Team can be defined as:

$$\mathbb{F}_{Person} =< SSN, Name, BirthDate >: P \times T \rightarrow I \times S \times D$$

$$\mathbb{F}_{Company} =< CName, CAddress >: C \times T \rightarrow S \times S$$

$$\mathbb{F}_{Team} =< Team\#, TeamMember >: T' \times T \rightarrow I \times 2^{P}$$

In addition, the $\mathbb{F}$ components of functional schemata corresponding to role types Employee, Employer, TeamLeader and SupervisedTeam can be defined as:

$$\mathbb{F}_{Employee} =< Salary, StartDate, OfficePhone >: E \times E' \times T \rightarrow R \times D \times S$$

$$\mathbb{F}_{Employer} =< Salary, StartDate, OfficePhone >: E' \times E \times T \rightarrow R \times D \times S$$

$$\mathbb{F}_{TeamLeader} =< Goal >: TL \times ST \times T \rightarrow S$$

$$\mathbb{F}_{SupervisedTeam} =< Goal >: ST \times TL \times T \rightarrow S$$

Note that as indicated in definition 3.8, in the $\mathbb{F}$ component of a functional schema, the order of sets in the Cartesian product is important. For example, in $\mathbb{F}_{TeamLeader}$ above, the first set of the Cartesian product is the set of team leaders TL whereas in $\mathbb{F}_{SupervisedTeam}$, the first set is the set of supervised teams ST. A team leader may supervise

one to many supervised teams whereas a supervised team may only be supervised by a team leader.

In addition to natural types and role types, phase types are also very useful in object-oriented and conceptual modeling. Examples of phase types include Child, Teenager, and Adult of Person, Town and Metropolis of City, ActiveCar and WreckedCar of Car, Caterpillar and Butterfly of Lepidopteron. In a phase type partition of a natural type or phase type, the subtypes should be constructed such that they are mutually disjoint and constitute a *total* partition of this supertype, i.e., an instance of the supertype should have a corresponding instance in exactly one phase subtype. Figure 4.7 shows a conceptual model including a phase type partition Child, Teenager, and Adult of natural type Person. It indicates that only adult persons can be employees.

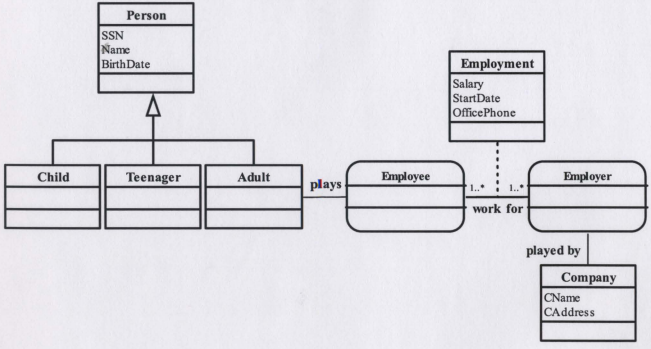

Fig. 4.7. A conceptual model with a phase type partition

Note that, in UML, an is-a partition of a classifier is a *generalization set* which "defines a particular set of Generalization relationships that describe the way in which a general Classifier (or superclass) may be divided using specific subtypes" [72, p. 71]. It is represented by connecting all subclasses to one single hollow arrowhead symbol. A generalization set constitutes a partition of the class pointed to by the symbol.

For convenience, since the attribute compartment of a role type is empty, whenever the intended meaning is clear (especially when the role type does not have role subtypes or role supertypes) it is more convenient to represent role types as named places in relationships. For example, in Figure 4.8, role types Employee and Employer are represented as named places and phase type Adult is related to natural type Company directly.

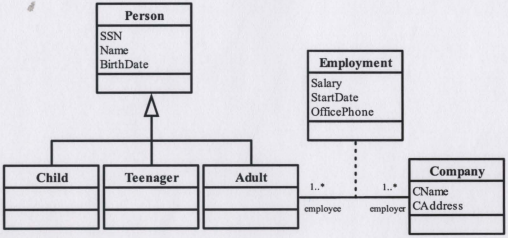

Fig. 4.8. The simplified conceptual model of Figure 4.7

4.3.4.3 The Counting Problem

Compared with our approach discussed above, as far as we know, none of the existing conceptual modeling languages in the literature differentiates intrinsic attribute from mutual attribute and furthermore realizes that all the attributes of a role type should be mutual attributes which are shared by all entities participating in the relationship. Consequently in their approaches, intrinsic attributes which actually are attributes of natural/phase type can occur in a role type, and moreover a role type does not have to be related to other role type(s). This practice may cause *unstable* conceptual models in the situations that a natural/phase type instance can play two or more roles of the same role type simultaneously, i.e., the counting problem.

For example, in Entity-Relationship approach (ER), there is only entity type, no explicit role type. Role types are represented as named places in relationships. As a result, all intrinsic and mutual attributes are placed by modelers in entity types or relationship types arbitrarily. Figure 4.9 illustrates an ER conceptual model adapted from [69].

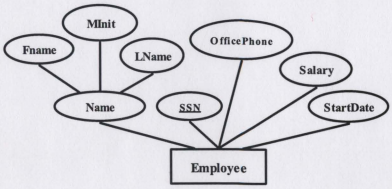

Fig. 4.9. An ER conceptual model extracted from [69]

In Figure 4.9, the entity type (Employee) is role type in our model. However, for entity type Employee, unlike attributes OfficePhone, Salary and StartDate which model Bunge-attribute functions representing general nonbinding mutual properties between, say, employees and employers, the attributes Name and SSN model Bunge-attribute functions representing actually general intrinsic properties of persons that play role type Employee. These intrinsic attributes are still valid even after a person ceases to be an employee thus loses all mutual attributes valid only in the employment relationship. Therefore instead of being placed in role type Employee, they should be placed in an additional natural type Person. Figure 4.10 shows the corresponding model segment using our metamodel.

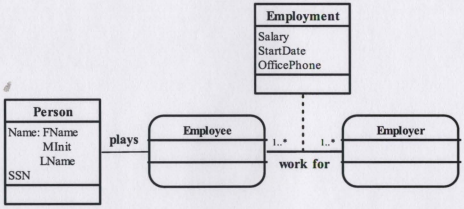

Fig. 4.10. The corresponding model of Figure 4.9 using our metamodel

In the situations that a person cannot have more than one job at the same time (e.g., a person cannot be a secretary and a technician simultaneously), when mapping conceptual models in Figure 4.9 and Figure 4.10 to a relational database schema, we get one relation Employee and two relations Person and Employment respectively. It is

usually not necessary to map also Employee and Employer in Figure 4.10 to two relations in a relational database schema because they have no intrinsic attributes.

However, in the situations that, for example, a person can be a secretary and technician simultaneously with different office phones, salaries, and start dates, the corresponding relational database schema of the conceptual model in Figure 4.9 must be modified accordingly. Now attributes OfficePhone, Salary, and StartDate of Employee are multivalued and when mapping this conceptual model to a relational database schema, we need to create a new relation R which includes the three attributes OfficePhone, Salary and StartDate, plus the primary key SSN of Employee as a foreign key in R. The primary key of R is the combination of all four attributes. Thus, unlike in the previous situations, the resulting relational database schema has two relations. Like Person relation in the corresponding relational database schema of Figure 4.10, now the Employee relation includes only attributes Name and SSN. It is thus clear that using ER approach, it is possible that the resulting relational database schema has to evolve after it has already been in existence for some time.

In contrast, for the conceptual model in Figure 4.10, when a person can be a secretary and a technician at the same time, this fact has no impact on the resulting relational database schema at all. In this case, one may simply insert into Employment relation two different records of a person with different employers in order to differentiate different jobs held by the same person. Thus the relational database schema resulting from the conceptual model in Figure 4.10 is more stable with respect to requirements change and thus more suited to capture dynamic and evolutionary aspects of real world

applications. More details on how to map various conceptual models based on our metamodel to relational database schemata will be discussed in Chapter 5.

Our approach described here bears some resemblance to the Object Role Modeling (ORM) approach proposed by Halpin in [70]. Unlike ER modeling (and our approach), ORM does not use attributes. As argued by Halpin [70], "The first problem with using attributes in the initial models is that they are often unstable. … So do not agonize over whether to model a particular feature as an attribute or relationship. Just model it as a relationship". An example he uses to illustrate this point is the attributes "country-Represented" and "birthplace" of entity type Athlete, both of which are based on the domain Country. He then argues that later on when we add the fact that Country has population, we would have to model Country as an entity type. However, different from our approach that has an ontological foundation, ORM is mainly focused on implementation considerations. As a result, it does not distinguish intrinsic attributes from mutual ones. In fact, in our approach, intrinsic attributes are owned by a class/type exclusively, thus should not be modeled as relationships.

