UNIVERSITY OF CRETE
DEPARTMENT OF COMPUTER SCIENCE

Contextualization:
An Abstraction Mechanism for Information Modeling

Manos Theodorakis

Doctoral Dissertation

Heraklion, Crete
June 2001

UNIVERSITY OF CRETE
DEPARTMENT OF COMPUTER SCIENCE

Contextualization:
An Abstraction Mechanism for Information Modeling

Dissertation submitted by
Manos Theodorakis
in partial fulfillment of the requirements for
the PhD degree in Computer Science

Author:

Manos Theodorakis

Examination Committee:

Panos Constantopoulos, Professor, Dept. of Computer Science, University of Crete, Advisor

Nicolas Spyratos, Professor, Dept. of Computer Science, University of Paris-Sud, France

Timos Sellis, Professor, Dept. of Electrical and Computer Engineering, National Technical University of Athens

Panos Trahanias, Associate Professor, Dept. of Computer Science, University of Crete

Dimitris Plexousakis, Assistant Professor, Dept. of Computer Science, University of Crete

Evangelos Markatos, Assistant Professor, Dept. of Computer Science, University of Crete

Manolis Koubarakis, Assistant Professor, Dept. of Electronic and Computer Engineering, Technical University of Crete

Accepted by:

Panos Constantopoulos
Chairman of Graduate Studies

June 2001

Contextualization:
An Abstraction Mechanism for Information Modeling

Manos Theodorakis

Doctoral Dissertation

Department of Computer Science

University of Crete, Greece

Abstract

Users of very large information bases are generally concerned with a small portion of the information stored. Different users need to view the same information differently according to their needs. Designers, on the other hand, need to structure information in accordance to the user group it is addressed. It is therefore important for information bases to provide their users and designers with tools that allow them to isolate desired portions of information and manage them in their own ways, e.g., by naming information objects using familiar names, grouping objects together according to their own perception, and so on. In other words it is important for information systems to provide an abstraction mechanism that reflects the intuitive notion of context.

The aim of this thesis is to introduce a notion of context in information modeling providing a structuring mechanism for large information bases, and to establish a formal notion of context along with a general framework for maintaining and querying contexts.

Specifically, we define a set of operations for manipulating contexts. These operations support context creation, update, copy, union, intersection, and difference. In particular, our operations of context union, intersection, and difference are different from these of set theory as they keep track of the context involved. However, they also satisfy the important properties of commutativity, associativity, and distributivity.

Then, we show how a particular semantic data model (the Telos data model) can be incorporated into the proposed contextualized framework. Thus, we enhance our notion of context by structuring its contents through the traditional abstraction mechanisms, i.e., classification, generalization, and attribution, and we study the interactions between contextualization and these mechanisms as well as the constraints that govern such interactions.

Finally, we present a query language for contextualized information bases. The query language can be used for navigating through contexts, as well as for retrieving information of interest.

We demonstrate the applicability of our mechanism to several areas requiring features which rely on partitioning such as views, workspaces, versions, thesauri fields and facets.

Supervisor:
Panos Constantopoulos,
Professor of Computer Science,
University of Crete

In memory of my father.

Acknowledgments

First, I would like to thank my supervisor Prof. Panos Constantopoulos, for providing me with the guidance and support over the course of my research. I would also like to thank the other members of my advisory committee (Prof. Nicolas Spyratos and Dr. Anastasia Analyti). More importantly, I want to thank them for teaching me what constitutes quality research.

I would like to thank the members of my dissertation committee from the University of Crete, Professors Dimitris Plexousakis, Panos Trahanias, and Evangelos Markatos, as well as the external members Professors Timos Sellis (National Technical University of Athens) and Manolis Koubarakis (Technical University of Crete), for their helpful comments and questions. It is my honor to have them in my committee.

Especially, I want to express my gratitude to Prof. Nicolas Spyratos. Nicolas has spent many hours with me discussing all of the topics of my thesis and writing papers. His way of thinking is really impressive, been both analytical and abstractive as needed. His suggestions on how to go on with research was crucial. Our collaboration and his consultation continued, even when we were not physically together. The most important think I learned from Nicolas is that it is possible to work hard and be really good, but at the same time maintain an excellent sense of humor and an appreciation for the finer things in life. Both as a friend and as an advisor, Nicolas is one of the most genuine and caring person I have ever known. I feel really lucky to have worked with him. Thank you!

Many thanks to Anastasia Analyti. Anastasia has spent countless hours with me discussing many details of this research, developing theories, and writing papers. In retrospect, her patience with me during our first period of cooperation (where she spent many hours discussing, evaluating, and extending my thoughts) seems worthy of some kind of prize. Her mathematical abilities are astounding. I learned from her what deep mathematical thinking means and how to find bugs in hard mathematical algorithms, even when they are deep hidden. She gave me consistent support and encouragement, and helped me with some of the most difficult proofs within the dissertation.

Thanks to Nicolas Spyratos and Yiannis Tzitzikas for their valuable comments concerning my presentation.

Thanks to Vassilis Christophides for providing me with bibliography and for his valuable comments during trial presentations.

Thanks to Dimitris Plexousakis for our discussions and the bibliography concerning graph theory.

Thanks to Mina Akaishi for developing a prototype implementation of context mechanism in the IntelligentPad system.

I would also like to thank all of my friends who helped me to stay sane and enjoy life. I

am in debt to many of you, including: Polivios, Yannis, Carola, Polykarpos, Katerina, Nikos, Antonis, and many others. When I was down, you lifted me up, made me smile and gave me the strength to persevere.

Finally, I would like to acknowledge my parents and my brother for their support. Especially, I want to thank my father who instilled in me the desire to learn and he never stopped believing in me. This thesis is devoted to him.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The notion of context is of fundamental importance in cognitive psychology, linguistics, and computer science. In computer science, a number of formal or informal definitions of some notion of context have appeared in several areas, such as artificial intelligence [51, 73, 46], software development [92, 45, 99, 106, 108, 56, 63], databases [7, 44, 49, 2, 27, 55, 86], machine learning [76, 130, 71], and knowledge representation [84, 122, 126, 121, 135, 31, 116, 26, 18]. See also [80] for a general survey on the subject.

However, all these notions of context are very diverse and serve different purposes. In software development the notion of context appears in the form of views [7, 44, 49, 98, 2], aspects [92], and roles [45, 99], for dealing with data from different perspectives, or even in the form of workspaces which are used to support cooperative work [56]. In machine learning, context is treated as environmental information for concept classification [76, 130, 71]. In the so called multiple databases, context appears as a collection of meta-attributes for capturing class semantics [55]. In artificial intelligence, the notion of context appears as a means of partitioning knowledge into manageable sets [51], or as a logical construct that facilitates reasoning activities [73, 46]. In particular, in the area of knowledge representation, the notion of context appears as an abstraction mechanism for partitioning an information base into possibly overlapping parts (e.g. [84, 121, 126, 135, 31, 116, 26, 18]).

This thesis focuses on the area of knowledge representation, where most research assumes an information base that describes a particular application based on an information model. In order to deal with the complexity of the data stored in an information base, abstraction mechanisms, such as classification, generalization and attribution, are used for their organization. An abstraction mechanism that is by far less well understood and studied is *contextualization*.

Contextualization can be seen as an abstraction mechanism which allows partitioning and packaging of information being added to an information base [82]. Its importance is stemming from the fact that in any modeling task there are often differences of opinion or perception among those gathering, providing, or using information. But even within the same group of people, there is a need to give descriptions of objects with respect to particular viewpoints or situations. Moreover, in very large information bases users are generally concerned with a small portion of the information stored. Different users need to view the same information differently according to their own needs. It is therefore important for information bases to provide their users and designers with tools that allow them to isolate desired portions of information and manage them in their own ways, e.g., by naming information objects using

familiar names, grouping objects together according to their own perception, and so on. In other words, it is important for information systems to provide an abstraction mechanism that reflects the intuitive notion of *context*. Such a mechanism should therefore allow to

- view information from different viewpoints and / or in different level of details,
- isolate a portion of the information base,
- restructure its contents,
- name the contents using familiar names,
- eventually link the contents to other portions of interest,
- handling inconsistent information.

An important feature of context is that the same information object can have different meanings in different contexts. For example, a computer can be part of two different contexts, one providing information about the hardware components and the other about the software components. In these two contexts the term "computer" has two different meanings: in the hardware context a computer is thought of as an assembly of hardware components, whereas in the software context it is thought of as an assembly of software components. In addition, an information object may be meaningful in one context and meaningless in another. For example, it is meaningful to talk of an airplane in the context of 20th century but meaningless in the context of 16th century. More generally, *the information contained in a context is dependent on that context*.

The aim of this thesis is introduce a notion of context in information modeling providing a structuring mechanism for large information bases, and to establish a formal notion of context along with a general framework for maintaining and querying contexts. The proposed context model is intended to form a unifying generic framework in order to support the development and effective use of large information bases in various application areas, and especially, the development of applications which rely on partitionings of an information base, such as views, workspaces, versions, fields and facets of thesauri management systems.

A context is seen as a set of information objects, each object carrying a set of names and a reference to some other context [122, 123]. Contexts can share objects (i.e., an object can belong to more than one context) but names and references of objects are context-dependent. Our notion of context supports synonyms, homonyms, and anonyms.

Moreover, we define a set of operations for manipulating contexts [125, 121, 120]. These operations support context creation, update, copy, union, intersection, and difference. In particular, our operations of context union, intersection, and difference are different from these of set theory as they keep track of the context involved. However, they also satisfy the important properties of commutativity, associativity, and distributivity. Our model contributes to the efficient handling of information, especially in large information systems, and in distributed, cooperative environments, as it enables (i) representing (possibly overlapping) partitions of an information base; (ii) partial representations of objects, (iii) flexible naming (e.g. relative names, synonyms and homonyms), (iv) focusing attention, (v) handling inconsistent information, and (vi) combining and comparing different partial representations.

Then, we show how a particular semantic data model (the Telos data model) can be incorporated into the proposed contextualized framework [122, 123]. Thus, we enhance our notion

of context by structuring its contents through the traditional abstraction mechanisms, i.e., classification, generalization, and attribution. We show that, depending on the application, our notion of context can be used either as an alternative way of modeling or as a complement of the traditional abstraction mechanisms. It is important that we study the interactions between contextualization and the traditional abstraction mechanisms as well as the constraints that govern such interactions.

Finally, we present a query language for contextualized information bases. The query language can be used for navigating through contexts, as well as for retrieving information of interest [124].

We demonstrate the applicability of our mechanism to several areas requiring features which rely on partitioning such as views, workspaces, versions, thesauri fields and facets.

We begin this chapter by outlining issues that govern information bases and by presenting conventional abstraction mechanisms. We then present some problems appearing in a variety of information bases, and as we go along, we outline a solution to these problems introducing the *contextualization abstraction mechanism*. The proposed mechanism is presented in details and further justified in the following chapters.

1.1 Information modeling

Information modeling is concerned with the construction of computer-based symbol structures which capture the meaning of information and organize it in ways to make it understandable and useful to people [82]. Such symbol structures are referred to as *information base* (this term is a generalization of the terms *database* and *knowledge base*). From now on we shall refer to computer-based symbol structures, as *descriptions*. The part of the real world being modeled by an information base is referred to as *application*.

An *information model* is defined as a collection of description types, whose instances are used to describe an application, a collection of operations which can be applied to any valid description, and a collection of general integrity rules which defines the set of consistent description states, or changes of states. For example, regarding the *relational model* for databases, its basic description types include `table`, `tuple`, and `domain`, its associated operations include `add`, `remove`, `update` operations for tuples, and / or `union`, `intersection`, `join` operations for tables. The relational model includes a single integrity rule: No two tuples within a table can have the same key [82].

The information models proposed and used over the years have been classified into the following three different categories [82]:

1. *Physical Information Models*: Such models employ conventional data structures and other programming constructs to model an application in terms of records, strings, arrays, lists, variable names, B-trees, and the like.

2. *Logical Information Models*: Such models offer abstract mathematical symbol structures (e.g. sets, arrays, relations) for modeling purposes, hiding the implementation details from the user. Examples of these models are the relational and network models for databases.

3. *Conceptual Information Models*: Such models offer highly sophisticated and more expressive facilities for modeling applications and structuring information bases. These models use semantic terms for modeling an application, such as `Entity`, `Activity`, `Agent`, `Goal`. Moreover, they offer means for organizing information in terms of *abstraction mechanisms*, such as classification, generalization, and aggregation. Such models are supposed to model an application more directly and naturally [47, 90]. From now on, we call these models simply as *conceptual models*.

In the sequel, we focus the discussion on conceptual models, since they constitute the state-of-the-art on the field for more than two decades.

1.1.1 Information base framework

We assume that the real world consists of *real world objects* (concrete objects or concepts). We also assume that an information base consists of *model objects* (entities, relationships, attributes, activities, or whatever). Each real-world object is represented in the information base by a model object. In this thesis, whenever we use the term *object* we mean *model object*.

Objects can be *concrete* (e.g. `Marry`, `1998`, `Greece`), which are also called *tokens*, or *abstract* (e.g. `Person`, `Year`, `Country`), which are also called *higher-level* or *generic objects*. Moreover, objects can be *individuals* or *links*. Individuals are intended to represent entities (concrete ones such as `Marry`, or abstracted ones such as `Person`), while links represent binary relationships between entities. Three common kinds of links are *instance-of*, *ISA*, and *attribute* links.

Abstract objects are objects modeled using the abstraction mechanisms offered by conceptual models. Three of the most useful and common abstraction mechanisms are: *classification*, *generalization*, and *aggregation*, which are described in the following subsection.

1.1.2 Abstraction mechanisms

The key concept to master the complexity in information handling is what is usually called the *abstraction principle* [135]. In order to deal with the complexity of the data stored in an information base we use abstraction mechanisms for their organization. By definition, abstraction involves suppression of (irrelevant) detail, and retention of the essential information [135, 82]. In other words, an abstraction is a simplified description, or specification, of a system that emphasizes some of the system's details or properties while suppressing others [105]. As a result, the higher-level concept may be more easily understood and used.

The most common abstraction mechanisms used in information modeling are the following [52, 114, 109, 82]:

1. *Classification*: an object (entity, relationship, attribute, activity or whatever) within an information base is *classified* under one or more generic objects (*classes*).

2. *Generalization*: generic objects are organized in terms of a partial order relation determined by their generality / specificity.

3. *Aggregation*: objects are viewed as aggregates of their components or parts (also called *partOf*).

1.2 Some problems in information modeling

1.2.1 Relative semantics

For each information modeling system only a part of all possible facts, stated about reality, is of interest. Users could judge if a certain fact is relevant or not (according to their interests). Thus, human perception is applied to the real world to extract a slice of the reality. The problem is that in any modeling task there are often differences of perception or opinion among those gathering, providing, or using information.

Thus, the same real world object or real world slice can be seen, described, or interpreted differently by different people or user groups. For example, in a painting exhibition, a painting may be described differently from an expert (e.g. a painter, who knows details about painting techniques), than a non-expert (e.g. a curator, who may know only the style of the painting and its painter). But even the same person can give different descriptions of the same real world object, depending on the reason or the person to whom the description is addressed. For example, a painting would be described in simple terms if the description is addressed to a primary school, and in technical terms, if the description is addressed to painting experts.

Moreover, it is often the case that in different contexts, the same object have different semantics. For example, in the context of discussing different paintings, the expression "*fake Monet*" may convey information about the artist being somebody other than Monet, i.e., not Monet; whereas in the context of discussing details of various painting techniques, the expression "*fake Monet*" may convey information about the technique used, and state that the technique is similar, but not exactly like the technique used by Monet [53].

Summarizing, in conceptual modeling, objects are needed to be described and interpreted differently, if they are viewed from different viewpoints, i.e., descriptions of objects are needed to be context-dependent. Most of the existing information models lack this ability, and a new abstraction mechanism should be developed to support different viewpoints.

1.2.2 Handling inconsistent information

Inconsistency in an information base, when viewed purely logically, seems undesirable. Indeed the traditional approach to dealing with inconsistency in data is to employ means to restore consistency immediately. However, it is important to study the larger environment containing such information bases, and the circumstances surrounding the inconsistency. In [38], it is argued that within the larger environment, an inconsistency can be desirable and useful, if we know appropriate actions to handle it. In some cases we may wish to remove the inconsistency, and in other cases we may wish to keep it. In the last case, we must be capable of representing this information in the information base. We believe that inconsistent or contradictory information can be represented in the same information base as long as it is treated in different contexts.

Unfortunately, most of the systems support global descriptions and thus information about objects must be consistent, while inconsistent is meaningless. For example, it is meaningless for a person to be classified under both the class of "Tall People" and the class of "Short People". On the other hand, it is meaningful to say that a person is tall in the context of his / her family, and short in the context of his / her basketball team.

Summarizing, most of the existing information models do not handle inconsistent information. Treatment of inconsistent information is useful as in reality, different people often have different perspectives or opinions.

1.2.3 Focused information access

Another serious problem, especially in large information bases, is that of unfocused information access. In the majority of existing systems, queries are posed globally to the information base. This results in (i) retrieving useless or irrelevant information, (ii) worsening the performance of query time.

On the other hand, it would be useful to focus attention in delimited parts of the information base and perform any action (e.g. queries, updates) within these parts.

1.2.4 Naming problems

The most common naming scheme employed by a variety of conceptual models is the following: An object is associated with a *logical name* which supports logical reference to the object and identifies it externally[1]. The logical name of an individual is unique over the entire information base, whereas the logical name of a link is unique over the links emanating from the same object.

Naming schemes like the one above are not "natural" enough and several problems often arise: logical names can be ambiguous (e.g. homonymous objects), excessively long (especially in large information bases), unrelated to or unable to follow the changes of the environment of the named object [127, 126]. In natural language, similar problems are resolved by the context within which words are used.

1.3 Contextualization: a solution

A solution to these problems is offered by *contextualization*, which allows partitioning and packaging of information being added to an information base.

Adding contextualization to an information base has many advantages, including the following:

- *Modular representation*: A context suppresses details by allowing grouping or packaging of the descriptions added in an information base, as well as it allows access to the hidden detail.
- *Context-dependent semantics*: A given object may be represented and interpreted differently in different context delimited parts of the information base.
- *Focused information access*: A context delimits the parts of an information base that are accessible in a given way. Thus contexts can act as a focusing mechanism when searching for information.

[1] Internally, an object is associated with a *system-generated, globally unique identifier*, completely independent of any physical location (a *surrogate*[24]), which distinguishes it internally from all other objects.

- *Ability to handle inconsistent information*: Contradictory information can be represented in the same information base as long as it is treated in different contexts.

As a consequence, contextualization can resolve many of the problems described in the previous section. The expressive power of that mechanism makes it capable of supporting the development and effective use of large information bases in various applications. Examples of applications include thesaurus management systems, scientific catalogues, encyclopedias, multilingual applications, web-based applications, cooperative work, distributed and federated databases.

In conceptual modeling, a *context* is seen as a higher-level conceptual entity that groups or packages together other conceptual entities from a particular standpoint. Contexts allow us to focus on the objects of interest, as well as to name each of these objects using one or more convenient names [85, 121].

It is important to note that the same object can belong to different contexts and may have different names in each context. Thus, to the extent that names convey meaning, this is one way of providing context-dependent interpretation of objects. As we shall see, there are more ways in which the notion of context supports relative interpretation.

A context can be created on the basis of one or more criteria, such as:

- *temporal*, e.g. the map of Greece through the centuries, where each century constitutes a different context;
- *spatial*, e.g. the economy of different regions of Greece, where each region constitutes a different context;
- *functional*, e.g. the usage of a knife in different human activities, where each activity constitutes a different context;
- *structural*, e.g. the organization of a computer from a hardware or from a software point of view, where each point of view constitutes a different context; and so on.

The aim of this thesis is to establish a formal notion of context along with a general framework for maintaining and querying contexts.

In the following subsections, we first present two previous approaches to contextualization and their limitations. We then present two new approaches. The first approach extends the previous approaches by presenting a more general framework. However, this extended approach shares with the previous approaches the assumption that contexts are objects themselves. The shortcomings of this assumption are then explained, and a novel approach is presented, where contexts and objects are of different sorts. In addition, in this approach objects within a context are structured using the usual abstraction mechanisms.

1.3.1 Conventional approaches

In the area of knowledge representation, Mylopoulos and Motschnig-Pitrik [84] proposed a general mechanism for partitioning information bases using the concept of context. They introduced a generic framework for contexts and discussed naming conventions, operations on contexts, authorization, and transaction execution. However, they impose a strict constraint on

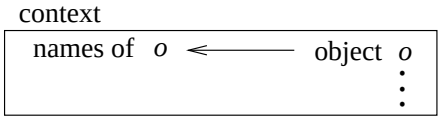
naming, whereby objects (called *information units*) are assigned unique names w.r.t. a context. Because of this constraint, several naming conflicts appear in operations among contexts, which the authors resolve in rather arbitrary ways. In addition, operations among contexts, such as *union* (called *addition*) and *intersection* (called *product*), are deprived of such useful properties as commutativity, associativity, and distributivity, and thus also can yield unexpected results. In [84], the major problem of the context union and context intersection operations is that it is possible for an object in the output context to have *no* name, even though it originally had one or more names. This can happen if an object of one input context has a name in common with an object of the other input context. For example, consider two contexts c and c' which correspond to two companies, the contents of c and c' being the employees of these two companies, respectively. Assume now that an employee in the first company has the same name with another employee in the second company. Then, the union of the contexts c and c' contains these two employees, but one of them will have no name. Such results might seriously hinder the applicability of this otherwise appealing framework.

In [126, 127], Theodorakis and Constantopoulos proposed a naming mechanism based on the concept of context, in order to resolve several naming problems that arise in information bases, such as object names being ambiguous, excessively long, or unable to follow the changes of the environment of the object. However, that approach imposes a hierarchical structure on contexts, i.e., a context may be contained in only one other context, which is rather restrictive.

1.3.2 Combining and extending conventional contextualization approaches

In this thesis, we try to combine the advantages of these previous two approaches and overcome their limitations by introducing a more general and more complete framework for context [125, 121, 120].

In particular, like in [84], a context is treated as a special object which is associated to a set of objects and a lexicon, i.e., a binding of names to these objects. However, in our model, an object is allowed to have more than one names, even in the same context, as shown in the following diagram:



For example, in the context of a research group, the object o can be a researcher with his social name (e.g. “John”) and his nickname within the group (e.g. “The_Hacker”) as two of its names.

This offers more flexibility and expressiveness and can handle the naming of real world entities in a more “natural” way, as it is possible for two objects to have the same name, even in the same frame of reference. This common name assignment may occur either accidentally, or by virtue of a common characteristic of the two objects (expressed through the common name).

In our model, naming conflicts that may appear during operations on contexts are resolved through a sophisticated, yet intuitive naming mechanism. Specifically, the following situations can be handled: (i) *Synonyms*: different names that have been assigned to the same object w.r.t. the same or different contexts; (ii) *Homonyms*: different objects that have the same name w.r.t. the same or different contexts; and (iii) *Anonyms*: objects with no name w.r.t. a context. An object is externally identified w.r.t. a context by using (simple) object names w.r.t. that context,

or composite names that are formed by taking into account the nesting of contexts. Object names and composite names w.r.t. a context form *name paths* w.r.t. that context. We distinguish an important class of contexts, called *well-defined*. Every object contained in a well-defined context possesses a unique name path w.r.t. that context.

The present model offers a set of operations for manipulating contexts. These operations provide support for creating, updating, combining, and comparing contexts. The most involved of the operations are those for combining and comparing contexts, namely context union, context intersection, and context difference. We prove that the class of well-defined contexts enjoys a closure property: the union, intersection, or difference of two well-defined contexts yields a well-defined context. Name ambiguities are resolved by adding to the resulting context, views of the objects as seen from the input contexts. Besides being used for name disambiguation, these views carry useful information about the input context. Finally, it should be mentioned that our context union and context intersection operations are commutative, associative, and distributive, with the benefits that these properties usually carry.

1.3.3 Contextualization as an independent abstraction mechanism for conceptual modeling: A novel approach to contextualization

Having extended conventional approaches, we have increased the expressive power and flexibility of contextualization, but there are still some useful capabilities that are lacking:

1. One-to-one correspondence between real world objects and (model) objects.

   Contexts have been defined to be objects themselves. This implies that two different contexts (i.e., objects) are needed in order to represent two different perceptions of the same real world object. For example, suppose that a person `A` wants to describe the environment of a university as a context containing information about the university, as viewed from his / her point of view (call this context c). Suppose, now, that another person `B` wants to describe the *same* university as a context also containing information about the university, but viewed from a different point of view (call this context c'). Certainly, context c contains different information than context c', i.e., they are two different contexts, and thus objects, although they refer to the same real world object (the same university). So, it is not at all clear whether two different contexts refer to the same real-world object or not. In other words, *there is not an one-to-one mapping from real-world objects to model objects, whereas there is an one-to-one mapping from different perceptions of real-world objects to model objects*. This can be confusing, and the user needs to keep the mapping between the real world object and its descriptions in the model, explicitly.

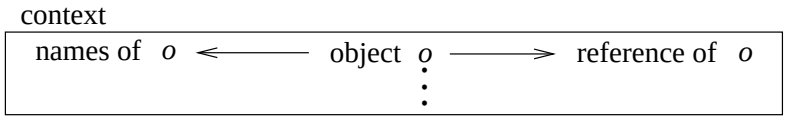
2. Structuring the contents of a context.

   The definition of context allows for a simple representation of objects, i.e., an object can have an associated set of names. However, the objects of a context can be related in more complex ways using the usual abstraction mechanisms of classification, generalization, and attribution. In the case that the context model supports the usual abstraction mechanisms, is there any relation between contextualization and those mechanisms?

To offer these capabilities, we enhance the notion of context in two ways [122, 123]:

1. We introduce references from objects to contexts.

The contents of a context is a set of objects, each of which is associated with a set of names (as before), with the following *new* feature: We allow each object of a context to be associated with another context that we call its *reference*. Thus, each object of a context is now associated with a set of names on one hand and (possibly) with a reference on the other, as in the following diagram:



Roughly speaking, the reference of the object points to information available about the object. In this extended model, real-world objects are represented by model objects in an one-to-one correspondence. A context is a collection of objects that supports encapsulation, and can be seen as a special kind of objects. Multiple representations of the same real world object are supported through the object reference. Thus, the university of the previous example would be represented as a model object o, whereas the reference of o in context of person `A` would be the context c. On the other hand, the reference of object o in the context of person `B` would be the context c'.

2. We allow the objects of a context to be structured through the traditional abstraction mechanisms of classification, generalization and attribution[2]. We study how these three abstraction mechanisms interact with contextualization, in particular how instance-of, ISA, and attribute links between objects affect the definition of their references.

So, *a context is a structured set of objects, in which each object is associated with a set of names and (possibly) a reference.*

1.3.4 Querying contextualized information bases

Contextualized information bases need a special treatment in order to answer queries. Thus, we propose a general framework for querying information bases which supports contextualization [124]. In particular, we focus on the following issues: (i) accessing information in a context structure using paths of names or paths of references, (ii) retrieval of contextualized information by defining useful fundamental query operations on contexts such as select, project, generate (which allows the reorganization of contexts structure), and path select.

In addition to the fundamental operations there are several other derived operations, such as *context union*, *context intersection*, and *context difference*. We extend the functionality of querying by allowing traversal of the context hierarchy using complex path expressions. Finally, we illustrate the usefulness of our contextualization mechanism by presenting higher level query operations that enable users to explore a contextualized information base. These higher level operations include focusing on a context of interest, searching the context structure for specific information, and making cross references of a concept from one context to another in order to obtain alternative representations of that object.

[2]By "attribution" we mean the assignment of an intrinsic attribute to an object as well as the declaration of its (binary) relationships to other objects. The abstraction mechanism of *aggregation* is a limited form of attribution [52].

1.4 Outline of the dissertation

The rest of the dissertation is organized as follows:

In Chapter 2, we motivate the use of contextualization as an abstraction mechanism for supporting the development and effective use of large information bases in various applications.

In Chapter 3, we briefly review the notion of context in several areas including, cognitive psychology, linguistics, and computer science.

In Chapter 4, we combine and extend conventional approaches on contextualization focusing on issues such as naming, operations on contexts, and properties of the operations on contexts.

In Chapter 5, we discuss in detail an example of using context in a cooperative environment.

In Chapter 6, we present a novel notion of context by (i) separating contexts from objects and introducing the reference of an object, and (ii) allowing the contents of a context to be structures with the traditional abstraction mechanisms, and studying the ways in which these mechanisms interact with the contextualization.

In Chapter 7, we present a theory for contextualized information bases.

In Chapter 8, we present a query language for contextualized information bases. The query language can be used for navigating through contexts, as well as for retrieving information of interest.

In Chapter 9, we demonstrate how our model addresses and provides uniform support of several features of applications of contextualization, such as views, workspaces, versions, thesauri fields and facets.

Finally, in Chapter 10, we make some suggestions for further research and concluding remarks.

Comparison of our work with related works is given in the chapters which present the main lines of our mechanism; these are Chapters 4, 6, and 8.

Chapter 2

Motivation

> *"Once upon a time, there was a little village, lost in the desert. All its inhabitants were blind. A great king passed by, followed by his army. He was riding an enormous elephant. The blind people herald of it. They had heard a great deal about elephants and were moved by a great desire to touch this fabulous animal, to get an idea of what it was. About ten of them, let's say the notables, set out. They begged the king for permission to touch the elephant. —"I give you permission, touch it!" said the king. One of them touched its trunk, another its foot, another its flanks, one was raised up so that to feel its ears, another seated on its back and given a ride. The blind men went back enchanted to their village. All the other blind people crowded round them, asking them greedily what sort of thing this fantastic beast, the elephant, was. The first said: "It is a big pipe that raises itself mightily, curls, and woe to you if it catches you!" Another said: "It is a hairy pillar." Another: "It is a wall, like a fortress, and it, too, is hairy." Another, the one who had felt the ear: "It's not a wall at all; it's a carpet of thick wool coarsely worked, which moves when you touch it." And the last cried: "What's that nonsense you 're telling? It's an enormous walking mountain." " [58]*
>
> **Nikos Kazantzakis**

One thing in which people differ is beliefs about concepts. A characteristic example is given in the above piece of text written by Nikos Kazantzakis [58]. In the above story, five people were asked to define the concept of an "elephant". Each of them has observed an elephant from a different point of view, and, based on his/her experience, he/she has given a description of the elephant. All five descriptions are subsequently different. Observing these descriptions we can make several remarks: People agree on some (e.g. "*it is hairy*"), but not all aspects of knowledge about the elephant. There are cases where knowledge is complementary (e.g. "*It is a wall, like a fortress, and it, too, is hairy*"), and others contradictory (e.g. "*It's not a wall at all*"). There are also cases where the description of one person does not make any sense to the others (e.g. "*What's that nonsense you 're telling?*"). That is, some descriptions are meaningful only in the context of the individual observer but possibly not meaningful at all to the rest.

All these five descriptions have been given under the assumption that the observers were blind and they had never seen, heard, or met an elephant before. In the context of a non-blind person who has seen an elephant either in TV, or in a zoo, or even in the jungle, all these five definitions does not make any sense. Eleven such persons have been asked to define the concept of an "elephant", and the answers are shown in Table 2.1. These definition are

PERSON	DEFINITION OF THE CONCEPT ELEPHANT
PERSON 1	- Elephant is a mammal animal. - It's one of the largest living animals on the earth. - Most elephants can be found in Africa and Southeast Asia. - Hunting elephants is prohibited in world. - Physical characteristics of this animal is different than any other.
PERSON 2	- Largest living land animal; with four legs, gray skin, large floppy ears and trunk; highly intelligent and docile; herbivore; lives in Africa or Asia.
PERSON 3	- A large mammal, weighs several tons. - Land-dwelling, 4-legged, nearly hairless. - Characterized by long snout + large ears. - Lives in herds, long childhood. - Eats plants, likes. - Lives in tropical areas, native to Africa & Asia (some parts). - Often found in zoos + circuses. - Strong, may be trained to help man. - Sensitive to sun (likes to cover itself in mud).
PERSON 4	- has big ears - huge body - four legs - long nose - gray color - pick up food with nose - spray water or sand by nose - small eyes compare with his body - has two long teeth that is bound to front - eats grasses ?
PERSON 5	- Life developed created from elephant DNA.
PERSON 6	A mammal that is located in either Africa or Asia, a large beast that roans through the jungle on the Savahana. They are vegetarians. The elephant usually has a tusk + a long snout that is used to grab thing to place in mouth. Usually they travel in herbs. They are use as work animals in Asia. The bull is for the mail that protect the herd.
PERSON 7	- large land-based mammal found primarily in the Southern Hemisphere - has a long, trunk as a nose which acts as a fifth hand - migratory living habits - stands 10-15 ft. tall - weighs from 2-4 tons. - pulls up the big tent at the circus ! - eats peanuts - great memory - big feet
PERSON 8	Large mammal which can be found in Africa + Asia Size Weight Color Origin Class Large heavy gray or brown Asia, Africa mammal
PERSON 9	An elephant is an animal belongs to the group of mammals. It is quite large. There are 2 types of elephants - Asian and African. Elephants have enlongated nose (trunk) which is used both for bathing and feeding itself.
PERSON 10	- Mammal - 4 legs - Very large size - gray in color - long nose (trunk) - Indigenous to Africa - Ivory tusks - Large ears - Intelligent (trainable)
PERSON 11	A large mammal, with 4 legs, tusks, a trunk, thick skin, generally gray in color, found in Africa and Asia, and some circus'. Eats grass and leaves. Travel in packs.

Table 2.1: Eleven different descriptions of the concept of "elephant", written by eleven adults (taken by [53]).

totally different from the five above. I could say that no comparison make sense, since the underlying assumptions are totally different (some observers are blind and some others are not). However, comparing the last eleven definitions we can make similar remarks as those for the first five definitions, i.e., knowledge is overlapping, complementary, contradictory, and sometimes nonsense w.r.t. a new observer. Thus, we can say that each of these definitions are meaningful only within the context of the individual who has given the definition.

In general, it is often the case that different persons have different opinions or perceptions of the reality. This is because different people have different background, experiences, way of thinking, and interests. This is a part of our life.

Turning now from reality to knowledge representation, we see a close relationship between describing things in reality (e.g. the descriptions of an elephant given above), and describing things in an information base. In any modeling process, there are often differences of perception or opinion among those gathering, providing, or using information. If we want to keep track of agreements or disagreements of opinions, information models should be expressive and flexible enough to support situations like this. Unfortunately, most of the existing information models lack of this capability.

One way to provide this capability is through context mechanisms, i.e., mechanisms for partitioning or grouping the knowledge into parts (which can also constitute reference environments), and give descriptions with respect to those parts, or so called *contexts*.

In this chapter we study the notion of context and examine its importance in information modeling. The first question we try to answer is "*What is actually a context?*". Then, we examine the importance of context in information modeling by its uses. We conclude this chapter by listing the advantages of using the notion of context in information modeling.

2.1 What is a context?

Currently, there is no unique definition about what is a context. Different approaches use their own assumptions to define a context and use it for different purposes. Some examples of context definitions follow:

- According to english dictionaries, the term *context* has two primary meanings:
  - the parts of a discourse that surround a language unit and help to determine its interpretation (can throw light on its meaning), and
  - the interrelated conditions in which something exists or occurs.
- In conceptual modeling:
  - "... it often appears useful to represent the situations themselves within the model. This is the case if several entities are to be viewed from the same situation. The latter then can be interpreted as a higher-order conceptual entity, namely a frame (space) enclosing the conceptual entities viewed. In accordance with [97], we refer to the denotation of a situation in a model as a *context* and represent it as a special symbolic structure. ... A context typically is a meaningful slice of the model or, in a special case, represent the global context" [80].

– "Many practical, real-life applications of concept learning are impossible to address without taking into consideration the background of the concept, its frame of reference, and the practical situation and circumstances of its occurrence, shortly its *context*" [71].

- In logic, "The *context* of a set of logical sentences (a theory) is defined as being: those assumptions of the environment, typically not feasibly axiomatisable and hence formalized as an abstract object, relative to which the language and the sentences in that language are stated for the theory" [103].

For the purposes of this thesis, in the context of information modeling, a *context* is viewed as a reference environment relatively to which descriptions of real world objects are given. The notion of context may be used to represent real world partitions, divisions, or in general, groups of information, such as situations, viewpoints, workspaces, or versions. The difference between a real world situation and a context is the following. A situation records the state of the world as it is, independently of how it is represented in the mind of an agent. A situation is complete as it records all the state of the world. Whereas, contexts are partial as they represent situations and hence capture different perspectives or record different levels of detail of a particular situation.

However, despite the lack of a uniform treatment of context in the literature, we can examine its importance by its uses. We therefore begin by listing some of the different kinds of uses of context and continue by arguing on the benefits of using context in information modeling.

2.2 Use of context

To get a feeling of what kind of features a context mechanism should have, consider some uses of it:

1. **Modularity.**

   The notion of context has been used for suppression of (irrelevant) details by allowing grouping or packaging of descriptions added in an information base. This grouping has also been used for emphasizing the essential information while hiding some inessential details.

   The notion of modularity has been used to represent real-world situations or partitions. Each such representation constitutes a different context. These representations are never complete (as we stated in the previous subsection); they are partial along some dimensions (such as time and space), which are dependent on the use of them. Different kinds of partitions that are used in modeling are described below (they are not meant to be mutually exclusive; indeed, some of them are specializations of others).

   (a) *Temporal.*

   Partitions can be partial along the temporal dimension (retaining temporal information). For example, the map of Greece through the centuries, the employees of a company the year 1997, or the painter lived in the period of Baroque, where each century, the year 1997, and the period of Baroque constitute a different context.

(b) *Spatial.*

Partitions can be partially represented along the spatial dimension. For example, the economy of different regions of Greece, where each region constitutes a different context.