4.3.4.4 Object Migration

Since a natural type is rigid, its instances cannot migrate to other natural types in a natural type hierarchy. For phase types however, since they are anti-rigid, their instances may migrate to other phase types in the partition of a phase/natural type hierarchy during their lifetime when the distinguishing intrinsic attribute(-s) is changed. An instance of a natural/phase type may become an instance of a role type when it participates in a

relationship thus acquires some mutual attribute(-s) (It still remains an instance of the natural/phase type). It may further acquire more mutual attributes thus become an instance of a sub role type or a new role type, or it may lose some mutual attributes thus migrate to a super role type or not be an instance of a role type any more.

For example, in Figure 4.11, Person, Man, and Woman are natural types; Child, Teenager, and Adult are phase types. Usually an instance of Man cannot migrate to Woman in his lifetime. But an instance of Child may possibly migrate to Teenager and Adult at some point in time. For role types, in Figure 4.6, a person may acquire mutual properties Salary, StartDate and OfficePhone shared with an employer thus become an employee. An employee may acquire mutual property Goal shared with a supervised team thus become a team leader.

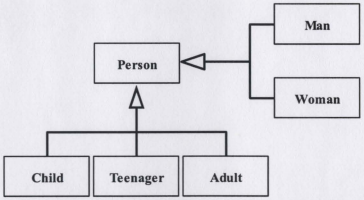

Fig. 4.11. An example of natural type and phase type partitions

Different from our approach presented here, in [53], Marcos and Cavero propose a classification of taxonomic hierarchies with different constraints on object migration for conceptual modeling based on Aristotle's definition of *essence* vs. *accident* and *change*

vs. *motion*. They identify three taxonomic hierarchies as constructors for conceptual modeling: IS-A (which is similar to natural type hierarchy), ROLE (which is similar to role type hierarchy), and STATE (which is similar to phase type hierarchy).

Similarly, Wieringa et al. propose to use dynamic classes and role classes as two ways to model object migration between classes [58]. In [58], a static partition is a partition whose subclasses are disjoint and their instances cannot migrate between each other (which is similar to natural type partition). A dynamic partition is a partition whose subclasses are disjoint at any particular point in time but their instances can migrate between each other (which is similar to phase type partition). A static class is an element of a static partition or an intersection of static classes; a dynamic class is an element of a dynamic partition of a (dynamic or static) class. Each intersection of a dynamic class with another (static or dynamic) class is also a dynamic class. In a static partition, classes are not necessarily rigid. For example, a partition with Boy and Girl as subclasses and Child as superclass is static, but classes Child, Boy and Girl are not rigid.

The problem with [53] and [58] is that, when discussing object migration, without paying attention to the ontological nature of different "kinds" of types (natural type, phase type, and role type), the authors focus on different types of class partitions. Consequently, their argument "we cannot have a static partition of a dynamic class" [58, p. 66] is not guaranteed, nor is the argument "So if we descend in the taxonomic structure, once we meet a dynamic partition, we will not meet a static partition anymore" [58, p. 66]. Consider Figure 4.12. Here, Child, Teenager, and Adult constitute a dynamic partition of Person. Therefore they are dynamic classes. But Boy and Girl constitute a static partition of Child.

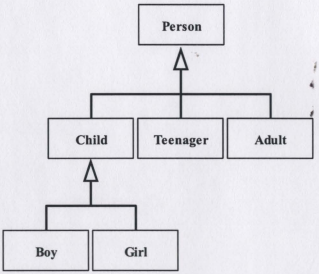

Fig. 4.12. A static partition of a dynamic class

In a phase type partition of a natural/phase type, object migration between these phase types is subject to some dynamic integrity constraints. For example, in the partition Child, Teenager, and Adult of Person, a person who is a child may migrate from Child to Teenager and further from Teenager to Adult during his/her lifetime, but the reverse direction of migration is forbidden (An adult cannot migrate to Teenager, and a teenager cannot migrate to Child). In practice, a migration diagram [58] can be used to describe the way in which instances of a natural/phase type can move through its phase type partition. An example is shown in Figure 4.13.

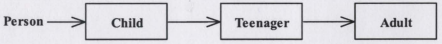

Fig. 4.13. The migration diagram of a phase type partition of Person

Similarly, in some role type partitions of role types (such as UndergraduateStudent and GraduateStudent of Student), instances of a role subtype may migrate to other role subtypes subject to some constraints (e.g., an undergraduate student can migrate to GraduateStudent, but not vice versa). In this case, migration diagram can also be employed to describe these constraints.

4.3.5 Discussion on Steimann's Features of Roles

In this section, we reconsider those features ascribed to roles by Steimann as discussed in section 4.2.5.1.

1. *A role comes with its own properties and behavior.* Indeed, in our model, roles are types, and they recruit their instances from natural/phase types playing them. However, in contrast with natural/phase types which have only intrinsic attributes owned by themselves exclusively, role types have only mutual attributes shared with other role type(s). Moreover, as discussed in section 3.3.5, an association class cannot have operations or methods. Instead, operations and methods that change mutual attribute values of an association class must be placed in participating role classes/types of the association.
2. *Roles depend on relationships.* Yes. In our model, since role types have only mutual attributes, they cannot occur without being related by their mutual attributes to other role type(s). Moreover, since a relationship type is actually a tuple of mutual attributes, we may also say that relationships depend on roles.

In this sense, Entity-Relationship (ER) modeling is actually Entity-Role modeling.

3. *An object may play different roles simultaneously*. Yes. As indicated in Figure 4.5, a natural/phase type can play zero to many role types simultaneously. Similarly, an instance of a natural/phase type can play different roles of different role types simultaneously.
4. *An object may play the same role several times, simultaneously*. Yes. In our model, an instance of a natural/phase type can play the same role type or participate in the same relationship type several times simultaneously. Each participation can be uniquely identified by a combination of the identities of the instance and the other instances it is related to in that particular participation.
5. *An object may acquire and abandon roles dynamically*. Yes. An instance of a natural/phase type may acquire some mutual attribute(-s) thus a role by participating in some relationship. At the same time, it remains an instance of the natural/phase type. It may later on cease to be involved in the relationship thus lose those mutual attribute(-s) and abandon the role.
6. *Objects of unrelated types can play the same role*. Yes. As indicated in Figure 4.5, a role type can be played by zero to many natural/phase types simultaneously. Therefore, instances of unrelated natural/phase types can play the same role type.
7. *Roles can play roles*. Yes.
8. *Different roles may share structure and behavior*. Yes. As indicated in Figure 4.5 and exemplified in Figure 4.6, different role types can be organized in

generalization/specialization hierarchies to enable sharing of structure and behavior.

9. *An object and its roles share identity.* Yes. In our model, an instance of a natural/phase type and its roles are the same object corresponding to the same real world thing, thus have the same identity.
10. *An object and its roles have different identities.* No. This view is contradictory to feature 9 above. In fact, since an object and its roles model the same thing in the real world, they should not have different identity.

4.4 Conclusion

In this chapter, we propose an ontological metamodel of classifiers based on OntoClean methodology and Guizzardi et al.'s ontological profile. Then, after incorporating this metamodel into our ontological framework built on Bunge's ontology, we explore its implications for object-oriented and conceptual modeling as well as information system design and implementation. In particular, we focus on discussing the definition, properties, and representation of the notion of roles in the literature. Indeed, because most role modeling approaches treat attributes of a role type as intrinsic attributes owned by the role type exclusively, therefore they cannot handle satisfactorily the counting problem and the related role identity problem. In contrast, our approach is founded on Bunge's ontology and conforms to the fundamental features identified in the literature, thus provides an ontological semantics for the notion of roles suitable for conceptual modeling. In the next

chapter, we further demonstrate the usefulness of our metamodel by applying it to conceptual database modeling.

Chapter 5

Application: Conceptual Database Modeling

5.1 Introduction

Conceptual modeling is an important phase in designing a successful database application. The Entity-Relationship (ER) model [73][74] is a popular high-level conceptual data model. It was developed by Peter Chen in 1976 and has undergone substantial modification and refinement through the years. Frequently, this model and its variations are used for the conceptual design of a database application. In this chapter, we elaborate on our model proposed in Chapter 4, comparing it with ER model with respect to conceptual database modeling in order to demonstrate conceptual and practical usefulness of our model. Then we describe the process of mapping a conceptual database model based on our metamodel into a relational database schema. Note that, since no uniformly

accepted set of notation exists for the elements of the ER model, we will use the notation employed by Elmasri and Navathe in [69].

5.2 Conceptual Database Modeling Using Our Proposed Metamodel

The ER model describes data in terms of entities, relationships, and attributes. The basic object represented is an entity, which represents a "thing" in the real world. A relationship captures how two or more entities are related to one another. An entity type defines a collection of entities that have the same attributes. Each entity type has attributes, which represent particular properties of the entities of this entity type. Each entity has its own value(s) for each attribute. Similarly, a relationship type among n entity types defines a collection of relationships among entities from these entity types. A relationship type can have attribute(s), each relationship of the relationship type has its own value(s) for each attribute.