(c) *Functional.*

People or concepts may play different roles in different contexts. For example, someone may be professor of a university or a general manager of a company, where each role (professor, manager) constitutes a different context. Another example is the usage of a knife in different human activities, where each activity constitutes a different context.

(d) *Physical.*

Often we need to view a real world object as a physical object, e.g. consisting of physical parts, having an appearance. Each of these types of physical characteristics constitutes a context. Such context may be the structure, appearance, or texture of a physical object.

(e) *Personal Beliefs.*

People often agree in some things and disagree in others. Personal spaces or environments that locate individual beliefs constitute different contexts.

(f) *Theory.*

Contexts can be used for representing a theory or a topic of a theory, such as a theory of mathematics, a theory of weather in winter, an architecture, or a style [46].

(g) *Tailored representations.*

Representation of situations tailored for specific applications or uses. For example, different demonstrations of a painting collection addressed to different user groups, such as elementary school students, or painting experts, where each type of presentation constitutes a context.

(h) *Viewpoints.*

How things are may depend on one's perspective on them. In software engineering [63, 28, 36], the construction of a complex model involves many agents. These agents have different views of the system they are trying to describe. These views are partial or incomplete descriptions which arise because of different responsibilities or roles assigned to the agents. The combination of an agent and its view that the agent holds form a *viewpoint* [33], where each viewpoint constitutes a context.

(i) *View schemas.*

A view schema in an object-oriented database [98, 2, 81], or in a relational database [44, 7] constitutes a context, whereas the view itself constitutes the contents of the context.

(j) *Multiversion objects.*

A multiversion object refers to a set of versions of a generic object [20, 56]. Therefore, a multiversion object can be seen as a context containing these versions.

(k) *Configurations.*

A configuration is the binding between a version of a composite object and the particular versions of its components [56]. Therefore, a configuration of a composite object can be seen as a context containing the particular versions of its components.

(l) *Workspaces.*

A workspace refers to a virtual space in which objects are designed and manipulated under the responsibility of an individual designer, or a group of designers [56]. A workspace constitutes a context that contains the designing objects.

(m) *Information bases in multidatabases.*

In a multidatabase environment, each information base constitutes a context.

(n) *Explicating environment.*

The knowledge sharing community has recognized the need for explicating context when transferring information from one agent to another [100]. The problem is to find the necessary environment within which a piece of information preserves its own meaning when exchanging it among different agents. The environment for exchanging meaningful information constitutes a context.

2. **Relative semantics/Multiple representations**.

A real-world object can be described differently with respect to different point of views. For example, a particular person might be represented differently as a student of a University, as an employee of a company, as a member of a diver's club, or as a physical entity consisting of physical parts.

3. **Granularity and accuracy**.

It is often the case that different descriptions of the same object make a different trade-off between *efficiency*, *simplicity*, and *accuracy*. A large system typically comprise a large number of theories that capture different perspectives, levels of detail, or are tailored to particular tasks. Contexts allow to represent the same object or fact with different degree of granularity and accuracy. For example, representations of paintings might be given in different degree of granularity and accuracy, if they are addressed to different user groups such as painting experts, curators, or kids.

4. **Focus attention/Localizing reasoning**.

Intelligent reasoning in the face of large amounts of information is based on establishing an appropriate context for reasoning (and reason within the context scope rather than the entire information base), and then being able to move between related contexts, preserving as much information deduced in other context(s) as possible.

5. **Find, combine, and compare different representations**.

The user must be able to find different partial representations of an object or a situation, as well as the contexts they are located. Partial representations of an entity / situation may be either combined to obtain a wider view of the entity / situation, or compared to obtain their commonalities and differences.

6. **Exception Handling**.

Exceptions usually appear and handled within special situations, i.e., specific contexts (e.g. exception in SGML, exceptions in representing and manipulating SGML semi-structured documents [22]).

7. **Handling Inconsistency**.

   Inconsistency in an information base, when viewed purely logically, seems undesirable. Indeed the traditional approach to dealing with inconsistency in data is to employ means to restore consistency immediately. However, it is important to study the larger environment containing such information bases, and the circumstances surrounding the inconsistency. It is argued that within the larger environment, an inconsistency can be desirable, and useful, if we know appropriate actions to handle it [38, 37, 32]. In some cases we may wish to remove the inconsistency, and in other cases we may wish to keep it. In the last case, inconsistent or contradictory information can be handled as long as it is treated in different contexts.

8. **Advanced naming/Resolving naming ambiguities**.

   Context contributes to naming by providing the necessary environment for resolving naming ambiguities such as homonyms (the same name may represent different entities in different contexts) and synonyms (different names may refer the same entity in different contexts). For example, the name "Holms" refers to the detective Holmes in the context of Sherlock Holmes stories; it refers to judge Holmes in the context of the US legal history; and perhaps does not refer to anybody in the context of European paintings.

   Finally, context mechanisms provide us with the ability to using finite and frugal vocabularies throughout an information base (as naming is relative to contexts and thus excessively long names are avoided). In addition, names are consistent with and follow the changes of the environment of the named object [127, 126].

9. **Multilinguality**.

   The language used for describing knowledge about objects is chosen to fit the intended domain. For example, the word "fennel" do not make any sense in the context of greek language, whereas the word "μαραθος" it does.

10. **Background knowledge**.

    Contexts provide us with the ability to describe the set of assumptions underlying a system of a specific task. Examples of background knowledge are: socioeconomic background, the goals of the conversants, the status of the people who provide or describe information (e.g. they are blind), etc.

2.3 Benefits of using context

The notion of context is important since it can capture many of the interesting aspects of the way we understand the world, such as relativity, locality, partiality, and context-dependence. Introducing the notion of context in computer science in general, and in information modeling in particular, contributes to the efficient handling of information. The advantages of the notion of context in information modeling and knowledge-based systems are the following [107]:

- *Economy of representation*: Different contexts can limit the parts of the information base that are accessible, effectively allowing the representation of multiple information bases in a single structure.

- *Economy of reasoning*: Contexts can act as a focusing mechanism when accessing information in a information base. Focusing on the information of interest may permit efficient reasoning since reasoning is limited in the scope of a context rather than the entire information base.

- *Allowing inconsistent information*: contradictory information can be accommodated in the same information base as long as it is viewed from different contexts.

- *Flexible semantics*: Contexts may dictate not only the accessible information base fragments, but also their semantics. The same object may be described and interpreted differently in different contexts. For example, the word “glass” may represent different real world entities in the context of “eye sight” and in the context of “wine”. Contexts may help to resolve lexical ambiguities (e.g. homonyms and synonyms) which appear in several information bases.

Chapter 3

Review of the Notion of Context

In this chapter, we review the notion of context as it appears in the literature of several areas (e.g. cognitive psychology, linguistics, and computer science), progressively focusing on aspects which are relevant to this dissertation.

In computer science, a number of formal or informal definitions of some notion of context have appeared in several areas, such as artificial intelligence [51, 73, 46], software development [92, 45, 99, 106, 108, 56, 63], databases [7, 44, 49, 2, 27, 55, 86], machine learning [76, 130, 71], and knowledge representation [84, 122, 126, 121, 135, 31, 116, 26, 18].

However, all these notions of context are very diverse and serve different purposes. In software development the notion of context appears in the form of views [7, 44, 49, 98, 2], aspects [92], and roles [45, 99], for dealing with data from different perspectives, or even in the form of workspaces which are used to support cooperative work [56]. In machine learning, context is treated as environmental information for concept classification [76, 130, 71]. In the so called multiple databases, context appears as a collection of meta-attributes for capturing class semantics [55]. In artificial intelligence, the notion of context appears as a means of partitioning knowledge into manageable sets [51], or as a logical construct that facilitates reasoning activities [73, 46]. In particular, we focus on the area of knowledge representation, where the notion of context appears as an abstraction mechanism for partitioning an information base into possibly overlapping parts (e.g. [84, 121, 126, 135, 31, 116, 26, 18]).

3.1 Linguistics

The term ‘context’ is used in many different ways in the linguistic literature, referring for example to the preceding discourse or text, to the physical environment, or to the domain of discourse.

In the Webster Dictionary, the term ‘context’ is defined as:

1. The parts of a discourse that surround a word or passage and can throw light on its meaning, and
2. The interrelated conditions in which something exists or occurs.

The first notion of context defined above deals with things such as syntactic construction involved in a utterance, structure between multi-sentence segments of text and it is known

in the literature as *linguistic context*. The second notion of context defined above deals with things such as relevant knowledge, speakers' (writers') and hearers' (readers') beliefs, goals, intentions, attitudes, common (shared) knowledge, and sensory information (vision, smell, taste, touch). This notion of context is also known as *non-linguistic context*.

The notion of context plays an important role in language comprehension. People naturally use contextual information in order to convey a specific meaning. Langacker claimed that all linguistic units are context-dependent [66]. People frequently employ the same form to convey more than one meanings in everyday speech. This is the case of *homonyms*, which are either *homographs* (e.g. 'bat' the mammal or 'bat' the athletic implement) or *polysemous* words (e.g. the word 'line' in 'a line of code', 'a line on the blackboard', or 'a line in the bank'). On the other hand, different forms are often used to convey the same meaning. This is the case of *synonyms* (e.g. 'cheap' and 'inexpensive'). These ambiguities are resolved by the context within which words are used. For example, the same meaning can be expressed by different words in the dialects of different cultural or professional groups [57]. Conversely, the same word can be used by different groups to convey different meanings, or even within one group to convey different meanings in different situations.

In artificial languages, the opposite of the previous assumption, that is, one meaning can have only one linguistic representation, is also true. This clearly requires using absolute or global "names". An interesting question in artificial language design is how to extend a name by suffixing or prefixing constructs in order to make it unique in a given context or, given a name, how to restrict the context so that this name may convey a unique meaning.

What is common to the various uses of the term 'context' in the linguistic literature is that they all refer to *factors, relevant to the understanding of communicative behavior* [14, 68, 69]. According to these factors, context is distinguished into five categories [14]:

1. *Linguistic context*: the surrounding linguistic material (textual or spoken). Closely related to what is sometimes called *dialogue history*.

2. *Semantic context*: the underlying goals that the participant of a dialogue want to be achieved (such as reaching a decision or setting up a meeting).

3. *Physical context*: the physical circumstances in which an interaction takes place (such as place or time).

4. *Social context*: communicative rights, obligations and constraints of each participant. Examples of social context include the type of institutional setting in which a dialogue occurs, or the type of communicative event that a dialogue represents (such as information-seeking dialogue or doctor-patient interview).

5. *Cognitive context*: the participants' beliefs, intentions, plans and other attitudes.

Moreover, for each of these 'dimensions' of context the author of [14] distinguishes between *global aspects*, which are fixed at the beginning of a dialogue and remain constant throughout, and *local aspects*, whose value develop and change through the dialogue. Global aspects are also known as *background knowledge*.

3.2 Software development

Within software development, context has appeared as *viewpoints* in software engineering during the specifications of a system. Viewpoints are defined as the combination of an agent and the view that the agent holds for the system and allow to collect system information from multiple perspectives [63, 28, 36].

In software engineering and CAD applications the notion of context appears as *versions* (a significant, semantically meaningful snapshot of an object at a point in time), *workspaces* (named repositories for supporting the sharing of objects among multiple designers), and *configurations* (compatible system components) [56].

In programming languages, the parts of the program which are visible to a particular program segment are determined by the *scopes* and *scope rules* using *scope resolution operators*. Recently, in an object-oriented framework, the high-level knowledge base programming language **K** defines context as a "sub-knowledge base" by specifying an intentional association pattern among classes [108]. In addition to this framework *aspects* [92], *roles* [45, 99, 136, 113, 5, 89] and *conceptual slices* (views) [106] have been introduced to support multiple state and multiple behavior of the same objects.

3.3 Machine learning

Many practical applications in concept learning necessitate the use of context in learning and particularly in classification of concepts, and the performance task alike. The context of a concept is referred to as the background of the concept, its frame of reference, and the particular situation and circumstances of its occurrence [71].

Michalski [76, 77] was the first to call attention to context dependency and proposed the method of two-tiered characterization of concepts: the *base core description* describes the basic properties of the concept, whereas the *inferential concept recognition* scheme is suggested to cover context-dependent aspects of the concept.

Other approaches [130] provide definition of contextual attitude and systematize various strategies for utilizing the information about context in the process of learning. In their experiments, they demonstrate that the use of context can result in substantially more accurate classification of concepts.

3.4 Networks

Naming and addressing issues appear in networks, in the sense that the address space is divided into domains, subdomains, and so on, and domains never overlap and subdomains are always strictly nested within domains yielding a tree of domains and subdomains [48, 95, 129]. Domains and subdomains are considered as contexts which decompose the address space into non-overlapping parts. Names are given with respect to a specific domain or subdomain. Similar organizational hierarchies of contexts appear in UNIX file systems and WWW addresses.

3.5 Context in the Web

The World Wide Web is a large collection of heterogenous documents. Web pages are unlike typical documents in traditional databases. Pages can be active (e.g., animations, Java), can be automatically generated in real-time (e.g., current stock prices), and may contain multimedia (e.g., video, sound). The authors of Web pages have very diverse backgrounds, knowledge, cultures, and aims. Furthermore, the availability of metadata is inconsistent (for example, some authors use the HTML heading tags to denote headings and subheadings in their text, while others use different methods, such as the HTML font tags or images).

Web search engines generally treat search requests in isolation. The results for a given query are identical, independent of the user, or the context in which the user made the request. Next›generation search engines will make increasing use of context information, either by using explicit or implicit context information from users, or by implementing additional functionality within restricted contexts. Greater use of context in web may help increase competition and diversity on the web [67, 42, 88].

There are search engines (e.g. Inquirus) that are said they support context information, but what they actually do is to provide context information to the query by adding more keywords.

In [88], the notion of context is similar to that of *Generic Concept*. A Generic Concept is a form of logical object (a kind of a *contextualized abstract view* over the contents of large semantically related document collections) whose purpose is to cross-relate, collate, and summurize the meta-data descriptions of semantically related network-accessible data.

3.6 Artificial intelligence

In artificial intelligence contexts have been introduced as means of partitioning knowledge into manageable sets [51], or they have been considered as logical constructs that facilitate reasoning activities [73, 46]. A precursory idea of context can be traced back to Peirce's *existential graphs* [93]. Existential Graphs use a logical form of context called a *cut* which shows in a topological manner the scope of a negative context on a sheet of paper (the *sheet of assertion*). Sowa [111, 112] introduced *conceptual graphs* as an extension of the existential graphs and defined *contexts* as concepts whose referent contains one or more conceptual graphs (Sowa's *situations*). Hendrix [50, 51] expanded semantic networks (based on existential graphs) through *partitioning* contexts. Unlike Peirce, Hendrix allowed overlapping contexts.

Partitions and partition inheritance are supported in SB-ONE knowledge representation workbench [62]. Using SB-ONE, knowledge can be assigned to partitions. These partitions may be completely independent each other, or they may be ordered in inheritance hierarchies.

Lately, contexts have been proposed as an important means for formalizing reasoning. Contexts first appeared in declarative artificial intelligence as a possible solution to the problem of generality [72]. Most of the architectures that have been proposed for symbolic artificial intelligence systems assumes that there is a knowledge base consisting of a set of expressions (sentences) which convey some truth about the domain. The contextual effects on an expression are often so rich that they can not be captured completely in the logic. Thus, contexts have been defined to be rich objects [75] in that a context can not be completely described and can be thought of as the reification of context dependencies of the sentences associated with the context. Contexts are treated as *formal objects* and have been made first class objects in

first-order logic so that they can serve as a frame of reference for sentences (which are relative to some context) and that it is possible to formulate rules which relate one context to another [73, 46, 74]. The modality $ist(c, p)$ (pronounced as "is true") has been used to assert that the proposition p is true in the context c.

Since then, contexts have found uses in various artificial intelligence applications, including: managing large knowledge bases [46], translating knowledge [15], modeling knowledge and belief [41], integrating heterogeneous databases [30], planning [16], common sense reasoning [74].

Coherently with the notion of context described above, Attardi [6] uses viewpoints to represent the notion of relativized truth such as beliefs, situations and knowledge. Viewpoints denote sets of sentences which represent the assumptions of a theory.

3.7 Semantic model clustering

In "real life" applications, it is often the case that semantic data models become large and complex, and thus difficult to understand. Several techniques cope with this problem by decomposing the global schema into smaller, more manageable partitions, called *entity clusters* [31, 116, 26, 128, 18, 25, 39, 116, 135, 96, 133].

In [116], several kinds of clustering are defined, all of which are supported by the framework proposed in this thesis (in particular see Chapter 4):

1. *Dominance grouping*: An object o is grouped together with its related objects into a cluster that represents the same real world entity as o, but at a different level of abstraction. In our framework, we support this kind of grouping by allowing the reference of an object to contain the object itself.

2. *Abstraction grouping*: Objects participating in abstractions such as classification, generalization, and attribution are grouped in a cluster. In our framework, we support this kind of grouping by allowing objects related by instance-of, ISA, and attribute links to be grouped together with these links in a context.

3. *Relationship grouping*: A relationship together with its participating entities are grouped into a cluster. In our framework, we support this kind of grouping, as relationships are represented by attributes and a context may contain any kind of object, i.e., individual or attribute.

In all approaches in the literature, dominance grouping is based on the following name convention: each cluster should have the same name as the object it represents. By contrast, in our framework, dominance grouping is based on object ids, allowing objects to have different names at different levels of abstraction.

In [26], an additional kind of clustering is defined, called *relationship abstraction*, which abstracts a number of relationships into a higher-level relationship. In our framework, we support this kind of clustering through the concept of reference.

Our framework differs substantially, from all of the above approaches in the following:

- A global schema is not a requirement for modeling the real world. Rather, it is possible that information about an object can only be found scattered across contexts.

- We support relative naming and relative semantics w.r.t. a context. Within different contexts, information about the same object can even be conflicting. Thus, information is meaningful only within a context, and its validity outside it cannot be directly assumed (unless explicitly declared).

- We distinguish between objects and contexts.

- We support the interaction of our contextualization mechanism with the traditional abstraction mechanisms.

Although in semantic model clustering the schema is decomposed, instances of both low-level objects and high-level objects (i.e., clusters) are globally defined. A solution to this problem is given in [26] by providing an algorithm to extract an instance graph for a higher-level object, i.e., to decompose the instances according to the cluster. In our framework, instance-of links are context-dependent and the user can explicitly declare the instances of high-level objects. In particular, in our framework, the instance graph of a high-level object corresponds to the instance-of links directed towards the object, along with the references of these links, and the instance-of links within these references, recursively. In [26], a higher-level object can be defined as a view derived by a query expression over a semantic model. We think that view support is an important issue and we intend to address it for our framework in future work.

3.8 Nested associations

Work in [65] deals with the problem of abstracting complex associations between objects of a conceptual model in order to make large data schemas more comprehensive. Towards this goal, the authors define an *enclosing class* as an abstraction which encapsulates a set of local classes. Additionally, they define an *enclosing association class* as an abstraction which associates a source enclosing class with a destination enclosing class, and encapsulates a set of local classes, as well as local associations.

Intuitively, our concepts of object reference and attribute reference cover the concepts of enclosing class and enclosing attribute class. However, in [65], the main emphasis is placed on nested associations, and issues such as relative naming and relative semantics, as well as the interaction between the proposed abstraction and the abstractions of classification and generalization are not considered.

In [39], a leveled entity-relationship model is proposed, where higher-level entities encapsulate lower-level entities, similarly to our concept of object reference. However, the authors do not support the notion of attribute reference. Moreover, naming, semantics, and instances of objects are globally defined. In [39], the authors argue that a relationship to a subentity from a higher-level entity breaks the encapsulation of the entity containing the subentity. To solve this encapsulation problem, they propose the notion of *aspect* that works as a window that makes a lower-level object to appear at a higher-level object. Though encapsulation is an important issue, we do not examine it in this work. We consider this as an authorization issue that can be handled on top of our general framework mechanism.

3.9 Context in categorization

Categorization is one of the basic mental processes in cognition. We, as human beings, can categorize various types of objects, events, and states of affairs, and our categorizations depend on the circumstance and perspective (i.e., how things are depend on one's point of view on them). Barwise and Sligman [8] use *natural regularities* to study the role of context in categorization. To be more specific consider the following example of regularity as it appears in [102]: 'swans are white', which express an intuitive sense that all swans are white. Although this is the general intuition about swans, there might be exceptions and we can find swans which are black (this can happen in Australia). Therefore, this sentence can be evaluated only in appropriate contexts, such as in Europe, outside zoos, and so forth. The appropriate context wouldn't be a problem if we could completely specify all contextual factors. However, in many cases it is impossible to state all the relevant contextual factors. In [8], a notion of context is captured through the notion of *perspective*. Different perspectives simply give rise to different ways of classifying things. For example, distances can be classified using either inches from Manos' perspective or centimeters from Panos' perspective.

3.10 Information bases and multidatabases

In multidatabase environments and heterogeneous information systems, database integration has to deal with naming conflicts of two types, *homonyms* and *synonyms*, because the global schema of the integrated database is usually generated by merging one or more user-oriented schemas [11, 9]. Some significant alternatives for representing context are the following:

1. *Semantic proximity proposal* [55]. Context is defined as a collection of meta-attributes for capturing the semantic similarity between objects.

2. *Context building approach* [86]. Context is defined as the knowledge that is needed to reason about another system, for the purpose of answering a query, and

3. *Context interchange approach* [100, 43]. Many separately-developed systems must ensure *semantic interoperability* to meaningfully exchange information. That is the individual systems must agree in the meaning of their exchange data. The key to the context interchange approach to achieving interoperability is the notion of *context*, which is used to refer to the (implicit) assumptions underlying the interpretation of data. Context is defined as the meaning, content, organization and properties of exchanging data and individual agents (i.e., sources or receivers) keep locally contextual information for their data. A shared ontology is used to map disparate local contexts to a set of common concepts that might appear in an application. To achieve the meaningful data exchange between a source and a receiver, a context mediator is used to convert (using the shared ontology) the data from the source context into the receiver context.

In traditional databases, *views* present a consistent partition of the database [44, 7]. Such mechanisms have been adopted in object-oriented databases [98, 2, 19, 94] (see [81] for a survey) and semantic data models [26]. Moreover, *contexts* have been proposed as a partitioning scheme of hypertext databases [27, 17, 40], and *perspectives* as a mechanism for organizing and manipulating groups of nodes and links in a hypertext network [91].

HAM [17] is a general purpose abstract machine that supports contexts. In HAM, a graph usually contains all the information regarding a general topic and contexts are used to partition the data within a graph. Therefore, a context may contain nodes, links, or other contexts. Contexts are organized hierarchically, i.e., a context is contained in only one other context. By contrast, in our model, a context may be contained in more than one contexts. Contexts in HAM have been used to support configurations, private workspaces, and version history trees [27]. HAM provides a set of context editing, context inquiry, and context attribute operations. All the context editing operations of HAM, namely *createContext*, *destroyContext*, *compactContext*, and *mergeContext*, can be simulated in our model using its operations. On the other hand, HAM does not support name relativism. Inquiries on and attributes of contexts can be supported by our model (see chapter 8), however they are outside of the scope of this chapter.

In [110], the notion of context is used to support collaborative work in hypermedia design. A context node contains links, terminal nodes, and other context nodes. Furthermore, context nodes are specialized into annotations, public bases, hyperbases, private bases, and user contexts. Using this notion of context, the authors define operations *check-in* and *check-out* for hypermedia objects. However, there is no support for name relativism, neither are generic operations on contexts provided.

The notion of context has also appeared in the area of heterogeneous databases [101, 86, 55]. There, the word "context" refers to the implicit assumptions underlying the manner in which an agent represents or interprets data. To allow exchange between heterogeneous information systems, information specific to them can be captured in specific contexts. Therefore, contexts are used for interpreting data. At present our model cannot be compared with these works, because it does not address heterogeneous databases, as we assume a single Information Base (which guarantees that real world objects are represented by unique objects in the Information Base).

Furthermore, a context is taken to define a view of the objects in a repository [10] and it is typically used to define a set of objects that an engineer is manipulating for a particular task. Recently, mechanisms for partitioning information bases with contexts have been proposed, which introduce a generic framework for contexts and deal with naming conventions, authorization, transaction execution and overlapping contexts [84, 85]. Other approaches employ context as a way to face the complexity of information base update [23, 131], or to develop a naming mechanism in information bases [126, 117, 119].

Specifically, in [84], Mylopoulos and Motschnig-Pitrik proposed a general mechanism for partitioning information bases using the concept of context. They introduced a generic framework for contexts and discussed naming conventions, operations on contexts, authorization, and transaction execution. However, they impose a strict constraint on naming, whereby objects (called *information units*) are assigned unique names w.r.t. a context. Because of this constraint, several naming conflicts appear in operations among contexts, which the authors resolve in rather arbitrary ways. In addition, operations among contexts, such as *union* (called *addition*) and *intersection* (called *product*), are deprived of such useful properties as commutativity, associativity, and distributivity, and thus also can yield unexpected results. In [84], the major problem of the context union and context intersection operations is that it is possible for an object in the output context to have *no* name, even though it originally had one or more names. This can happen if an object of one input context has a name in common with an object of the other input context. For example, consider two contexts c and c' which correspond to two companies, the contents of c and c' being the employees of these two companies, respectively.

Assume now that an employee in the first company has the same name with another employee in the second company. Then, the union of the contexts c and c' contains these two employees, but one of them will have no name. Such results might seriously hinder the applicability of this otherwise appealing framework.

In [126, 127], Theodorakis and Constantopoulos proposed a naming mechanism based on the concept of context, in order to resolve several naming problems that arise in information bases, such as object names being ambiguous, excessively long, or unable to follow the changes of the environment of the object. These naming problems also resolved in this dissertation. The information base is assumed to be structured with the traditional abstraction mechanisms of classification, generalization, and attribution. A context is identified by a node and a link class, called *pivot elements* of the context. A link class could be either an attribute class, or the instance-of relation, or the ISA relation. The contents of a context consist of objects and links which are associated with the pivot elements of the context. This definition of context allows the information base to be decomposed into partitions on the basis of one of the traditional abstraction mechanisms. Finally, relative naming is supported, as well as nesting of non contexts. However, that approach imposes a hierarchical structure on contexts, i.e., a context may be contained in only one other context, which is rather restrictive. Context defined in the dissertation is more general and can be easily embedded to any data model.

In this thesis, we try to combine the advantages of these previous two approaches and alleviate their shortcomings by introducing a more general and more complete framework for context. Our objective is to establish a formal notion of context to support the development and effective use of large information bases in various application areas, especially in distributed, cooperative environments.

The notion of context introduced in this work, supports nesting of contexts, context overlapping, and relative naming, yet it advances with respect to [84, 85, 126] mainly along the following lines:

- We distinguish between objects and contexts. Objects represent real world concepts, whereas contexts are collections of objects. Within each context, local names and semantics are assigned to objects, as well as references (which are also contexts) for describing objects in more detail. Thus, a real world concept (e.g. the geographical viewpoint of Greece) is represented by an object which can have different detailed descriptions (i.e., references) within different contexts (e.g. the 15th century and the 20th century). This is certainly a modeling capability not offered by the other approaches.

- Our formal notion of context is easy to understand and easy to embed in an existing data model. We show how contextualization can be embeded in a data model which supports the traditional abstraction mechanisms of classification, generalization, and attribution and how contextualization interacts with these abstraction mechanims. Works in [84, 85] lack this interaction. In [85], the notion of context is introduced in the Telos data model, where each context is considered to be at the Token level, i.e., an atomic object, and contexts do not participate in classification or generalization hierarchies, something which is supported by our contextualization mechanism.

- Our operations of context union, intersection, and difference are different from these of set theory as they take into account the notion of context, and keep track of the origin of data in the result. However, context union and intersection also satisfy the important

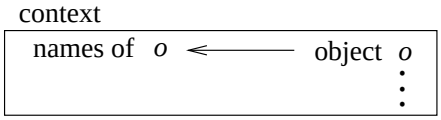
properties of commutativity, associativity, and distributivity, which offer flexibility in the execution of these operations. Operations on contexts defined in [84, 85] lack these abilities. Thus, information from the original contexts may get lost in the result of an operation as conflicts appear and their conflict resolution strategy may cause units to be inaccessible in the resulting context.

Chapter 4

Combining and Extending Conventional Contextualization Approaches

In this chapter, we develop a model for context, which alleviates the shortcomings of the above proposals. It also can serve as the basis for effectively employing context in large information bases in various applications, especially in distributed, cooperative environments. An example of such an application is given in Chapter 5. Subsequently, in Chapter 6, we introduce an extension to the notion of context that leads to establishing context as an important conceptual modeling mechanism. Work of this chapter published in [125, 121, 120].

Like in [84], we consider a context to be a special object which is associated with a set of objects and a lexicon i.e., a binding of names to these objects. However, in our model, an object is allowed to have more than one names, even in the same context, as shown in the following diagram:



For example, in the context of a research group, the object o can be a researcher with his social name (e.g. "John") and his nickname within the group (e.g. "The_Hacker") as two of its names.

This offers more flexibility and expressiveness and can handle the naming of real world entities in a more "natural" way, as it is possible for two objects to have the same name, even in the same frame of reference. This common name assignment may occur either accidentally, or by virtue of a common characteristic of the two objects (expressed through the common name). In our model, naming conflicts that may appear during operations on contexts are resolved through a sophisticated, yet intuitive naming mechanism. Specifically, the following situations can be handled:

- *Synonyms*: different names that have been assigned to the same object in the same or different contexts;

- *Homonyms*: different objects that have the same name in the same or different contexts; and

- *Anonyms*: objects with no name in a context.

An object is externally identified using *name paths* w.r.t. a context. These name paths are either the object names w.r.t. that context, or composite names that are formed by taking into account the nesting of contexts. We distinguish an important class of contexts, called *well-defined*. Every object contained in a well-defined context possesses a unique name path in that context.

The present model offers a set of operations for manipulating contexts. These operations provide support for creating, updating, combining, and comparing contexts. The most involved of the operations are those for combining and comparing contexts, namely context union, context intersection, and context difference. The above three operations are different from these of set theory as they take into account the notion of context. We prove that the class of well-defined contexts enjoys a closure property: the union, intersection, or difference of two well-defined contexts yields a well-defined context. Name ambiguities are resolved by adding to the resulting context views of the objects as seen from the input contexts. Besides being used for name disambiguation, these views carry useful information, as we demonstrate in the example of Chapter 5. Finally, the context union and context intersection operations defined here are commutative, associative, and distributive, with the benefits that these properties usually carry.

4.1 The notion of context

In information modeling, a *context* is a higher-level conceptual entity that describes a group of conceptual entities from a particular standpoint [80]. The conceptual entities described can be contexts themselves, thus allowing for nesting of contexts. Conceptual entities are named with respect to a context as part of their description.

Examples of contexts are:

- *Information bases*: An information base describes a set of conceptual entities from the point of view of its designer. Certainly, the designer's viewpoint is influenced by the particular needs of the targeted users.

- *View schemas*: A view schema in an object-oriented database [98, 2, 81], or in a relational database [44, 7] describes the conceptual entities in the view according to the person that defined that view.

- *Multiversion objects*: A multiversion object refers to a set of versions of a generic object [20, 56]. Therefore, a multiversion object can be seen as a context in which the particular versions are contained.

- *Configurations*: A configuration is the binding between a version of a composite object and the particular versions of its components [56]. Therefore, a configuration of a composite object can be seen as a context containing a particular set of versions of its components.

- *Workspaces*: A workspace refers to a virtual space in which objects are created and manipulated under the responsibility of an individual person, or a group of persons [56]. Therefore, a workspace can be seen as a context in which the objects are viewed according to the responsibilities of the persons involved.

An information base can be considered as a repository of objects. Objects represent atomic or collective real world entities, attributes, (binary) relationships, or primitive values. We denote by $\mathcal{O}$ the set of all objects.

Contexts are taken as a special kind of objects that represent real world reference environments such as partitions, viewpoints, situations, or workspaces. We shall call all objects which are not contexts, *simple objects*. Contexts allow us to focus on a set of objects of interest, as well as to name each of these objects using one or more convenient names. Informally, we think of a context as containing objects, each object being associated with a set of names.

Definition 4.1 Context.
Contexts are a special kind of objects which can be thought of as containing objects, each object being associated with a set of names. Let $\mathcal{CXT}$ be the set of all contexts. Then, $\mathcal{CXT} \subseteq \mathcal{O}$. $\diamond$

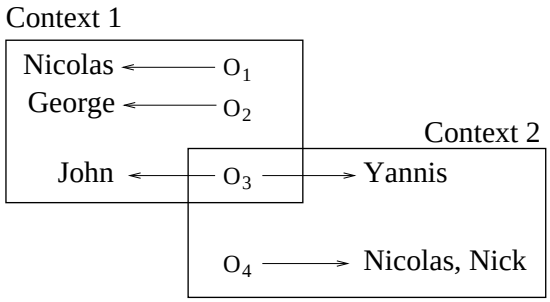


Figure 4.1: The notion of context.

For example, Figure 4.1 illustrates two contexts, `Context_1` and `Context_2`, which represent the environment of two companies. The employees of those companies are represented by objects o_1 to o_4. `Context_1` contains the objects o_1, o_2, and o_3, and associates them with names `Nicolas`, `George`, and `John`, respectively. `Context_2` contains the objects o_3 and o_4, and associates them with names `Yannis`, and `Nicolas` or `Nick`, respectively. The employee represented by object o_3 works for both companies and is called *John* in the first company, whereas *Yannis* in the second.

In order to treat contexts more formally we need the concept of lexicon, i.e., a binding of names to objects in which an object may have zero, one or more names.

Definition 4.2 Lexicon.
Let $\mathcal{N}$ be the set of all atomic names and $\mathcal{P}(\mathcal{N})$ the power set of $\mathcal{N}$. A *lexicon* is a mapping l of the form:

$$l_O : O \longrightarrow \mathcal{P}(\mathcal{N})$$

where O is a set of objects ($O \subseteq \mathcal{O}$). A lexicon associates each object in O with a set of names. The objects in O are called *objects of* the lexicon l_O and denoted by $objs(l_O)$. We denote by $\mathcal{LEX}$ the set of all lexicons. $\diamond$

Note that an object of a lexicon may be associated with an empty set of names.

We shall often think of a lexicon l as a set of pairs of the form $o : l(o)$. In other words, if $objs(l) = \{o_1, \ldots, o_k\}$ then we shall write $l = \{o_1 : l(o_1), \ldots, o_k : l(o_k)\}$.

The following is an example of a lexicon:

$$l_1 = \{o_1 : \texttt{Panos}, o_2 : \texttt{head}, o_3 : \texttt{Manos}, o_4 : \texttt{Nicolas}, \texttt{Nick}\}$$

where $objs(l_1) = \{o_1, o_2, o_3, o_4\}$. We depict a lexicon as follows:

$$l_1 = \begin{cases} o_1 : \texttt{Panos} \\ o_2 : \texttt{head} \\ o_3 : \texttt{Manos} \\ o_4 : \texttt{Nicolas}, \texttt{Nick} \end{cases}$$

As already mentioned, we think of a context as containing objects, each object being associated with a set of names. Formally, this is expressed by associating each context c with a lexicon The context c can be used to focus on the objects of the lexicon, as well as to assign relative names to these objects.

Definition 4.3 Context lexicon.

A *context lexicon* is a total function of the form:

$$lex : \mathcal{CXT} \longrightarrow \mathcal{LEX}$$

which associates a context with a lexicon, which we shall call the *lexicon of c*. For each context *c*, objects of $lex(c)$ are also called *objects of c*, and denoted by $objs(c)$. That is, $objs(c) = objs(lex(c))$. $\diamond$

Let c be a context with lexicon $\{o_1 : N_1, \ldots, o_k : N_k\}$. We shall use the following notation and terminology:

- The objects $o_1, \ldots, o_k$ are called the *objects* of c and their set is denoted by $objs(c)$.
- We shall say that c *contains* $o_1, \ldots, o_k$.
- The names in N_i are called the names of o_i in c, or the *c-names* of o_i. The set N_i will also be denoted by $names(o_i, c)$.

A similar notation and terminology is used for a lexicon as well.

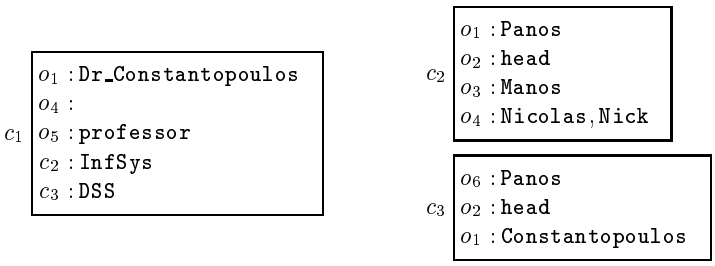


Figure 4.2: Example of contexts.

As an example, consider a context c_1 which represents an institute (see Figure 4.2). Context c_1 contains five objects in its lexicon, o_1, o_4, o_5, c_2, and c_3. Object o_1 is a simple object whose

c_1-name is `Dr_Constantopoulos`, and represents a specific person. Object o_4 is a simple object as well which represents an entity that is known to exist within the context c_1 but we do not know its name yet. Object o_5 represents the notion of professor (and not a particular person who happens to be a professor). Objects c_2 and c_3 are themselves contexts whose c_1-names are `InfSys` and `DSS`, respectively. Context c_2 represents the environment of the Information Systems Lab and describes the objects of that lab. Context c_3 represents the environment of the Decision Support Systems Lab and describes the objects of that lab. The objects contained in contexts c_2 and c_3 are as shown in Figure 4.2. Note that object o_1 has only one c_2-name (`Panos`), whereas object o_4 has two c_2-names (`Nicolas` and `Nick`). Also note that the same object can be contained in more than one context under the same or different names. For instance, object o_1 is contained in three contexts c_1, c_2, and c_3. The c_1-name of object o_1 is `Dr_Constantopoulos`, its c_2-name is `Panos`, whereas its c_3-name is `Constantopoulos`. Note also that two different objects, o_1 and o_6, have the same name in two different contexts (c_2 and c_3).