Moreover, in ER, role is a named place in a relationship. Role names are assigned to entity types participating in relationships. Role names are useful when the same entity type participates in a relationship type more than once in different roles. In these cases, role names are essential for distinguishing the meaning of each participation [69]. However, as we argued in Chapter 4, viewing roles as named places fails to acknowledge the fact that roles come with their own properties and behavior, and this weakness can be resolved by regarding roles as types in their own right. Moreover, we argue that since

roles are mere labels of types, it is thus impossible to construct role type generalization/specialization hierarchies leading to better-organized conceptual models.

Furthermore, different from our model, ER does not differentiate natural types, phase types, and role types. Thus it cannot be used to represent different "kinds" of types with different ontological nature as well as their relationships.

In Chapter 4, we proposed that natural/phase type has only intrinsic attributes owned by itself exclusively whereas role type has only mutual attributes shared with other role type(s). All the (mutual) attributes of the role type are placed in the association class of the association connecting this role type to other role type(s), thus in a UML conceptual model, a role type cannot occur without being related to other role type(s). In this section, we compare our model with ER diagrams with respect to conceptual database modeling in order to demonstrate the advantages and usefulness of our model.

5.2.1 The Counting Problem and Multivalued Attribute

In section 4.3.4.3, we described the *Counting Problem*, namely the situations in which instances counted in their roles yield a greater number than the same instances counted by the objects or phases playing the roles. We then demonstrated using an example that since there is no explicit role type in ER and all intrinsic and mutual attributes are placed by modelers in entity types or relationship types arbitrarily, the ER approach has a serious flaw in dealing with dynamic and evolutionary aspects of real world applications, namely, relational database schemata resulted from ER models may have to be modified after having been put into use for some time. For a detailed example, refer to section 4.3.4.3.

In conceptual database modeling, a multivalued attribute in ER is an attribute that can have a set of values for the same entity. For example, a CollegeDegrees attribute of a person is a multivalued attribute. A person may have zero, one, two, or more degrees issued by the same or different college(s). Thus instead of modeling a Bunge-attribute function representing an intrinsic property of a person, CollegeDegrees models a Bunge-attribute function representing a general nonbinding mutual property. In fact, intuitively, in order to differentiate each value of a multivalued attribute, an external entity is required (we may have as many external entities as its values). Therefore we suggest that multivalued attributes in an ER schema always reflect mutual properties in our model. As a result, in our model, a natural/phase type cannot have a multivalued attribute as its component. Contrarily, any attribute of a role type may be a multivalued attribute.

5.2.2 A More Complicated Example

In Chapter 4, we gave several example conceptual models using our metamodel. In this section, we present a more complicated example with multiple inheritance. Figure 5.1 is an ER diagram with a generalization/specialization lattice adapted from [69, p. 84] for a university database. Attributes SSN, Name, Sex, Address, and BirthDate are intrinsic attributes of natural type Person. Attributes Salary, MajorDept, and PercentTime are mutual attributes of role types Employee, Student, and StudentAssistant respectively. As discussed in sections 4.3.4.3 and 5.2.1, these mutual attributes cannot be owned exclusively by the role types. A corresponding conceptual model of Figure 5.1 based on our metamodel is illustrated in Figure 5.2.

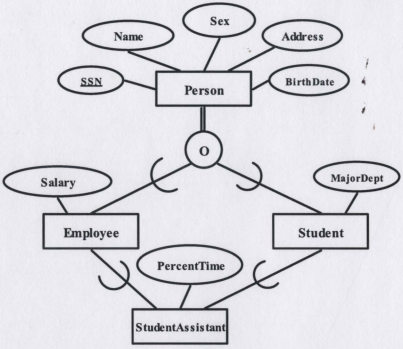

Fig. 5.1. A generalization/specialization lattice for a university database

In Figure 5.2, Person and University are natural types (We add to University a key attribute Name). Student, Employee, StudentAssistant, Employer, StudentEmployer, and UniversityEnrolled are all role types. Mutual attribute Salary is shared by Employee (which is played by Person) and Employer (which is played by University). Similarly, mutual attribute MajorDept is shared by Student (which is played by Person) and UniversityEnrolled (which is played by University). Also, mutual attribute PercentTime is shared by StudentAssistant (which is a subtype of role types Employee and Student) and StudentEmployer (which is a subtype of role types Employer and UniversityEnrolled).

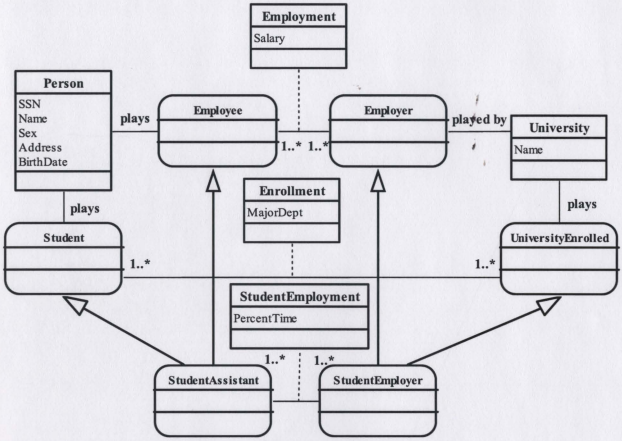

Fig. 5.2. The corresponding model of Figure 5.1 using our metamodel

It is clear that the conceptual model in Figure 5.2 expresses more real world semantics than Figure 5.1. In fact, these real world semantics are implicit in Figure 5.1 and exist only in modelers' mind. But in Figure 5.2, they are represented explicitly. We deem it a positive aspect of our approach because, as argued by Evermann and Wand [25, p 147], one of the detrimental effects of the lack of rich, formal languages specific to conceptual modeling is that information system "development projects might begin without explicitly modelling the application domain and instead must rely on implicit assumptions of developers". Moreover, in Figure 5.2, the identifier attribute *Name* of the newly created natural type University is important because it will be used later as part of a

combined key attribute to identify each relationship instance of relationship types Employment, Enrollment, and StudentEmployment associated to the role types played by University.

5.2.3 Union Type (Category) vs. Role Type

In conceptual database modeling, a *union* type or *category* is a subclass (in a single superclass/subclass relationship with more than one superclass, each of which represents a different entity type) which represents a collection of objects that is (a subset of) the union of distinct entity types [69]. For example, Figure 5.3 (adapted from [69, p. 86]) illustrates two union types Owner and Registered_Vehicle. An owner may own a number of registered vehicles, and a registered vehicle may be owned by a number of owners. An owner of a vehicle can be a person, a bank, or a company. Similarly, a registered vehicle can be a car or a truck.

In UML, a subclass may have multiple superclasses. The extension of the subclass is the intersection of the extensions of all superclasses. I.e., each instance of the subclass is also an instance of every superclass. For example, an engineering manager in the subclass Engineering_Manager is also a member of superclasses Engineer and Manager. On the other hand, the extension of a category is (a subset of) the union of the extensions of all superclasses. Each instance of the category must belong to *only one* superclass.

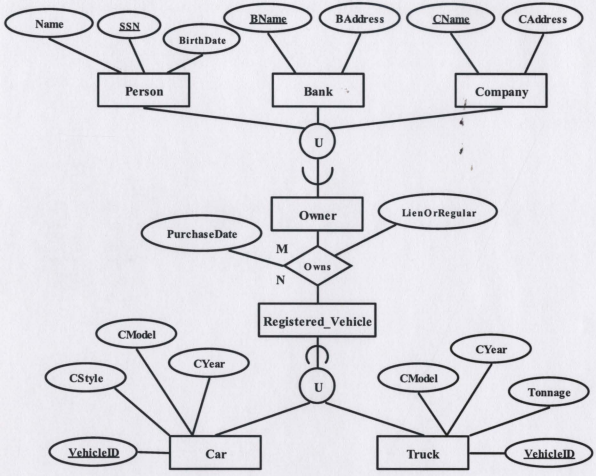

Fig. 5.3. Two union types: Owner and Registered_Vehicle

However, we suggest that, instead of being a new modeling concept, a union type or category is actually a role type in our metamodel (which is anti-rigid and dependent). To be more precise, it is formal role in OntoClean or rolemixin in Guizzardi et al.'s ontological profile. Consequently, we may model Owner and Registered_Vehicle as role types in our metamodel. Figure 5.4 illustrates the corresponding model based on our metamodel. In Figure 5.4, Person, Bank, and Company are natural types that can play role type Owner. Similarly, Car and Truck are natural types that can play role type Registered_Vehicle. Since a role type recruits its instances from natural/phase types, an

owner can be a person, bank, or company. Similarly, a registered vehicle can be a car or a truck. All attributes of the relationship *Owns* (PurchaseDate and LienOrRegular) are placed in association class Ownership.