Recall that an object may represent real world attributes or binary relationships. We call these objects *link objects*. Link objects have a *source* and a *destination* object. This information is represented in our model by a triplet $< o_l, o_s, o_d >$, where o_l is a link object, and o_s and o_d are its source and destination, respectively. As any object, link objects are also defined w.r.t. a context. The link objects of a context c are determined by the function $links(c)$, which is defined as follows:

$$links(c) = \{< o_l, o_s, o_d > | o_l, o_s, o_d \in objs(c)\}.$$

Definition 4.4 Recursive Containment.
We say that *a context c recursively contains object o* if either c contains o, or there is a context contained in c that recursively contains o. This is denoted by $o \in^\star c$. $\diamond$

For instance, in Figure 4.2, context c_1 recursively contains object o_2, as c_1 contains c_2 and c_2 contains o_2, i.e., $o_2 \in^\star c_1$. We shall call *nested subcontext* of a context c, any context that is recursively contained in c.

We can *refer* to every object of a context c either by using one of its c-names, or by using a composite name, in case the object is contained in a nested subcontext of c. A *composite name* is a sequence of dot-separated names which are composed by taking into account the nesting of contexts, as shown in the following definition.

Definition 4.5 Name paths of an object in a context.
Let c be a context and let o be an object recursively contained in c. The set of all *name paths* of o in c, denoted by $npaths(o, c)$, is defined as follows:

$$npaths(o, c) = names(o, c) \cup compositeNames(o, c)$$
$$compositeNames(o, c) = \{r.n \mid \exists c' \in^\star c \wedge n \in names(o, c') \wedge r \in npaths(c', c)\}$$

The set of all name paths of all objects in all contexts is denoted by $\mathcal{NP}$. $\diamond$

For example (see Figure 4.2), we can refer to object o_1 of context c_1 either by using the name `Dr_Constantopoulos`, or by using the composite names `InfSys.Panos`, or `DSS.Constantopoulos`.

Note that a name path r in a context c may be *ambiguous*, in the sense that it may refer to more than one objects. That is, a name path r is ambiguous if there are two objects o, o' such that $r \in npaths(o, c) \cap npaths(o', c)$. It is possible for all name paths of an object o recursively

contained in a context c to be ambiguous, i.e.,

$$npaths(o,c) \subseteq \bigcup_{o' \in^\star c \wedge o' \neq o} npaths(o',c).$$

For example, in Figure 4.3, within context c_1, the name paths of object o_2, i.e. A, C, and D.F, are all ambiguous, as $\texttt{A} \in names(o_1, c_1)$, $\texttt{C} \in names(o_3, c_1)$, and $\texttt{D.F} \in npaths(o_4, c_1)$.

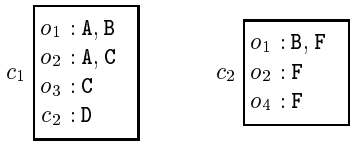


Figure 4.3: Example of ambiguous name paths.

However, in practice, at least one unique name path of an object is required to be used for external identification. Thus, we distinguish an important class of contexts that possess at least one unique name path for every object and we call these contexts *well-defined*. An acyclicity contraint is also imposed.

Definition 4.6 Well-defined lexicon.
A lexicon l is called *well-defined* iff it satisfies the following conditions:

1. **Unique name path.**
   For every object recursively contained in l, there is a unique name path in l, i.e., for all objects o, o' of l:
   $o \neq o' \Rightarrow \exists r \in npaths(o,l) : \ \forall r' \in npaths(o',l), r \neq r'$.
2. **Acyclicity.**
   For every nested subcontext c' of l, it holds: $c' \notin^\star c'$. $\diamond$

Definition 4.7 Well-defined context.
A context c is called *well-defined* iff its lexicon is well-defined and $c \notin^\star c$. $\diamond$

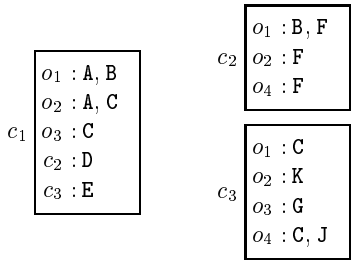
In the example of Figure 4.2, contexts c_1, c_2, and c_3 are well-defined. Another example is shown in Figure 4.3, where context c_1 is not well-defined as there is at least an object recursively contained in c_1 with non unique name paths in c_1, (e.g. the object o_2 or the object o_3 or the object o_4). Context c_2 is not well-defined as well. On the other hand, if we add the context c_3 in the contents of c_1 (see Figure 4.4) then c_1 becomes well-defined. Note that, in Figure 4.4, c_1 is a well-defined context although its subcontexts c_2 and c_3 are not.

Acyclicity is an important property of a context c, as it ensures that the set of name paths $npaths(o,c)$ of any object o recursively contained in c can be computed in finite time.

Proposition 4.1 Finite length and set of name paths.
Let c be a well-defined context, and let o be an object recursively contained in c. Then, the following hold:

1. Every name path of o in c has finite length, and
2. The set $npaths(o,c)$ is finite. $\diamond$

Proof: It follows easily from Definition 4.5 and the fact that all contexts contained in c satisfy the acyclicity property. □



Context c_1 is well-defined whereas contexts c_2 and c_3 are not.

Figure 4.4: Example of well-defined and non well-defined contexts.

We can assume a special context that recursively contains *all* objects of interest in a given application. We refer to this context as the *Information Base* (IB). As mentioned, a user can refer to an object using name paths. A name path to an object can be either *absolute*, i.e., in context IB, or *relative*. As a convention, if the name path is prefixed by @ then it is an *absolute name path*, otherwise it is a *relative name path*. Relative name paths are resolved with respect to a context specified by the user, which we call the *Current Context* (CC). The user sets the CC through the Set Current Context operation, introduced in the following section.

In order to guarantee that every object has a unique absolute name path, we require that the IB is a well-defined context. Therefore, we introduce the following axiom:

Axiom 4.1 Well-defined Information Base.
The context IB is a well-defined context. $\diamond$

Support for relative naming of objects is an important feature of our model. The following situations can be handled:

- *Synonyms*: Two different name paths w.r.t a context are called *synonymous*, if they refer to the same object. We view synonyms as alternative ways for externally identifying the same object. This is an important feature of our model because people often refer to the same concept using different names. For example, in Figure 4.2, the name paths `Nick` and `Nicolas` (which are the english and the french name of a person) in context c_2 are synonyms, as they refer to the same object o_4. Similarly, the name paths `Dr_Constantopoulos`, `InfSys.Panos`, and `DSS.Constantopoulos` in context c_1 are synonyms, as they refer to the same object o_1.

- *Homonyms*: Two different objects are called *homonymous* in a given context if they have a common name path in that context. If these two objects are recursively contained in a well-defined context c, then there exists a unique name path to each of these objects in c. Note that there always exists such a context, because IB recursively contains every object and it is a well-defined context, by assumption.

- *Anonyms*: An object o is called *anonymous* in a context c, if o is associated with no name in c, i.e., $names(o, c) = \emptyset$. Intuitively, this is possible when an object is contained in a context but we are not interested in naming it in that context, or we do not know its name yet. However, there is no problem with the external identification of o, if there is a well-defined context c' such that $npaths(o, c') \neq \emptyset$, and IB is such a context. For example, in Figure 4.2, the object o_4 in context c_1 is anonymous.

4.2 Operations on Contexts

In this section we present six operations on contexts: lookup, browsing, update, copy, union, intersection and difference. The presentation is informal, and uses illustrative examples.

Our definitions (both formal and informal) make use of two auxiliary concepts, namely, source context and derived context. Every context created by a single, explicit call of the operation $createCxt$ is called a *source context*.

In order to simplify the presentation, we introduce an auxiliary function $src(c)$ that returns the source of context c: if $src(c) = c'$ then c is a derived context and c' is its source, and if $src(c) = c$ then c is a source context.

With the above conventions in mind we now turn to the presentation of the operations.

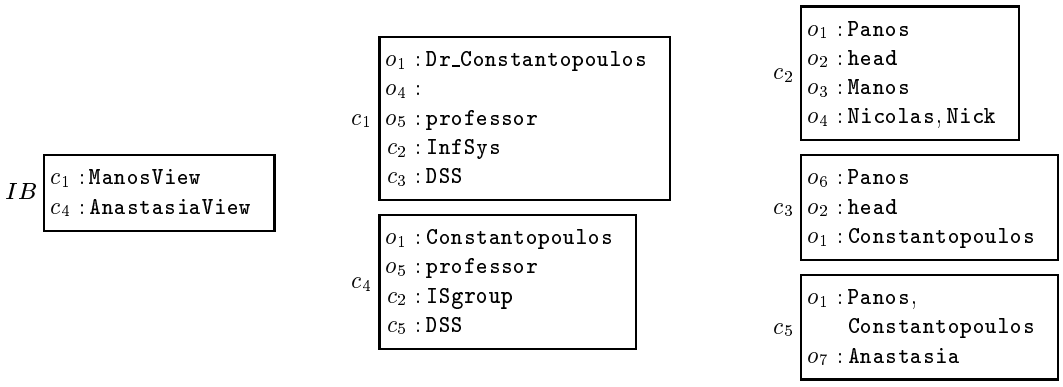


Figure 4.5: An Information Base context.

Consider the Information Base illustrated in Figure 4.5. Context IB contains two contexts c_1 and c_4, namely `ManosView` and `AnastasiaView`, respectively. These contexts represent the views of Manos and Anastasia regarding the Institute. Context c_4 contains the already seen objects o_1, o_5 and c_2, as well as a new context c_5 that represents the view of Anastasia regarding the Decision Support Systems lab. The fact that both contexts c_1 and c_4 share context c_2 indicates that both Manos and Anastasia have the same view for the Information Systems lab.

4.2.1 Lookup operations

- **lookup(r)**

  This operation takes as input a name path `r` and returns the set of objects o such that $\mathtt{r} \in npaths(o, c)$, where: $c = IB$ if `r` is absolute, or $c =$CC, otherwise. $\diamond$

- **lookupOne(r)**

  This operation takes as input a name path `r` and returns an object o such that: $r \in npaths(o, c)$ and $|\, npaths(o, c)\,| = 1$, where: $c = IB$, if `r` is absolute, or $c =$CC, otherwise. $\diamond$

The computational algorithms of the lookup operations are shown in Figure 4.6.

lookup(**Input** $r : \mathcal{NP}$; **Output** $O : \mathcal{P}(\mathcal{O})$).

/* This operation takes as input a reference r and returns all objects with reference. */

1. If r starts with @ then
   O is the set of all objects o such that $r \in npaths(o, IB)$
   else O is the set of all objects o such that $r \in npaths(o, \mathtt{CC})$.

2. End.

lookupOne(**Input** $r : \mathcal{NP}$; **Output** $o : \mathcal{O}$).

/* This operation takes as input a reference r and returns the object referenced by r, if it is just one. Otherwise, it returns ERROR. */

1. $O = lookup(r)$.

2. If the cardinality of the set O is one then
   return the element o of O
   else ERROR.

3. End.

Figure 4.6: The algorithms of the lookup operations.

4.2.2 Browsing operations

- **Set current context: SCC(r)**

  This operation takes as input a name path `r`[1] to a context (call it c), and sets the current context to be the context c. $\diamond$

  Example: The operation $SCC(@.\mathtt{ManosView.InfSys})$ sets the CC to c_2, and the operation $\overline{SCC(@)}$ sets the CC to IB.

The computational algorithm of this operation is shown in Figure 4.7.

SCC(**Input** $r : \mathcal{NP}$).

/* This operation takes as input a reference r to a context and sets CC to be this context. */

1. $c = lookup(r)$.

2. $\mathtt{CC} = c$; /* CC holds the current context of the user issuing the command */

3. End.

Figure 4.7: The algorithm of the operation SCC.

[1] In all operations, if a name path is ambiguous, an error message is returned.

4.2.3 Update operations

- **Create context: createCxt(l)**

  This operation takes a lexicon l as input, and returns a context (call it c) such that $lex(c) = l$. Additionally, it sets $src(c) = c$. $\diamond$

  Example: The operation $createCxt(\{o_1 : \texttt{Panos}, c_1 : \texttt{institute}\})$ results in the creation of a new context (call it c_{10}) with lexicon:

  $$lex(c_{10}) = \begin{cases} o_1 : \texttt{Panos} \\ c_1 : \texttt{institute}. \end{cases}$$

- **Insert an object into a context: insert($o, N,$ r)**

  This operation takes as input an object o, a set of names N and a name path r to a context (call this context c), and either inserts $(o : N)$ into the lexicon of c if object o is not contained in c or adds the names in N to the c-names of o. Additionally, it sets $src(c) = c$. This is because, as a new object has been inserted into c, c is thought as a derivation of the original source of c. $\diamond$

  Example: The operation $insert(o_{20}, \{\texttt{Nicolas}, \texttt{Nick}\}, @.\texttt{ManosView.DSS})$ results in the insertion of $o_{20} : \texttt{Nicolas}, \texttt{Nick}$ into the context c_3:

  c_3

o_6 : Panos
o_2 : head
o_1 : Constantopoulos
o_{20} : Nicolas, Nick

  Note that synonyms or homonyms may occur as a result of an $insert$ operation.

- **Delete an object from a context: deleteObj(o, r)**

  This operation takes as input an object o and a name path r to a context, and deletes the pair $(o : N)$ from the lexicon of that context. $\diamond$

- **Delete an object name from a context: deleteName(o, n, r)**

  This operation takes as input an object o, a name n, and a name path r to a context (call this context c), and deletes the name n from the c-names of o. $\diamond$

Note that a $deleteName$ operation may produce an anonym.

The computational algorithms of the update operations are shown in Figure 4.8.

4.2.4 Copy operations

- **Copy context: copyCxt(r)**

  This operation takes as input a name path r to a context (call this context c) and returns a new context (call it c') such that $lex(c') = lex(c)$. In other words:
  $copyCxt(\texttt{r}) = createCxt(lex(c))$. $\diamond$

createCxt(**Input** $l : \mathcal{L}$; **Output** $c : \mathcal{CXT}$).

/* This operation takes a lexicon l as input, and returns a new context c with lexicon l. */

1. Create a new context c such that $lex(c) = l$.
2. Set $src(c) = c$.
3. End.

insert(**Input** $o : \mathcal{O}, N : \mathcal{P}(\mathcal{N}), r : \mathcal{NP}$).

/* This operation takes as input an object o, a set of names N, and a reference r to a context (call this context c). Then, it either inserts $o{:}N$ into the lexicon of c if object o is not contained in c, or adds the names contained in N to the c-names of o. */

1. $c = lookupOne(r)$.
2. If $o{:}N' \in lex(c)$ then
   replace $o{:}N'$ by $o{:}N' \cup N$ in $lex(c)$
   else add $o{:}N$ into $lex(c)$.
3. Set $src(c) = c$.
4. End.

deleteObj(**Input** $o : \mathcal{O}, r : \mathcal{NP}$).

/* This operation takes as input an object o and a reference r to a context, and deletes the pair $o{:}N$ from the lexicon of that context. */

1. $c = lookupOne(r)$.
2. Delete the pair $o{:}N$ from $lex(c)$.
3. End.

deleteName(**Input** $o : \mathcal{O}, n : \mathcal{N}, r : \mathcal{NP}$).

/* This operation takes as input an object o, a name n, and a reference r to a context (call this context c), and deletes the name n from the c-names of o. */

1. $c = lookupOne(r)$.
2. If $o{:}N \in lex(c)$ then
   replace $o{:}N$ by $o{:}N - \{n\}$ in $lex(c)$.
3. End.

Figure 4.8: The algorithms of the update operations.

Example : The operation $copyCxt(@.\texttt{ManosView})$ returns a new context (call it c_{11}) shown as follows:

c_{11}

o_1 : Dr_Constantopoulos
o_4 :
o_5 : professor
c_2 : InfSys
c_3 : DSS

- **Deep copy context: deepCopyCxt(r)**

This operation takes as input a name path r to a context (call this context c), and returns a new context (call it c') that contains the simple objects of c and deep copies of the contexts contained in c, (i.e., copies of those contexts together with their recursive expansions). In case a context c'' is contained in two or more contexts that are recursively contained in c, then c'' is copied only once (i.e., c' does not recursively contain multiple copies of the same context). The computational algorithm of this operation is shown in Figure 4.9. ◇

Example 1: The operation $deepCopyCxt(@.\texttt{ManosView})$ returns a new context (call it c'_1) which contains copies of contexts c_2 and c_3 (call them c'_2 and c'_3) as shown in the following picture:

c'_1

o_1 : Dr_Constantopoulos
o_4 :
o_5 : professor
c'_2 : InfSys
c'_3 : DSS

c'_2

o_1 : Panos
o_2 : head
o_3 : Manos
o_4 : Nicolas, Nick

c'_3

o_6 : Panos
o_2 : head
o_1 : Constantopoulos

Example 2: The operation $deepCopyCxt(@)$ returns a new context (call it c_{12}) which contains deep copies of contexts c_1 and c_4 (call them c'_1 and c'_4). Context c'_1 contains copies of contexts c_2 and c_3 (call them c'_2 and , c'_3) whereas context c'_4 contains copies of contexts c_2 and c_5 (these are context c'_2 and c'_5). Note that although context c_2 is contained in both contexts c_1 and c_4 it is copied only once (context c'_2).

c_{12}

c'_1 : ManosView
c'_4 : AnastasiaView

c'_1

o_1 : Dr_Constantopoulos
o_4 :
o_5 : professor
c'_2 : InfSys
c'_3 : DSS

c'_4

o_1 : Constantopoulos
o_5 : professor
c'_2 : ISgroup
c'_5 : DSS

c'_2

o_1 : Panos
o_2 : head
o_3 : Manos
o_4 : Nicolas, Nick

c'_3

o_6 : Panos
o_2 : head
o_1 : Constantopoulos

c'_5

o_1 : Panos, Constantopoulos
o_7 : Anastasia

copyCxt(**Input** $r : \mathcal{NP}$; **Output** $c' : \mathcal{CXT}$).

/* This operation takes as input a reference r to a context (call this context c) and returns a new context c' such that $lex(c') = lex(c)$. The computational algorithm of this operation is shown in Figure 4.9. */

1. $c = lookupOne(r)$.
2. $c' = createCxt(lex(c))$.
3. End.

deepCopyCxt(**Input** $r : \mathcal{NP}$; **Output** $out\_c : \mathcal{CXT}$).

/* This operation takes as input a reference r to a context (call this context c) and returns a new context out_c. Context out_c contains the original simple objects of c, and deep copies of the contexts contained in c. */

1. $c = lookupOne(r)$.
2. Let $RecCxt$ be the contexts recursively contained in c.
3. $OrigCxt = RecCxt \cup \{c\}$.
4. $CopiedCxt = \emptyset$.
5. While $OrigCxt \neq \emptyset$ do
   (a) Find context $c' \in OrigCxt$ which is not contained in any other context in $OrigCxt$.
   (b) $c'' = copyCxt(c')$.
   (c) If $c = c'$ then $out\_c = c''$.
   (d) If c' is contained in some contexts in $CopiedCxt$ then
   replace c' with c'' in the lexicon of these contexts.
   (e) $OrigCxt = OrigCxt - \{c'\}$.
   (f) $CopiedCxt = CopiedCxt \cup \{c''\}$.
6. End.

deepCopyCxt(**Input** $r : \mathcal{NP}, d : Integer$; **Output** $out\_c : \mathcal{CXT}$).

/* This operation takes as input a reference r to a context (call this context c) and an integer d, and returns a new context out_c. Context out_c contains the original simple objects of c, and deep copies of the contexts contained in c up to depth d. */
The same like $deepCopyCxt$ except for Step 2:

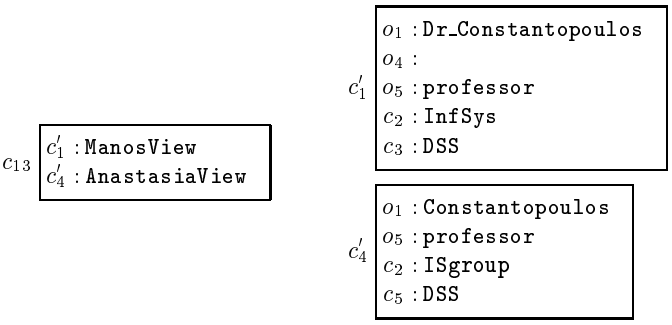
2 Let $RecCxt$ be the contexts recursively contained in c up to depth d.

Figure 4.9: The algorithms of the copy operations.

- **Deep copy context up to depth d: deepCopyCxt($\mathtt{r}, d$)**

  This operation takes as input a name path $\mathtt{r}$ to a context (call this context c) and an integer d, and returns a new context (call it c') such that: if $d = 1$ then $c' = copyCxt(r)$ otherwise if $d > 1$ then c' contains the simple objects of c and deep copies up to depth $d - 1$ of the contexts contained in c. In case a context c'' is contained in two or more contexts that are recursively contained in c, then c'' is copied only once at the shallowest level (i.e., c' does not recursively contain multiple copies of the same context). The computational algorithm of this operation is shown in Figure 4.9. $\diamond$

  Example: The operation $deepCopyCxt(@, 2)$ returns a new context (call it c_{13}) which contains copies of contexts c_1 and c_4 (call them c'_1 and c'_4).



4.2.5 Union operation

- **Union: $\mathtt{r_1} \uplus \mathtt{r_2}$**

  This operation takes as input two parameters r_1 and r_2 and returns a lexicon as a result. We distinguish three cases:

  1. If r_1 and r_2 are both lexicons, then the operation returns a lexicon l such that (let $O_1 = objs(r_1)$ and $O_2 = objs(r_2)$):
     (a) $objs(l) = O_1 \cup O_2$.
     (b) For each object $o \in objs(l)$: $\quad l(o) = \begin{cases} r_1(o) \cup r_2(o), & \text{if } o \in O_1 \cap O_2 \\ r_1(o), & \text{if } o \in O_1 \text{ and } o \notin O_2 \\ r_2(o), & \text{if } o \in O_2 \text{ and } o \notin O_1 \end{cases}$
     (c) Find all contexts of l with the same source (call this source c) and merge them into a new context with source c.
  2. If r_1 is a lexicon and $\mathtt{r_2}$ is a name path to a context (call this context c_2), then the operation returns a lexicon l such that:

     $$l = r_1 \uplus (lex(c_2) \uplus \{c_2 : \{str(\mathtt{r_2})\}\}).$$

     In other words, we add the context c_2 to the lexicon of c_2, and use the name $str(r_2)$ as one of its names (where the function $str(\mathtt{r})$ converts a name path $\mathtt{r}$ to a name by replacing dots by underscores).
  3. If $\mathtt{r_1}$ and $\mathtt{r_2}$ are both name paths to contexts (call these contexts c_1 and c_2), then the operation returns a lexicon l such that:

     $$l = (lex(c_1) \uplus \{c_1 : \{str(\mathtt{r_1})\}\}) \uplus (lex(c_2) \uplus \{c_2 : \{str(\mathtt{r_2})\}\}).$$

lexUnion(**Input** $O : \mathcal{P}(\mathcal{O});\ l_1, l_2 : \mathcal{L};$ **Output** $l : \mathcal{L}$).

/* This operation takes as input a set of object O and two lexicons l_1 and l_2, and returns a lexicon l. Lexicon l contains objects of O that are also contained in l_1 or l_2. The names of each object o of l is the union of the names of o w.r.t. l_1 with the names of o w.r.t. l_2. */

1. Let $l = \emptyset$.
2. For each $o \in O$ do
   If $o \in objs(l_1) \cap objs(l_2)$ then $l = l \cup \{o\text{:}\, names(o, l_1) \cup names(o, l_2)\}$
   else if $o \in objs(l_1)$ then $l = l \cup \{o\text{:}\, names(o, l_1)\}$
   else $l = l \cup \{o\text{:}\, names(o, l_2)\}$
3. End.

merge(**Input** $l : \mathcal{L};$ **Output** $out\_l : \mathcal{L}$).

/* This operation takes as input a lexicon l and merges its subcontexts $c_1, \ldots, c_k$ with the same source context, i.e., $src(c_1) = \ldots = src(c_k)$. */

1. $c = createCxt(l)$.
2. Let $RecCxt$ be the contexts recursively contained in c.
3. $OrigCxt = RecCxt \cup \{c\}$.
4. While $OrigCxt \neq \emptyset$ do
   (a) Find context $c' \in OrigCxt$ which is not contained in any other context in $OrigCxt$.
   (b) Let $M = \{c_1, \ldots, c_k\} \subseteq \mathcal{CXT}$,
   where $\forall i \in \{1, \ldots, k\} :\ c_i \in objs(c')\ \wedge\ src(c_i) = src(c_1)$.
   (c) If $M \neq \emptyset$ then
      i. $N_m = names(c_1, c') \cup \ldots \cup names(c_k, c')$.
      ii. If $\exists c_i :\ c_i = src(c_i)$ then $c_m = c_i$
      else $c_m = createCxt(lex(c_1) \uplus \ldots \uplus lex(c_k))$. /* Merges the lexicon of contexts $c_1, \ldots, c_k$ */
      iii. Set $src(c_m) = src(c_1)$.
      iv. For $i \in \{1, \ldots, k\}$ do
      If $c_i \neq src(c_i)$ then $deleteObj(c_i, c')$.
      v. $insert(c_m, N_m, c')$.
      vi. If $c_m \neq src(c_m)$ then
      $OrigCxt = OrigCxt \cup \{c_m\}$. /* *merge* will be called for c_m as well */
   (d) $OrigCxt = OrigCxt - \{c'\}$.
5. $out\_l = lex(c)$.
6. End.

Figure 4.10: The algorithms of the operations $lexUnion$ and $merge$.

1. Lexicon Union

$\uplus$ (**Input** $l_1, l_2 : \mathcal{L}$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two lexicons and returns their union. */

1. $out\_l = lexUnion(objs(l_1) \cup objs(l_2), l_1, l_2)$.
2. $out\_l = merge(out\_l)$. /* the operation $merge$ is shown in Figure 4.10 */
3. End.

2. Context-Lexicon Union

$\uplus$ (**Input** $r_1 : \mathcal{NP}$, $l_2 : \mathcal{L}$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input a reference r_1 to a context and a lexicon and returns the union between this context and this lexicon. */

1. $c_1 = lookupOne(r_1)$.
2. $l_1 = lex(c_1) \ \uplus \ \{ \ c_1 : \{str(r_1)\} \ \}$.
3. $out\_l = l_1 \ \uplus \ l_2$.
4. End.

3. Context Union

$\uplus$ (**Input** $r_1, r_2 : \mathcal{NP}$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two references r_1 and r_2 to two contexts and returns the union of these contexts. */

1. $c_1 = lookupOne(r_1)$.
2. $c_2 = lookupOne(r_2)$.
3. $l_1 = lex(c_1) \ \uplus \ \{ \ c_1 : \{str(r_1)\} \ \}$.
4. $l_2 = lex(c_2) \ \uplus \ \{ \ c_2 : \{str(r_2)\} \ \}$.
5. $out\_l = l_1 \ \uplus \ l_2$.
6. End.

Figure 4.11: The algorithms of the union operations.

The computational algorithm of this operation is shown in Figures 4.10 and 4.11. ◇

Note that, in Case 1, if an object belongs to both lexicons then we can refer to it in the output lexicon, using any of its names in the two input lexicons. In Case 2 (where the second parameter is a context), context c_2 is added to the output lexicon under the name $\mathtt{r}_2$. Intuitively, this adds a view over the objects of the combined lexicons as seen from c_2. We name this view $\mathtt{r}_2$ to record the fact that this view has been referred to by the user as $\mathtt{r}_2$[2]. Similarly, in Case 3 (where both inputs are contexts), contexts c_1 and c_2 are added to the output lexicon under the names $\mathtt{r}_1$ and $\mathtt{r}_2$, respectively.

It is important to note that of context union operation keeps track of the contexts the results come from, since the original contexts involved the operation are contained in the results as well (e.g., context c_2 in Case 2).

Example 1: Assume that CC has been set to c_1. Then, the operations $lex(\mathtt{InfSys}) \uplus lex(\mathtt{DSS})$ and $\mathtt{InfSys} \uplus \mathtt{DSS}$ return the lexicons l_1 and l_2, respectively, such that:

$$l_1 = \begin{cases} o_1 : \mathtt{Panos}, \\ \quad \mathtt{Constantopoulos} \\ o_2 : \mathtt{head} \\ o_3 : \mathtt{Manos} \\ o_4 : \mathtt{Nicolas}, \mathtt{Nick} \\ o_6 : \mathtt{Panos} \end{cases} \qquad l_2 = \begin{cases} o_1 : \mathtt{Panos}, \\ \quad \mathtt{Constantopoulos} \\ o_2 : \mathtt{head} \\ o_3 : \mathtt{Manos} \\ o_4 : \mathtt{Nicolas}, \mathtt{Nick} \\ o_6 : \mathtt{Panos} \\ c_2 : \mathtt{InfSys} \\ c_3 : \mathtt{DSS} \end{cases}$$

Note that object o_1 has two names: one originating from c_2 and the other from c_3. Note also that InfSys and DSS are name paths (w.r.t. the CC) of contexts c_2 and c_3, respectively. Intuitively, the union of InfSys and DSS contains the objects of l_1, as well as two views (contexts c_2 and c_3) over these objects, as seen from the Information Systems and DSS lab, respectively.

Example 2: Assume that the current context is the context IB, i.e., CC = IB. The operation $\mathtt{ManosView} \uplus \mathtt{AnastasiaView}$ combines the views of Manos and Anastasia to get a wider view of the Institute, and returns the following lexicon:

$$l_3 = \begin{cases} o_1 : \mathtt{Dr\_Constantopoulos}, \mathtt{Constantopoulos} \\ o_4 : \\ o_5 : \mathtt{professor} \\ c_2 : \mathtt{InfSys}, \mathtt{ISgroup} \\ c_3 : \mathtt{DSS} \\ c_5 : \mathtt{DSS} \\ c_1 : \mathtt{ManosView} \\ c_4 : \mathtt{AnastasiaView} \end{cases}$$

Note that there are two different contexts c_3 and c_5 with the same name. However, no ambiguity is caused, as these contexts also belong to contexts c_1 and c_4, respectively. Therefore, we can refer to c_3 and c_5 uniquely through the name paths ManosView.DSS and AnastasiaView.DSS, respectively.

[2]Obviously, the user can change this name using the operations: *deleteName* and *insert*.

4.2.6 Intersection operation

- **Intersection plus:** $\mathtt{r_1} \uplus_{\mathtt{d}} \mathtt{r_2}$

To define the intersection operation we need first to introduce the function $ComO$. Let l_1, l_2 be lexicons. We define $ComO(l_1, l_2) = objs(l_1) \cap objs(l_2)$. This operation takes as input two parameters r_1 and r_2, and returns a lexicon as a result. It also takes as input an integer d which is the depth of cleaning nested subcontexts from non-common objects. We distinguish three cases:

1. If r_1 and r_2 are both lexicons, then the operation returns a lexicon l defined as follows (let $I = ComO(r_1, r_2)$):
   (a) If $o \in I$ then $o \in objs(l)$ and $l(o) = r_1(o) \cup r_2(o)$.
   (b) If $d > 1$ then
   If $o \notin I$ and o is a context recursively containing up to depth d an object of I then:
      i. Make a deepcopy of o up to depth d (call it c), and set the source of its copy context to be equal to the source of the original context.
      ii. Remove from c and from every context recursively contained in c (i) any simple object that is not in I, and (ii) any context that is not in I and does not recursively contain objects in I.
      iii. Add c to $objs(l)$ and define: $l(c) = \begin{cases} r_1(o), & \text{if } o \in objs(r_1) \\ r_2(o), & \text{if } o \in objs(r_2) \end{cases}$
   (c) Find all contexts of l with the same source (call this source c) and merge them into a new context with source c.
2. If r_1 is a lexicon and $\mathtt{r_2}$ is a name path to a context (call this context c_2), then the operation returns a lexicon l such that:
   $$l = r_1 \uplus_d (lex(c_2) \uplus \{c_2' : \{str(\mathtt{r_2})\}\}),$$
   where c_2' is a new context such that $lex(c_2') = lex(c_2)$.
3. If $\mathtt{r_1}$ and $\mathtt{r_2}$ are both name paths to contexts (call these contexts c_1 and c_2), then the operation returns a lexicon l such that:
   $$l = (lex(c_1) \uplus \{c_1' : \{str(\mathtt{r_1})\}\}) \uplus_d (lex(c_2) \uplus \{c_2' : \{str(\mathtt{r_2})\}\})$$
   where c_1' and c_2' are new contexts such that $lex(c_1') = lex(c_1)$ and $lex(c_2') = lex(c_2)$.

The computational algorithm of this operation is shown in Figures 4.12 and 4.13. $\diamond$

Note that, if an object belongs to both lexicons, then we can refer to it in the output lexicon using any of its names in the two input lexicons. In Case 2 (where the second parameter is a context), we add to the output lexicon a new context c_2' with name $\mathtt{r_2}$. Intuitively, this adds a view over the objects of the output lexicon as seen from c_2. Context c_2' results from c_2 after removing from it and its nested subcontexts all simple objects that are not contained in $ComO(r_1, lex(c_2))$. The same holds in Case 3.

Parameter d determines how deep the nested subcontexts of the result will be cleaned from non common objects (i.e., objects not contained in $ComO(l_1, l_2)$). In fact, parameter

elimObj(**Input** $O : \mathcal{P}(\mathcal{O}); C : \mathcal{P}(\mathcal{CXT})$).

/* This operation takes as input a set of objects O and set of contexts C, and works as follows: The objects in O are eliminated from each context in C. If a context $c \in C$ is shared by another context $c' \in C$, then c is not eliminated from the objects of c'. */

1. While $C \neq \emptyset$ do
   (a) Find context $c \in C$ which does not contain any other context in C.
   (b) For each $o \in objs(c)$ do
   If $o \notin O$ then $deleteObj(o, c)$.
   (c) If $c \neq \emptyset$ then $O = O \cup \{c\}$.
   (d) $C = C - \{c'\}$.
2. End.

BodyInterPlus(**Input** $l_1, l_2 : \mathcal{L},\ c_1, c_2 : \mathcal{CXT},\ d : Integer$; **Output** $out\_l : \mathcal{L}$).

/* This operation takes as input two lexicons and an integer d and returns their intersection up to depth d. It also takes as input two contexts which are two views over the objects of the result (if an input context is *NIL* it is not taken into account). */

1. $I = ComO(l_1, l_2)$.
2. $out\_l = lexUnion(I, lex(l_1), lex(l_2))$. /* the operation $lexUnion$ is shown in Figure 4.10 */
3. $ComC = I \cap \mathcal{CXT}$. /* $ComC$ stands for Common Contexts */
4. Let $RecCxt$ be the contexts recursively contained in l_1 or l_2 up to depth d, i.e., in depth $d_i < d$.
   /* contexts contained in l_1 or l_2 are in depth 1. */
5. If $c_1 \neq NIL$ then $RecCxt = RecCxt \cup \{c_1\}$.
6. If $c_2 \neq NIL$ then $RecCxt = RecCxt \cup \{c_2\}$.
7. $OrigCxt = RecCxt - ComC$.
8. $CopiedCxt = \emptyset$.
9. While $OrigCxt \neq \emptyset$ do
   (a) Find context $c \in OrigCxt$ which is not contained in any other context in $OrigCxt$.
   /* c is contained either in l_1 or in l_2 */
   (b) $c' = copyCxt(c)$.
   (c) Set $src(c') = src(c)$.
   (d) If $c \in objs(l_1)$ then $out\_l = out\_l \uplus \{c' : names(c, l_1)\}$
   else If $c \in objs(l_2)$ then $out\_l = out\_l \uplus \{c' : names(c, l_2)\}$
   (e) If c is contained in some contexts in $CopiedCxt$
   then replace c with c' in the lexicon of these contexts.
   (f) $OrigCxt = OrigCxt - \{c\}$.
   (g) $CopiedCxt = CopiedCxt \cup \{c'\}$.
10. $elimObj(I, CopiedCxt)$.
11. $out\_l = merge(out\_l)$. /* the operation $merge$ is shown in Figure 4.10 */
12. End.

Figure 4.12: The algorithms of the operations $elimObj$ and $BodyInterPlus$.

1. Lexicon Intersection Plus

Θ (**Input** $l_1, l_2 : \mathcal{L}, d : Integer$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two lexicons and an integer d, and returns the intersection up to depth d of these two lexicon. */

1. $out\_l = BodyInterPlus(l_1, l_2, NIL, NIL, d)$.
   /* this operation is shown in Figure 4.12 */
2. End.

2. Context-Lexicon Intersection Plus

Θ (**Input** $r_1 : \mathcal{NP},\ l_2 : \mathcal{L}, d : Integer$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes a reference r_1 to a context and a lexicon, and an integer d, and returns the intersection up to depth d between this context and this lexicon. */

1. $c_1 = lookupOne(r_1)$.
2. $c_1' = copyCxt(c_1)$.
3. $l_1 = lex(c_1) \uplus \{c_1' : \{str(r_1)\}\}$.
4. $out\_l = BodyInterPlus(l_1, l_2, c_1', NIL, d)$.
   /* this operation is shown in Figure 4.12 */
5. End.