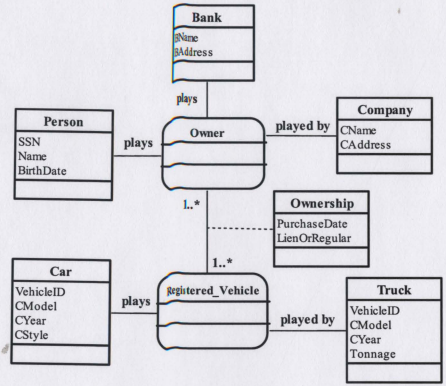

Fig. 5.4. The corresponding model of Figure 5.3 using our metamodel

Also note that, different from Figure 5.3, in Figure 5.4, the cardinality constraint of Owner and Registered_Vehicle in relationship *Owns* is 1..*. This is because, as we discussed in section 4.3.4.2, a role type cannot have optional associations with other role type(s).

5.2.4 Role Identity

The notion of object identity is a crucial issue in many areas of computer science, including programming languages, database systems, computer networks, and operating systems. In [29] and [58], Wieringa and his colleague compare object identity with role identity. In their model, a role type has its own instances (roles) with separate global unique identity different from other roles as well as instances of natural types (objects) and phase types (phases). This view is motivated to model some real world situations such as *a person plays exactly three employee roles simultaneously, with different salary and office number*, i.e., the counting problem described in sections 4.2.5.1 and 5.2.1. In this case, each employee role instance of a person object is a quasi-object in that it describes a state of the corresponding object playing the role in a particular context. This view is also shared by [59][57] and [64].

Although it looks appealing, this approach is problematic because a conceptual model should be a faithful representation of someone's perception of a real world domain. According to Shanks, Tansley, and Weber [27], a conceptual model is faithful if it is accurate, complete, conflict free, and non-redundant. Thus an object with a unique identity in a conceptual model should correspond to a distinct thing in the real world. Furthermore, the requirement that each role instance has a global unique identity different from other roles as well as instances of natural types and phase types is quite artificial.

In ER, there is no role type. Entity types that are actually role types (e.g., Employee and Student) usually inherit object identity from their superclasses. For example, in Figure 5.1, Employee, Student, and StudentAssistant inherit key attribute SSN from Person. This practice is not feasible when, for example, a person can be student in two different universities. Here, a Person entity corresponds to two Student entities,

therefore a single key attribute SSN of the person is not adequate to distinguish these two studentships.

As Steimann's role model in [54][55][56], in our model, instances of role types are directly recruited from natural/phase types. Moreover, we suggest that, role instances in [29][58][59][57][64] are actually relationship instances, or instances of association classes that do not have counterpart in the real world. These relationship instances can be naturally distinguished from each other as well as from instances of natural types, phase types, and role types by a combination of the identities of participating objects. For example, in Figure 5.2, natural type Person plays role type Student, natural type University plays role type UniversityEnrolled. An instance of Enrollment of a student in an enrolled university can be uniquely identified by the combination of key attributes SSN of the person and Name of the university. Similarly, an instance of Employment of an employee by an employer can be uniquely identified also by the combination of key attributes SSN of the person and Name of the university.

Note that the situation becomes more complicated when a role type can be played by more than one natural/phase types. If these natural/phase types have the same key attribute, role instances can inherit this key attribute directly. For example, in Figure 5.4, natural types Car and Truck have the same key attribute VehicleID. Therefore instances of role type Registered_Vehicle can inherit this key attribute directly. If however the natural/phase types playing the role type have different key attributes, we cannot use any of them exclusively to identify all instances of the role type. For convenience, a new key attribute may be specified for the role type. For example, also in Figure 5.4, natural types Person, Bank, and Company have different key attributes SSN, BName, and CName.

Therefore, a new key attribute OwnerID can be used to identify instances of role type Owner. Consequently, instances of relationship type Ownership can be uniquely identified by the combination of key attributes OwnerID and VehicleID.

5.2.5 Integrity Constraints

In ER, relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. Among them, the *participation constraint* "specifies whether the existence of an entity depends on its being related to another entity via the relationship type" [69, p. 57]. There are two types of participation constraints – total and partial. An example for total participation constraint is *every employee must work for an employer*. An example for partial participation constraint is *not every employee manages a department*. Total participation is also called *existence dependency*. However, as indicated before, in our model, a role type cannot have optional associations (or partial participation) with other role type(s). In fact, in the example for partial participation constraint, instead of role type Employee, it is role type Manager that has a management relationship with role type ManagedDepartment. In this case, the relationship type management is total participation for both Manager and ManagedDepartment.

In a phase type partition of a natural or phase type, the phase subtypes should be constructed such that they are mutually *disjoint* and constitute a *total* partition of this supertype, i.e., any instance of the supertype should have a corresponding instance in exactly one phase subtype. For example, in Figure 5.5, phase subtypes Child, Teenager,

and Adult are mutually disjoint and constitute a total partition of natural type Person. The disjointness integrity constraint can be implemented in SQL2 using three CREATE ASSERTION statements as illustrated below (suppose relations Person, Child, Teenager, and Adult have the same key attribute SSN):

```
CREATE ASSERTION PHASETYPEPARTITION_CONSTRAINT1
CHECK( NOT EXISTS (SELECT * FROM CHILD C, TEENAGER T
                   WHERE C.SSN=T.SSN));

CREATE ASSERTION PHASETYPEPARTITION_CONSTRAINT2
CHECK( NOT EXISTS (SELECT * FROM TEENAGER T, ADULT A
                   WHERE T.SSN=A.SSN));

CREATE ASSERTION PHASETYPEPARTITION_CONSTRAINT3
CHECK( NOT EXISTS (SELECT * FROM CHILD C, ADULT A
                   WHERE C.SSN=A.SSN));
```

Because a phase type partition is a total partition, in a relational database schema, if we insert an entity into a relation that represents the supertype of a phase type partition, this entity must also be inserted into exactly one relation that represents an appropriate phase subtype. For example, in Figure 5.5, if we insert a person entity that is a child into relation Person, this entity is mandatorily inserted into relation Child. On the other hand, if we delete an entity from *a set of* relations that represent a phase type partition (i.e., this entity does not belong to any phase subtype of the partition any more), this entity must also be deleted from the relation representing the supertype of the phase type partition. In Figure 5.5, if an adult becomes deceased thus deleted from relation Adult, he or she

cannot belong to any phase subtype of partition Person any more, thus is mandatorily deleted from relation Person. We may use CREATE TRIGGER statement to implement this integrity constraint as illustrated below:

```
CREATE TRIGGER DELETEPHASE
AFTER DELETE ON ADULT
FOR EACH ROW
    DELETE * FROM PERSON WHERE SSN=OLD.SSN;
```

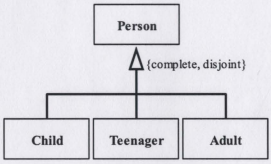

Fig. 5.5. An example of a phase type partition of a natural type Person

Moreover, it is argued by Wieringa et al. that to construct taxonomic structures, three classification principles should be followed [58, p. 69]. Among them, the first principle states that, for each is-a partition of a class, the classification principle that governs the division in its subtypes should be *clear*, *unambiguous*, *singular*, and *uniform* for all subtypes. An example of non-uniform classification principle is "a division of animals into domestic animals, poisonous snakes and others". Here, one subtype is defined along the dimension domestic-wild animals and another along the dimension poisonous-nonpoisonous snake. Following the above principles, in natural type hierarchies, partitions should be constructed such that its subtypes are mutually disjoint.

For example, in Figure 5.6, natural subtypes Car and Truck are mutually disjoint, so are natural subtypes Man and Woman. The corresponding assertion implementation for Figure 5.6 (b) can be illustrated below (suppose relations Person, Man, and Woman have the same key attribute SSN):

```
CREATE ASSERTION NATURALTYPEPARTITION_CONSTRAINT
CHECK( NOT EXISTS (SELECT * FROM MAN M, WOMAN W
                   WHERE M.SSN=W.SSN));
```

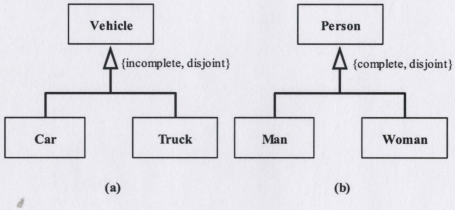

Fig. 5.6. An example of (a) a *partial* natural type partition, and (b) a *total* natural type partition

Note that a natural type partition does not necessarily exhaust its supertype. Figure 5.6 (a) illustrates a *partial* natural type partition Truck and Car of Vehicle, Figure 5.6 (b) illustrates a *total* natural type partition Man and Woman of Person. Consequently, in a relational database schema, if we insert an entity into a relation that represents the supertype of a natural type partition, this entity does not necessarily have to be inserted into a relation that represents a natural subtype. For example, in Figure 5.6 (a), if we

insert an entity that is a motorcycle into relation Vehicle, this entity does not have to be inserted into either Car or Truck.