3. Context Intersection Plus

Θ (**Input** $r_1, r_2 : \mathcal{NP}, d : Integer$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two references r_1 and r_2 to two contexts and an integer d, and returns the intersection up to depth d of these contexts. */

1. $c_1 = lookupOne(r_1)$.
2. $c_2 = lookupOne(r_2)$.
3. $c_1' = createCxt(lex(c_1))$.
4. $c_2' = createCxt(lex(c_2))$.
5. $l_1 = lex(c_1) \uplus \{\ c_1' : \{str(r_1)\}\ \}$.
6. $l_2 = lex(c_2) \uplus \{\ c_2' : \{str(r_2)\}\ \}$.
7. $out\_l = BodyInterPlus(l_1, l_2, c_1', c_2', d)$.
   /* this operation is shown in Figure 4.12 */
8. End.

Figure 4.13: The algorithm of the operation Intersection Plus.

d is used in practice to face up with the complexity of recursion (cleaning of nested subcontexts from non-common objects).

Parameter d increases the expressiveness of intersection in the following way: if d is equal to 1 the result contains all common objects. This is the most common type of intersection. However, if d is greater than 1, the result contains not only the common objects but also the subcontexts that contain these common objects in any depth less than or equal to d. For example, imagine two contexts: one containing the terminology used in Chemistry and the other the terminology used in Biology. Both contexts contain subcontexts representing departments of Chemistry and Biology, respectively, that contain the terminology used in these departments. The intersection of these two contexts for $d = 1$ will result in the common terminology of Chemistry and Biology, as well as in their common departments. However, the same intersection for $d \geq 1$, say $d = 5$, will result not only in their common terminology and departments, but also in a mass of departments and subdepartments in depth 5 that use this common terminology. Note that departments and subdepartments contain only the common terminology, while the rest of the information has been removed from them.

In the rest of the thesis, whenever parameter d is not used it is assumed to be infinite.

Example 1: The operation $lex(\texttt{InfSys}) \uplus lex(\texttt{DSS})$, returns the lexicon:

$$l_4 = \begin{cases} o_1 : \texttt{Panos}, \\ \quad\ \texttt{Constantopoulos} \\ o_2 : \texttt{head}. \end{cases}$$

Note that $I = \{o_1, o_2\}$. Therefore, objects o_1 and o_2 are added to the output lexicon in Step 1(a). Note that like in the Union operation, object o_1 has two names.

Example 2: The operation $\texttt{InfSys} \uplus \texttt{DSS}$, returns the following lexicon:

$$l_5 = \begin{cases} o_1 : \texttt{Panos}, \texttt{Constantopoulos} \\ o_2 : \texttt{head} \\ c_2'' : \texttt{InfSys} \\ c_3'' : \texttt{DSS} \end{cases} \qquad c_2'' \boxed{\begin{array}{l} o_1 : \texttt{Panos} \\ o_2 : \texttt{head} \end{array}}$$

$$c_3'' \boxed{\begin{array}{l} o_2 : \texttt{head} \\ o_1 : \texttt{Constantopoulos} \end{array}}$$

Note that $I = \{o_1, o_2\}$. Contexts c_2'' and c_3'' are derived from contexts c_2 and c_3 after removing all simple objects not in I and thus, $src(c_2'') = src(c_2) = c_2$ and $src(c_3'') = src(c_3) = c_3$ (Step 1(b)ii). Contexts c_2'' and c_3'' are added to the output lexicon in Step 3.

Example 3: The operation $\texttt{ManosView} \uplus \texttt{AnastasiaView}$ computes the commonalities of the views of Manos and Anastasia, and returns the following lexicon:

$$l_6 = \begin{cases} o_1 : \texttt{Dr\_Constantopoulos}, \\ \quad \texttt{Constantopoulos} \\ o_5 : \texttt{professor} \\ c_2 : \texttt{InfSys}, \texttt{ISgroup} \\ c'_3 : \texttt{DSS} \\ c'_5 : \texttt{DSS} \\ c'_1 : \texttt{ManosView} \\ c'_4 : \texttt{AnastasiaView} \end{cases}$$

c'_3 | $o_1 : \texttt{Constantopoulos}$

c'_5 | $o_1 : \texttt{Panos}, \texttt{Constantopoulos}$

c'_1 |
$o_1 : \texttt{Dr\_Constantopoulos}$
$o_5 : \texttt{professor}$
$c_2 : \texttt{InfSys}$
$c'_3 : \texttt{DSS}$

c'_4 |
$o_1 : \texttt{Constantopoulos}$
$o_5 : \texttt{professor}$
$c_2 : \texttt{ISgroup}$
$c'_5 : \texttt{DSS}$

Note that the set I of the Intersection algorithm is $\{o_1, o_5, c_2\}$. That is, objects o_1, o_5, and c_2 are the common objects of c_1 and c_4. These objects are added to the lexicon of the intersection in Step 1(a) of the Intersection algorithm. Contexts c'_3 and c'_5 are copies of contexts c_3 and c_5 after removing all simple objects not in I. Contexts c'_3 and c'_5 are added to the lexicon of the intersection in Step 1(b) of the Intersection algorithm. These contexts represent views over the objects in I as seen from c_3 and c_5, respectively. Contexts c'_1 and c'_4 are copies of contexts c_1 and c_4 after removing all simple objects not in I, and all contexts not in I which do not recursively contain objects in I. Contexts c'_1 and c'_4 are added to the lexicon of the intersection in Step 3 of the Intersection algorithm. Contexts c'_1 and c'_4 represent views over the objects in I as seen from c_1 and c_4, respectively.

Example 4: The operation $lex(\texttt{ManosView}) \ \pitchfork_1 \ lex(\texttt{AnastasiaView})$ computes the commonalities of the the contents of views of Manos and Anastasia in depth 1, and returns the following lexicon:

$$l_7 = \begin{cases} o_1 : \texttt{Dr\_Constantopoulos}, \\ \quad \texttt{Constantopoulos} \\ o_5 : \texttt{professor} \\ c_2 : \texttt{InfSys}, \texttt{ISgroup} \end{cases}$$

Example 5: The operation $\texttt{ManosView} \ \pitchfork_1 \ \texttt{AnastasiaView}$ computes the commonalities of the views of Manos and Anastasia in depth 1, and returns the following lexicon:

$$l_8 = \begin{cases} o_1 : \texttt{Dr\_Constantopoulos}, \\ \quad \texttt{Constantopoulos} \\ o_5 : \texttt{professor} \\ c_2 : \texttt{InfSys}, \texttt{ISgroup} \\ c'_1 : \texttt{ManosView} \\ c'_4 : \texttt{AnastasiaView} \end{cases}$$

c'_1 |
$o_1 : \texttt{Dr\_Constantopoulos}$
$o_5 : \texttt{professor}$
$c_2 : \texttt{InfSys}$

c'_4 |
$o_1 : \texttt{Constantopoulos}$
$o_5 : \texttt{professor}$
$c_2 : \texttt{ISgroup}$

- **Intersection times:** $\mathbf{r_1} \ \pitchfork_{d} \ \mathbf{r_2}$

  It is defined like Intersection Plus except for the following Steps:

  1(a). If $o \in I$ then $o \in objs(l)$ and $l(o) = r_1(o) \cap r_2(o)$.

  2. $l = r_1 \ \pitchfork_{d} \ (lex(c_2) \ \uplus \ \{c'_2 : \{str(\mathbf{r_2})\}\})$

  3. $l = (lex(c_1) \uplus \{c'_1 : \{str(\mathbf{r_1})\}\}) \ \pitchfork_{d} \ (lex(c_2) \uplus \{c'_2 : \{str(\mathbf{r_2})\}\})$

  The computational algorithm of this operation is shown in Figure 4.14. $\diamond$

lexIntersection(**Input** $O : \mathcal{P}(\mathcal{O});\ l_1, l_2 : \mathcal{L};$ **Output** $l : \mathcal{L}$).

/* This operation takes as input a set of object O and two lexicons l_1 and l_2, and returns a lexicon l. Lexicon l contains objects of O that are also contained in l_1 and l_2. The names of each object o of l is the intersection the l_1-names of o with the l_2-names of o. */

1. Let $l = \emptyset$.
2. For each $o \in O$ do
   If $o \in objs(l_1) \cap objs(l_2)$ then $l = l \cup \{o : names(o, l_1) \cap names(o, l_2)\}$
3. End.

BodyInterTimes(**Input** $l_1, l_2 : \mathcal{L},\ c_1, c_2 : \mathcal{CXT},\ d : Integer$; **Output** $out\_l : \mathcal{L}$).

/* The same like operation $BodyInterPlus$, shown in Figure 4.12, except for Step 2: */

2. $out\_l = lexIntersection(I, lex(l_1), lex(l_2))$.

Intersection Times($\cap$ **).**

1. Lexicon Intersection Times

$\cap$ (**Input** $l_1, l_2 : \mathcal{L}, d : Integer;$ **Output** $out\_l : \mathcal{L}$)

1. $out\_l = BodyInterTimes(l_1, l_2, NIL, NIL, d)$.
2. End.

2. Context-Lexicon Intersection Times

$\cap$ (**Input** $r_1 : \mathcal{NP},\ l_2 : \mathcal{L}, d : Integer;$ **Output** $out\_l : \mathcal{L}$)

1. $c_1 = lookupOne(r_1)$.
2. $c_1' = copyCxt(c_1)$.
3. $l_1 = lex(c_1) \uplus \{c_1' : \{str(r_1)\}\}$.
4. $out\_l = BodyInterTimes(l_1, l_2, c_1', NIL, d)$.
5. End.

3. Context Intersection Times

$\cap$ (**Input** $r_1, r_2 : \mathcal{NP}, d : Integer;$ **Output** $out\_l : \mathcal{L}$)

1. $c_1 = lookupOne(r_1)$.
2. $c_2 = lookupOne(r_2)$.
3. $c_1' = createCxt(lex(c_1))$.
4. $c_2' = createCxt(lex(c_2))$.
5. $l_1 = lex(c_1) \uplus \{\ c_1' : \{str(r_1)\}\ \}$.
6. $l_2 = lex(c_2) \uplus \{\ c_2' : \{str(r_2)\}\ \}$.
7. $out\_l = BodyInterTimes(l_1, l_2, c_1', c_2', d)$.
8. End.

Figure 4.14: The algorithm of the operation Intersection Times.

Motivation and use of parameter d are similar to these in the Intersection Plus operation.

Example 1: The operation $lex(\texttt{InfSys}) \ \text{⍟} \ lex(\texttt{DSS})$, returns the lexicon:

$$l'_4 = \begin{cases} o_1 : \\ o_2 : \texttt{head}. \end{cases}$$

Example 2: The operation $\texttt{InfSys} \ \text{⍟} \ \texttt{DSS}$, returns the following lexicon:

$$l'_5 = \begin{cases} o_1 : \\ o_2 : \texttt{head} \\ c''_2 : \texttt{InfSys} \\ c''_3 : \texttt{DSS} \end{cases}$$

c''_2 | $o_1 : \texttt{Panos}$, $o_2 : \texttt{head}$

c''_3 | $o_2 : \texttt{head}$, $o_1 : \texttt{Constantopoulos}$

Example 3: The operation $\texttt{ManosView} \ \text{⍟} \ \texttt{AnastasiaView}$ computes the commonalities of the views of Manos and Anastasia, and returns the following lexicon:

$$l'_6 = \begin{cases} o_1 : \\ o_5 : \texttt{professor} \\ c_2 : \\ c'_3 : \texttt{DSS} \\ c'_5 : \texttt{DSS} \\ c'_1 : \texttt{ManosView} \\ c'_4 : \texttt{AnastasiaView} \end{cases}$$

c'_3 | $o_1 : \texttt{Constantopoulos}$

c'_5 | $o_1 : \texttt{Panos}, \texttt{Constantopoulos}$

c'_1 | $o_1 : \texttt{Dr\_Constantopoulos}$, $o_5 : \texttt{professor}$, $c_2 : \texttt{InfSys}$, $c'_3 : \texttt{DSS}$

c'_4 | $o_1 : \texttt{Constantopoulos}$, $o_5 : \texttt{professor}$, $c_2 : \texttt{ISgroup}$, $c'_5 : \texttt{DSS}$

Note that the common objects o_1 and o_2 of lexicon l'_6 are without any name. This means that although these two objects are known to both Manos and Anastasia, they use different set of names to describe them.

4.2.7 Difference operation

- **Difference:** $\mathbf{r_1 \ \ominus_d \ r_2}$

This operation takes as input two parameters r_1 and r_2, and returns a lexicon as a result. We distinguish four cases:

1. If r_1 and r_2 are both lexicons, then the operation returns a lexicon l such that (let $D = objs(r_1) - objs(r_2)$ and $I = objs(r_1) \cap objs(r_2)$):
   (a) If $o \in D$ then $o \in objs(l)$ and $l(o) = r_1(o)$.
   (b) If $d > 1$ then
   If $o \in I$ and o is a context recursively containing up to depth d an object of D then:
      i. Make a deepcopy of o up to depth d (call it c), and set the source of its newly derived context (copy) to be equal to the source of the original context.

ii. Remove from c and from every context recursively contained in c, any simple object that is not in D.

iii. Add c to $objs(l)$ and define: $l(c) = r_1(o)$.

(c) No other object is in $objs(l)$.

2. If r_1 is a lexicon and $\mathtt{r_2}$ is a name path to a context (call this context c_2), then the operation returns a lexicon l such that:

$$l = r_1 \ominus_d lex(c_2).$$

3. If $\mathtt{r_1}$ is a name path to a context (call this context c_1) and r_2 is a lexicon, then the operation returns a lexicon l such that:

$$l = (lex(c_1) \uplus \{c_1' : \{str(\mathtt{r_1})\}\}) \ominus_d r_2,$$

where $lex(c_1') = lex(c_1)$.

4. If $\mathtt{r_1}$ and $\mathtt{r_2}$ are both name paths to contexts (call these contexts c_1 and c_2), then the operation returns a lexicon l such that:

$$l = (lex(c_1) \uplus \{c_1' : \{str(\mathtt{r_1})\}\}) \ominus_d (lex(c_2) \uplus \{c_2 : \{str(\mathtt{r_2})\}\}),$$

where $lex(c_1') = lex(c_1)$.

The computational algorithm of this operation is shown in Figures 4.15 and 4.16. $\diamond$

Note that, in cases 3 and 4, if the operands are name paths to contexts then the Difference operation operates on their respective lexicons.

Motivation and use of parameter d in the Difference operation are similar to these in the intersection operation.

Example 1: The operation $lex(\mathtt{InfSys}) \ominus lex(\mathtt{DSS})$, returns the lexicon:

$$l_9 = \begin{cases} o_3 : \mathtt{Manos} \\ o_4 : \mathtt{Nicolas}, \mathtt{Nick}. \end{cases}$$

Note that objects o_3 and o_4 are objects contained in c_2 but not in c_3. That is, $D = \{o_3, o_4\}$. These objects are added to the output lexicon in Step 1(a). Also, note that $I = \{o_1, o_2\}$. As I does not contain any context, Step 1(b) is not executed.

Example 2: The operation $\mathtt{ManosView} \ominus \mathtt{AnastasiaView}$ computes the differences between the views of Manos and Anastasia, and returns the lexicon:

$$l_{10} = \begin{cases} o_4 : \\ c_2''' : \mathtt{InfSys} \\ c_3 : \mathtt{DSS} \\ c_1''' : \mathtt{ManosView} \end{cases}$$

c_2''' | $o_4 : \mathtt{Nicolas}, \mathtt{Nick}$

c_1''' | $o_4 :$
 $c_2''' : \mathtt{InfSys}$
 $c_3 : \mathtt{DSS}$

Note that o_4 and c_3 are objects contained in c_1 but not in c_4. Note also that the Difference operation is not recursively applied to the nested subcontexts of `ManosView` and `AnastasiaView`. Therefore, if the user wants to go into more depth, he has to call explicitly the operation `ManosView.InfSys` $\ominus$ `AnastasiaView.InfSys`.

BodyDiff(**Input** $l_1, l_2 : \mathcal{L},\ c_1 : \mathcal{CXT},\ d : Integer$; **Output** $out\_l : \mathcal{L}$).

1. Let $DifO = objs(c_1) - objs(c_2)$.
2. $out\_l = lexUnion(DifO, l_1, \emptyset)$.
   /* the operation $lexUnion$ is shown in Figure 4.10 */
3. Let $DifC = DifO \cap \mathcal{CXT}$.
4. Let $RecCxt$ be the contexts recursively contained in l_1 up to depth d, i.e., in depth $d_i < d$.
   /* contexts contained in l_1 are in depth 1. */
5. If $c_1 \neq NIL$ then $RecCxt = RecCxt \cup \{c_1\}$.
6. $OrigCxt = RecCxt - DifC$.
7. $CopiedCxt = \emptyset$.
8. While $OrigCxt \neq \emptyset$ do
   (a) Find context $c \in OrigCxt$ which is not contained in any other context in $OrigCxt$.
   /* c is contained in both l_1 and l_2 */
   (b) $c' = copyCxt(c)$.
   (c) Set $src(c') = src(c)$.
   (d) $out\_l = out\_l \uplus \{c' : names(c, l_1)\}$.
   (e) If c is contained in some contexts in $CopiedCxt$ then
   replace c with c' in the lexicon of these contexts.
   (f) $OrigCxt = OrigCxt - \{c\}$.
   (g) $CopiedCxt = CopiedCxt \cup \{c'\}$.
9. $elimObj(DifO, CopiedCxt)$.
   /* the operation $elimObj$ is shown in Figure 4.12 */
10. $out\_l = merge(out\_l)$.
    /* the operation $merge$ is shown in Figure 4.10 */
11. End.

Figure 4.15: The algorithm of the operation *BodyDiff*.

1. Lexicon Difference

$\ominus$ (**Input** $l_1, l_2 : \mathcal{L}, d : Integer$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two lexicons and an integer d, and returns the difference up to depth d of these two lexicon. */

1. $out\_l = BodyDiff(l_1, l_2, NIL, d)$.
   /* the operation $BodyDiff$ is shown in Figure 4.15 */
2. End.

2. Context-Lexicon Difference

$\ominus$ (**Input** $r_1 : \mathcal{NP}$, $l_2 : \mathcal{L}, d : Integer$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes a reference r_1 to a context and a lexicon, and an integer d, and returns the difference up to depth d between this context and this lexicon. */

1. $c_1 = lookupOne(r_1)$.
2. $c_1' = copyCxt(c_1)$.
3. $l_1 = lex(c_1) \uplus \{c_1' : \{str(r_1)\}\}$.
4. $out\_l = BodyDiff(l_1, l_2, c_1', d)$.
   /* the operation $BodyDiff$ is shown in Figure 4.15 */
5. End.

3. Context Difference

$\ominus$ (**Input** $r_1, r_2 : \mathcal{NP}, d : Integer$; **Output** $out_l : \mathcal{L}$)

/* This operation takes as input two references r_1 and r_2 to two contexts and an integer d, and returns the difference up to depth d of these contexts. */

1. $c_1 = lookupOne(r_1)$.
2. $c_2 = lookupOne(r_2)$.
3. $c_1' = copyCxt(c_1)$.
4. $c_2' = copyCxt(c_2)$.
5. $l_1 = lex(c_1) \uplus \{ c_1' : \{str(r_1)\} \}$.
6. $l_2 = lex(c_2) \uplus \{ c_2 : \{str(r_2)\} \}$.
7. $out\_l = BodyDiff(l_1, l_2, c_1', d)$.
   /* the operation $BodyDiff$ is shown in Figure 4.15 */
8. End.

Figure 4.16: The algorithm of the Difference operation.

4.3 Properties of the operations

In the course of execution of the Union, Intersection, and Difference operations, nested subcontexts are copied and merged into new contexts. This implies that even the same operation, if executed twice, will result into two different lexicons. However, these two lexicons will bear the equivalence relation defined below.

Definition 4.8 Equivalence relation $\sim$.

We define the equivalence relation $\sim$ between contexts or lexicons, as follows:

1.Let c and c' be contexts. Then

$$c \sim c' \Leftrightarrow (c = c') \ \vee \ (lex(c) \sim lex(c') \ \wedge \ src(c) = src(c'))$$

2.Let l and l' be lexicons. Then

$$\begin{aligned} l \sim l' \Leftrightarrow \ & (\forall o \in \mathcal{S} : \ o{:}N \in l \Leftrightarrow o{:}N \in l') \wedge \\ & (\forall c \in \mathcal{CXT} : \\ & (c{:}N \in l \Rightarrow \ \exists c' : \ c'{:}N \in l' \ \wedge \ c \sim c') \wedge \\ & (c{:}N \in l' \Rightarrow \ \exists c' : \ c'{:}N \in l \ \wedge \ c \sim c')) \end{aligned}$$

where $\mathcal{S}$ denotes the set of simple objects. $\diamond$

It can be easily seen that the relation $\sim$ is *reflexive*, *symmetric*, and *transitive*, i.e., an equivalence relation.

It turns out that the operations of Union and Intersection have the properties of commutativity, associativity, and distributivity over lexicons and contexts, just like ordinary set union and intersection. These properties are important as they offer flexibility in the execution of operations. Specifically, commutativity allows one to ignore the order between two operands. Associativity allows to omit an indication of precedence, in expressions with more than one instance of the operator. Finally, distributivity allows to factor out or to distribute an operand, so as to optimize further processing.

Proposition 4.2 .

Let A, B, and C be references to contexts or lexicons. The following properties hold:

- **Commutativity**:
  (1) $A \uplus B \sim B \uplus A$
  (2) $A \owedge B \sim B \owedge A$
  (3) $A \oplus B \sim B \oplus A$
- **Associativity**:
  (4) $(A \uplus B) \uplus C \sim A \uplus (B \uplus C)$
  (5) $(A \owedge B) \owedge C \sim A \owedge (B \owedge C)$
  (6) $(A \oplus B) \oplus C \sim A \oplus (B \oplus C)$
- **Distributivity**:
  (7) $(A \owedge B) \uplus C \sim (A \uplus C) \owedge (B \uplus C)$
  (8) $(A \uplus B) \owedge C \sim (A \owedge C) \uplus (B \owedge C)$ $\diamond$

Proof:

(1), (2), (3) **Commutativity**.

The reader can easily verify this property by looking at the code of the Union and Intersection operations.

(4) **Union Associativity**.

Let $l_1 = (A \uplus B) \uplus C$, and $l_2 = A \uplus (B \uplus C)$.
We shall prove that: $l_1 \sim l_2$, that is:

$$\forall o \in \mathcal{O}, N \in \mathcal{P}(\mathcal{N}) : \quad o{:}N \in l_1 \Leftrightarrow o{:}N \in l_2 \ \vee \ (\ \exists o' : \ \ o'{:}N \in l_2 \ \wedge \ o \sim o').$$

We will first prove the forward derivation. The backwards derivation is proved similarly.
Let A, B, C be lexicons. We distinguish the following cases:

1. o is a simple object.
2. o is a context.
   (a) o is a context such that $src(o) = o$.
      i. No merging takes place between o and other cleaned[3] contexts during the computation of l_1.
      ii. o is produced by merging o with one or more cleaned contexts.
   (b) o is a cleaned context (i.e., $src(o) \neq o$).
      i. There is only one context o' with $src(o') = src(o)$ contained in the lexicons A, B, or C.
      ii. There exist more than one objects o_i with $src(o_i) = src(o)$ in the lexicons A, B, or C.

Cases 1 and 2.(a).i

1. If $o \in objs(C)$ and $o \notin objs(A \uplus B)$ then we have $o \notin objs(A)$ and $o \notin objs(B)$, and $N = C(o)$.
   Hence, $o{:}N \in B \uplus C$ and $o{:}N \in l_2$.
2. If $o \in objs(A \uplus B)$ and $o \notin objs(C)$ then we have that either $o \in objs(A)$, or $o \in objs(B)$, or both.
   (a) If $o \in objs(A)$ and $o \notin objs(B)$ then $N = A(o)$.
   Hence, $o \notin objs(B \uplus C)$ and because $o{:}N \in A$ we have $o{:}N \in l_2$.
   (b) If $o \in objs(B)$ and $o \notin objs(A)$ then similarly to the previous case we can prove that $N = B(o)$ and $o{:}N \in l_2$.
   (c) If $o \in objs(A)$ and $o \in objs(B)$ then $N = A(o) \cup B(o)$.
   On the other hand, $o{:}B(o) \in B \uplus C$ and $o{:}A(o) \cup B(o) \in A \uplus (B \uplus C)$.
   Hence, $o{:}N \in l_2$.
3. If $o \in objs(A \uplus B)$ and $o \in objs(C)$ then similarly to the previous case we can prove that $N = A(o) \cup B(o) \cup C(o)$ and $o{:}N \in l_2$.

[3] A context c is *cleaned* if $src(c) \neq c$.

Case 2.(a).ii

Without loss of generality, assume that there is context $o_1 \in objs(A)$ with $src(o_1) = src(o)$, $o \in objs(B)$, and there is no context $o_3 \in objs(C)$ with $src(o_3) = src(o)$ (the rest of the cases are proved similarly).
Then, during the computation $A \uplus B$, the $merge$ operation (called at Step 2 of the Lexicon Union algorithm given in Figure 4.11) merges o_1 with o.
The result of this merging is again the context o, but now o has names $A(o_1) \cup B(o)$.
Note that as there is no context $o_3 \in objs(C)$ with $src(o_3) = src(o)$ no other merging will take place and thus, l_1 will contain o with names $N = A(o_1) \cup B(o)$.

On the other hand, $B \uplus C$ contains o with names $B(o)$.
Then, the $merge$ operation (called at Step 2 of the Lexicon Union algorithm computing l_2) merges o_1 with o resulting again in the context o, but now with names $A(o_1) \cup B(o)$.
Hence, $o{:}N \in l_2$.

Case 2.(b).i

Without loss of generality, assume that o' is contained in only one of A, B, or C (call this lexicon D) and $o = o'$.
Similarly to the previous cases we can prove that $N = D(o)$ and $o{:}N \in l_2$.

Case 2.(b).ii

Without loss of generality, assume that there are contexts $o_1 \in objs(A)$ and $o_2 \in objs(B)$ such that $src(o_1) = src(o_2) = src(o)$, and there is no context $o_3 \in objs(C)$ with $src(o_3) = src(o)$.
Then, $o \in objs(A \uplus B)$ and o is produced by merging o_1, o_2 through the $merge$ operation (called at Step 2 of the Lexicon Union algorithm computing $A \uplus B$).
Hence, $N = A(o_1) \cup B(o_2)$.

On the other hand, note that $o_2{:}B(o_2) \in B \uplus C$.
Therefore, there is a context o' such that $o'{:}N \in l_2$, which is produced by merging o_1, o_2 through the $merge$ operation (called during the computation of l_2).
Obviously, $o \sim o'$.

Let A, B be lexicons, and C be a reference to a context (call this context c).
Then, $l_1 = (A \uplus B) \uplus C = (A \uplus B) \uplus (lex(c) \uplus \{c{:}str(C)\})$,
and $l_2 = A \uplus (B \uplus C) = A \uplus (B \uplus (lex(c) \uplus \{c{:}str(C)\}))$.
As $lex(c) \uplus \{c : str(C)\}$ is a lexicon and associativity holds among lexicons, it follows that associativity holds among A, B, C as well.

Similarly, we can prove the associativity property in the case that any of A, B, or C is a context.

(5), (6) **Intersection Associativity**.

In the following, we will prove the property (6). We can prove the property (5) similarly.

Let $l_1 = (A \Cap B) \Cap C$, and $l_2 = A \Cap (B \Cap C)$.
We shall prove that: $l_1 \sim l_2$, that is:

$$\forall o \in \mathcal{O}, N \in \mathcal{P}(\mathcal{N}) : \quad o{:}N \in l_1 \Leftrightarrow o{:}N \in l_2 \ \vee \ (\ \exists o' : \ \ o'{:}N \in l_2 \ \wedge \ o \sim o').$$

We will first prove the forward derivation. The backwards derivation is proved similarly.

Let A, B, C be lexicons. We distinguish the following cases:

1. o is a simple object.

2. o is a context.

   (a) o is a context such that $src(o) = o$.

      i. No merging takes place between o and other cleaned contexts during the computation of l_1.

   (b) o is a cleaned context (i.e., $src(o) \neq o$).

      i. There is only one context c contained in A, B, or C that recursively contains objects in $ComO(A \uplus B, C)$ and $src(c) = src(o)$.

      ii. There exist more than one contexts c_i that recursively contain objects contained in $ComO(A \uplus B, C)$ and $src(c) = src(o)$.

Cases 1 and 2.(a).i

As o is not a cleaned subcontext, $o \in ComO(A \uplus B, C)$ and $N = (A \uplus B)(o) \cup C(o)$.
Therefore, $o \in objs(A \uplus B)$ and $o \in objs(C)$.
From this it follows that $o \in ComO(A, B)$ and $(A \uplus B)(o) = A(o) \cup B(o)$.
Thus, $o \in objs(A)$ and $o \in objs(B)$.
It now easily follows that $o{:}\, B(o) \cup C(o) \in B \uplus C$ and thus $o{:}\, A(o) \cup (B(o) \cup C(o)) \in l_2$.
Hence, $o{:}\, N \in l_2$.

Case 2.(b).i

Without loss of generality, assume that there is context $c \in objs(A)$ and $c \notin objs(B) \cup objs(C)$.
Then, during the operation $A \uplus B$, a new cleaned context c' is produced in the Step 9b of Lexicon Intersection Algorithm by copying context c.
Then, the objects of c' which are not in I or which do not recursively contain objects in I are eliminated from c' through the operation $elimObj(ComO(A, B), \{c', \ldots\})$.
Thus, $c'{:}\, A(c) \in objs(A \uplus B)$.
Similarly, during the operation $(A \uplus B) \uplus C$, the cleaned context o is produced by copying c'.
Context o is cleaned through the operation $elimObj(ComO(A \uplus B, C), \{o, \ldots\})$.
Note also that $N = A(c)$.

Similarly, on the other hand, during the operation $A \uplus (B \uplus C)$ a new cleaned context c'' is produced by copying c such that $c''{:}\, A(c) \in l_2$.
Context c'' is cleaned through the operation $elimObj(ComO(A, B \uplus C), \{o, \ldots\})$.
It can be easily proved $ComO(A \uplus B, C) = ComO(A, B \uplus C)$.
Hence, $c''{:}\, N \in l_2$ and $o \sim c''$.

Case 2.(b).ii

Without loss of generality, assume that there exist contexts $c_1 \in objs(A)$ and $c_2 \in objs(B)$ such that $src(c_1) = src(c_2) = src(o)$, and there is no context $c_3 \in objs(C)$ which recursively contain objects in $ComO(A \uplus B, C)$ and $src(c_3) = src(o)$.
Then, during the operation $A \uplus B$, two new cleaned contexts, c_1' and c_2', are produced by

copying contexts c_1 and c_2, respectively.
Then, contexts c_1' and c_2' are cleaned through the operation:

$$elimObj(ComO(A, B), \{c_1', c_2', \ldots\}).$$

It also holds that $src(c_1') = src(c_1)$ and $src(c_2') = src(c_2)$.
As $src(c_1) = src(c_2)$ then $src(c_1') = src(c_2')$ and contexts c_1' and c_2' are merged through the $merge$ operation, and a new context c' is produced with names $A(c_1) \cup B(c_2)$.
Then, during the operation $(A \uplus B) \uplus C$, the cleaned context o is produced by copying c'.
Context c' is cleaned through the operation $elimObj(ComO(A \uplus B, C), \{o, \ldots\})$.
Also, $N = A(c_1) \cup B(c_2)$.

On the other hand, during the operation $B \uplus C$, a new cleaned context c_2'' is produced by copying c_2 and having name $B(c_2)$.
Context c_2'' is cleaned through the operation $elimObj(ComO(B, C), \{c_2'', \ldots\})$.
Then, during the operation $A \uplus (B \uplus C)$, two new cleaned contexts, c_1'' and c_2''', are produced by copying contexts c_1 and c_2'', and $A(c_1)$ and $B(c_2)$, respectively.
Then, contexts c_1'' and c_2''' are cleaned through the operation

$$elimObj(ComO(A, B \uplus C), \{c_1'', c_2''', \ldots\}).$$

As $src(c_1'') = src(c_2''')$, contexts c_1' and c_2''' are merged through the $merge$ operation and a new context c'' is produced with names $A(c_1) \cup B(c_2)$.
As $ComO(A \uplus B, C) = ComO(A, B \uplus C)$, it follows that $c'' : N \in l_2$ and $o \sim c''$.

Let A, B be lexicons, and C be a reference to a context (call this context c).
Then, there are contexts c', c'' such that

$$l_1 = (A \uplus B) \uplus C = (A \uplus B) \uplus (lex(c) \uplus \{c' : str(C)\})$$

and

$$l_2 = A \uplus (B \uplus C) = A \uplus (B \uplus (lex(c) \uplus \{c'' : str(C)\})).$$

As $lex(c) \uplus \{c' : str(C)\}$ and $lex(c) \uplus \{c'' : str(C)\}$ are lexicons and associativity holds among lexicons, it follows that associativity holds among A, B, C as well.

Similarly, we can prove the associativity property in the case that any of A, B, or C is a context.

(7) **Distributivity**.

Let $l_1 = (A \uplus B) \uplus C$, and $l_2 = (A \uplus C) \uplus (B \uplus C)$.
We shall prove that: $l_1 \sim l_2$, that is:

$$\forall o \in \mathcal{O}, N \in \mathcal{P}(\mathcal{N}) : \quad o : N \in l_1 \Leftrightarrow o : N \in l_2 \ \vee \ (\ \exists o' : \ \ o' : N \in l_2 \ \wedge \ o \sim o').$$

We will first prove the forward derivation. The backwards derivation is proved similarly.

Let A, B, C be lexicons. We distinguish the following cases:

1. o is a simple object.

2. o is a context.

   (a) o is a context such that $src(o) = o$.

      i. No merging takes place between o and other cleaned contexts during the computation of l_1.

      ii. o is produced by merging o with one or more cleaned contexts.

   (b) o is a cleaned context (i.e., $src(o) \neq o$).

Cases 1 and 2.(a).i

Then, o is contained in either $A \curlywedge B$, or C, or both.
Without loss of generality, assume that o is contained in $A \curlywedge B$, but not in C.
Then, $o \in ComO(A, B)$ and $N = A(o) \cap B(o)$.
Therefore, $o \in objs(A)$ and $o \in objs(B)$.
Thus, $o{:}A(o) \in A \uplus C$ and $o{:}B(o) \in B \uplus C$.
From this it follows that $o \in ComO(A \uplus C, B \uplus C)$ and $o{:}(A \uplus C)(o) \cap (B \uplus C)(o) \in l_2$.
Hence $o{:}A(o) \cup B(o) \in l_2$.

Case 2.(a).ii

Without loss of generality, assume that o is contained in $A \curlywedge B$, and there is a cleaned context c contained in C such that $src(c) = o$.

Then, on one hand, $o \in ComO(A, B)$ and $(A \curlywedge B)(o) = A(o) \cap B(o)$.
Therefore, $o \in objs(A)$ and $o \in objs(B)$.
The $merge$ operation (called during the computation of l_1) merges o with c resulting again in the context o, but now with names $(A \curlywedge B)(o) \cup C(c)$.
Hence, $N = (A(o) \cap B(o)) \cup C(c)$.

On the other hand, the $merge$ operation (called during the computation of $A \uplus C$) merges o with c, resulting again in the context o, but now with names $A(o) \cup C(c)$.
Similarly, the $merge$ operation (called during the computation of $B \uplus C$) merges o with c, resulting again in the context o, but now with names $B(o) \cup C(c)$.
Therefore, $o \in ComO(A \uplus C, B \uplus C)$ and $o{:}(A \uplus C)(o) \cap (B \uplus C)(o) \in l_2$.
That is $o{:}(A(o) \cup C(c)) \cap (B(o) \cup C(c)) \in l_2$.
Hence, since distributivity of set union and set intersection is hold, $o{:}N \in l_2$.

Case 2.(b)

Without loss of generality, assume that there is a context c contained in $A \curlywedge B$ such that $src(c) = src(o)$, and there is a context c' contained in C such that $src(c') = src(o)$.

As o is contained in $A \curlywedge B$, assume that there are contexts c_1, c_2 contained in A, B, respectively, which both recursively contain objects in $I = ComO(A, B)$ and $src(c_1) = src(c_2) = src(o)$.
Then, during the operation $A \curlywedge B$, two new cleaned contexts, c_1' and c_2' are produced in the Step 9b of Lexicon Intersection Algorithm by copying contexts c_1 and c_2, respectively.
Then, the objects of c_1' and c_2' that are not in I or do not recursively contain objects in I are eliminated from these contexts through the operation $elimObj(I, \{c_1', c_2', \ldots\})$.
As $src(c_1') = src(c_2') = src(o)$, contexts c_1' and c_2' are merged through the $merge$ operation, and the new context c is produced with names $A(c_1) \cap B(c_2)$.

Then, during the operation $(A \upharpoonright B) \uplus C$, contexts c, c' are merged through the $merge$ operation and the context o is produced with names $N = (A(c_1) \cap B(c_2)) \cup C(c')$.

On the other hand, during the operation $A \uplus C$, contexts c_1 and c' are merged through the $merge$ operation, and a new context c_1'' is produced with names $A(c_1) \cup C(c')$.
Similarly, during the operation $B \uplus C$, contexts c_2 and c' are merged through the $merge$ operation, and a new context c_2'' is produced with names $B(c_2) \cup C(c')$.
Note that contexts c_1'' and c_2'' recursively contain objects in I and thus they also recursively contain objects in $I' = ComO(A \uplus C, B \uplus C)$.
Then, during the operation $(A \uplus C) \upharpoonright (B \uplus C)$, contexts c_1'' and c_2'' are first cleaned through the operations $elimObj(I', \{c_1'', c_2'', \ldots\})$, and then merged through the $merge$ operation.
This will produce a new context c'' with names $(A(c_1) \cup C(c')) \cap (B(c_2) \cup C(c'))$.
Hence, $c'' : N \in l_2$ and, since I' contain contexts equivalent to the contexts contained in $I \cup objs(C)$, $o \sim c''$.