On the other hand, if we delete an entity from *any* relation that represents a subtype of a natural type partition, this entity must also be deleted from the relation representing the supertype of the partition. For example, in Figure 5.6 (b), if a man/woman entity is deleted from relation Man/Woman, he/she is mandatorily deleted from relation Person. The corresponding trigger implementation for Figure 5.6 (b) can be illustrated below:

```
CREATE TRIGGER DELETEOBJECT1
AFTER DELETE ON MAN
FOR EACH ROW
    DELETE * FROM PERSON WHERE SSN=OLD.SSN;


CREATE TRIGGER DELETEOBJECT2
AFTER DELETE ON WOMAN
FOR EACH ROW
    DELETE * FROM PERSON WHERE SSN=OLD.SSN;
```

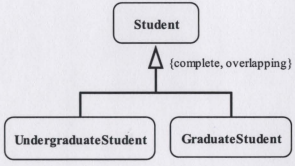

Fig. 5.7. An example of a total role partition of role type Student

Compared with natural/phase type partition, usually a role type partition can be *overlapping*. This is because an object or phase may play multiple roles simultaneously. For example, in Figure 5.7, if a person can be an undergraduate student in one university and a graduate student in another, then this partition is overlapping. Furthermore, a role type partition can be *partial* or *total*. Figure 5.7 illustrates a total partition: every student is either an undergraduate student or a graduate student.

Moreover, as discussed in section 4.3.4.4, instances of a natural type cannot migrate to other natural types in a natural type hierarchy. However, this is not the case for phase and role types. In a phase type partition of a natural/phase type, object migration between these phase types is subject to some dynamic integrity constraints. A migration diagram can be used to describe the way in which instances of a natural/phase supertype can move through its phase type partition. For example, Figure 5.8 illustrates a phase type partition Caterpillar and Butterfly of Lepidopteron along with its migration diagram. A caterpillar can migrate to Butterfly, but not vice versa.

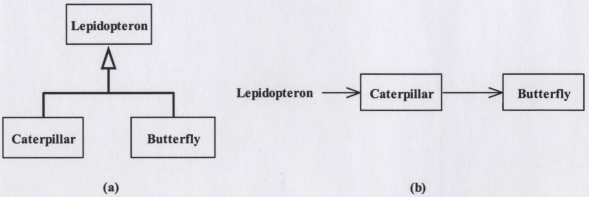

Fig. 5.8. A phase type partition and its corresponding migration diagram

Similarly, migration diagrams can also be employed to describe dynamic integrity constraints of role type partitions of role types. Figure 5.9 shows the corresponding migration diagram of Figure 5.7.

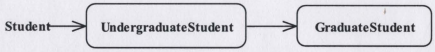

Fig. 5.9. The corresponding migration diagram of Figure 5.8

5.2.6 Converting An ER Model Into A Conceptual Model Based On Our Metamodel

Given an ER model, we may convert it into a conceptual model based on our metamodel by following the steps described below. Note that we do not consider weak entity types in this chapter.

Step 1: For each entity type in the ER model, decide whether it is a natural/phase type or a role type, and then use appropriate notation (rectangle for natural/phase type and oval for role type) to represent it. Typical natural types are Person, University, and Department, etc. Typical phase types are Man, Woman, and Adult, etc. Typical role types are Employee, Student, and Customer, etc. For example, in Figure 5.1, entity type Person is natural type, entity types Employee, Student, and StudentAssistant are role types. Note that, all union types in ER models are converted into role types in our models.

Also note that, in our model, natural/phase types contain attributes modeling Bunge-attribute functions representing only general intrinsic properties of things whereas role types contain attributes modeling Bunge-attribute functions representing only general nonbinding mutual properties shared by things. However, in some ER models, intrinsic and mutual attributes are placed by modelers in entity types arbitrarily. For example, in Figure 4.9 (section 4.3.4.3), attributes Name and SSN are intrinsic attributes of Person, but they are placed in entity type Employee, which is a role type. In this case, we remove from the role type all the intrinsic attributes and put them into a newly created natural/phase type playing this role type. For example, in Figure 4.10 (section 4.3.4.3), a new natural type Person is created with attributes Name and SSN, which plays role type Employee.

In the case that a natural/phase type participates in a relationship with other type(s) in the ER model (which is not allowed in our model), we may need to create a new role type that participates in the relationship and is played by the natural/phase type. For example, if in the original ER model, natural type University participates in relationship Enrollment with role type Student, then we may create a new role type UniversityEnrolled to replace University in relationship Enrollment, and let University play UniversityEnrolled.

Moreover, as demonstrated in Figures 5.1 and 5.2, sometimes we need to create new natural/phase types with at least an identifier attribute (e.g., University with the identifier attribute Name) and role types (Employer, UniversityEnrolled, and StudentEmployer) in order for our model to be considered complete. Here, the

identifier attribute of the created natural/phase type is important because, as we discussed in section 5.2.4, it can be used later (as part of a combined key attribute) to identify each relationship instance associated to the role types played by this natural/phase type. Also note that, in Figure 5.2, the creation of additional role types Employer, UniversityEnrolled, and StudentEmployer is necessary because they share mutual attributes with role types Employee, Student, and StudentAssistant respectively.

Step 2: Remove all generalization/specialization relationships between natural/phase types and role types in the original ER model. Relate using *plays* relationship each role type to the corresponding natural/phase type(s) playing it. Keep intact the generalization/specialization relationships between natural/phase types and between role types. For example, in Figure 5.2, whereas the two generalization/specialization relationships between Employee and StudentAssistant and between Student and StudentAssistant are kept intact, those between natural type Person and role types Employee and Student are replaced by two *plays* relationships.

For newly created natural/phase types in step 1, relate them using generalization/specialization relationship if appropriate. Repeat this for newly created role types as well. For example, in Figure 5.2, two generalization/specialization relationships are created between newly created role types Employer and StudentEmployer and between UniversityEnrolled and StudentEmployer.

Step 3: For each natural/phase type, place its attributes in the attribute compartment of the type. Relate using an association each role type to other role type(s) with which it shares its attributes. These attributes are placed in the attribute compartment of an association class (with a proper name) attached to the association.

5.3 Mapping Conceptual Models Based On Our Metamodel Into Corresponding Relational Database Schema

In this section, we describe the process of mapping a conceptual database model based on our metamodel into a relational database schema.

Step 1: For each natural/phase type N/P in the conceptual database model, create a relation R that includes all the attributes of N/P. These attributes model Bunge-attribute functions representing general intrinsic properties of the things represented by instances of the natural/phase type. Choose one of the key attributes of N/P as primary key for R.

For example, from Figure 5.2, we create the relations Person and University for natural types Person and University, as shown in Figure 5.10. We choose SSN and Name as primary keys for the relations Person and University, respectively.

In another example, from Figure 5.4, we create the relations Person, Bank, Company, Car, and Truck for the corresponding natural types, as shown in Figure 5.11. We choose SSN, BName, and CName as primary keys for the relations Person, Bank, and Company, choose VehicleID as primary key for the relations Car and Truck.

Person

SSN	Name	Sex	Address	BirthDate

University

Name

Employment

SSN	Name	Salary

Enrollment

SSN	Name	MajorDept

StudentEmployment

SSN	Name	PercentTime

Roles

RoleName	PlayerName	RelationshipName	ID
Employee	Person	Employment	SSN
Student	Person	Enrollment	SSN
StudentAssistant	Person	Employment	SSN
StudentAssistant	Person	Enrollment	SSN
StudentAssistant	Person	StudentEmployment	SSN
Employer	University	Employment	Name
UniversityEnrolled	University	Enrollment	Name
StudentEmployer	University	Employment	Name
StudentEmployer	University	Enrollment	Name
StudentEmployer	University	StudentEmployment	Name

Fig. 5.10. Relational database schema diagram for the conceptual model in Figure 5.2

Step 2: For each natural/phase type generalization/specialization with m subtypes $\{S_1, S_2, ..., S_m\}$ and supertype S, create a relation R for S and m relations $\{R_1, R_2, ..., R_m\}$ for the subtypes. R contains the key attribute k of S as primary key and all common attributes. Each R_i contains k as primary key and the attributes specific to that subtype.