Let A, B be lexicons, and C be a reference to a context (call this context c).
Then, $l_1 = (A \upharpoonright B) \uplus C = (A \upharpoonright B) \uplus (lex(c) \uplus \{c{:}\, str(C)\})$,
and $l_2 = A \upharpoonright (B \upharpoonright C) = (A \upharpoonright (lex(c) \uplus \{c{:}\, str(C)\}) \upharpoonright (B \upharpoonright (lex(c) \uplus \{c{:}\, str(C)\})$.
As $lex(c) \uplus \{c : str(C)\}$ is a lexicon, and associativity holds among lexicons, it follows that associativity holds among A, B, C as well.

Similarly, we can prove the associativity property in the case that any of A, B, or C is a context.
□

For example (see Figure 4.5), assume the current context to be the context IB. The operation

$$(\texttt{ManosView.InfSys} \upharpoonright \texttt{AnastasiaView.DSS}) \uplus \texttt{ManosView} \tag{4.1}$$

computes the commonalities between the Information Systems lab as seen from Manos and the DSS lab as seen from Anastasia and then combines these commonalities with the view of Manos for the Institute to get a wider view of it. Let l_1 be the intermediate lexicon returned by the operation

`ManosView.InfSys` $\upharpoonright$ `AnastasiaView.DSS`

and let l_2 be the lexicon returned by the Operation 4.1. Then we have:

$$l_1 = \begin{cases} o_1 : \texttt{Panos} \\ c_2' : \texttt{ManosView\_InfSys} \\ c_5' : \texttt{AnastasiaView\_DSS} \end{cases}$$

c_2' $\boxed{o_1 : \texttt{Panos}}$

c_5' $\boxed{o_1 : \texttt{Panos}, \texttt{Constantopoulos}}$

$$l_2 = \begin{cases} o_1 : \texttt{Panos}, \texttt{Dr\_Constantopoulos} \\ o_4 : \\ o_5 : \texttt{professor} \\ c_2 : \texttt{ManosView\_InfSys}, \texttt{InfSys} \\ c_3 : \texttt{DSS} \\ c_5' : \texttt{AnastasiaView\_DSS} \\ c_1 : \texttt{ManosView} \end{cases}$$

Note that during the computation of the operation $l_1 \uplus$ `ManosView` context c_2 is merged with context c_2' into context c_2 as $src(c_2') = src(c_2) = c_2$ (see Step 1(c) of the Union algorithm on page 44 and the detailed algorithms of the operations of merge and union shown in Figures 4.10 and 4.11), respectively.

On the other hand, the operation

$$(\texttt{ManosView.InfSys} \uplus \texttt{ManosView}) \upharpoonright (\texttt{AnastasiaView.DSS} \uplus \texttt{ManosView}) \tag{4.2}$$

yields two wider views of the Institute as seen from Manos by (i) combining `ManosView`, context c_1, with the Information Systems lab as seen from Manos, context c_2, (call the returned lexicon l_3) and (ii) combining `ManosView` with the DSS lab as seen from Anastasia, context c_5, (call the returned lexicon l_4), and then computes the commonalities of these two wider views. Let l_5 be the lexicon returned by the Operation 4.2. Then we have:

$$l_3 = \begin{cases} o_1 : \texttt{Panos}, \\ \quad \texttt{Dr\_Constantopoulos} \\ o_2 : \texttt{head} \\ o_3 : \texttt{Manos} \\ o_4 : \texttt{Nikos}, \texttt{Nick} \\ o_5 : \texttt{professor} \\ c_2 : \texttt{ManosView\_InfSys}, \\ \quad \texttt{InfSys} \\ c_3 : \texttt{DSS} \\ c_1 : \texttt{ManosView} \end{cases} \qquad l_4 = \begin{cases} o_1 : \texttt{Panos}, \\ \quad \texttt{Constantopoulos}, \\ \quad \texttt{Dr\_Constantopoulos} \\ o_4 : \\ o_5 : \texttt{professor} \\ o_7 : \texttt{Anastasia} \\ c_2 : \texttt{InfSys} \\ c_3 : \texttt{DSS} \\ c_5 : \texttt{AnastasiaView\_DSS} \\ c_1 : \texttt{ManosView} \end{cases}$$

$$l_5 = \begin{cases} o_1 : \texttt{Panos}, \\ \quad \texttt{Dr\_Constantopoulos} \\ o_4 : \\ o_5 : \texttt{professor} \\ c_2 : \texttt{ManosView\_InfSys}, \\ \quad \texttt{InfSys} \\ c_3 : \texttt{DSS} \\ c_5'' : \texttt{AnastasiaView\_DSS} \\ c_1 : \texttt{ManosView} \end{cases} \qquad c_5'' \boxed{\begin{array}{l} o_1 : \texttt{Panos}, \\ \quad \texttt{Constantopoulos} \end{array}}$$

According to property (6), lexicons l_2 and l_5 are equivalent.

We now define an important class of lexicons, called *operational lexicons*, which is closed over the operations Union, Intersection, and Difference. This closure property is expressed in Lemma 4.1 and Theorem 4.1. In the following, we shall call *root context* any context contained in a lexicon l (resp. context c) which is not recursively contained in any other context contained in l (resp. c).

Definition 4.9 Operational lexicon.
A lexicon l is called *operational* iff

1. it is a well-defined lexicon,
2. if c is a root context of l then $src(c)$ is well-defined, and
3. any object of l which is not a root context is recursively contained in a root context of l. $\diamond$

Lemma 4.1 Closure of the operationality property: two lexicons.
Let l_1, l_2 be two operational lexicons. Assume that every root context c of l_1 (resp. l_2) has a name n in l_1 (resp. l_2) such that there is no name n in l_2 (resp. l_1). Then the operation $l_1 \odot l_2$, where $\odot \in \{ \uplus , \Cap , \ominus \}$, results in an operational lexicon. $\diamond$

Proof:

Let $l = l_1 \odot l_2$. We shall prove that l is an operational lexicon, that is:

1. We will first prove that l is a well-defined context.

   We will prove that for each object o of l there is a unique reference of o w.r.t. l.
   Let first o be a context, which comes from a root context c of l_1 or l_2 ("comes from" means that either (i) o is the context c, or (ii) o is the result of cleaning the context c, or (iii) o is the result of merging a context of l_1 with a context of l_2, one of which is c).
   Assume that c is a context of l_1 (proceed similarly if c is a context of l_2).
   From the definition of the Union, Intersection, and Difference operations, we have

$$names(c, l_1) \subseteq names(o, l) \tag{4.3}$$

As c is a root context of l_1, there is a name $n \in names(c, l_1)$ such that there is no name n w.r.t. l_2.
Equation (4.3) implies that $n \in names(o, l)$.
We will prove that n is a unique reference of o w.r.t. l.
Assume that there is another object o' such that $n \in npaths(o', l)$. Then, there is an object o'' contained in l_1 or l_2 such that o' comes from $o'' \neq o$.
Then, $n \in names(o'', l_1)$ or $n \in names(o'', l_2)$.
However, this is impossible because n is a unique name w.r.t. l_1 and there is no name n w.r.t. l_2.

Any other object o of l comes from objects that are not root contexts, but they are recursively contained in a root context (call this context c).
Since o is contained in l, there must be a context c' of l coming from c (this is because of the definition of the Union, Intersection and Difference operations).
Since c is well-defined, c' is well-defined as well.
Thus, there is a unique reference r of o w.r.t c'.
As c is a root context, we proved above that there is n such that n is a unique name of c' w.r.t. l.
Thus, $n.r$ is a unique reference of o w.r.t. l.

We shall now prove that every nested subcontext of l satisfies the acyclicity property.
As every nested subcontext of l_1, l_2 satisfies the acyclicity property, it can be easily seen that every nested subcontext of l satisfies the acyclicity property as well.

2. We will now prove that if c is a root contexts of l then $src(c)$ is well-defined.

   Let c be a root context of l. Then, c either (i) is a root context of l_1 or l_2, or (ii) is the result of cleaning a root context c' of l_1 or l_2, or (iii) is the result of merging a context c' of l_1 with a context c'' of l_2.
   For case (i), as l_1 and l_2 are operational contexts, $src(c)$ is a well-defined context.
   Similarly for case (ii), $src(c) = src(c')$, and hence $src(c)$ is a well-defined context.
   For case (iii), c' or c'' should be root context.
   Thus, $src(c')$ or $src(c'')$ is a well-defined context.
   Therefore, $src(c) = src(c') = src(c'')$ is a well-defined context as well.

3. We will now prove that any object of l which is not a root context is recursively contained in a root context of l.

   Let o be an object of l which is not a root context.
   Then, o comes from an object o' of l_1 or l_2, which either (i) is a root context of l_1 or l_2, or (ii) is recursively contained in a root context of l_1 or l_2.
   Consider first case (i), and without loss of generality, assume that o' is a root context of l_1.
   As o is not a root context of l, o' is recursively contained in a root context c' of l_2.
   Let c be the context of l which comes from c'.
   Obviously, c is a root context of l and o is recursively contained in c.
   Consider now case (ii), and without loss of generality, assume that o is recursively contained in a root context c' of l_1.
   Let c be the context of l which comes from c'.
   Obviously, c is a root context of l and o is recursively contained in c. □

Theorem 4.1 Closure of the operationality property: arbitrary number of lexicons.
Let $l_1, \ldots l_k$ be operational lexicons. If every root context c of l_i has a name n in l_i such that there is no name n in any of $l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_k$, then the sequence of operations $l_1 \odot_1 \ldots \odot_{k-1} l_k$, where the operations $\odot_i \in \{ \uplus, \text{⍟}, \text{⍟}, \ominus\}$ are executed in any order, results in an operational lexicon. $\diamond$

Proof: From Lemma 4.1, the operation $l_i \odot_i l_{i+1}$ results in an operational context.
Therefore, we can compute the sequence of operations $l_1 \odot_1 \ldots \odot_{k-1} l_k$ through a sequence of computations of the form $l \odot l'$, where l and l' are operational lexicons and satisfy the condition of Lemma 4.1.
Thus, the sequence of operations $l_1 \odot_1 \ldots \odot_{k-1} l_k$ will result in an operational lexicon. $\square$

The following theorem expresses that the Union, Intersection, and Difference operations preserve the well-definedness property of contexts.

Theorem 4.2 Closure of the well-definedness of contexts.
Let $\mathtt{r_1}, \ldots, \mathtt{r_k}$ be name paths of the well-defined contexts $c_1, \ldots, c_k$. If $str(\mathtt{r_i})$ is not a name of an object in any of $c_1, \ldots, c_k$, then the sequence of operations $\mathtt{r_1} \odot_1 \ldots \odot_{k-1} \mathtt{r_k}$, where the operations $\odot_i \in \{ \uplus, \text{⍟}, \text{⍟}, \ominus\}$ are executed in any order, results in a well-defined lexicon. $\diamond$

Proof: Note that for any $i \leq k$

$$c_i \odot_i c_{i+1} = (lex(c_i) \ \uplus \ \{c_i' : str(r_i)\}) \odot_i (lex(c_{i+1}) \ \uplus \ \{c_{i+1}' : str(r_{i+1})\})$$

where c_i' and c_{i+1}' are determined according to the particular operation $\odot_i$ (see the definitions of the Union, Intersection, and Difference operations).
Note also that as c_i is a well-defined lexicon, $l_i = lex(c_i) \ \uplus \ \{c_i' : str(r_i)\}$ results in an operational lexicon with root context c_i'.
As $str(r_i)$ is not a name of an object w.r.t. each lexicon $l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_k$, all conditions of Theorem 4.1 are met and hence $r_1 \odot_1 \ldots \odot_{k-1} r_k$ results in a well-defined lexicon. $\square$

The closure of well-definedness of context under the operations of context union, intersection, and difference ensures that unique external identification of objects and acyclicity are preserved, after applying the above operation on contexts. Thus, no naming conficts and no cycles will appear in the resulting contexts. Operations on contexts defined in other works [84, 85] lack this ability. Thus, in these works, information from the original contexts may get lost in the result of an operation, since conflicts appear and their conflict resolution strategy may cause units to be inaccessible in the resulting context. Operations in these works do not satisfy the properties of commutativity, associativity, and distributivity.

4.4 Discussion

As mentioned in the introduction, the notion of context has appeared in several areas and has been treated in various ways depending on the purposes of the particular application. However, the semantics given to the notion of context in those areas are not always the same and the various semantics are not always comparable. In this section, we compare our approach with other approaches that treat the notion of context in a comparable way.

Our model has been mainly inspired by the work of Mylopoulos and Motschnig-Pitrik [84], and incorporates previous work by Theodorakis and Constantopoulos [126]. Specifically, in [84], Mylopoulos and Motschnig-Pitrik proposed a general mechanism for partitioning information bases using the concept of context. They introduced a generic framework for contexts and discussed naming conventions, operations on contexts, authorization, and transaction execution. However, they impose a strict constraint on naming, whereby objects (called *information units*) are assigned unique names w.r.t. a context. Because of this constraint, several naming conflicts appear in operations among contexts, which the authors resolve in rather arbitrary ways. In addition, operations among contexts, such as *union* (called *addition*) and *intersection* (called *product*), are deprived of such useful properties as commutativity, associativity, and distributivity, and thus also can yield unexpected results. In [84], the major problem of the context union and context intersection operations is that it is possible for an object in the output context to have *no* name, even though it originally had one or more names. This can happen if an object of one input context has a name in common with an object of the other input context. For example, consider two contexts c and c' which correspond to two companies, the contents of c and c' being the employees of these two companies, respectively. Assume now that an employee in the first company has the same name with another employee in the second company. Then, the union of the contexts c and c' contains these two employees, but one of them will have no name. Such results might seriously hinder the applicability of this otherwise appealing framework.

In [126, 127], Theodorakis and Constantopoulos proposed a naming mechanism based on the concept of context, in order to resolve several naming problems that arise in information bases, such as object names being ambiguous, excessively long, or unable to follow the changes of the environment of the object. These naming problems also resolved by contextualization mechanism defined in this chapter. The information base is assumed to be structured with the traditional abstraction mechanisms of classification, generalization, and attribution. A context is identified by a node and a link class, called *pivot elements* of the context. A link class could be either an attribute class, or the instance-of relation, or the ISA relation. The contents of a context consist of objects and links which are associated with the pivot elements of the context. This definition of context allows the information base to be decomposed into partitions on the basis of one of the traditional abstraction mechanisms. Finally, relative naming is supported, as well as nesting of non contexts. However, that approach imposes a hierarchical structure on contexts, i.e., a context may be contained in only one other context, which is rather restrictive. Context defined in the dissertation is more general and can be easily embedded to any data model.

HAM [17] is a general purpose abstract machine that supports contexts. In HAM, a graph usually contains all the information regarding a general topic and contexts are used to partition the data within a graph. Therefore, a context may contain nodes, links, or other contexts. Contexts are organized hierarchically, i.e., a context is contained in only one other context. By contrast, in our model, a context may be contained in more than one contexts. Contexts in HAM have been used to support configurations, private workspaces, and version history trees [27]. HAM provides a set of context editing, context inquiry, and context attribute operations. All the context editing operations of HAM, namely *createContext*, *destroyContext*, *compactContext*, and *mergeContext*, can be simulated in our model using its operations. On the other hand, HAM does not support name relativism. Inquiries on and attributes of contexts can be supported by our model (see chapter 8), however they are outside of the scope of this chapter.

In [110], the notion of context is used to support collaborative work in hypermedia design. A context node contains links, terminal nodes, and other context nodes. Furthermore, context nodes are specialized into annotations, public bases, hyperbases, private bases, and user contexts. Using this notion of context, the authors define operations *check-in* and *check-out* for hypermedia objects. However, there is no support for name relativism, neither are generic operations on contexts provided.

The notion of context has also appeared in the area of heterogeneous databases [101, 86, 55]. There, the word "context" refers to the implicit assumptions underlying the manner in which an agent represents or interprets data. To allow exchange between heterogeneous information systems, information specific to them can be captured in specific contexts. Therefore, contexts are used for interpreting data. At present our model cannot be compared with these works, because it does not address heterogeneous databases, as we assume a single Information Base (which guarantees that real world objects are represented by unique objects in the Information Base).

4.5 Summary

In this chapter, we developed a model for representing contexts in information bases along with a set of operations for creating, updating, combining, and comparing contexts. A context is treated as a special object which is associated to a set of objects and a lexicon, i.e., a binding of names to these objects. Contexts may overlap, in the sense that an object may be contained in more than one contexts simultaneously. Contexts may also be nested, in the sense that a context may contain other contexts. Also, a context may be contained in more than one contexts.

The main contributions of this work are:

- It allows an object to have zero, one, or more names, not necessarily unique, w.r.t. a context. Therefore, we can handle synonymous, homonymous, and anonymous objects. Possible name ambiguities are resolved by assuming that objects contained in well-defined contexts have *at least one* unique external identification (i.e., absolute name).
- The operations of context union, intersection satisfy the important properties commutativity, associativity, and distributivity, which offer flexibility in the execution of these operations.
- The operations of context union, intersection, and difference preserve the well-definedness of contexts. This ensures that unique external identification of objects is preserved, after applying the above operations on contexts.

Chapter 5

Applying Context in a Cooperation Environment

The potential usefulness of context in supporting such constructs as versions, configurations ans workspaces has already been indecated in [84]. In this chapter we present a comprehensive example that illustrates the use of context, as defined in Chapter 4, in a cooperation environment, where workspaces, versions and configurations all are important ingredients. In the course of the example we show how application-specific high-level comments can be built employing the context construct.

5.1 Cooperation environment

A cooperation environment is usually organized into named repositories, called *workspaces*, to allow workers to share information concerning the work done on an object, in a secure and orderly manner [56, 21]. In a cooperation environment, there are three kinds of workspaces: *public*, *group*, and *private*.

The public workspace contains fully verified (i.e., released) and finished object versions, which have reached absolute stability and cannot be updated or deleted. However, any worker can read this workspace, and can add new object versions to it.

The group workspace contains object versions that have reached reasonable stability, and therefore can be shared by two or more workers. Thus, the combination of work-in progress between different workers is achieved. This process is necessary before a version is finalized and migrates to the public workspace. Object versions of the group workspace cannot be updated but they can be deleted.

The private workspace consists of a number of user workspaces. Each user workspace is owned and can be accessed only by a specific user. User workspaces contain temporary object versions which are expected to undergo a significant amount of update before reaching a reasonably stable state (and moved to the group or to the public workspace). Therefore, object versions of a user workspace can be updated or deleted by its user.

Object versions can be moved into and out of the public workspace through the $check\text{-}in$ and $check\text{-}out$ operations, and into and out of the group workspace through the $import$ and $export$ operations. A user checks a version out of the public workspace into his private workspace,

where he can make changes. The new version is possibly exported to the group workspace for integration testing with other objects. To correct errors, the version has to be imported to the private workspace. Finally, a new verified version is checked in the public workspace and is linked (through a version history link) to the original public version from which it was derived. At this point, the version history of the object has been updated.

An object is, in general, composed of other objects that are either atomic or composite. In our model, a version of an atomic object can be thought of as a simple object. Recall that a simple object is an object of the Information Base that is not a context. A *configuration* is a version of a composite object, composed of particular versions of its components. Therefore, a configuration can be thought of as a context that contains versions of its components. We refer to contexts that represent configurations as *configuration contexts*.

A version history of an object can be thought of as a context that contains (a) versions of the object, and (b) links from one version to another that indicate version derivation. We call such contexts, *history contexts*. The context types described above, are ISA-related as shown in Figure 5.1, thus forming a hierarchy of contexts.

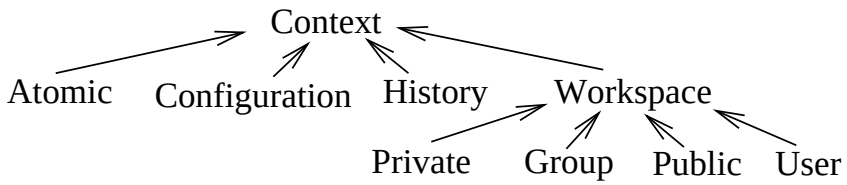


Figure 5.1: Context types of the cooperation environment.

A cooperation environment can be thought of as an Information Base (IB), containing six contexts: ATOMIC, CONFIG, HISTORY, PUBLIC, PRIVATE, and GROUP (see Figure 5.2). The context ATOMIC contains all versions of atomic objects. The context CONFIG contains all configuration contexts, and the context HISTORY contains all history contexts. The context PUBLIC contains all objects in the public workspace, which we assume to be history contexts, and the context PRIVATE contains all the user contexts. A user context may contain history contexts, configuration contexts, and atomic objects. A user context may also contain results of operations on contexts. The context GROUP essentially contains results of operations on contexts.

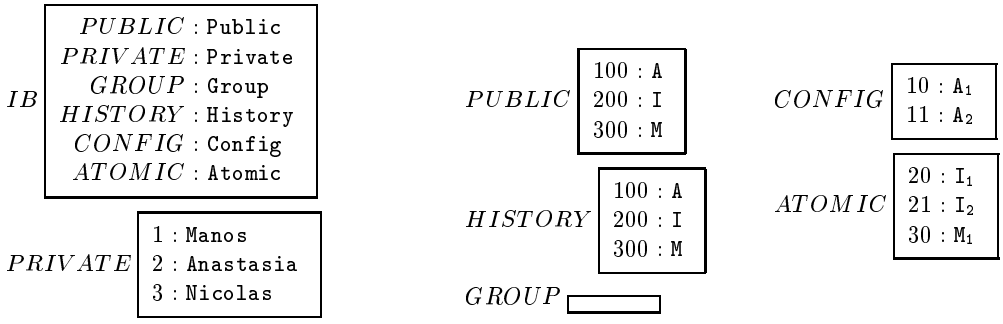


Figure 5.2: Initial lexicons of IB and the six contexts of the cooperation scenario.

5.2 Cooperation scenario

We consider a cooperation scenario in which three authors cooperate on the revision of an article, composed of an introduction and a main section. The initial state of our cooperation scenario is shown in Figures 5.2 and 5.3. In Figure 5.3, we use the following conventions: A symbol of the form $o : \mathtt{n}_1, \mathtt{n}_2, \ldots$ denotes object o with names $\mathtt{n}_1, \mathtt{n}_2, \ldots$, e.g. $100 : \mathtt{A}$ denotes object 100 with a single name `A`. Solid line rectangles represent workspaces, dashed line rectangles represent history contexts, rounded solid line boxes represent configuration contexts, and thick dots represent atomic objects.

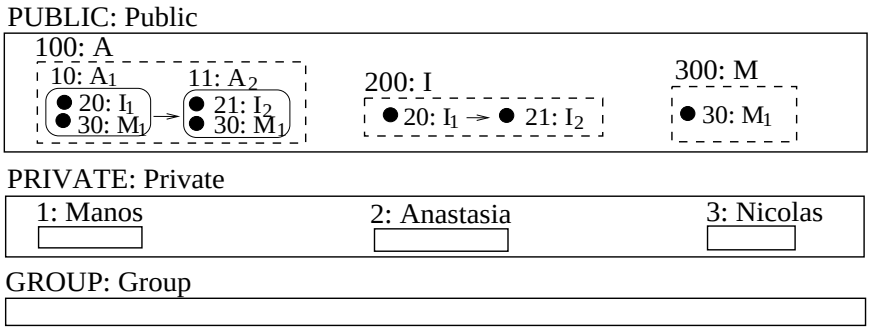


Figure 5.3: Initial state of the cooperation scenario.

Specifically, the initial state of the Information Base is as follows (see Figures 5.2 and 5.3):

- The context PUBLIC contains a history context for the article, and a history context for each component of the article. The history context for the article is context 100 with name `A`, the history context for the introduction is context 200 with name `I`, and the history context for the main section is the context 300 with name `M`, as shown in Figures 5.2 and 5.3. The names `A`, `I` and `M` stand for "Article", "Introduction" and "Main section", respectively[1]. Here, versions of the introduction and the main section are simple objects, as any piece of (unstructured) text is considered to be an atomic object. Context 100 contains two contexts (these are 10 and 11) representing two different versions of the article, as well as a link object from context 10 to context 11. Similarly, contexts 200 and 300 contain versions of the introduction and the main section, respectively, as well as link objects.

- The context PRIVATE contains three user contexts, one for each author. The first author is assigned the user context 1 with name `Manos`, the second author is assigned the user context 2 with name `Anastasia`, and the third author is assigned the user context 3 with name `Nicolas`.

- The context GROUP is initially empty.

We refer to a user workspace as the *home workspace* of the corresponding user. We assume that each user has his own variable *current context* (`CC`) whose initial value is his home workspace. For each user, the value of the variable `Username` is his login name. Also, the

[1] In practice one would use meaningful names instead of `A`, `I` and `M`, e.g. `Article_on_Contexts` instead of `A`.

name of his home workspace in the context PRIVATE, is his login name. Finally, the value of the variable Home is the global name path of the home workspace of the user. For example, for user $Manos$, CC $= 1$, Username $=$ Manos, and Home $=$ @.Private.Manos. In the following, whenever we refer to the variables CC, Home, and Username we use their values. Variables are written in a special character font to be distinguished from strings.

5.3 Cooperation commands

For the revision of the article, each author has four commands at his disposal, as described below. These commands are high level operations, implemented using the context operations of the model. The full code of the operations is given in Appendix A. An example of their use is given in the following section.

- **check-out(**r, n**)**

  This operation takes as input a name path r in the public workspace and a name n, and does the following:

  1. Copies the history context of the version referred to by r, from the public workspace into the home workspace of the user, under the same name.
  2. Copies the version referred to by r (call this version v), from the public workspace into the CC (call this copy v').
  3. Adds v' into the copy of the history context, under the name n.
  4. Updates the copy of the history context by adding a link from v to v'. ⋄

- **check-in(**r, h, n**)**

  This operation takes as input a name path r w.r.t. CC, a name path h w.r.t. the public workspace, and a name n. Then, it copies the version referred to by r from the CC into the history context of the public workspace referred to by h, under the name n. ⋄

- **export(**$exportedListOfContexts, exportedCxtName$**)**

  This operation takes as input a set of name paths $exportedListOfContexts$ w.r.t. the CC, and a name $exportedCxtName$. Then, it does the following:

  1. Creates a context (call it c), which contains a copy of the context referenced by each name path contained in the input set.
  2. Inserts the context c into the group workspace, under the name $exportedCxtName$. ⋄

- **import(**r, n**)**

  This operation takes as input a name path r w.r.t. the group workspace, and a name n. Then, it copies the context referenced by r from the group workspace into the CC, under the name n. ⋄

5.4 A cooperation session

In this section, we present and discuss the commands issued by each author during a cooperation session. These commands are shown in Figure 5.4.

Commands by Manos

User Manos checks-out version A_2 of the article, and copies it as version A_3 to his home workspace (see Figure 5.5). This is done through the command $check\text{-}out(A.A_2, A_3)$. As the user wants to revise version A_3, he focuses on context A_3. This is done through the command $SCC(A_3)$. As he wants to revise the main section, he checks-out object M_1 to his home workspace (replacing the object M_1 contained in context A_3 by a new version of the main section, named M_2, as shown in Figure 5.5). This is done through the operation $check\text{-}out(M.M_1, M_2)$. The local editing of M_2 is indicated by three dots in Figure 5.4.

After revision is completed, Manos needs to exchange information with the other authors for further revision. To this end, he needs to create the necessary environment which works as a coordinating unit for comparing the versions prepared by the different authors, before the final version is checked in the public workspace. This comparison requires knowledge about which authors have edited a particular version, and what changes have been made to it. Specifically, he uses the command $export(\{A, M\}, \texttt{Manos\_changes})$ to create a context named `Manos_changes` that contains copies of the history contexts of the edited objects, i.e., copies of the contexts 101 and 301 that represent the history of the article and its main section, respectively (see Figure 5.6). These contexts contain the original versions of the article and its main section, as well as their new versions created by user Manos.

Commands by Anastasia

Concurrently, user Anastasia also checks-out version A_2 of the article, and copies it as version A_3 in her home workspace[2] (see Figure 5.7). As she wants to revise version A_3, she focuses on context A_3. She then checks-out I_2 and M_1, and copies them as I_3 and M_2 in her home workspace (see Figure 5.7). Anastasia can now start editing I_3 and M_1. Once editing is finished, she exports her modifications to the group workspace for further revision (see Figure 5.8). This is done through the command $export(\{A, I, M\}, \texttt{Anastasia\_changes})$

Commands by Nicolas

Finally, user Nicolas imports contexts `Manos_changes` and `Anastasia_changes`, which contain modifications made by Manos and Anastasia, to his home workspace under the names `Manos` and `Anastasia`, respectively (commands 3.(a) and 3.(b) in Figure 5.4). As Nicolas wants to unify these modifications, he issues the commands 3.(c) to 3.(i), shown in Figure 5.4, and he creates the context 603 (assigned the variable `histCxt`) with name `Histories` in his home workspace (see Figure 5.9).

Context 603 contains three history contexts: (i) context 600 (assigned the variable `history_A`) with name `A`, which contains the whole history of the article after the modifications made on it by Manos and Anastasia; it also contains information about who made each modification (contexts 102 and 104); (ii) context 601 (assigned the variable `history_I`) with name `I`, which

[2] Note that she uses the same name A_3 as Manos did for naming a different version of the article. However, there is no ambiguity as the two A_3's are contained in different contexts.

1. Commands by user Manos.
/* CC = 1, Home = @.Private.Manos, Username = Manos */

(a) $check\text{-}out(\mathtt{A}.\mathtt{A}_2, \mathtt{A}_3)$
(b) $SCC(\mathtt{A}_3)$
(c) $check\text{-}out(\mathtt{M}.\mathtt{M}_2, \mathtt{M}_3)$
(d) $\cdots$
(e) $SCC(\mathtt{Home})$
(f) $export(\{\mathtt{A}, \mathtt{M}\}, \mathtt{Manos\_changes})$

2. Commands by user Anastasia.
/* CC = 2, Home = @.Private.Anastasia, Username = Anastasia. */

(a) $check\text{-}out(\mathtt{A}.\mathtt{A}_2, \mathtt{A}_3)$
(b) $SCC(\mathtt{A}_3)$
(c) $check\text{-}out(\mathtt{I}.\mathtt{I}_2, \mathtt{I}_3)$
(d) $check\text{-}out(\mathtt{M}.\mathtt{M}_1, \mathtt{M}_2)$
$\cdots$
(e) $SCC(\mathtt{Home})$
(f) $export(\{\mathtt{A}, \mathtt{I}, \mathtt{M}\}, \mathtt{Anastasia\_changes})$

3. Commands by user Nicolas.
/* CC = 3, Home = @.Private.Nicolas, Username = Nicolas */

(a) $import(\mathtt{Manos\_changes}, \mathtt{Manos})$
(b) $import(\mathtt{Anastasia\_changes}, \mathtt{Anastasia})$
(c) $\mathtt{history\_A} = createCxt(\mathtt{Manos.A} \uplus \mathtt{Anastasia.A}))$
(d) $\mathtt{history\_I} = createCxt(\{\,\} \uplus \mathtt{Anastasia.I}))$
(e) $\mathtt{history\_M} = createCxt(\mathtt{Manos.M} \uplus \mathtt{Anastasia.M}))$
(f) $\mathtt{histCxt} = createCxt(\{\})$
(g) $insert(\mathtt{history\_A}, \{\mathtt{A}\}, \mathtt{histCxt})$
(h) $insert(\mathtt{history\_M}, \{\mathtt{M}\}, \mathtt{histCxt})$
(i) $insert(\mathtt{histCxt}, \{\mathtt{Histories}\}, \mathtt{Home})$
(j) $\mathtt{changesLex} = lex(\mathtt{Manos.A}) \uplus lex(\mathtt{Anastasia.A}) \uplus$
$lex(\mathtt{Manos.M}) \uplus lex(\mathtt{Anastasia.M}) \uplus lex(\mathtt{Anastasia.I}) \uplus$
$\{lookupOne(\mathtt{Manos}) : \mathtt{Manos}, lookupOne(\mathtt{Anastasia}) : \mathtt{Anastasia}\} \ominus$
$lex(@.\mathtt{Public.A}) \ominus lex(@.\mathtt{Public.I}) \ominus lex(@.\mathtt{Public.M})$

(k) $insert(createCxt(\mathtt{changesLex}), \mathtt{Changes}, \mathtt{Home})$
(l) $\mathtt{final\_A} = createCxt(\{lookupOne(\mathtt{Changes}.\mathtt{I}_3) : \mathtt{I}_3,$
$lookupOne(\mathtt{Changes.Manos.M}.\mathtt{M}_2) : \mathtt{M}_2\})$

(m) $insert(\mathtt{final\_A}, \mathtt{A}_3, \mathtt{Home})$
(n) $check\text{-}in(\mathtt{A}_3.\mathtt{I}_3, \mathtt{I}, \mathtt{I}_3)$
(o) $check\text{-}in(\mathtt{A}_3.\mathtt{M}_2, \mathtt{M}, \mathtt{M}_2)$
(p) $check\text{-}in(\mathtt{A}_3, \mathtt{A}, \mathtt{A}_3)$

Figure 5.4: User commands during a cooperation session.

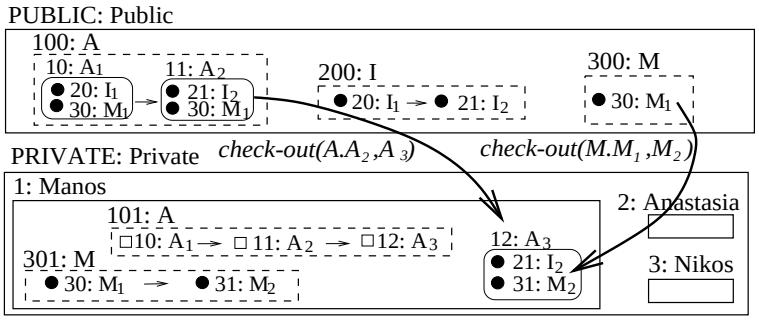


Figure 5.5: Manos' interaction with the public workspace.

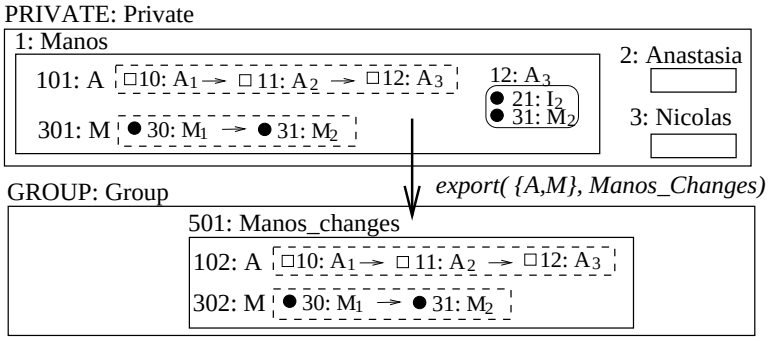


Figure 5.6: Manos' export to the group workspace.

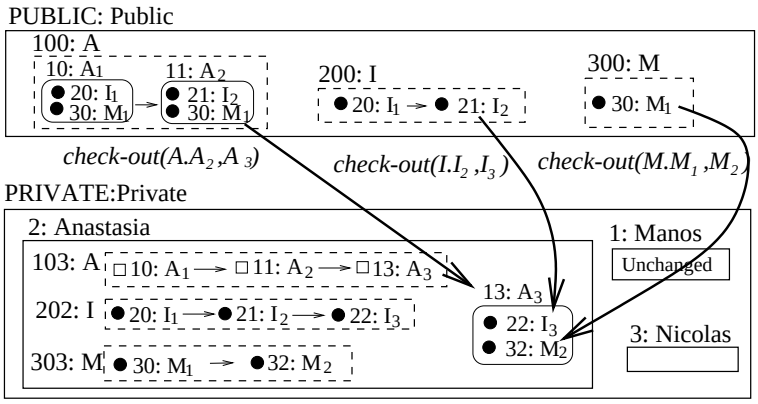


Figure 5.7: Anastasia's interaction with the public workspace.

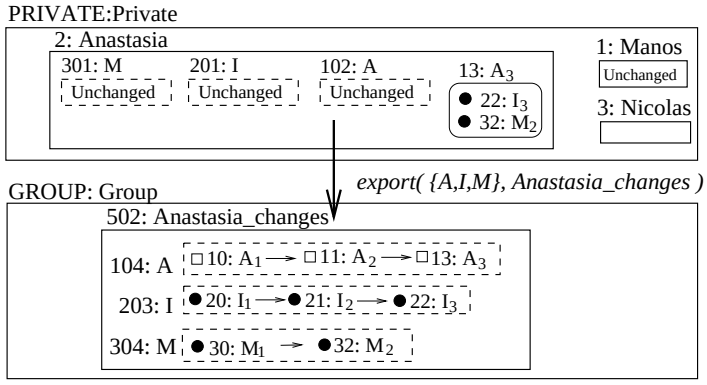


Figure 5.8: Anastasia's export to the group workspace.