Person

SSN	Name	BirthDate	OwnerID

Bank

BName	BAddress	OwnerID

Company

CName	CAddress	OwnerID

Car

VehicleID	CModel	CYear	CStyle

Truck

VehicleID	CModel	CYear	Tonnage

Ownership

OwnerID	VehicleID	PurchaseDate	LienOrRegular

Roles

RoleName	PlayerName	RelationshipName	ID
Owner	Person	Ownership	OwnerID
Owner	Bank	Ownership	OwnerID
Owner	Company	Ownership	OwnerID
Registered_Vehicle	Car	Ownership	VehicleID
Registered_Vehicle	Truck	Ownership	VehicleID

Fig. 5.11. Relational database schema diagram for the conceptual model in Figure 5.4

Step 3: For each role type T, since it has no intrinsic attributes of its own, we may create a relation R that includes only a primary key attribute. If the role type is played by only one natural/phase type, include as the primary key attribute of R the primary key attribute of the relation that corresponds to the natural/phase type playing this role type. If, however, the role type is played by more than one natural/phase types, if these natural/phase types have the same key attribute, include as the primary key attribute of R this key attribute; otherwise, if these natural/phase types have different key attributes, a new key attribute must be specified for the role type and included as the primary key attribute of R. Moreover, we also add this key attribute as foreign key to each relation corresponding to a natural/phase type playing the role type.

Note that, since R contains only a primary key and does not provide additional information, usually we do not need to include R in a relational database schema. In practice, in order to reverse-engineer high-level conceptual models such as that in Figure 5.2 from existing relational database schema, we may store structure information on role types (metadata) into a relation called Roles which has four attributes RoleName, PlayerName, RelationshipName, and ID. For each role type in a conceptual model based on our metamodel, attribute RoleName contains the name of the role type, PlayerName contains the name of (one of) its player (a natural or phase type), RelationshipName contains the name of (one of) the relationship type this role type participates in, and ID contains the name of the primary key of the relation corresponding to the role type. Because

the values of attributes RoleName, PlayerName, and RelationshipName can uniquely identify a tuple in relation Roles, the combination of the three attributes can be used as the primary key of Roles.

For example, from Figure 5.2, relation Roles can be created as shown in Figure 5.10. Relations corresponding to role types Employee, Student, and StudentAssistant get their primary key SSN from relation Person. Relations corresponding to role types Employer, UniversityEnrolled, and StudentEmployer get their primary key Name from relation University.

In another example, from Figure 5.4, relation Roles can be created as shown in Figure 5.11. Whereas relation corresponding to role type Registered_Vehicle gets its primary key VehicleID from relations Car and Truck, a new key attribute OwnerID is included as the primary key of the relation corresponding to role type Owner and also added as foreign key to relations Person, Bank, and Company.

Step 4: For each association class A in the conceptual model, create a relation R that includes all the attributes of A. Include as foreign key attributes in R the primary keys of the relations that represent the participating role types. If the association is a binary 1:1 relationship type, any primary key of the relations representing the two participating role types can be used as primary key of R. If the association is a binary $1:N$ relationship type, the primary key of the relation representing the participating role type on the N-side can be used as primary key of R. If the

association is a binary $M:N$ relationship type, the combination of the primary keys of the relations representing the participating role types will be used as primary key of R. If the association is a n-ary relationship type, where $n > 2$, the primary key of R will be the combination of the primary keys of the relations representing the participating role types whose cardinality constraints are not 1.

Because each instance of the association class has an existence dependency on each role instance (Actually, this role instance is a natural/phase type instance) it relates, the propagate (CASCADE) option for the referential triggered action should be specified on the foreign keys in the relation corresponding to the association class.

For example, from Figure 5.2, we create the relations Employment, Enrollment, and StudentEmployment for the corresponding association classes, as shown in Figure 5.10. The primary key of relation Employment/Enrollment/ StudentEmployment is the combination of SSN from (the implicit) relation Employee/Student/StudentAssistant and Name from (the implicit) relation Employer/UniversityEnrolled/StudentEmployer.

In another example, from Figure 5.4, we create the relation Ownership for the corresponding association class, as shown in Figure 5.11. Similarly, the primary key of relation Ownership is the combination of OwnerID from (the implicit) relation Owner and VehicleID from (the implicit) relation Registered_Vehicle.

5.4 Conclusion

In this chapter, we compare conceptual models created using our metamodel to those created using ER approach with respect to conceptual database modeling. Then we describe how to map a conceptual model based on our metamodel into relational database schema. We demonstrate using examples that relational database schemata generated using our approach are more stable with respect to a kind of requirements change, i.e., a change in multiplicity of roles. However, an experimental assessment of our claim is needed in the future. Moreover, by distinguishing among natural types, phase types, and role types, a number of real world semantics and rules can be implemented as integrity constraints of a relational database schema. In Appendix, an even more complex example is given illustrating a university conceptual schema.

Chapter 6

Conclusions

6.1 Review of The Research

Developing a conceptual model that faithfully represents the domain it is intended to represent is of critical importance for successful information system development. Although it is widely held that UML could be used both for modeling software, and for modeling the problem domain that is supported by a system, this expectation is problematic because of UML's implementation oriented origin. In this thesis, our research objective is to develop an ontological core of UML for conceptual modeling based on Bunge's ontology, focusing on static aspects. In Chapter 3, we developed an ontological semantic framework for a core set of UML's model constructs (object, attribute, class/type, association, link, association class, state, state transition, and operation). The choice of these constructs in our ontological UML core is driven by Bunge's ontology. We also analyze consequences for conceptual modeling using UML

based on this semantic mapping. In particular, we focused on UML association and link and indicate that links and their corresponding associations in UML class diagrams are of fundamentally different ontological nature from those in UML collaborations. Thus we deem the so-called *Baseless Link Problem* discussed in the UML literature as an unpleasant consequence of UML's implementation oriented origin from object-oriented programming, which disappears naturally in our approach.

In Chapter 3, we proposed that Bunge-functional schema is modeled by UML-class/type. However, Bunge does not give guidelines as to how to differentiate different "kinds" of functional schemata thus different "kinds" of UML-classes/types. Accordingly, in Chapter 4, we investigate other ontological theories (OntoClean and Guizzardi et al.'s ontological profile), based on which we propose an ontological metamodel of classifiers and incorporate it into our framework built on Bunge's ontology. Moreover, in Chapter 4, we focus on the definition, properties, and representation of the notion of roles in the literature. In object-oriented and conceptual modeling, role is a powerful modeling concept. However, a lot of confusion exists on its definition, properties, and representation. Most role models proposed up to now have been primarily based on implementation considerations. In our metamodel, instead of viewing attributes of a role type as intrinsic attributes owned by the role type exclusively, our approach regards attributes of a role type as mutual attributes shared by participating role types, thus handles the counting problem and the related role identity problem and conforms to the fundamental features identified in the literature.

To demonstrate the conceptual and practical usefulness of our metamodel, in Chapter 5, conceptual models created using our metamodel are compared to those created

using ER approach with respect to conceptual database modeling. Then we describe how to map a conceptual model based on our metamodel into relational database schema. We demonstrate using examples that relational database schemata generated using our approach are more stable with respect to a kind of requirements change, i.e., a change in multiplicity of roles. However, an experimental assessment of our claim is needed in the future. Indeed, since in ER, there is only entity type, no explicit role type, all intrinsic and mutual attributes are placed by modelers in entity types or relationship types arbitrarily. Therefore, using the ER approach, it is possible that the resulting relational database schema has to evolve after it has already been in existence for some time. This situation must be taken seriously since, as many authors have argued, the cost of repairing requirements errors during maintenance may be two orders of magnitude greater than that of correcting them during RE. Moreover, in our approach, by distinguishing among natural types, phase types, and role types, a number of real world semantics and rules can be implemented as integrity constraints of a relational database schema.

6.2 Future Research

There are a lot of potential future research directions for the research. In this section, some interesting problems that need further investigation are given:

- In conceptual modeling, notions of Part and Whole are of significant importance. In Chapter 3, we argue that the distinction between UML-composition/aggregation and between Bunge-aggregate/system is along different dimensions. In fact, there

are various kinds of mereologies, or formal ontological theories of part, whole, and related concepts. One potentially fruitful area of research is to analyze characteristics of different part-whole relationships and further investigate how to incorporate them into our ontological core.

- In this thesis, we focus on discussing static aspects of UML. As argued in Chapter 3, the dynamic aspects of UML are quite confusing – there is no precise definition even for such fundamental notion as state. Consequently, research is needed to investigate and develop an ontological foundation for UML dynamic aspects, and we believe that Bunge's ontology is the appropriate candidate.

Appendix

An Example

In this section, we present a more substantial realistic example as illustrated in Figure a.1 (adapted from [69, p. 91]). The university database is used to keep track of faculty and their research projects, as well as students and their majors, transcripts, and registration. The corresponding model of Figure a.1 using our metamodel is shown in Figure a.2.

In Figure a.2, relationships in Figure a.1 like Chairs and Belongs between Faculty and Department are replaced by relationships Chair between DepartmentHead (subclass of Faculty) and DepartmentEnrolled and Employment between Faculty and DepartmentEnrolled. Similarly relationships Advisor and Committee between Faculty and Grad_Student are replaced by relationships Supervision between Advisor and Grad_Student and Co_Supervision between Committee_Member and Grad_Student. Both Advisor and Committee_Member are subclasses of Faculty. Meanwhile, two mutual

attributes StartDate and ResearchTopic are added to Supervision and one mutual attribute StartDate is added to Co_Supervision. Similarly, a mutual property Prerequisite is also added to the relationship between role types CourseOffered and DepartmentEnrolled and a mutual property StartDate is added to the relationship Chairs between DepartmentHead and DepartmentEnrolled.

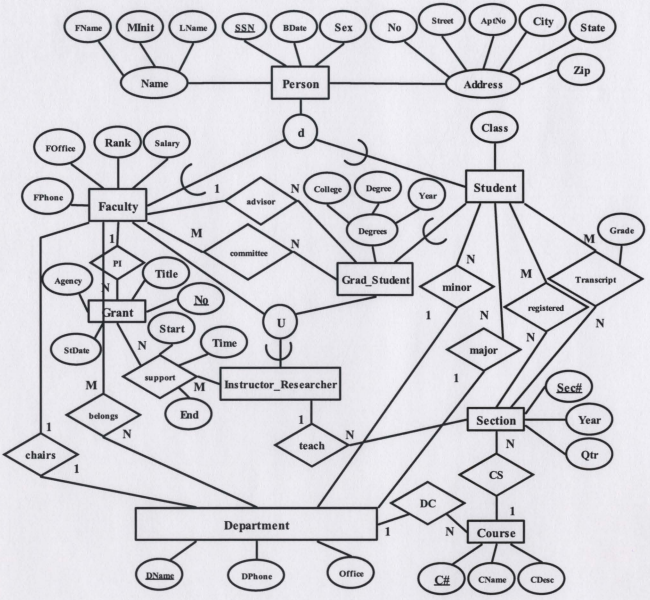

Fig. a.1. An EER conceptual schema for a university database [69, p. 91]

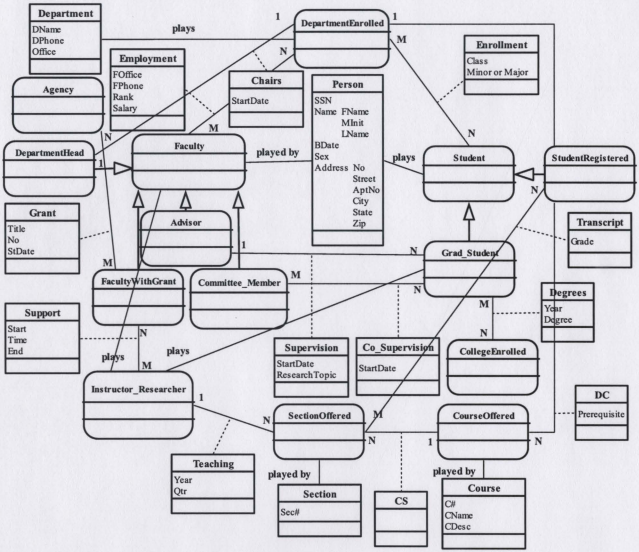

Fig. a.2. The corresponding model of Figure a.1 using our metamodel

As discussed in section 4.3.2, role can play role. In Figure a.2, role types Faculty and Student also play role type Instructor_Researcher. Moreover, there is no relationship between Grant and Instructor_Researcher. Instead, a new role type FacultyWithGrant is created. Entity type Grant and relationship type Support in Figure a.1 are represented as

relationships between Agency and FacultyWithGrant as well as between FacultyWithGrant and Instructor_Researcher.

Bibliography

[1] Object Management Group: UML Specification 1.5

[2] Evermann, J., Wand, Y., *Toward Formalizing Domain Modeling Semantics in Language Syntax*, IEEE Transactions on Software Engineering, Vol. 31, No. 1, January 2005, pp. 21 - 37.

[3] Opdahl, A., Henderson-Sellers, B., *Ontological Evaluation of the UML using the Bunge-Wand-Weber Model*, Software and Systems Modelling, vol. 1, 1, pp. 43-67, 2002.

[4] Glinz, M.: *Problems and Deficiencies of UML as a Requirements Specification Language*, Proceedings of the Tenth International Workshop on Software Specification and Design (IWSSD-10), San Diego, pp 11-22, IEEE Computer Society Press, Nov. 2000.

[5] Dori, D.: *Why significant UML change is unlikely*, Commun. ACM 45(11): 82-85, 2002.

[6] Simons, A. J.H.; Graham, I.: *37 Things that Don't Work in Object-Oriented Modeling with UML*, ECOOP 98 Workshop on Precise Behavioral Semantics.

[7] Simons, A. J.H.; Graham, I.: *30 Things that go wrong in object modelling with UML 1.3*, chapter 17 in *Behavioral Specifications of Businesses and Systems eds*, H Kilov, B Rumpe, I Simmonds, Kluwer Academic Publishers, 1999, 237-257.

[8] Evans, A.S., Clark, A.N., *Foundations of the Unified Modeling Language*, 2nd Northern Formal MethodsWorkshop, Ilkley, Electronic Workshops in Computing, Berlin: Springer Verlag, 1998.

[9] Evans, A.S. et al., *Meta-Modelling Semantics of UML*, in Kilov, Haim (ed.) Behavioural Specifications for Businesses and Systems, Dordrecht, Holland: Kluwer Academic Publisher, 1999.

[10] Evans, A.S., Kent, S., *Core Meta-Modelling Semantics of UML: the pUML Approach*, UML'99 The Unified Modeling Language – Beyond the Standard, 2nd International Workshop, Fort Collins, CO. Berlin: Springer Verlag, pp. 99–115, 1999.

[11] Evermann, J. and Wand, Y., *Towards Ontologically-Based Semantics for UML Constructs*, in: H.S. Kunii, S. Jajodia, A. Solvberg , (Eds). Conceptual Modeling – ER 2001, Lecture Notes in Computer Science #2224, Springer,2001, pp. 341-354 (Transactions of the 20th International Conference on Conceptual Modeling (ER2001), Yokohama, Japan, November 27-30, 2001.

[12] Evermann, J. and Wand, Y., *An Ontological Examination of Object Interaction in Conceptual Modeling*, in J. Parsons and O. Sheng (Eds.), *Proceedings of the Eleventh Workshop on Information Technologies and Systems* (WITS'01), New Orleans, LA, December 2001.

[13] Parsons, J., Wand, Y., *Using Objects for Systems Analysis*, Communications of the ACM, (40)12, pp. 104-110, 1997.

[14] Wand, Y., *A proposal for a formal model of objects*, In Won Kim and Frederick H. Lochovsky, editors, Object-Oriented Concepts, Databases, and Applications, chapter 21, pages 537—539, Addison-Wesley, 1989, ACM Press Frontier Series.

[15] Wand, Y., Monarchi, D.E., Parsons, J., Woo, C.C., *Theoretical Foundations for Conceptual Modelling in information systems development*, *Decision Support Systems,* (15), pp. 285-304, 1995.

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

[17] Wand, Y., Weber, R., *An Ontological Model of an Information System*, IEEE Transactions on Software Engineering, 11, p 1282-1290, 1990.

[18] Wand, Y., Weber, R., *An Ontological Evaluation of Systems Analysis and Design Methods*, in E. Falkenberg and P. Lindgreen (Eds.), *Information Systems Concepts - An In-Depth Analysis,* Amsterdam: North-Holland, 1989, pp. 79-107.

[19] Wand, Y., Weber, R., *On the Ontological Expressiveness of Information Systems Analysis and Design Grammars*, Journal of Information Systems, October 1993, pp. 217-237.

[20] Wand, Y., Weber, R., *Towards a theory of Deep Structure of Information Systems*, Journal of Information Systems, (5), pp. 203-223, 1995.

[21] Weber, R., Zhang, Y., *An Ontological Evaluation of NIAM's Grammar for Conceptual Schema Diagrams*, Information Systems Journal, April 1996, pp. 147-170, 1996.

[22] Guizzardi, G., Herre, H., Wagner G., *Towards Ontological Foundations for UML Conceptual Models*, 1st International Conference on Ontologies, Databases and Application of Semantics (ODBASE'02), Irvine, California, USA, 2002.

[23] Guizzardi, G.; Herre, H.; Wagner, G.: *On the General Ontological Foundations of Conceptual Modeling*, ER 2002: 65-78.

[24] Mylopoulos, J: Conceptual modeling and telos, In: Locoupoulos P, Zicardi R (eds) Conceptual Modeing, Databases, and Cases, Wiley, New York, 1992.