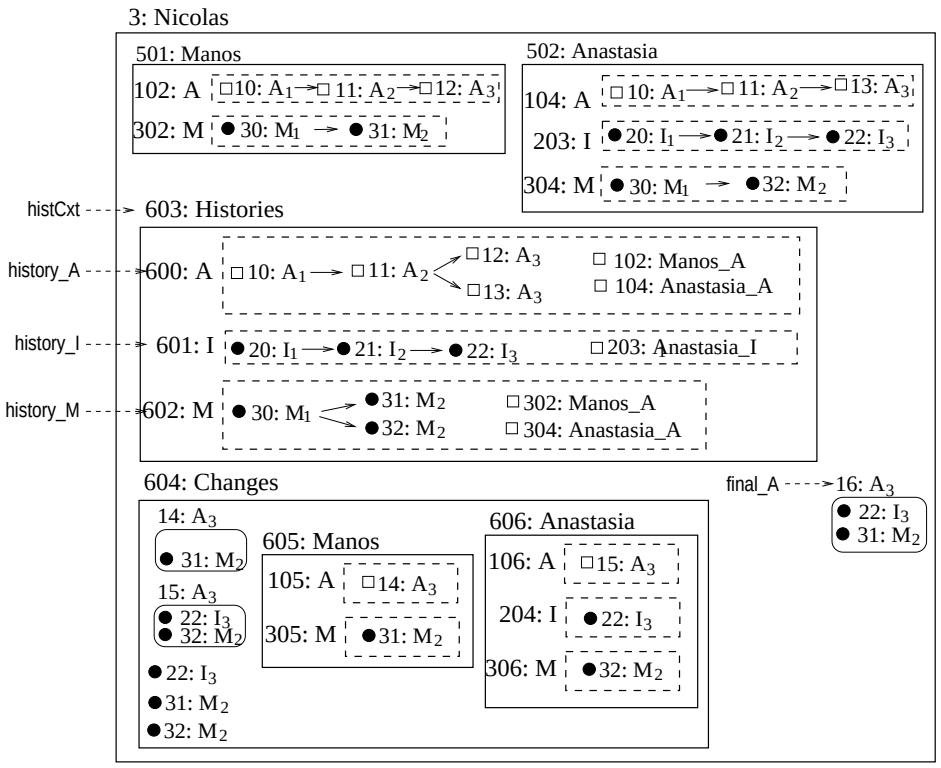


Figure 5.9: The context of user Nicolas.

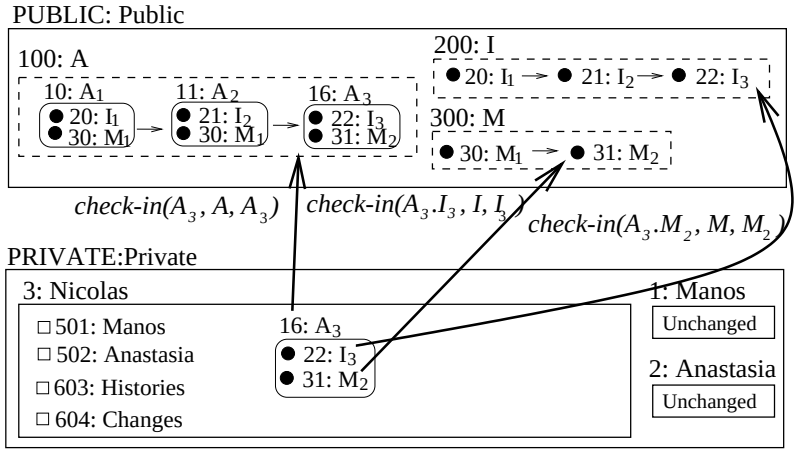


Figure 5.10: Nicolas' check-in to the public workspace.

contains the history of the introduction after the modifications made on it by Anastasia (Manos did not modify the introduction); and (iii) context 602 (assigned the variable `history_M`) with name `M`, which contains the history of the main section after the modifications made on it by Manos and Anastasia; it also contains information about who made each modification (contexts 302 and 304). Then, he can see that versions 12 and 13 (with name A_3) of the article are two parallel versions of version 11 and that version 12 was created by Manos (`Manos_A`.A_3) and version 13 by Anastasia (`Anastasia_A`.A_3).

Nicolas then wants to isolate the changes made by Manos and Anastasia and get rid of the whole history of the article and its parts contained in the public workspace. Thus, he creates the context 604 with name `Changes`, by issuing the commands 3.(j) and 3.(k) shown in Figure 5.4. This context contains the modifications made by Manos and Anastasia (objects 14, 15, 22, 31 and 32), as well as where these modifications appear within the structure of imported contexts 501 and 502; thus two new views of the structures of contexts 501 and 502 are created, which are contexts 605 and 606, respectively (see Figure 5.9).

Then, Nicolas studies modifications and creates the final version of article (command 3.(l) in Figure 5.4) composed by the version 22 (`Changes`.I_3) of the introduction made by Anastasia and the version 31 (`Changes.Manos.M`.M_2) of the main section made by Manos. This final version is checked in the public workspace through the commands 3.(n) to 3.(p) (see Figure 5.10).

We would like to stress that the purpose of the example presented here was to illustrate the use of context in a simple cooperation environment. The commands *check-in*, *check-out*, *import* and *export*, are examples of simple communication commands that can be implemented using the context operations of our model.

In a more complex environment, however, the users will most likely need information on various aspects of the cooperation. For example, in a software engineering project, where several groups are developing software in parallel, a coordinating unit may need to compare modules coming from various groups, before merging them into a single module. Such information can be obtained through more sophisticated higher level commands that can also be implemented using the context operations of the model.

The Information Base can be organized in a number of different ways. Choosing the appropriate organization is a design problem that depends on the application. However, this problem lies outside the scope of this thesis.

Chapter 6

Contextualization as an Independent Abstraction Mechanism for Conceptual Modeling

In Chapter 4 we presented an extension of previous models of context, along with a formalization, which increased the expressive power and flexibility and solved certain operational problems of those models. This model of context was demonstrated at work in Chapter 5. We now identify certain additional properties which, if possessed by the context construct, this could stand as an abstraction mechanism for conceptual modeling in its own right. We then extend the definition of context, we demonstrate its use in modeling and discuss its interaction with other abstraction mechanisms. Work of this chapter published in [122, 123].

With the context construct in its current form (i) the model of the world generated contains objects that are not homographic to those in the real world modeled; and (ii) structuring of the contents of a context is not directly supported.

1. One-to-one correspondence between real world objects and (model) objects.

   Contexts have been defined to be objects themselves. This implies that two different contexts (i.e., objects) are needed in order to represent two different perceptions of the same real world object. For example, suppose that a person `A` wants to describe the environment of a university as a context containing information about the university, as viewed from his / her point of view (call this context c). Suppose, now, that another person `B` wants to describe the *same* university as a context also containing information about the university, but viewed from a different point of view (call this context c'). Certainly, context c contains different information than context c', i.e., they are two different contexts, and thus objects, although they refer to the same real world object (the same university). *There is not an one-to-one mapping from real-world objects to model objects, whereas there is an one-to-one mapping from different perceptions of real-world objects to model objects.* So, there could be confusion as to whether two different contexts refer to the same real-world object or not. We feel that it would be preferable to maintain an one-to-one mapping between real world objects and their representations in the model, as well as between different perceptions of the real world objects and contexts.

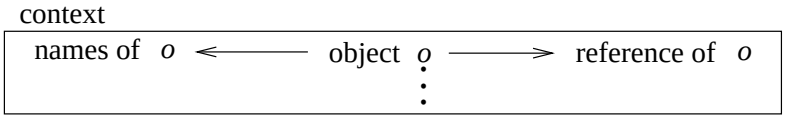
2. Structuring the contents of a context.

   The definition of context allows for a simple representation of objects, i.e., an object can have an associated set of names. However, the objects of a context could be related in more complex ways using the usual abstraction mechanisms of classification, generalization, and attribution. Then the question arises: How does contextualization interact with classification, generalization, and attribution when all four are supported by the conceptual model? In other words, are there any constraints that govern their joint use?

In this chapter, we enhance the notion of context in two ways:

1. We introduce references from objects to contexts.

   The contents of a context is a set of objects, each of which is associated with a set of names (as before), with the following *new* feature: We allow each object of a context to be associated with another context that we call its *reference*. Thus, each object of a context is now associated with a set of names on one hand and (possibly) with a reference on the other, as in the following diagram:



   Roughly speaking, the reference of the object points to information available about the object.

   In this extended model, real world objects are represented by model objects in an one-to-one correspondence. A context is a collection of objects that supports encapsulation. Multiple representations of the same real world object are supported through the object reference. Thus, the university of the previous example would be represented as a model object o, whereas the reference of o in the context of person `A` would be the context c. On the other hand, the reference of object o in the context of person `B` would be the context c'.

2. We allow the objects of a context to be structured through the traditional abstraction mechanisms of classification, generalization and attribution[1]. We study how these three abstraction mechanisms interact with contextualization, in particular how instance-of, ISA, and attribute links between objects affect the definition of their references.

So, *a context is a structured set of objects, in which each object is associated with a set of names and (possibly) a reference.*

It is important to note that this notion of context allows to group together such things as class instances, classes, metaclasses, subclasses, superclasses, attributes, ISA links, and instance-of links.

The extended notion of context introduced here enriches the modeling capabilities of the traditional abstraction mechanisms in two significant ways:

1. *Expressive power*: By supporting relative semantics, i.e., relative naming and relative descriptions, and by interacting with the traditional abstraction mechanisms, context provides new modeling capabilities.

[1]By "attribution" we mean the assignment of an intrinsic attribute to an object as well as the declaration of its (binary) relationships to other objects. The abstraction mechanism of *aggregation* is a limited form of attribution [52].

2. *Modularity*: By retaining the essential information and hiding inessential details, context helps to increase comprehensibility and communicability in complex applications such as information retrieval over the web, cooperative work in distributed environments, large engineering databases, scientific catalogs, etc.

In this chapter, we present the mechanism of context and the ways in which contexts interact with each other and with the traditional abstraction mechanisms of conceptual modeling.

We first define the notion of context *without* structuring of its objects, and we discuss some of its modeling capabilities. Then we introduce structure through the traditional abstraction mechanisms. We discuss the ways in which contexts interact with each other and with the traditional abstraction mechanisms.

6.1 The notion of context revisited

Suppose we want to talk about Greek islands by simply using their names without further description. Let us consider the island of Crete. We can represent this island by an *object identifier*, say o_1, and by associating this identifier with the name `Crete`. We write $names(o_1) = \{\texttt{Crete}\}$ and we denote this as follows[2]:

$$\texttt{Crete} : o_1$$

Next, let us consider the island of Santorini. Following a similar approach, we represent this island by an object identifier o_2 and by associating it with the name `Santorini`. However, the island of Santorini is also known under the name Thera. So this time, we associate o_2 with the set of names $\{\texttt{Santorini}, \texttt{Thera}\}$, i.e., this time we write $names(o_2) = \{\texttt{Santorini}, \texttt{Thera}\}$ and we denote this as follows:

$$\texttt{Santorini}, \texttt{Thera} : o_2$$

Finally, let us consider one of those tiny, uninhabited islands of Greece that happen to be nameless. We represent such an island by an object identifier o_3 and by associating it with no name, i.e., we write $names(o_3) = \{\}$ and we denote this as follows:

$$: o_3$$

Continuing in the same way, we can represent every Greek island in a similar manner. The set of all such representations is what we call a *context* and we represent it by a *context identifier*, say c_1, as shown in Figure 6.1.

Suppose next we want to talk about the Greek mainland by simply using the names of each region of Greece without further description. Proceeding in a similar way as in the case of Greek islands, we can create a second context, say c_2, as shown in Figure 6.1.

Suppose now that we want to talk about geography of Greece seen as a division of Greece into islands and mainland. First, let us consider the islands. We can represent the islands by an object identifier, say o, and by associating it with the name `Islands`, i.e., $names(o) = \{\texttt{Islands}\}$. However, the object o is a higher level object that collectively represents all Greek islands, i.e., the object o collectively represents the contents of context c_1. In other words, if we want to see what o means at a finer level of detail, then we have to "look into" the contents of c_1. Thus we call context c_1 the *reference* of object o, and we write $ref(o) = c_1$. Summarizing our discussion on islands, we write $names(o) = \{\texttt{Islands}\}$ and $ref(o) = c_1$, and we denote this as follows:

[2]In this chapter, the terms *object* and *object identifier* will be used interchangeably.

Islands : o ⋯▷ c_1

Following a similar reasoning, we can represent the mainland by an object identifier, say o', and by associating it with the name `Mainland` and the reference c_2. We can now group together the islands and the mainland to form a context c, as shown in Figure 6.1. Then, geography of Greece can be represented by an object identifier o'' and by associating it with context c (Figure 6.1).

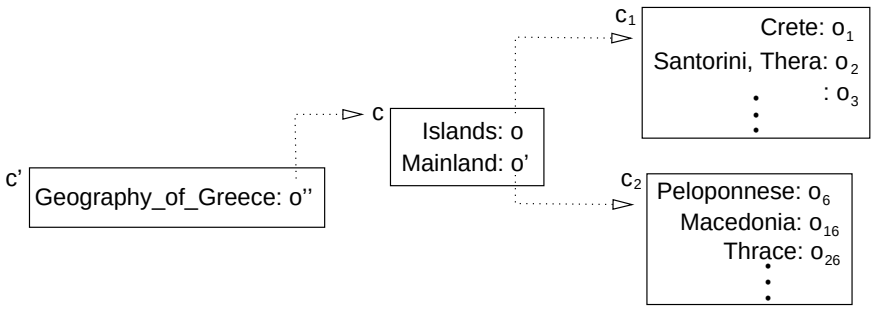


Figure 6.1: An example of context structure: geography of Greece

The previous examples suggest the following informal definition of context (in its simplest form): *a context is a set of objects of interest, each object having zero, one or more names, and zero or one references.* For the purposes of this chapter we shall give an informal statement of the definition of context. A formal theory is presented in Chapter 7.

Definition 6.1 Context.

A context c is defined as a set of objects, denoted by $objs(c)$, such that each object o

1. is associated with a set of names, called *the names of o in c*, denoted by $names(o, c)$;
2. is associated with zero or one context, called *the reference of o in c*, denoted by $ref(o, c)$.
3. is either an individual or a link (attribute, instance-of, or ISA);
4. can be related to other objects through attribute, instance-of or ISA links. ◇

The terms *object*, *individual* and *attribute* are used in the sense of Telos [83, 64]. We will not be concerned with this interpretation until Section 6.4.

The reason why we use the symbols $names(o, c)$ and $ref(o, c)$, instead of $names(o)$ and $ref(o)$ used in the previous examples, is that an object can belong to different contexts and may have different names and / or reference in each context. That is, *names and references are context-dependent*.

In our previous examples, while explaining the construction of a context, we followed a bottom-up approach. That is, we started from simple objects and built up contexts which were later on referenced by higher level objects ("moving" from right to left in Figure 6.1). Clearly, we could have followed the opposite construction, i.e., a top-down approach ("moving" from left right in Figure 6.1). Or, we could follow a mixed approach.

This flexibility is important in conceptual modeling and give (among other things) the possibility of *modular design*, i.e., retaining at each level of abstraction the essential information and hiding inessential details (by "encapsulating" them in the form of a context).

Let us see a top-down definition of a context. Suppose we are defining a context containing guides to Greece and wish to model a tourist guide as one of its objects. Within this context we represent the guide as follows:

Tourist_Guide: o_4 ⋯▷ c_3

The next stage is to define the context c_3 that contains the information concerning the tourist guide. The context c_3 is shown in Figure 6.2.

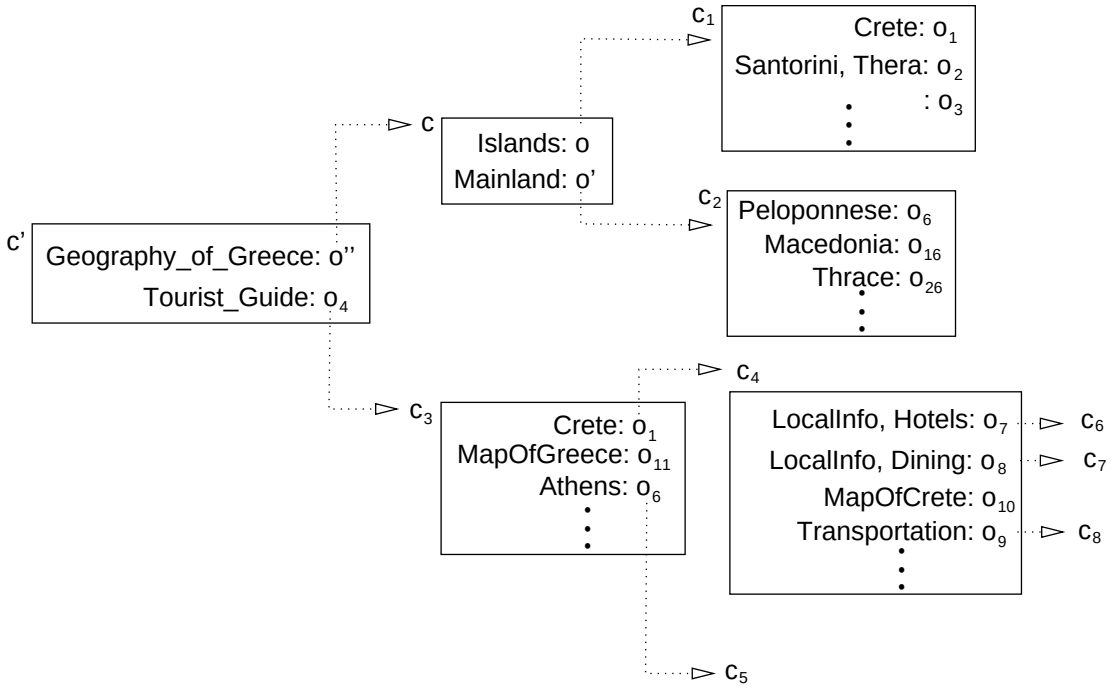


Figure 6.2: An example of context structure: tourist guide and geography of Greece

We then define the contexts c_4, c_5, and so on. Context c_4 is shown in Figure 6.2, whereas the remaining contexts are not. Context c_4 contains tourist information concerning Crete such as hotels, dining, a map of Crete, transportation, etc. Looking now at the definition of context c_4, we see that we have to define contexts c_6, c_7, c_8, and so on (their definition is not shown in the figure). Context c_6 will list the hotels in Crete and provide access to further information such as addresses and telephone numbers. Context c_7 will provide access to information about dining in Crete, e.g. a list of restaurants, local dishes, and so on, and context c_8 will give transportation information.

Note that object o_1 is shared by context c_3 and context c_1, and in context c_3 it has a reference (context c_4), whereas in context c_1 it does not (see Figure 6.2).

The notion of context supports a simple and straightforward way of referencing objects at any level of detail. Consider, for example, the tourist guide of Greece in Figure 6.2. Suppose that, currently, we are in the context containing the tourist guide, and we want to look at Cretan hotels. To do so we can "go" from object o_4 (`Tourist_Guide`) to object o_1 (`Crete`) and then to object o_7 (`Hotels`). We indicate this as follows: $o_4.o_1.o_7$, i.e., by forming a path of object identifiers. If the last object in the path has a reference, this points to a context that contains the information of interest.

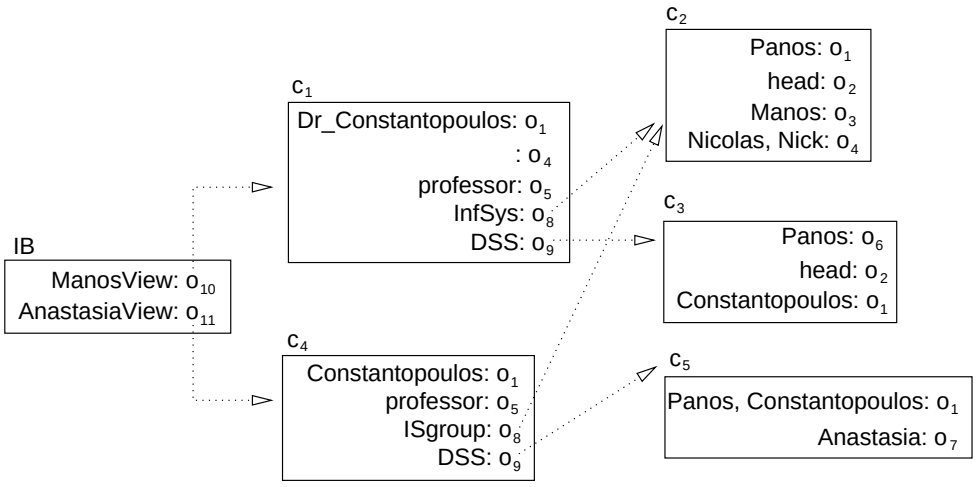


Figure 6.3: An example of context structure: Two views of the same institute.

Let us now consider another example, shown in Figure 6.3, which represents the views of two persons regarding an institute. This is actually the running example used in Chapter 4 to present the notion of context as seen in that chapter (see Figure 4.5). Figure 6.3 illustrates the example shown in Figure 4.5 of Chapter 4 but being modeled using the extended notion of context described in this chapter. Indeed, in Figure 6.3, context IB contains two objects, o_{10} and o_{11}, namely `ManosView` and `AnastasiaView`, respectively. These objects represent the views of Manos and Anastasia regarding an institute. Object o_{10} is associated with the reference c_1 and object o_{11} is associated with the reference c_4. Contexts c_1 and c_4 contain the information concerning the views of Manos and Anastasia for the institute, respectively.

Let us now focus on context c_1 which contains five objects, namely o_1, o_4, o_5, o_8 and o_9. Objects o_8 and o_9 represent the Information Systems Lab and the Decision Support Systems Lab, and are associated with the names `InfSys` and `DSS`, respectively. Further information is available for object o_8, which is modeled by associating it with the reference c_2. Context c_2 contains information about the Information Systems Lab as seen from Manos' point of view. Further information is also available for object o_9 (it is associated with the reference c_3) and context c_3 contains information about the Decision Support Systems Lab as seen from Manos' point of view. Note that context c_4 contains objects o_8 and o_9 as well. The object o_8 is associated with the reference c_2 as well, but the object o_9 is associated with another reference, i.e., context c_5. Intuitively, this expresses that Manos and Anastasia have the same view for the Information Systems Lab (context c_2), whereas they have different views for the Decision Support Systems Lab.

In Figure 4.5, context c_1 contains five objects, two of which are contexts, namely c_2 and c_3, which represent Manos' view for the Information Systems Lab and the Decision Support Systems Lab, respectively. Moreover, context c_4 contains the contexts c_2 and c_5, which represent Anastasia's view for the Information Systems Lab and the Decision Support Systems Lab, respectively. However, the problem in this representation is that there is no way to express that contexts c_3 and c_5 are descriptions of the same real world object, namely the Decision Support Systems Lab. Whereas, in Figure 6.3, the same information is expressed by representing the Decision Support Systems Lab with the object o_8 and then associating it with the reference c_2 in context c_1 and the reference c_5 in context c_4.

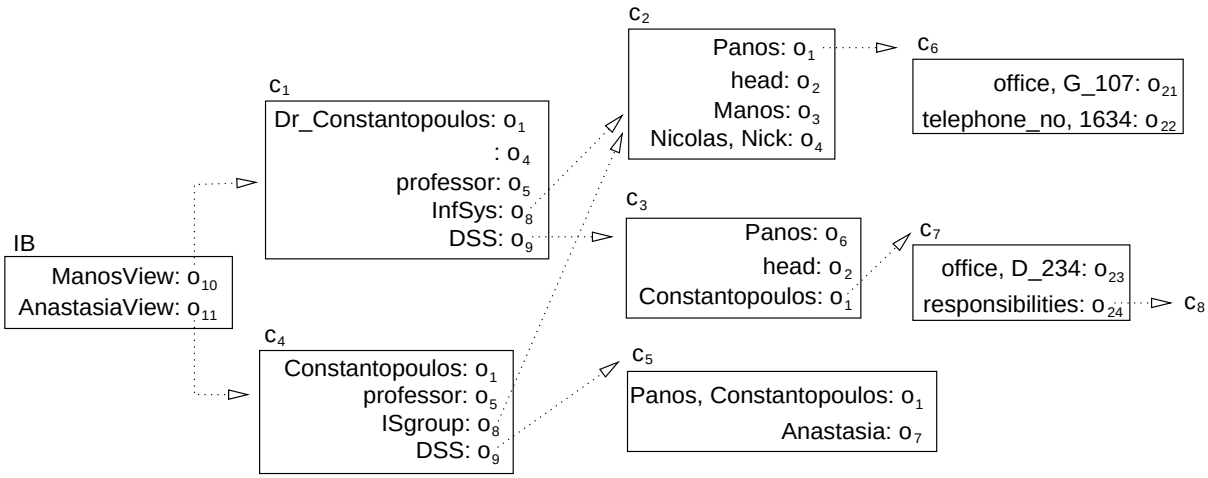


Figure 6.4: An example of context structure evolution.

The extended notion of context introduced here increases the scalability of the model as, during the information base evolution, new information can be easily added to the information base by (i) inserting new objects in a context, (ii) inserting new names in the names of an object in a context, and (iii) associating an object with a reference. The later is important since it makes the difference between the notion of context described in this chapter from the previous approaches described in Chapter 4.

For example, assume that further information about the object o_1 is available both in context c_2, concerning the office number and the telephone number of that person in the Information System Lab, and in context c_3 concerning the office number and the responsibilities in the Decision Support Systems Lab. To add this information we associate object o_1 with the reference c_6 in context c_2 and with the reference c_7 in context c_3 as shown in Figure 6.4. On the other hand, it is not easy to add this information in traditional approaches of context (see Chapter 4). To do this we should replace object o_1 in contexts c_2 and c_3 with contexts c_6 and c_7, respectively. This is not a desirable result because we lose that context c_6 and c_7 contain information about the same person, represented by the object o_1.

Summarizing, we say that the extended notion of context introduced here increases the expressive power, flexibility and scalability of the model.

6.2 Features of context

Let us summarize the features of context supported by context definition introduced in this chapter. Some of these features (see Items 1-5 in the following list) are also supported by the definition of context introduced in Chapter 4.

1. **Object sharing or overlapping contexts.**
   An object can belong to one or more different contexts. When contexts share objects we say that contexts overlap. This feature is useful when we want to view an object under different perspectives.

2. **Context-dependent object names.**
   The same object can have different names in different contexts (in which it belongs).

This is very convenient, because a name which has a clearly understood meaning in one context may not do so in another.

3. **Synonyms.**
   The same object can have different names in the same context. That is, alternative ways for naming the same object are supported. This is the case of synonymous objects.

4. **Homonyms.**
   Two different objects can have the same name within a context. This is the case of homonymous objects.

5. **Anonyms.**
   An object may have no name within a context. This is the case of anonymous objects.

6. **Context-dependent references.**
   The same object can have different references within different contexts. In other words, references are context-dependent.

7. Two different objects, whether or not they belong to the same or different contexts, can have the same reference. This is convenient, as a given context can be reachable through different object paths.

8. From within a given context, we can "reach" any object that belongs to the reference of an object within that context (and, recursively, any object that lies on a path).

6.3 Contextualization as an abstraction mechanism

Contextualization constitutes an abstraction mechanism in the sense that a context c "encapsulates" its contents and thus any object referencing c can be seen as the abstraction of the contents of c. For example, in context c_3 of Figure 6.2, the object o_1 (`Crete`) is the abstraction of the contents of c_4.

Contextualization can be used in either an alternative or a complementary capacity with regard to the other modeling mechanisms, depending on the application. For example, referring to Figure 6.1, the object o (`Islands`) can also be modeled as a class with a set-valued attribute *names*, and the objects o_1, o_2, o_3, can be modeled as instances of o. However, referring to Figure 6.2, it is less obvious how to model the object o_4 (`Tourist_Guide`) by means of the traditional abstraction mechanisms. Indeed, the objects o_1, o_6, o_{11} can hardly be considered as instances of o_4, or, the objects o_7, o_8, o_9, o_{10} as instances of o_1. Thus, the use of contexts for describing the tourist guide of Greece seems to be more appropriate.

Roughly speaking, the modeling power of contexts stems from the fact that one can group together quite dissimilar things as the contents of a context, regardless of any structural relationships they may have. In fact, no such relationships are required to hold the contents of the context together.

In summary, modeling with contexts offers several capabilities, including the following:

1. Modeling an object under different perspectives, by associating it with different references in different contexts.

2. Modular representation, by providing at each level of abstraction an overview of the available information in the form of items that each provide access to relevant detail.

3. Top-down, bottom-up, or mixed modeling.

Moreover, as we shall shortly see, the combination of contextualization with the traditional abstraction mechanisms provides further modeling capabilities.

6.4 Structuring the contents of a context

We now turn to the structuring possibilities presented by definition 6.1.

For the purposes of this thesis, we shall assume that the objects of a context can be structured as in a Telos information base [83, 64].

A Telos information base consists of structured objects built from two kinds of primitive units: *individuals* and *attributes*. An important and distinctive feature of Telos is that individuals and attributes are treated uniformly, and are referred to as "*objects*" (in [83] objects are referred to as "*propositions*"). Individuals represent entities (atomic ones, such as `John`, or collective ones, such as `Person`), while attributes represent directed binary relationships between or intrinsic characteristics of entities. Every attribute consists of a *source*, a *label*, and a *destination*.

Objects (individuals or attributes) are organized along three dimensions, referred to as the *classification*, *generalization*, and *attribution* dimensions [52, 114, 83, 64].

The classification dimension calls for each object to be an *instance-of* one or more *classes*. Classes are themselves objects, therefore they can be instances of other, more abstract classes. Generally, objects are classified into:

> *tokens*, i.e., objects having no instances and intended to represent atomic entities in the domain of discourse;
>
> *simple classes*, i.e., objects having only tokens as instances;
>
> *metaclasses*, i.e., objects having only simple classes as instances;
>
> *metametaclasses*, and so on.

This classification defines an unbounded hierarchy of levels of ever more abstract objects. All tokens are classified under the class $\mathtt{L_0\_Class}$, all simple classes under the class $\mathtt{L_1\_Class}$, all metaclasses under the class $\mathtt{L_2\_Class}$, and so on, and $\mathtt{L_0}$, $\mathtt{L_1}$, $\mathtt{L_2}$, etc. are called *instantiation levels*. Classification is treated as a form of weak typing mechanism: the classes which a structured object is an instance of determine the kinds of attributes it can have and the properties it must satisfy.

Classes at the same instantiation level can be specialized along *generalization* or *ISA hierarchies*. For example, the class `Person` may have subclasses such as `Employee`, `Professor`, and `Student`. As a class may be a subclass of more than one classes, the ISA hierarchy is not necessarily a tree. Attributes of a class which are not tokens can be inherited by subclasses, this inheritance being strict rather than default [83].

Finally, in the attribution dimension an object is seen as the aggregate of its attributes.

Instance-of and ISA relations, as well as attributes, are also referred to as instance-of, ISA and attribute links, respectively. They all have source and destination objects, while attributes in addition have objects identifiers and optional labels. In our model, for reasons of uniformity, instance-of and ISA links are treated as objects too.

Every object of a context is assumed to belong to one and only one level of the instantiation hierarchy, i.e., it is assumed to be one and only one of the following: a token, a class, a metaclass, and so on. To this effect, it is assumed that every context supports a number of system classes, named $\texttt{L}_0\texttt{\_Class}$, $\texttt{L}_1\texttt{\_Class}$, $\texttt{L}_2\texttt{\_Class}$, and so on, and that every object is an instance of one and only one of these classes.

Each context is assumed to be equipped with

- A predicate for defining the objects that are attributes links. This is the predicate

  $$attr(att\_obj, from, to)$$

  declaring that object att_obj is an attribute link with source object $from$ and destination object to.

- A predicate for defining the objects that are instance-of links. This is the predicate

  $$in(in\_obj, from, to)$$

  declaring that object in_obj is an instance-of link in which the object $from$ is an instance of the object to.

- A predicate for defining the objects that are ISA links. This is the predicate

  $$isa(isa\_obj, from, to)$$

  declaring that object isa_obj is an ISA link in which the class $from$ is a subclass of class to.

Note that, as attribute, instance-of, and ISA links are objects, a link can have zero, one or more names. Each of these names corresponds to a Telos label.

Consider, for example, modeling employees using a class whose instances have three attributes: name, salary and address. Using our definition of context, this modeling can be done as shown in Figure 6.5(a), where o is the employee class, and the three attribute declarations define the objects o_1, o_2 and o_6 as attribute classes from class o to classes o_4, o_5 and o_3, respectively. Figure 6.5(b) shows a more convenient representation of context c, where the attributes o_1, o_2 and o_6 are represented by arrows. For example, $attr(o_1, o, o_4)$ is represented by the arrow from o to o_4. This declares that an instance of the class `Employee` can have an attribute, instance of class `Name`, that connects it to an instance of class `String`. Note that attribute o_6 has the same name as object o_3, something allowed by our definition of context. However, if referencing of objects is done through names, this may lead to ambiguities. We address this problem in [121, 120][3].

In the previous example, the employee information is modeled as the contents of a single context, i.e., a context containing the employee class *and* its attributes. An alternative way of

[3] A simple way to avoid this problem is to put the name of object o_6 in a verb form, e.g. `has_address`.

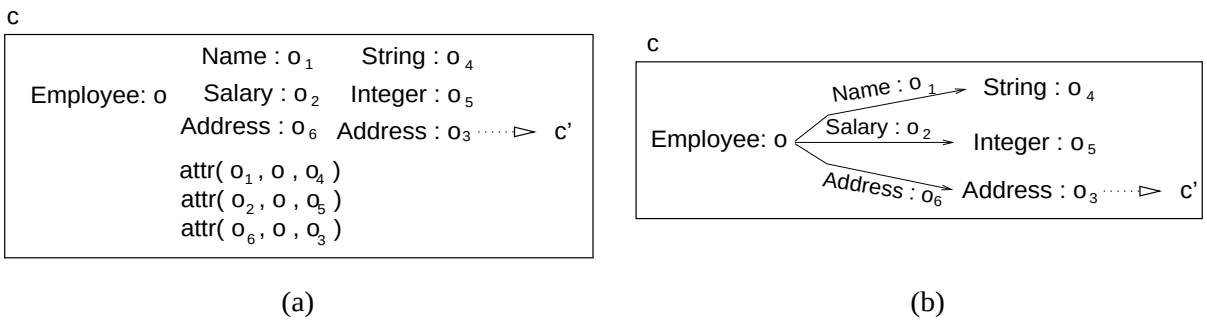


Figure 6.5: Modeling an employee using attributes

modeling an employee is the following: the employee class references a context containing the attribute information. This is shown in Figure 6.6(a), where the ISA declarations in context c define that `Name` is a subclass of `String` and `Salary` is a subclass of `Integer`. Figure 6.6(b) shows a more convenient representation of context c, where ISA links are represented by thick arrows. In fact, from now on, we shall use arrows to represent all relationships, with different kinds of arrows for the different relationship types.

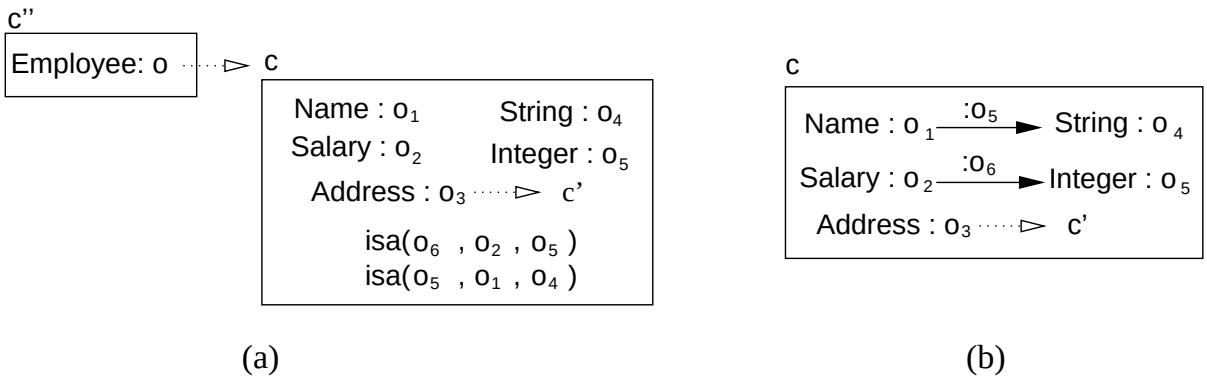


Figure 6.6: Modeling an employee using contexts

At first glance, the modeling of Figure 6.6 may look simpler than that of Figure 6.5. However, depending on the application, the one or the other could be preferred. The important point here is that the contextualization mechanism offers new alternative modeling solutions.

In the rest of the chapter, in order to simplify the pictorial presentation of contexts, the object identifiers of instance-of, ISA, and attribute links will be omitted from the pictures.

Figure 6.7 shows another example of context with structured contents, this time using all three abstraction mechanisms, i.e., classification, generalization, and aggregation. It is important to note that contextualization is orthogonal to the other abstraction mechanisms.