[25] Evermann, J.; Wand, Y.: *Ontology based object-oriented domain modelling: fundamental concepts*, Requirements Eng. 10 (2): 146-160, 2005.

[26] Evermann, J.: *The Association Construct in Conceptual Modelling - An Analysis Using the Bunge Ontological Model*, CAiSE 2005: 33-47.

[27] Shanks, G.; Tansley, E.; Weber, R.: *Using ontology to validate conceptual models*, Commun. ACM 46(10): 85-89, 2003.

[28] Opdahl, A.L.; Henderson-Sellers, B.: *Grounding the OML metamodel in ontology*, The Journal of Systems and Software 2001; 57(2):119-143.

[29] Wieringa, R.; Jonge, W. de: *The identification of objects and roles: object identifiers revisited*, Technical Report IR267, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, December 1991.

[30] Bunge, M.: *Treatise on Basic Philosophy (Volume 3), Ontology I: The Furniture of the World*, Boston: Reidel, 1977.

[31] Bunge, M.: Treatise on Basic Philosophy (Volume 4), Ontology II: A World of Systems, Boston: Reidel, 1979.

[32] Bonfatti, F.; Pazzi, L.: *Ontological foundations for state and identity within the object-oriented paradigm*, International Journal of Human-Computer Studies, vol. 43, no. 5/6, pp. 891-906, 1995.

[33] Evans, Andy S. et al., *The UML as a Formal Modeling Notation*, The Unified Modeling Language UML: Beyond the Notation, First International Workshop, Mulhouse, France, Berlin: Springer Verlag, 1999.

[34] Jackson, M., Software Requirements & Specifications – A lexicon of practice, principles and prejudices, ACM Press/Addison-Wesley, Workingham/England, 1995.

[35] Sommerville, I., Sawyer, P., *Requirements Engineering – A good practice guide*, Wiley Chichester/England, 1997.

[36] Offen, R.: *Domain understanding is the key to successful system development*, Requirements Eng., vol. 7. pp. 172-175, 2002.

[37] Davis, A.M., *Software Requirements: Objects, Functions, States*, Prentice-Hall, Upper Saddle River, NJ, 1993.

[38] Woodfield, S.N.: *The Impedance Mismatch Between Conceptual Models and Implementation Environments*, in ER'97 Workshop 4 Proceedings, 1997.

[39] Sowa, J.F., *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, New York; 1984.

[40] Weber, R.: *Ontological Foundations of Information Systems,* Buscombe Vicprint, Blackburn, Victoria, 1997.

[41] Stevens, P.: *On the interpretation of binary associations in the Unified Modelling Language*, Software and System Modeling 1(1): 68-79, 2002.

[42] Rumbaugh, J.; Jacobson, I.; Booch G.: *The Unified Modeling Language Reference Manual, Addison-Wesley Professional*, December 1998.

[43] Simons, P.: *Parts: a study in ontology*, Claredon Press, New York, 1987.

[44] Peirce, C. S.: *Scientific Metaphysics*, Vol. VI of the Collected Papers, Cambridge, Mass.: Harvard University Press, 1935 (original from 1892-1893).

[45] Guizzardi, G.: *Ontological foundations for structural conceptual models*, PhD thesis, University of Twente, the Netherlands, October 2005.

[46] Rosemann, M.; Green, P.: *Developing a metamodel for the Bunge-Wand-Weber ontological constructs*, Information Systems 27(2): 75-91, 2002.

[47] Guarino, N.; Welty, C.: *An overview of OntoClean*, in S. Staab, R. Studer (eds.), Handbook on Ontologies, Springer Verlag, pp. 151-159, 2004.

[48] Welty, C.; Guarino, N.: *Supporting ontological analysis of taxonomic relationships*, Data & Knowledge Engineering, 39(1):51-74, 2001.

[49] Guarino, N.; Welty, C.: *Evaluating ontological decisions with OntoClean*, Commun. ACM 45(2): 61-65, 2002.

[50] Guarino, N.; Welty, C.: *A formal ontology of properties*, In Rose Dieng (ed.), Proceedings of the 12th Int. Conf. On Knowledge Engineering and Knowledge Management, Springer Verlag LNCS.

[51] Guizzardi, G.; Wagner, G.; Guarino, N.; McBrien, P.; Rizopoulos, N.: *An Ontologically Well-Founded Profile for UML Conceptual Models*, CAiSE, 112-126, 2004.

[52] Guizzardi, G.; Wagner, G.; Sinderen, M.: *A Formal Theory of Conceptual Modeling Universals*, WSPI 2004.

[53] Marcos, E.; Cavero, J.M.: *Hierarchies in Object Oriented Conceptual Modeling*, OOIS Workshops 2002: 24-33.

[54] Steimann, F.: *On the representation of roles in object-oriented and conceptual modeling*, Data & Knowledge Engineering, 35 83-106, 2000.

[55] Steimann, F.: *A radical revision of UML's role concept*, UML 2000, 194-209.

[56] Steimann, F.: *Role = Interface: A merger of concepts*, Journal of Object-Oriented Programming 14:4 (2001) 23-32.

[57] Dahchour, M.; Pirotte, A.; Zimányi, E.: *A role model and its metaclass implementation*, Information Systems 29(3): 235-270, 2004.

[58] Wieringa, R.; Jonge, W. de; Spruit, P.: *Using Dynamic Classes and Role Classes to Model Object Migration*, Theory and Practice of Object Systems 1, 1 Page 61-83, 1995.

[59] Gottlob, G.; Schrefl, M.; Rock, B.: *Extending Object-Oriented Systems with Roles*, ACM Trans. Inf. Syst. 14(3): 268-296, 1996.

[60] *Roles, An Interdisciplinary Perspective*, 2005 AAAI Fall Symposium.

[61] Sowa, J.F.: *Using a lexicon of canonical graphs in a semantic interpreter*, in: M.W. Evens, ed., Relational Models of the Lexicon (Cambridge University Press, Cambridge, 1988) 113-137.

[62] Bachman, C.W.; Daya, M.: *The role concept in data models*, in: Proceedings of the Third International Conference on Very Large Databases, 1977, pp. 464-476.

[63] Guarino, N: *Concepts, attributes and arbitrary relations*, Data & Knowledge Engineering 8 (1992) 249-261.

[64] Loebe, F.: *An Analysis of Roles – Toward Ontology-Based Modelling*, Master's Thesis, University of Leipzig, 2003.

[65] Masolo, C.; Vieu, L.; Bottazzi, E.; Catenacci, C.; Ferrario, R.; Gangemi, A.; Guarino, N.: *Social roles and their descriptions*, In KR 2004, 267-277.

[66] Al-Jadir, L.; Leonard, M.; If we refuse the inheritance..., in: T.J.M. Bench-Capon, G. Soda, A.M. Tjoa (Eds.), Proceedings of the Tenth International Conference on Database and Expert Systems Applications, DEXA'99, Florence, Italy, Lecture Notes in Computer Science, Vol. 1677, Springer, Berlin, 1999.

[67] Steimann, F.: *The Role Data Model Revisited*, in: Roles, an interdisciplinary perspective, AAAI Fall Symposium, 2005.

[68] Masolo, C.; Guizzardi, G.; Vieu, L.; Bottazzi, E.; Ferrario, R.: *Relational Roles and Qua-individuals*, in: Roles, an interdisciplinary perspective, AAAI Fall Symposium, 2005.

[69] Elmasri, R.; Navathe, S.: *Fundamentals of Database Systems*, 3nd ed. AddisonWesley, 2000.

[70] Halpin, T. A.: *Business rules and object-role modeling*, DBP&D 9, No. 10, October 1996.

[71] Lieberman, H.: *Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems*, In N. Meyrowitz (Ed.), Object-oriented programming: Systems, languages and applications, Page 214-223, 1986.

[72] Object Management Group: *UML 2.0 Superstructure Specification*, August 2005.

[73] Chen, P.P.: *The Entity-Relationship Model - Toward a Unified View of Data*, ACM Transactions on Database Systems (TODS) Volume 1 Number 1: pp 9-36, 1976.

[74] Chen, P. P.: *Entity-Relationship Modelling: Historical Events, Future Trends, and Lessons Learned*, Springer-Verlag New York, Inc, pp. 296-310, 2002.

[75] Cris Kobryn: *UML 2001: A Standardization Odyssey*, Commun. ACM 42(10): 29-37, 1999.

[76] Booch, G.; Rumbaugh, J.; Jacobson, I.: *The Unified Modeling Language User Guide, Addison-Wesley Professional*, October 1998.

[77] Chandrasekaran, B.; Josephson, J.R.; Benjamins, V.R: *What are ontologies, and why do we need them?*, IEEE Intelligent Systems 14, 1 (Jan./Feb. 1999), 20–26.