In this example, company `CompanyA` is represented by object o_1. We can find more information about the company in context c_1, which contain information about employees, managers and services provided by the company. In particular, objects `Employee` and `Manager` represent the class of employees and managers, respectively. The salary paid by the company is represented by object o_5 and is an Integer. The fact that employees has a salary is represented by the object o_{11}. Similarly, object o_{10} represents the fact that managers `manages` employees. Individuals employees are represented by objects `Pol`, `Pit`, and `John`, where `John` manages `Pol` and `Pit`. Thus, `Pol` and `Pit` are instances of `Employee`, whereas `John` is instance of `Manager`. Attributes

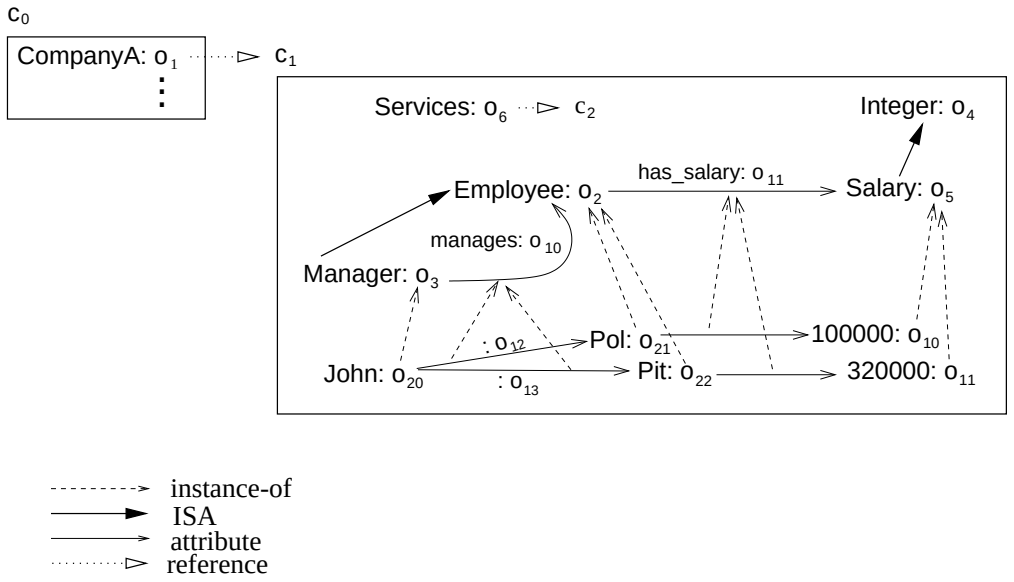


Figure 6.7: Modeling the employees of a company

o_{12} and o_{13} are instances of attribute class `manages` and represent that `John` manages `Pol` and `Pit`, respectively. Object o_6 represents collectively the services available by the company (more information is available in context c_2).

Figure 6.8 shows another example of context with structured contents representing recommendations for dining in a touristic place.

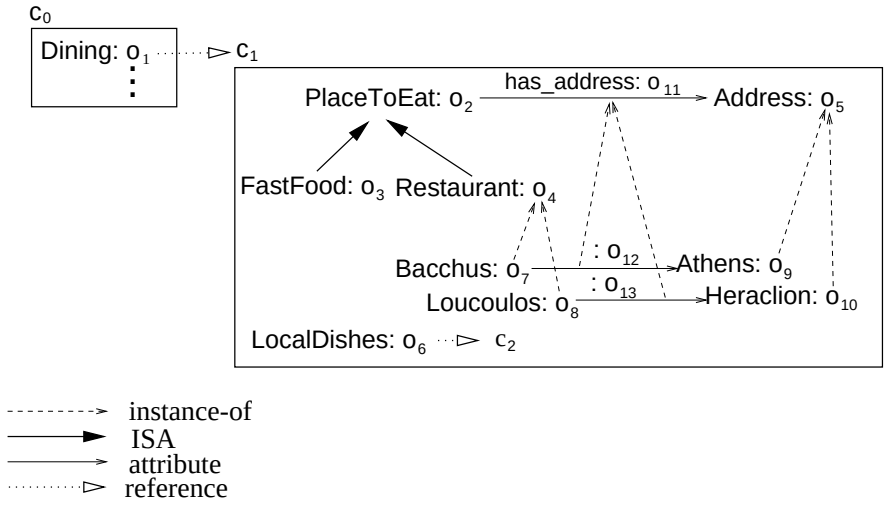


Figure 6.8: Structured contents of a context

It must be stressed that *all items in a context, including the predicate declarations, are defined relative to that context*. In other words, the scope of the definition of the contents of a context is the context itself.

For example, an object that belongs to two different contexts can have different attributes in each context. Similarly, an object o which is an instance of a class o' in a given context might

not be an instance of o' in a different context (assuming that o, o' belong to both contexts). And so on. This is illustrated in Figure 6.9. Contexts c_2 and c_4 contain information concerning geographic data about Crete during the 15th and 20th century, respectively. The object o_6 represents a place in Crete which is classified under `Village` in the 15th century, whereas the same place is classified under `City` in the 20th century. Note that object o_5 was called Chandax in the 15th century, whereas the same object is called Heraklion in the 20th century. Moreover, the 15th century description includes information on the fortification of the city, while that of the 20th century includes information on the airport.

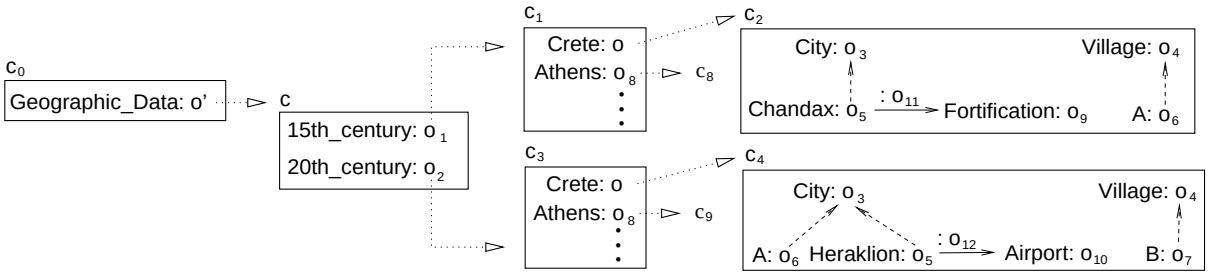


Figure 6.9: Context-dependent description

6.5 The interaction between abstraction mechanisms

In an information base, the interaction between different abstraction mechanisms provides useful information to the designers of the information base, but also implies certain constraints that must be satisfied to guarantee consistency of the stored information.

For example, if object o is instance of class o', and o' is subclass of o'', then o is instance of o''. This is useful information as it implies that o' inherits all attributes of o''. On the other hand, if class o_1 is subclass of class o_2 and o_2 is instance of metaclass o_3, then o_1 cannot be declared as subclass of o_3, and this is a constraint that must be satisfied to guarantee consistency of the stored information.

The interaction between the traditional abstraction mechanisms of conceptual modeling has been extensively studied in the literature (see [52, 114] for a survey). In this section, we study the interaction between the traditional abstraction mechanisms and the mechanism of contextualization.

6.5.1 Attribution and contextualization

In our enhanced definition of context, each object can be either an individual or a link (in this subsection the link is an attribute link), and has a set of names and zero or one reference. Now, if the object is an individual, then its reference can be *any* context. An attribute, however, cannot exist without a source and a destination, and this information must be part of its reference.

Therefore, if the object is an attribute, then its reference must contain at least a description of the source and the destination reference[4]. This can be done as shown in Figure 6.10. Let o be an attribute from object o_1 to object o_2 with references c_1 and c_2, respectively. Then, its

[4]We refer to the reference of the source (resp. destination) of an attribute as the *source* (resp. *destination*) reference.

reference c should contain two special objects: an object o_f named `from` with reference c_1 and an object o_t named `to` with reference c_2.

So from now on, we assume that the reference of every attribute is as shown in Figure 6.10. That is, it contains information about the source and the destination of the attribute. We also assume that an instance-of or ISA link may have a reference, and that this reference (if any) is of the same kind as the attribute link, i.e., a reference that contains information about the source and the destination of the link.

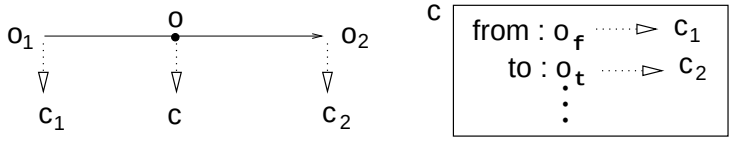


Figure 6.10: The reference of an attribute

Of course, apart from the minimal necessary information shown in Figure 6.10, the reference of an attribute may also contain other information. The question here is whether there are constraints that this "other information" should satisfy.

Let us call *traversal path* any path from an object in the source reference of the attribute to an object in the destination reference, such that every member of the path is an attribute, instance-of, or ISA link. We call *attribute path* any traversal path every member of which is an attribute. Intuitively, an attribute path defines an attribute from an object in the source reference to an object in the destination reference.

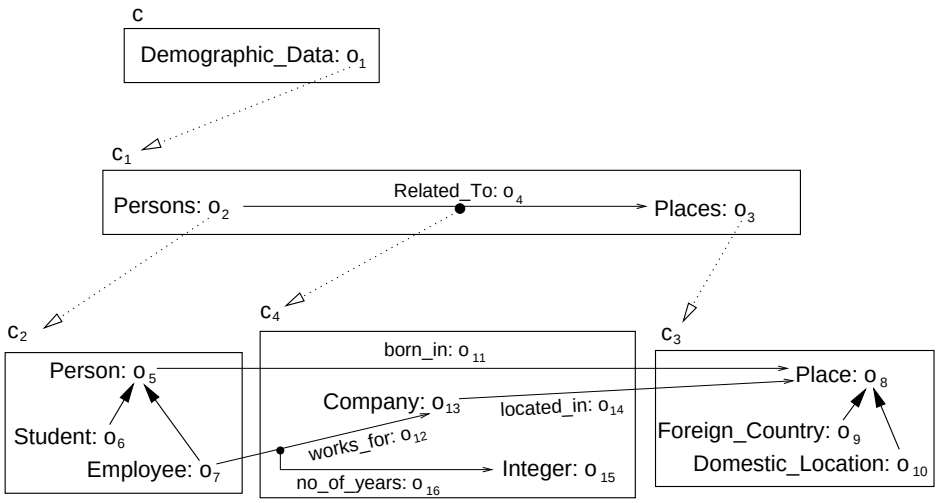
Now, the constraint that we propose for the reference of an attribute can be stated informally as follows: the attribute must collectively represent all traversal paths from objects in its source reference to objects in its destination reference. Clearly, in order for this requirement to be satisfied, all traversal paths must be attribute paths. Hence the following constraint on the information that the reference of an attribute can contain:

Constraint 6.1 Attribute Reference Constraint.
Every traversal path in the reference of an attribute is an attribute path. ◇

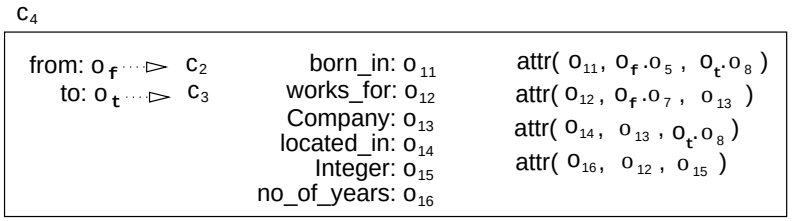
Figure 6.11 illustrates the interaction between attribution and contextualization in a top-down modeling of demographic data. The reference of attribute o_4 (`Related_To`) is context c_4. This reference contains two traversal paths that are both attribute paths. The first of these paths goes from object o_5 in context c_2 to object o_8 in context c_3, and consists of a single attribute: o_{11} (`born_in`). Within context c_4, this is defined as $attr(o_{11}, o_f.o_5, o_t.o_8)$ because objects o_f and o_t refer to the source and destination references of attribute o_4, respectively. The second path goes from object o_7 in context c_2 to object o_8 in context c_3 and consists of two attributes: o_{12} (`works_for`), from o_7 to o_{13}, and o_{14} (`located_in`), from o_{13} to o_8. Note that in Figure 6.11(a), context c_4 is given in a pictorial way, where the special objects o_f and o_t are omitted and predicates are depicted through arrows. Its actual definition is given in Figure 6.11(b).

6.5.2 Classification and contextualization

The interaction between classification and contextualization is similar to that between attribution and contextualization. Thus the reference of an instance-of link l should contain only



(a)



(b)

Figure 6.11: Interaction between attribution and contextualization

instance-of links from objects in the source reference of l to objects in the destination reference of l.

Constraint 6.2 Instance-of Reference Constraint.

Every traversal path in the reference of an instance-of link consists of a single instance-of link. ◇

If an object o is an instance of object o' then the reference of the instance-of link may classify objects in the reference of o into object classes in the reference of o'. Intuitively, we can say that the objects in the reference of o follow the "schema" defined in the reference of o'.

Probably the most relevant example of this interaction is the one relating a database schema with its instances. In Figure 6.12, object o_1 (`Instance`) is instance-of object o_1' (`Schema`). Note that the reference c_{in} of the instance-of link contains only instance-of links from objects of c to objects of c'.

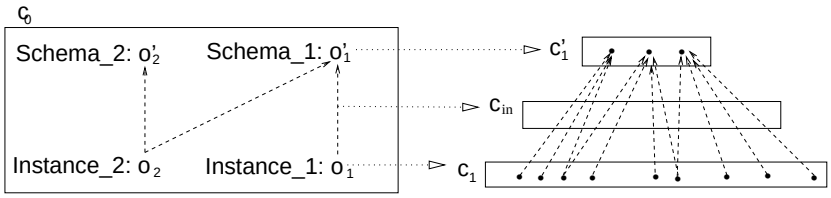


Figure 6.12: The reference of an instance-of link

Intuitively, within c_{in}, objects of c_1 are instances of objects of c_1'. For example, if c_1 contains a set of relational tuples and c_1' contains a relational database schema then the instance-of links relate tuples in c_1 with tables in c_1'. That is, the contents of c_{in} can be seen as a sort of metadata describing the association of instances to schemas.

Note that the separation between instance and schema allows for several sets of objects to share the same schema, and the same set of objects to be classified under different schemas. For example, in Figure 6.12, schemas o_1' (`Schema_1`) and o_2' (`Schema_2`) share the same instance set, represented by object o_1 (`Instance_1`). On the other hand, instance sets o_1 and o_2 share the same schema o_2' (`Schema_2`). A more realistic example for the second case is given in Figure 6.13, where object o_1 (`Company`) refers to the concept of Company, and objects o_2 (`CompA`) and o_3 (`CompB`) refer to two specific companies. Intuitively, the reference of o_1 corresponds to the schema of a company, and the references of o_2 and o_3 correspond to information about the particular companies. As objects o_2 and o_3 are instances of o_1, objects within the references of o_2 and o_3 are classified into classes in the reference of o_1. This classification takes place within the references c_4 and c_5 of the instance-of links from o_2 and o_3 to o_1, respectively. Of course, at a more abstract level, schemas may be considered as instance objects and thus, be classified to metaschemas.

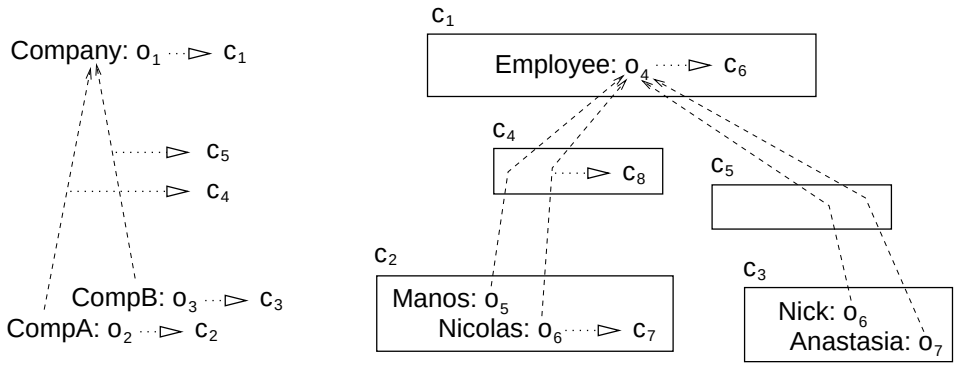


Figure 6.13: Interaction between classification and contextualization

Continuing with the example of Figure 6.13, consider the instance-of link from object o_2 (`CompA`) to object o_1 (`Company`). To satisfy the Instance-of Reference Constraint, the reference c_4 of this link may only contain instance-of links from objects in its source reference (context c_2) to objects in its destination reference (context c_1). In other words, within context c_4, objects of company `CompA` may only be classified to the classes in the schema of `Company`. Indeed, within context c_4, there is an instance-of link from object o_6 (`Nicolas`) to object o_4 (`Employee`). The reference c_7 of object o_6 contains information about the employee `Nicolas`, and the reference c_6 of object o_4 contains schema information about the class `Employee`. To satisfy the Instance-of Reference Constraint, the reference c_8 of the instance-of link may only classify information about the employee `Nicolas` to the classes in the schema of `Employee`. Therefore, recursive application of the Instance-of Reference Constraint implies classification of objects according to their entire recursive structure.

6.5.3 Generalization and contextualization

Generalization establishes a subclass-superclass relation between classes and is used to emphasize the similarities among classes with common superclasses and to hide their differences. The interaction between generalization and attribution is expressed by the well known mechanism of *attribute inheritance*. In our framework, in addition to attribute inheritance, we support a new mechanism that we call *reference inheritance*. Roughly speaking, according to reference inheritance, the reference of the subclass *inherits* the contents of the reference of the superclass.

Formally, reference inheritance is defined through a partial order over contexts that we call *context refinement*.

Definition 6.2 Context refinement.
We say that context c *refines* context c', or that context c is a refinement of c', iff

1.every object of c' is also an object of c,

2.the names of every object in c' are included in the names of the object in c,

3.every predicate of c' is also a predicate of c, and

4.the reference of every object of c refines the reference of the object in c'. $\diamond$

Note that the above definition of context refinement is recursive and that every context is a refinement of itself. We show in Chapter 7 that refinement is a partial pre-ordering, i.e., reflexive and transitive. Moreover, we show that context refinement is a partial ordering up to context equivalence, where context equivalence is defined as follows: two contexts are *equivalent* if they have (i) the same objects, (ii) the same names for each object, (iii) the same predicates, and (iv) the references of each object in the two contexts either both do not exist, or both exist and they are equivalent. Roughly speaking, two contexts are equivalent if they have the same contents, up to equivalence of the object references.[5].

The following constraint expresses the application of reference inheritance on ISA links.

Constraint 6.3 Reference Inheritance Constraint.
The source reference of an ISA link refines the destination reference of the link. $\diamond$

For example, in Figure 6.14, object o_2 (`Hospital`) is a subclass of object o_1 (`Organization`). The source reference of this ISA link (context c_2) is a refinement of the destination of the link (context c_1), as c_2 contains all the contents of c_1. Therefore, the reference inheritance constraint is satisfied. Intuitively, we can say that the contents of c_1 have been inherited by c_2.

To keep the contexts concise, we could eliminate duplications in the contents of the source reference of the ISA link. In this case, the complete contexts are obtained after the application of reference inheritance on the ISA links. A mechanism for eliminating duplications is proposed in Subsection 6.6.

Context refinement can be achieved in stages through the repetitive application of the following operations on the contents of a context:

1. the addition of a new object (possibly an attribute),

2. the specialization, generalization, or classification of an object (possibly an attribute),

[5]Notice the similarity between context equivalence and deep object equality in object oriented databases [4].

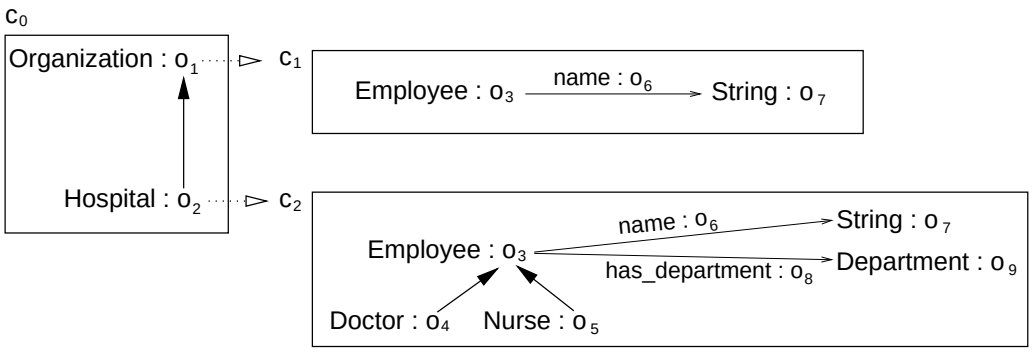


Figure 6.14: Interaction between generalization and contextualization

3. the addition of a name to an object,

4. the addition of a reference to an object, and

5. the application of the previous operations to the contents of a reference.

The resulting context is certainly a refinement of the original context, as it merely extends the information contained in that context (and no cancellation takes place). In this sense, refinement inheritance is a form of strict inheritance.

We now give a more involved example (see Figure 6.15). Let c_2 be a context describing medical services. Within c_2, the class o_4 (`Hospital`) and the class o_5 (`PrivateUnit`) are subclasses of the class o_3 (`Health_Care`). In accordance to the Reference Inheritance Constraint, the reference c_4 of `Hospital` and the reference c_5 of `PrivateUnit` are refinements of the reference c_3 of `Health_Care`. Specifically, contexts c_4 and c_5 inherit all the information contained in context c_3, including the object o_8 (`Agents`) that represents the concept of agent. Within context c_3, the reference c_8 of o_8 describes the `Agents` hierarchy in the general health care environment. Within context c_4, the reference c_9 of o_8 describes the `Agents` hierarchy in the hospital environment. Within context c_5, the reference c_{10} of o_8 describes the `Agents` hierarchy in the private unit environment. Note that although contexts c_9 and c_{10} refine context c_8, they describe different hierarchies. For example, c_9 indicates that the director of a hospital should be a doctor, whereas c_{10} indicates that the director of a private unit should be owner of the unit.

6.5.4 Classification, generalization and contextualization

The interaction between classification and generalization is usually expressed through the following constraint: If an object o is instance of a class o', and o' is subclass of o'', then o is instance of o''. In our framework this inference rule imposes a constraint on the references of the instance-of links. Let c_1 and c_2 be the references of the instance-of links from o to o' and from o to o'', respectively, as shown in Figure 6.16. The question is whether there is any relationship between the contents of c_1 and c_2. Indeed, we show that c_1 refines c_2.

Let c, c' and c'' be the references of the objects o, o' and o'', respectively. As c' refines c'' (see the Reference Inheritance Constraint, Constraint 6.3), it should hold that if an object w in c is classified in a class w'' in c'' then, w should also be classified in the inherited class w' in c'. That is, the instance-of links contained in c_2 should be inherited by c_1, and the reference of an

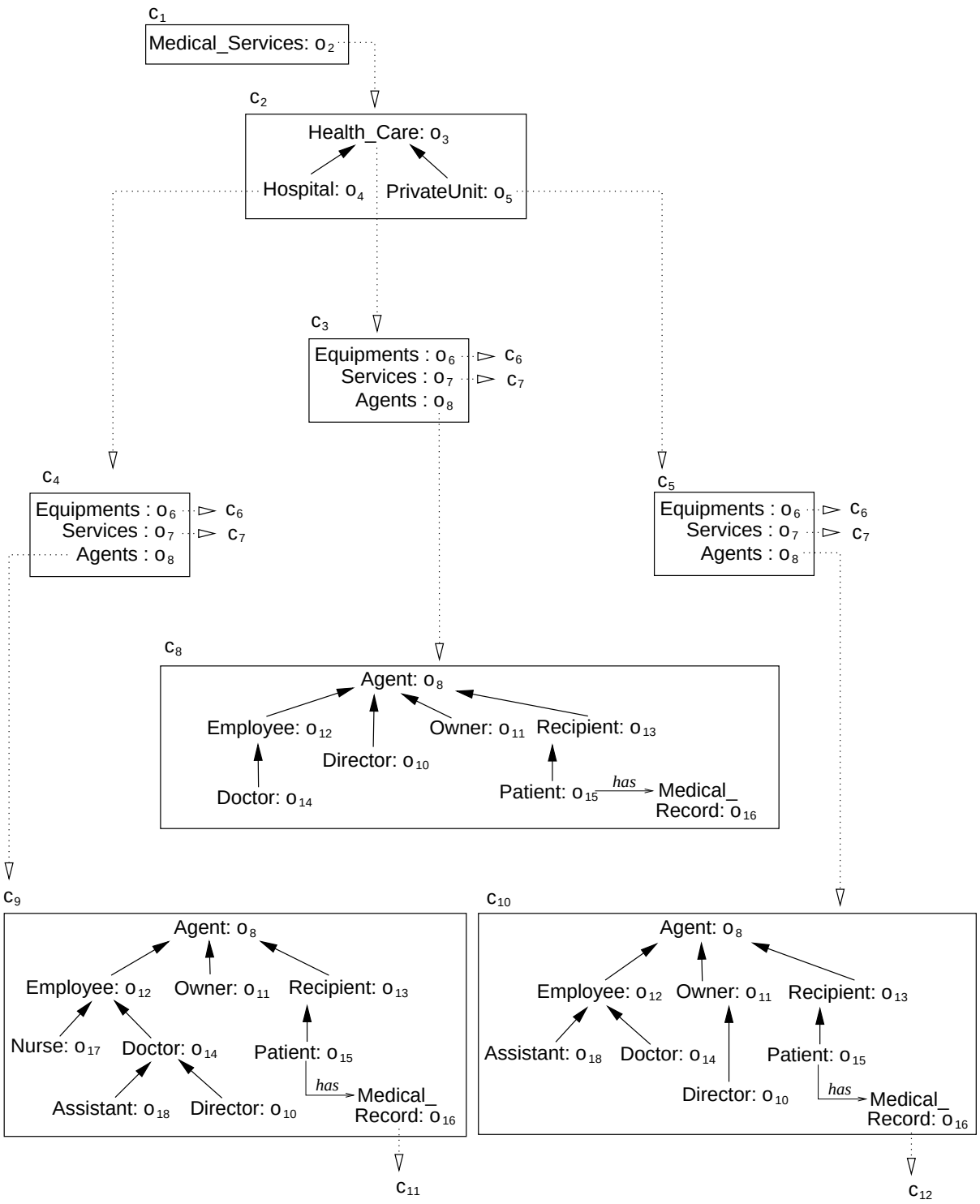


Figure 6.15: Interaction between generalization and contextualization: An example from a medical environment

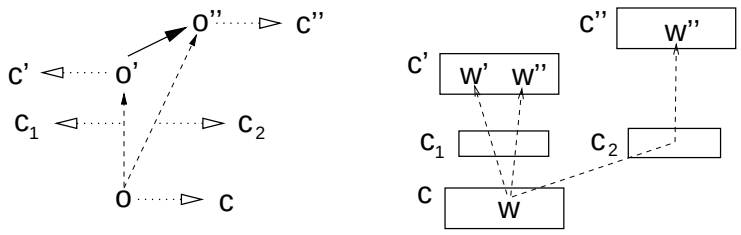


Figure 6.16: The interaction between classification, generalization, and contextualization

inherited link in c_1 should refine the reference of this link in c_2. This implies that c_1 should refine c_2. We refer to this constraint as *Instance-of Inheritance Constraint*.

Constraint 6.4 Instance-of Inheritance Constraint.
The reference of an instance-of link from an object o to a class o' refines the reference of any instance-of link from o to a superclass of o'. $\diamond$

Duplications of instance-of links in context c_1 can be avoided through an inheritance mechanism similar to that applied in the interaction between generalization and contextualization, described in the previous subsection. One such mechanism is proposed in Section 6.6.

6.6 Keeping contexts concise

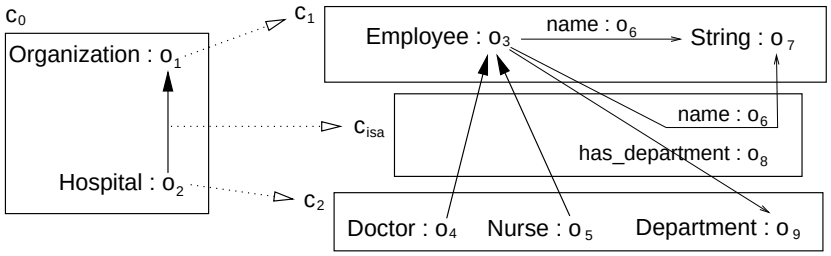
If a context c refines a context c' then the c "inherits" the contents of c'. This produces duplication of information contained in contexts c and c'. In this section we present two alternative techniques which avoid duplication of information between contexts related with context refinement. If we avoid such duplication of information we say that we keep contexts *concise*.

As we have already mentioned ISA links are treated as objects and thus they may have a reference within a context. This feature can be used to keep contexts concise in the following manner: Assume that the object o is ISA related with the object o' and that the reference of the lower-level object o is the context c whereas the reference of the higher-level object o' is the context c'. To keep contexts concise we eliminate duplications in the contents of context c, i.e., context c does not contain the information inherited from c'. However, context c may now contain relationships that connect objects of c with objects of c' (this is a consequence of refinement). In order for c to be consistent with the definition of context these relationships are also eliminated from the contents of c and are placed into the contents of the reference of ISA link. This is the minimum information that must be eliminated from context c in order to keep contexts concise. It is a modeling issue to decide whether more information is to be eliminated from context c and be placed into the contents of the ISA link reference.

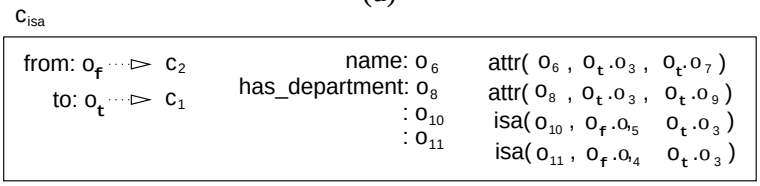
Conversely, when we say that the contents of the reference of the higher-level object as well as the contents of the reference of ISA links are *inherited* by the reference of the lower-level object, we mean that we can think of the reference of the lower-level object as a “virtual” context containing the contents of c, the contents of c', and the contents of the reference of the ISA link.

For example, Figure 6.17 illustrates the concise version of contexts shown in Figure 6.14. Another example is shown in Figure 6.18.

In order to maintain a concise model for context, we should be able to compute a complete



(a)



(b)

Figure 6.17: Maintaining contexts concise (example 1)

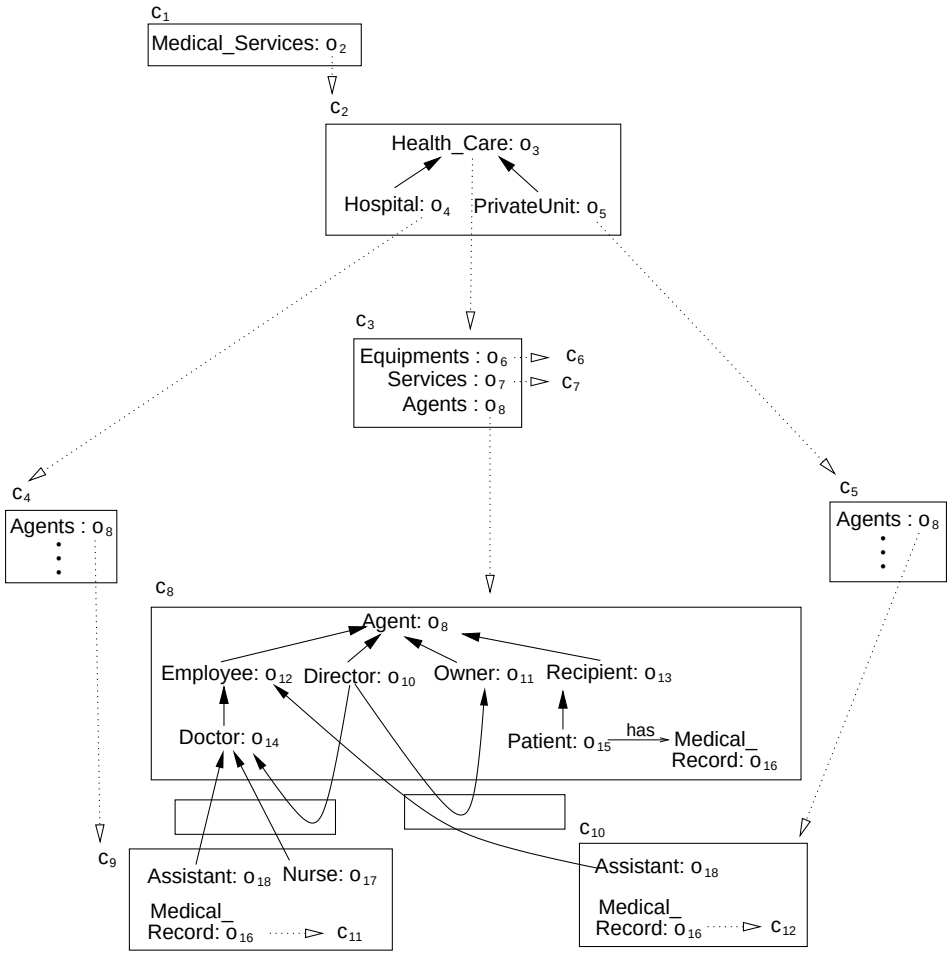


Figure 6.18: Maintaining contexts concise (example 2)

context from a concise one. In other words, given a concise context c we should be able to compute the complete one, i.e., a context that contains all the information inherited by c from all contexts refined by c. Then, all axioms of our context model hold for the complete contexts. Complete contexts are computed always w.r.t. a wider context. This is because, a context c can be refered by different objects within different contexts and the contexts that c refines depends on the superclasses of the objects that refer to c.

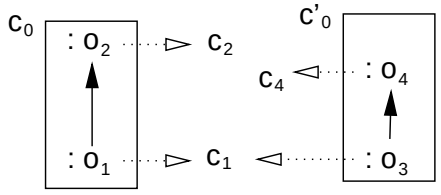


Figure 6.19: The computation of a complete context is dependend on a wider one

For example, in Figure 6.19, context c_1 refines context c_2 if we consider that the wider context is the context c_0. This is because, within context c_0, object o_1, which refers to context c_1, is a subclass of object o_2, which refers to context c_2. On ther other hand, context c_1 refines context c_4 if consider that the wider context is the context c_0'. This is because, within context c_0', object o_3, which refers to context c_1, is a subclass of object o_4, which refers to context c_4. The operation that computes the complete context of a context c w.r.t. a wider context c_0, is denoted by $complete_{c_0}(c)$, and is defined in the sequel.

- **complete**$_{c_0}(c)$

  This operation takes as input two contexts c_0 and c, and returns a context (call this context c_{out}) as a result. The computational algorithm of this operation is shown in Figure 6.20. This operation does the following:

  1. $cnts(c_{out}) = cnts(c)$.
  2. Find all objects o which refer to context c within context c_0.
  3. Find all immediate superclasses o' of o and the corresponding ISA link objects o_{isa}.
  4. For each superclass o' and the corresponding ISA link object o_{isa} do
     (a) If $c' = ref(c_0)(o')$ then
     Merge the contents of $complete_{c_0}(c')$ with the contents of c_{out}.
     (b) If $c_{isa} = ref(c_0)(o_{isa})$ then
     Merge the contents of c_{isa} with the contents of c_{out}, after having removed objects o_f and o_t from objects of c_{isa} as well as any reference of o_f and o_t that appear in the predicates of c_{isa}.

- **Merging the contents of two contexts**
  This operation merges the contents of a context c with the contents of c_{out} is done as follows:

  1. $objs(c_{out}) = objs(c_{out}) \cup objs(c)$.
  2. $attr\_pred(c_{out}) = attr\_pred(c_{out}) \cup attr\_pred(c)$.
  3. $in\_pred(c_{out}) = in\_pred(c_{out}) \cup in\_pred(c)$.

complete$_{c_0}$(c), where $c_0, c \in \mathcal{CXT}$.
/* This operation takes as input a context c and returns a new context c', which contains the contents of c as well as all information inherited by c from the contexts that c refines. */

1. $c_{out} = createCxt()$.
2. $cnts(c_{out}) = cnts(c)$.
3. Let $AbsO$ be the set of all objects o of c_0 each of which refers to context c, i.e., $AbsO = \{o \in objs(c_0) \mid ref(o, c_0) = c\}$.
4. For each object $o \in AbsO$ do
   /* Collect all superclasses of o and the corresponding ISA links. */
   (a) Let $SuperCls$ an empty set of objects.
   /* $SuperCls$ will contain the immidiate superclasses of o */
   (b) Let $ISAlinks$ be an empty set of objects.
   /* $ISAlinks$ will contain all ISA link object with source the object o */
   (c) For each object $o' \in objs(c_0)$ and object $o_{isa} \in objs(c_0)$ such that $isa(o_{isa}, o, o') \in isa\_pred(c_0)$ do
      i. $SuperCls = SuperCls \cup \{o'\}$.
      ii. $ISAlinks = ISAlinks \cup \{o_{isa}\}$.
5. For each object $o' \in SuperCls$ do
   $c_{out} = mergeCxts(c_{out}, complete_{c_0}(ref(o', c_0)))$.
6. For each object $o_{isa} \in ISAlinks$ do
   $c_{out} = mergeCxts(c_{out}, strip(ref(o_{isa}, c_0)))$.
7. End.

mergeCxts(**Input** $c_1, c_2 : \mathcal{CXT}$; **Output** $c_{out} : \mathcal{CXT}$).

1. $c_{out} = createCxt()$.
2. $objs(c_{out}) = objs(c_1) \cup objs(c_2)$.
3. $attr\_pred(c_{out}) = attr\_pred(c_1) \cup attr\_pred(c_2)$.
4. $in\_pred(c_{out}) = in\_pred(c_1) \cup in\_pred(c_2)$.
5. $isa\_pred(c_{out}) = isa\_pred(c_1) \cup isa\_pred(c_2)$.
6. For each object $o \in objs(c_{out})$ do
   (a) $names(o, c_{out}) = names(o, c_1) \cup names(o, c_2)$.
   (b) $ref(o, c_{out}) = mergeCxts(ref(o, c_1), ref(o, c_2))$.
7. End.

Figure 6.20: The algorithms of the operations $complete$ and $mergeCxts$.

4. $isa\_pred(c_{out}) = isa\_pred(c_{out}) \cup isa\_pred(c)$.

5. For each object $o \in objs(c)$ we have

   (a) $names(o, c_{out}) = names(o, c_{out}) \cup names(o, c)$.

   (b) $ref(o, c_{out}) = \begin{cases} ref(o, c), & \text{if } ref(o, c) \text{ is defined and} \\ & ref(o, c_{out}) \text{ is undefined} \\ merge(ref(o, c_{out}), ref(o, c)), & \\ & \text{if } ref(o, c) \text{ is defined and} \\ & ref(o, c_{out}) \text{ is defined} \end{cases}$

6.7 Context-based information bases

Any information base that supports contextualization, typically in addition to the traditional abstraction mechanisms, we shall call a *context-based information base.*

Adding contextualization to an information base carries several benefits, including the following:

- *Modular representation*: At each level of abstraction, an overview of available information can be presented, with access to the hidden detail.

- *Focused information access*: A context delimits the parts of an information base that are accessible in a given way. Thus contexts can act as a focusing mechanism when searching for information.

- *Context-dependent semantics*: A given object may be represented and interpreted differently in different context-delimited parts of the information base.

- *Ability to handle inconsistent information*: Contradictory information can be represented in the same information base as long as it is treated in different contexts.

However, in order to support contextualization within an information base the necessary environment for context creation should be specified. This environment, that we shall call *the information base environment*, must include three mutually disjoint sets: a set of names, a set of object identifiers and a set of context identifiers.

In fact, one can think of an information base as consisting of two parts (see Figure 6.21(a)): (i) the information base environment, and (ii) a special context, which contains the contents of the information base.

When several information bases can co-exist in the same system (e.g. in a multi-base), we can group together all information bases into a single context as shown in Figure 6.21(b). Indeed, each information base can be seen as an object o_i having a name, say $\texttt{IB}_i$, and a reference, say *IB$_i$-context*, whose contents are the contents of the information base $\texttt{IB}_i$. In the resulting context, (*MB-context* in Figure 6.21(b)), we can even structure the set of information bases as explained in the previous section, e.g. each information base can be related to other information bases through instance-of, ISA or attribute links.

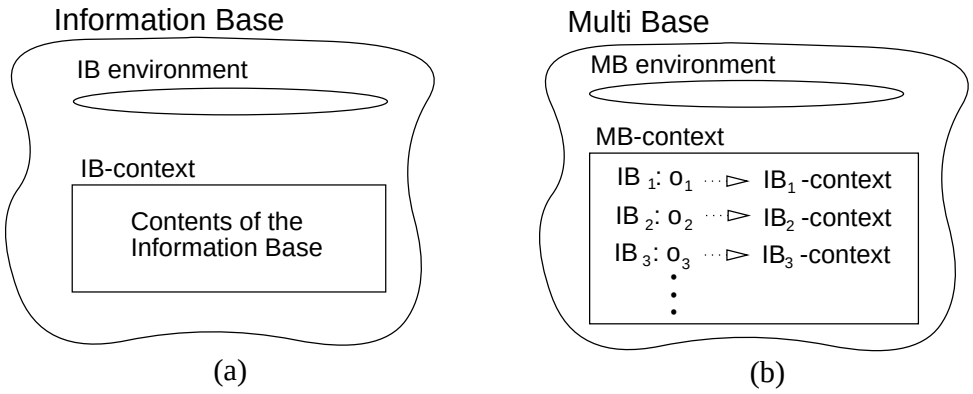


Figure 6.21: Context-based information bases

6.8 Summary

In this chapter, we extended the definition of context by introducing the notion of reference and structure: The reference of the object is another context which "hides" detailed information about the object. Within a context objects can be structured through the traditional abstraction mechanisms of classification, generalization and attribution. We also study how the contextualization mechanism interact with the traditional abstraction mechanisms.

Adding contextualization to an information base provides several modeling capabilities, including the following: (i) modular representation (at each level of abstraction, an overview of available information can be presented, with access to the hidden detail), (ii) focused information access (a context delimits the parts of an information base that are accessible in a given way), (iii) context-dependent semantics (a given object may be represented and interpreted differently in different context delimited parts of the information base), (iv) ability to handle inconsistent information (contradictory information can be represented in the same information base as long as it is treated in different contexts). (v) top-down, bottom-up, or mixed modeling.

Chapter 7

A Theory for Contextualized Information Bases

7.1 Information base contents

We now present a formal foundation of the contextualization abstraction mechanism.

We assume an information base consisting of the following three basic components:

- A set of *object identifiers*, or simply *objects*, denoted by $\mathcal{O}$.
- A set of *context identifiers*, or simply *contexts*, denoted by $\mathcal{CXT}$.
- A set of *atomic names*, denoted by $\mathcal{N}$.

Recall that contexts allow us to group, or package, information and that each context c is associated with a (possibly structured) set of objects, denoted by $objs(c)$. The contents of a context c consist of:

1. A set of objects, called *the objects of* c, denoted by $objs(c)$.
2. A mapping associating each object o of c with a set of names.
   This mapping is called *the lexicon of* c, and is denoted by $lex(c)$. The names bound to object o within context c are called *the names of* o *in* c: $lex(c)(o) = names(o, c)$.
3. A partial function associating an object o of c with a context.
   This function is called *the reference association of* c, and denoted by $rf(c)$. If $c' = rf(c)(o)$ then we say that c' is *the reference of* o in context c: $rf(c)(o) = ref(o, c)$.
4. A set of instance-of, ISA, or attribute links.
   These links relate two objects, which can be either both objects of c, or each one can be reached through a path starting from an object of c. Thus, in general, the source or destination of a link is a path starting from and object of c. Paths of length one are considered identical to the object consisting the path.

We have already mentioned that attribute links are first-class objects. For uniform treatment with attribute links, in the formal model, instance-of and ISA links are also considered to be first-class objects.

Accessing information in an information base often involves navigating from one object to another by following links [59]. Navigation relies on the notion of *path*. We distinguish two kinds of paths:

1. *reference paths* which support navigation through the reference of an object within a context, and
2. *structural paths* which support navigation through attribute, instance-of, and ISA links.

In order to formally define the contents of a context we first need to define the notion of reference path.

7.2 The notion of reference path

As the reference of an object within a context is also a context, references provide a means to traverse from an object o of one context c to the objects of another context via the reference of o in c. Dot notation (.) is used to specify this traversal. The sequence of traversed objects constitutes a kind of path, which we call *reference path*.

Definition 7.1 Reference Path.

A *reference path* $o_1.o_2.\ \cdots\ .o_{n-1}.o_n$ w.r.t. a context c_0 is a sequence of objects where each object o_{i+1} is contained in the reference of the previous object o_i. The reference of o_1 is taken in c_0 (call this reference c_1), the reference of o_2 is taken in c_1, and so on. The set of all reference paths in the context c_0, denoted by $\mathcal{RP}_{c_0}$, is defined as follows:

$$\mathcal{RP}_{c_0} = \{o_1.o_2.\ \cdots\ .o_{n-1}.o_n \mid n \in I\!N_0 \ \wedge\ (\forall 1 \leq i \leq n,\ o_i \in \mathcal{O}) \ \wedge \\ (\forall 1 \leq i < n,\ \exists c_i, c_{i-1} \in \mathcal{CXT} :\ o_i \in objs(c_{i-1}) \ \wedge\ c_i = ref(o_i, c_{i-1}))\}. \diamond$$

In fact, a reference path $o_1.o_2.\ \cdots\ .o_{n-1}.o_n$ in c_0 is used to reach object o_n in context c_{n-1} starting from object o_1 in c_0.

We denote by $\mathcal{RP}$, the set of all possible reference paths, that is: $\mathcal{RP} = \bigcup_{c \in \mathcal{CXT}} \mathcal{RP}_c$.

We introduce two primitive operations on paths[1]:

1. $first(p)$: it returns the first element of path p.
2. $rest(p)$: it returns the path resulting from path p after removing its first element.

Based on these primitive operations, a number of useful operations on paths can be defined:

[1]These are the well-known operations *car* and *cdr* used in LISP.

1. $length(p)$: it returns the length of path p.
2. $elem(i, p)$: it returns the ith element of path p (assuming $1 \leq i \leq length(p)$).
3. $second(p)$: it returns the second element of path p ($second(p) = elem(2, p)$).
4. $last(p)$: it returns the last element of path p ($last(p) = elem(length(p), p)$).

7.3 Contents of a context

Let:

- $\mathcal{L}$ be the set of all lexicons. A *lexicon* is a relation of the form: $\mathcal{O} \times \mathcal{P}(\mathcal{N})$ that associates objects with sets of names.
- $\mathcal{REF}$ be the set of all *reference associations*. A *reference association* is a relation of the form: $\mathcal{O} \times \mathcal{CXT} \cup \{NIL\}$ that associates objects with contexts or with the special symbol *NIL*.
- $\mathcal{IN}$ be the set of all triplets that define instance-of links. That is:

$$\mathcal{IN} = \{\langle o, p_f, p_t\rangle \mid o \in \mathcal{O},\ p_f, p_t \in \mathcal{RP}\}$$

- $\mathcal{ISA}$ be the set of all triplets that define ISA links. That is:

$$\mathcal{ISA} = \{\langle o, p_f, p_t\rangle \mid o \in \mathcal{O},\ p_f, p_t \in \mathcal{RP}\}$$

- $\mathcal{ATTR}$ be the set of all triplets that define attribute links. That is:

$$\mathcal{ATTR} = \{\langle o, p_f, p_t\rangle \mid o \in \mathcal{O},\ p_f, p_t \in \mathcal{RP}\}$$

We can now define the contents of a context:

Definition 7.2 Contents of a context.
We define *the contents of a context c*, denoted by $cnts(c)$, as the tuple
$\langle\ objs(c), lex(c), rf(c), attr(c), in(c), isa(c)\ \rangle$
where

1. $objs(c) \in \mathcal{P}(\mathcal{O})$ is a set of objects.
2. $lex(c) : objs(c) \longrightarrow \mathcal{P}(\mathcal{N})$ is a function which associates each object of c with a set of names ($lex(c) \subseteq \mathcal{L}$). We call this function the *lexicon of c*.
3. $rf(c) : objs(c) \longrightarrow \mathcal{CXT} \cup \{NIL\}$ is a function which associates an object of c with a context, called *the reference of* that object ($rf(c) \subseteq \mathcal{REF}$). We call this partial function the *reference association of c*.
4. $in(c) = \{\langle o, p_f, p_t\rangle \in \mathcal{IN} \mid o \in objs(c) \wedge p_f, p_t \in \mathcal{RP}_c\}$ is a set of triplets of the form $\langle o, p_f, p_t\rangle$, expressing that o is an instance-of link in context c from reference path p_f to reference path p_t.

5. $isa(c) = \{\langle o, p_f, p_t\rangle \in \mathcal{ISA} \mid o \in objs(c) \ \wedge \ p_f, p_t \in \mathcal{RP}_c\}$ is a set of triplets of the form $\langle o, p_f, p_t\rangle$, expressing that o is an ISA link in context c from reference path p_f to reference path p_t.

6. $attr(c) = \{\langle o, p_f, p_t\rangle \in \mathcal{ATTR} \mid o \in objs(c), \ p_f, p_t \in \mathcal{RP}_c\}$ is a set of triplets of the form $\langle o, p_f, p_t\rangle$, expressing that o is an attribute link in context c from reference path p_f to reference path p_t. $\diamond$

The fact that the source and the destination of a link are reference paths implies that (i) the related objects (i.e., last objects in the paths) are contextualized, i.e., they are viewed from the last accessed context navigating through the path, and (ii) it is taken into account the navigation to reach the last object in the path (i.e., two different paths reaching the same object defines two different links). For example, in Figure 6.11, o_{11} is an attribute link in c_4 from $o_f.o_5$ to $o_t.o_8$. As contexts c_2 and c_3 are the last accessed contexts navigating through the paths $o_f.o_5$ to $o_t.o_8$, respectively, the objects related by the attribute link o_{11} (i.e., objects o_5 and o_8) are contextualized within contexts c_2 and c_3, respectively.

Definition 7.3 Contextualized information base.

A *contextualized information base* is a tuple of the form:

$\langle \mathcal{O}, \mathcal{CXT}, \mathcal{N}, \mathcal{L}, \mathcal{IN}, \mathcal{ISA}, \mathcal{ATTR}, \mathcal{REF}, cnts \rangle$

where $cnts$ is a total function that associates a context with its contents. $\diamond$

A special kind of context is the *link context*, i.e., a context which is the reference of a link (attribute, instance-of, or ISA). Recall that the reference of an object contains information about this object (from a particular standpoint). As a link expresses an association between its source and destination objects, the reference of the link may contain some information about the source and destination reference of the link[2]. To store this information in the contents of a context, we add two special objects o_f and o_t, with names `from` and `to`, respectively. These two objects must be contained in every link context. As we have seen in subsection 6.5.1, if a link has as reference a link context c then the source and destination reference of the link should refine the references of objects o_f and o_t in c, respectively. In other words, information in the references of objects o_f and o_t in c should also be contained in the source and destination reference of the corresponding link. We denote the set of link contexts by $\mathcal{LCXT}$. Clearly, it holds: $\mathcal{LCXT} \subset \mathcal{CXT}$.

Definition 7.4 Contents of a link context.

The objects of a link context must contain the special objects, o_f and o_t, with names `from` and `to`, respectively. That is:

$\forall c \in \mathcal{LCXT}: \ o_f, o_t \in objs(c) \ \wedge \ \texttt{from} \in names(o_f, c) \ \wedge \ \texttt{to} \in names(o_t, c)$. $\diamond$

Definition 7.5 The empty context (C_{empty}).

In every information base, there is a special context C_{empty}, called *empty context*, which contains all built-in information[3] (objects, lexicon, and links) shared by all contexts. $\diamond$

[2] We refer to the reference of the source (resp. destination) of a link as the *source* (resp. *destination*) reference of that link (see also subsection 6.5.1).

[3] Built-in information is not needed to be created explicitly by the users.

For example, in an object-oriented database, the empty context will contain all built-in classes and their relationships.

7.4 Predicates and functions

In this section, we define a number of predicates and functions which will be used to express the axioms of our theory.

7.4.1 Predicates

We define the following predicates:

1. The predicate $In_c(o, p_f, p_t)$ expresses that o is an instance-of link in context c from p_f to p_t, and is defined as follows:

   $$\forall c \in \mathcal{CXT},\ o \in \mathcal{O},\ p_f, p_t \in \mathcal{RP}_c :$$

   $$In_c(o, p_f, p_t) \Leftrightarrow \langle o, p_f, p_t \rangle \in in(c).$$

2. The predicate $Isa_c(o, p_f, p_t)$ expresses that o is an ISA link in context c from p_f to p_t, and is defined as follows:

   $$\forall c \in \mathcal{CXT},\ o \in \mathcal{O},\ p_f, p_t \in \mathcal{RP}_c :$$

   $$Isa_c(o, p_f, p_t) \Leftrightarrow \langle o, p_f, p_t \rangle \in isa(c).$$

3. The predicate $Attr_c(o, p_f, p_t)$ expresses that o is an attribute link in context of c from p_f to p_t, and is defined as follows:

   $$\forall c \in \mathcal{CXT},\ o \in \mathcal{O},\ p_f, p_t \in \mathcal{RP}_c :$$

   $$Attr_c(o, p_f, p_t) \Leftrightarrow \langle o, p_f, p_t \rangle \in attr(c).$$

4. The predicate $IsLink_c(o)$ expresses that object o is a link in context c, and is defined as follows:

   $$\forall c \in \mathcal{CXT},\ o \in \mathcal{O} :$$

   $$IsLink_c(o) \Leftrightarrow\ \exists p_f, p_t \in \mathcal{RP}_c :\ \ Attr_c(o, p_f, p_t)\ \vee\ In_c(o, p_f, p_t)\ \vee\ Isa_c(o, p_f, p_t)$$

5. The predicate[4] $c \precsim c'$ expresses that context c *refines* context c'.[5]

[4] Note that $c \precsim c'$ is the infix notation of $\precsim (c, c')$.

[5] The notion of context refinement was introduced in subsection 6.5.3.

7.4.2 Functions

We define the following functions:

1. $From_c : \mathcal{O} \longrightarrow \mathcal{RP}_c$.

   This function is defined w.r.t. a context c, takes as input a link o, and returns the source of o in c. That is:

   $$\forall c \in \mathcal{CXT},\ o \in \mathcal{O},\ p \in \mathcal{RP}_c :$$

   $$From_c(o) = p \Leftrightarrow\ \exists p' \in \mathcal{RP}_c :\ \ Attr_c(o,p,p')\ \vee\ In_c(o,p,p')\ \vee\ Isa_c(o,p,p').$$

   We call this function *the source object of* o in context c.

   For example, in Figure 6.11, $From_{c_4}(o_{11}) = o_f.o_5$.

2. $To_c : \mathcal{O} \longrightarrow \mathcal{RP}_c$.

   This function is defined w.r.t. a context c, takes as input a link o, and returns the destination of o in c. That is:

   $$\forall c \in \mathcal{CXT},\ o \in \mathcal{O},\ p \in \mathcal{RP}_c :$$

   $$To_c(o) = p \Leftrightarrow\ \exists p' \in \mathcal{RP}_c :\ \ Attr_c(o,p',p)\ \vee\ In_c(o,p',p)\ \vee\ Isa_c(o,p',p).$$

   We call this function *the destination object of* o in context c.

   For example, in Figure 6.11, $To_{c_4}(o_{11}) = o_t.o_8$.

3. $Ref_c : \mathcal{RP}_c \longrightarrow \mathcal{CXT}$.

   This function is defined w.r.t. a context c, takes as input a reference path p in c and returns the reference of the last object in p w.r.t. the last accessed context navigating through p. That is:

   $$\forall c \in \mathcal{CXT}, p \in \mathcal{RP}_c :\ \ Ref_c(p) = \begin{cases} ref(p,c), & \text{if } length(p) = 1 \\ Ref_{ref(first(p),c)}(rest(p)), & \text{if } length(p) > 1 \end{cases}$$

   Note that the above definition is recursive and terminates when $length(p) = 1$. We call this function *the reference of* path p in context c.

   For example, in Figure 6.9, $Refc(o_1) = c_1$, $Refc(o_1.o) = c_2$ and $Refc(o_2.o) = c_4$.

4. $Names_c : \mathcal{RP}_c \longrightarrow \mathcal{P}(\mathcal{N})$.

   This function is defined w.r.t. a context c, takes as input a reference path p in c and returns the names of the last object in p w.r.t. the last accessed context navigating through p. That is:

   $$\forall c \in \mathcal{CXT}, p \in \mathcal{RP}_c :\ \ Names_c(p) = \begin{cases} names(p,c), & \text{if } length(p) = 1 \\ Names_{ref(first(p),c)}(rest(p)), & \text{if } length(p) > 1 \end{cases}$$

   Note that the above definition is recursive and terminates when $length(p) = 1$. We call this function *the names of* path p in context c.

   For example, in Figure 6.11, $Names_{c_1}(o_f) = \{\texttt{From}\}$ and $Names_{c_1}(o_t) = \{\texttt{To}\}$.

7.5 The notion of link path

In this section, we define the notion of link path and traversal path. As the source and the destination of a link (attribute, instance-of, and ISA) are objects (reached through reference paths), links provide a means to traverse from one object to another. For each such traversal, there is a corresponding sequence of traversed links. This sequence of traversed links (recall that links are objects themselves) constitutes a kind of path which we shall call *link path*. Link paths, similar to reference paths, are given in a context c.

Definition 7.6 Link path.

The set of *link paths* in a context c, denoted by $\mathcal{LP}_c$, is defined as follows:

$$\mathcal{LP}_c = \{o_1.o_2.\ \cdots\ .o_{n-1}.o_n \mid n \in I\!N_0\ \wedge\ \forall 1 \leq i < n :$$
$$IsLink_c(o_i)\ \wedge\ IsLink_c(o_{i+1})\ \wedge\ (From_c(o_{i+1}) = To_c(o_i))\ \vee\ From_c(o_{i+1}) = o_i)\}. \diamond$$

Intuitively, a link path in c is a sequence of links of c, where the source object of each link in the sequence coincides with the destination object of the previous link, or the previous link itself. A link path in c whose all members are either attributes, or instance-of, or ISA links in c will be called *attribute*, *instance-of*, or *ISA path* in c, respectively.

For example, in Figure 6.11, there are three link paths in context c_4: $p_1 = o_{12}.o_{14}$, $p_2 = o_{12}.o_{16}$, and $p_3 = o_{11}$.

If c is a link context then we have mentioned that c will contain the special objects o_f and o_t, whose reference is refined by the source and destination reference of the corresponding link, respectively. We shall call *traversal path in* c, any link path in c that traverses from an object reached through a reference path starting from o_f, to an object reached through a reference path starting from o_t. Specifically, the traversal path should satisfy the following: (i) the source of its first link is a reference path starting from o_f, and (ii) the destination of its last link is a reference path starting from o_t. Intuitively, a traversal path in a link context traverses from an object in the source reference, to an object in the destination reference of the corresponding link.

Definition 7.7 Traversal Path.

The set of all traversal paths in a link context c, denoted by $\mathcal{TP}_c$, is defined as follows:

$$\mathcal{TP}_c = \{p \in \mathcal{LP}_c \mid first(From_c(first(p))) = o_f\ \wedge\ first(To_c(last(p))) = o_t\}. \diamond$$

For example, in Figure 6.11, there are two traversal paths within context c_4: $p_1 = o_{12}.o_{14}$ and $p_3 = o_{11}$.

The set of all attribute paths in c which are also traversal paths is denoted as $\mathcal{ATP}_c$. The set of all instance-of paths in c which are also traversal paths is denoted as $\mathcal{INTP}_c$, and the set of instance-of traversal paths in c of length one is denoted by $\mathcal{INTP}^1_c$, i.e., $\mathcal{INTP}^1_c = \{p \in \mathcal{INTP}_c \mid length(p) = 1\}$.

7.6 Core axioms

In this section, we present the core axioms of our theory that should be satisfied by any contextualized information base. Additional axioms may be added to this core set of axioms

in order to support extra modeling features.

Axiom 7.1 Contextualized ISA reflexivity and transitivity.

1.ISA reflexivity.

Let c be a context. For every reference path p in c, there is an ISA link in c from p to p.

$$\forall c \in \mathcal{CXT},\ p \in \mathcal{RP}_c,\ \ \exists o \in \mathcal{O}:\ \ Isa_c(o,p,p).\ \diamond$$

2.ISA transitivity.

Let c be a context and p_1, p_2, p_3 be reference paths in c. If there are two ISA links in c from p_1 to p_2, and from p_2 to p_3, respectively, then there is an ISA link in c from p_1 to p_3.

$$\forall o_1, o_2 \in \mathcal{O},\ c \in \mathcal{CXT},\ p_1, p_2, p_3 \in \mathcal{RP},\ \ \exists o_3 \in \mathcal{O}:$$

$$Isa_c(o_1,p_1,p_2)\ \wedge\ Isa_c(o_2,p_2,p_3)\ \ \Rightarrow\ \ Isa_c(o_3,p_1,p_3).\ \diamond$$

The above axioms are contextualized versions of the ISA reflexivity and ISA transitivity properties of conventional object-oriented systems.

Axiom 7.2 Context Refinement.

If context c *refines* context c' then (i) every object of c' is also an object of c, (ii) the names of every object in c' are included in the names of the object in c, (iii) every instance-of, ISA, or attribute link in c' is also an instance-of, ISA, or attribute link in c, and (iv) the reference of every object of c' refines the reference of the object in c.

1.$\forall o \in \mathcal{O},\ c, c' \in \mathcal{CXT}:$

$$c \precsim c'\ \wedge\ o \in objs(c')\ \ \Rightarrow\ \ o \in objs(c)\ \wedge\ Names_{c'}(o) \subseteq Names_c(o).$$

2.$\forall c, c' \in \mathcal{CXT}:\ \ c \precsim c'\ \ \Rightarrow\ \ in(c') \subseteq in(c).$

3.$\forall c, c' \in \mathcal{CXT}:\ \ c \precsim c'\ \ \Rightarrow\ \ isa(c') \subseteq isa(c).$

4.$\forall c, c' \in \mathcal{CXT}:\ \ c \precsim c'\ \ \Rightarrow\ \ attr(c') \subseteq attr(c).$

5.$\forall o \in \mathcal{O},\ c, c', c'_1 \in \mathcal{CXT},\ \exists c_1 \in \mathcal{CXT}:$

$$c \precsim c'\ \wedge\ Ref_{c'}(o) = c'_1\ \ \Rightarrow\ \ Ref_c(o) = c_1\ \wedge\ c_1 \precsim c'_1.\ \diamond$$

This axiom is justified and exemplified in subsection 6.5.3.

Axiom 7.3 Refinement is reflexive and transitive.

The refinement relation between contexts is reflexive and transitive.

1.Reflexivity.

$$\forall c \in \mathcal{CXT}:\ \ c \precsim c.$$

2.Transitivity.

$$\forall c_1, c_2, c_3 \in \mathcal{CXT}:\ \ c_1 \precsim c_2\ \wedge\ c_2 \precsim c_3\ \ \Rightarrow\ \ c_1 \precsim c_3.\ \diamond$$

Axiom 7.4 Contextualized instance upward inheritance.

Let c be a context and p_1, p_2, p_3 be reference paths in c. If there is an instance-of link in c from p_1 to p_2, and an ISA link in c from p_2 to p_3, then there is an instance-of link in c from p_1 to p_3.

$$\forall o_1, o_2 \in \mathcal{O},\ c \in \mathcal{CXT},\ p_1, p_2, p_3 \in \mathcal{RP},\ \ \exists o_3 \in \mathcal{O}:$$
$$In_c(o_1, p_1, p_2)\ \wedge\ Isa_c(o_2, p_2, p_3)\ \ \Rightarrow\ \ In_c(o_3, p_1, p_3).\ \diamond$$

In conventional object-oriented systems, instances of classes are also instances of their superclasses. The above axiom is the contextualized version of this property.

Axiom 7.5 Interaction between attribution and contextualization.

Let c be a context and p_f, p_t be reference paths in c. If o is an attribute link in a context c from p_f to p_t, and c_a is the reference of o in c, then the references of p_f and p_t in c refine the references of the special objects o_f and o_t in c_a. Additionally, every traversal path in c is an attribute path in c.

$$\forall o \in \mathcal{O},\ c, c_f, c_t \in \mathcal{CXT},\ p_f, p_t \in \mathcal{RP},\ c_a \in \mathcal{LCXT}:$$
$$Attr_c(o, p_f, p_t)\ \wedge\ c_a = Ref_c(o)\ \wedge\ c_f = Ref_{c_a}(o_f)\ \wedge\ c_t = Ref_{c_a}(o_t)\ \ \Rightarrow$$
$$Ref_c(p_f) \precsim c_f\ \wedge\ Ref_c(p_t) \precsim c_t\ \wedge\ \mathcal{TP}_c = \mathcal{ATP}_c.\ \diamond$$

This axiom is justified and exemplified in subsection 6.5.1.

Axiom 7.6 Interaction between classification and contextualization.

Let c be a context and p_f, p_t be reference paths in c. If o is an instance-of link in c from p_f to p_t, and c_{in} is the reference of o in c, then the references of p_f and p_t in c refine the references of the special objects o_f and o_t in c_{in}. Additionally, every traversal path in c is an instance-of link in c.

$$\forall o \in \mathcal{O},\ c, c_f, c_t \in \mathcal{CXT},\ p_f, p_t \in \mathcal{RP},\ \ \exists c_{in} \in \mathcal{LCXT}:$$
$$In_c(o, p_f, p_t)\ \wedge\ c_{in} = Ref_c(o)\ \wedge\ c_f = Ref_{c_{in}}(o_f)\ \wedge\ c_t = Ref_{c_{in}}(o_t)\ \ \Rightarrow$$
$$Ref_c(p_f) \precsim c_f\ \wedge\ Ref_c(p_t) \precsim c_t\ \wedge\ \mathcal{TP}_c = \mathcal{INTP}_c = \mathcal{INTP}_c^1.\ \diamond$$

This axiom is justified and exemplified in subsection 6.5.2.

Axiom 7.7 Interaction between generalization and contextualization.

Let c be a context and p_f, p_t be reference paths in c. If there is an ISA link in c from p_1 to p_2 then the reference of p_1 in c refines the reference of p_2 in c.

$$\forall o \in \mathcal{O},\ c, c_1, c_2 \in \mathcal{CXT},\ p_1, p_2 \in \mathcal{RP}:$$
$$Isa_c(o, p_1, p_2)\ \wedge\ c_1 = Ref_c(p_1)\ \wedge\ c_2 = Ref_c(p_2)\ \ \Rightarrow\ \ c_1 \precsim c_2.\ \diamond$$

This axiom is justified and exemplified in subsection 6.5.3.

Axiom 7.8 Interaction between classification, generalization and contextualization.

Let c be a context and p, p', p'' be reference paths in c. If o_1 is an instance-of link in c from p to p', o is an ISA link in c from p' to p'', and o_2 is an instance-of link in c from p to p'', then the reference of o_1 in c refines the reference of o_2 in c.

$$\forall o, o_1, o_2 \in \mathcal{O},\ c, c_1, c_2 \in \mathcal{CXT},\ p, p', p'' \in \mathcal{RP}:$$
$$In_c(o_1, p, p')\ \wedge\ Isa_c(o, p', p'')\ \wedge\ In_c(o_2, p, p'')\ \wedge\ Ref_c(o_1) = c_1\ \wedge\ Ref_c(o_2) = c_2\ \ \Rightarrow$$
$$c_1 \precsim c_2.\ \diamond$$

This axiom is justified and exemplified in subsection 6.5.4.

Axiom 7.9 Inheritance of built-in information.

Every context refines the empty context.

$\forall c \in \mathcal{CXT}: \quad c \precsim C_{empty}. \diamond$

This axiom expresses that the contents of the empty context should be present in every context.

7.7 The equivalence and refinement relations

In this section, we give the formal definitions of the equivalence and refinement relations, described informally in subsection 6.5.3, and we prove some of their properties.

In order to define formally the equivalence relation we need to show how a context structure is represented as a labeled directed graph. Since contexts can be seen as labeled directed graphs and graph theory is a well-studied field, we can use graph theoretical problems to classify the problems concerning context relations (e.g. context equivalence).

7.7.1 Context structure as labeled directed graphs

In this subsection, we define the notion of 'labeled directed graph' as appears in graph theory and how two such graphs are isomorphic. The notion of isomorphic graphs is used in the following subsection to define context equivalence. We also show how a context structure is represented as a labeled directed graph.

Definition 7.8 Labeled directed graph.

A labeled directed graph is a triplet $G = \langle V, E, L \rangle$ where V is a finite set of vertices, $E \subseteq V \times V$ is the set of edges (which connect certain pair of vertices) and L is a function which assigns each vertex and each edge a label.

Definition 7.9 Labeled directed graph isomorphism.

Two labeled directed graphs, $G_1 = \langle V_1, E_1, L_1 \rangle$ and $G_2 = \langle V_2, E_2, L_2 \rangle$ are isomorphic iff there is a bijective function (isomorphism) $\pi : V_1 \to V_2$ such that

1. $\langle v_1, v_2 \rangle \in E_1$ iff $\langle \pi(v_1), \pi(v_2) \rangle \in E_2$,
2. $L_1(v) = L_2(\pi(v))$ for all $v \in V_1$, and
3. $L_1(\langle v_1, v_2 \rangle) = L_2(\langle \pi(v_1), \pi(v_2) \rangle)$ for all $\langle v_1, v_2 \rangle \in E_1$.

Let $Iso(G_1, G_2)$ denote the set of all isomorphisms between G_1 and G_2. $\diamond$

Lemma 7.1 Graph isomorphism transitivity.

If graphs G_1, G_2 are isomorphic and graphs G_2, G_3 are also isomorphic, then graphs G_1, G_3 are isomorphic too. $\diamond$

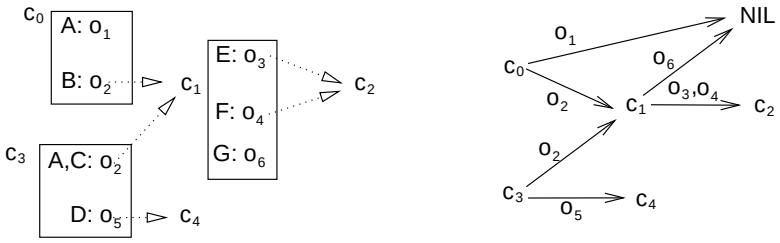
Proof: If $\pi \in Iso(G_1, G_2)$ and $\phi \in Iso(G_2, G_3)$, it is easy to prove that $\pi \circ \phi$ is an isomorphism between graphs G_1 and G_3. $\square$

By context structure we mean a structure of contexts and their nested subcontexts. In the structure we don't take into account links predicates (instance-of, ISA and attribute predicates) as well as the names of the objects, but we do take into account the objects of the contexts,

which are represented as labels of the graph eadges. Thus, a context structure can be seen as a labeled directed graph as follows:

- the contexts are represented as vertices of the graph,
- the subcontext c' of a context c is represented by the edge $\langle c, c' \rangle$,
- the navigation from a context c to its subcontext c' through the reference of an object o in c, i.e. $ref(o, c) = c'$, is represented by $o \in L(\langle c, c' \rangle)$, i.e. the label of edge $\langle c, c' \rangle$ is a set containing the object o. If there is another object o' such that $ref(o', c) = c'$ then $o' \in L(\langle c, c' \rangle)$ as well. In other words, the label of an edge $\langle c, c' \rangle$ is the set of all objects of c with reference c',
- $L(v) = \{\}$, for all $v \in V$, as contexts are assigned no name.

Figure 7.1 shows how a context structure is represented as a labeled directed graph.



(a) (b)

Figure 7.1: (a) A contexts structure, (b) its representation as a labeled directed graph.

Note that since each object within a context has either a *NIL* reference or a reference to another context and the objects of a context are represented as labels of edges originating from that context, it is easy to see that: if c is a context then

$$objs(c) = \bigcup_{\langle c,c' \rangle \in E} L\langle c, c' \rangle)$$

Each context is connected with a context structure which contains the context itself and its nested subcontexts. The labeled directed graph induced by the context structure of a context is denoted by $GR(c)$ and is defined formally as follows:

Definition 7.10 Labeled directed graph induced by a context c, $GR(c)$.

The labeled directed graph induced by the context c is defined as follows:

$$\forall c \in \mathcal{CXT} : \ GR(c) = \langle V, E, L \rangle$$

where

$$V = \{c\} \cup \{c' \in \mathcal{CXT} \mid c' = Ref_c(p), \ p \in \mathcal{RP}_c\}$$

$$E = \{\langle c_1, c_2 \rangle \mid c_1, c_2 \in V \ \wedge \ \exists o \in objs(c_1) : \ ref(o, c_1) = c_2\}$$

$$\forall \langle c_1, c_2 \rangle \in E : \ L(\langle c_1, c_2 \rangle) = \bigcup_{ref(o,c_1)=c_2} \{o\}$$

Now, we are ready to define the equivalence relation between contexts.

7.7.2 Context equivalence

Definition 7.11 Context equivalence ($\sim$).

Let c, c' be two contexts and $GR(c) = \langle V, E, L\rangle$, $GR(c')$ the graphs induced by c and c', respectively. *Context equivalence* is a relation between c and c', denoted by $c \sim c'$, is defined as follows:

$$
\begin{aligned}
c \sim c' \quad \Leftrightarrow \quad & (c = c' = \mathit{NIL}) \\
& \vee \\
& (\ \exists \pi \in Iso(GR(c), GR(c')) :\ \ \pi(c) = c' \wedge \\
& \qquad \forall v \in V : \\
& \qquad\quad attr(v) \ = \ attr(\pi(v)) \quad \wedge \\
& \qquad\quad in(v) \ = \ in(\pi(v)) \quad \wedge \\
& \qquad\quad isa(v) \ = \ isa(\pi(v)) \quad \wedge \\
& \qquad (\forall o \in objs(v) :\ \ names(o, v) = names(o, \pi(v)))) \diamond
\end{aligned}
$$

Two contexts are equivalent ($c \sim c'$) if c and c' are identical (possibly NIL) or they have an isomorphic context structure and the contents (except references) of each context coincide. Intuitively, this means that two contexts are equivalent if they are identical or their contents (except references) coincide, and the references of their common objects are equivalent.

Graph Isomorphism Problem: The complexity of an algorithm which decides whether two given contexts are equivalent is the same to the complexity of an algorithm which decides whether the context graphs of the given contexts are isomorphic. This is the well-known *graph isomorphism problem* [61, 78, 35, 60]. It is easy to see, based on result by Miller [78], that the graph isomorhism problems for (non-labeled, undirected) graphs and labeled directed graphs are many-one equivalent and therefore we do not distinguish between them. The graph isomorhism problem is clearly in the class of NP, and it is not known whether it is in P. It is also unknown whether the problem is NP-complete, but this seems to be unlikely since it would imply that the polynomial hierarchy collapses to its second level [12]. In fact, evidence that graph isomorphism is not NP-complete was given already by Mathon [70] who showed that the decision problem for graph isomorphism and its counting version (i.e., the problem to compute the number of isomorphisms of two given graphs) are polynomial-time Turing equivalent. This is a remarkable situation since for the known NP-complete problems the corresponding counting version seems to be much harder than the decision version.

An important property of the relation $\sim$ is that it is equivalence. Thus, in case context identifiers are not important, equivalent contexts can be used interchangeably for modeling information.

Proposition 7.1 Relation $\sim$ is an equivalence relation.

The relation $\sim$ on the set of contexts $\mathcal{CXT}$ is an equivalence relation, i.e., reflexive, transitive, and symmetric. $\diamond$

Proof: From the Definition 7.11 it follows immediately that $\sim$ is reflexive and symmetric. We will now prove that $\sim$ is also transitive, that is:

$$\forall c_1, c_2, c_3 \in \mathcal{CXT} :\ \ (c_1 \sim c_2) \ \wedge \ (c_2 \sim c_3) \Rightarrow c_1 \sim c_3$$

We distinguish the following cases:

1. $c_1 = c_2 = c_3 = \mathit{NIL}$.

Then, from the Definition 7.11, it easily follows that $c_1 \sim c_3$.

2. $c_1 \neq NIL$ and $c_2 \neq NIL$ and $c_3 \neq NIL$.

   (a) $c_1 = c_2 = c_3$ or $c_1 = c_2 \neq c_3$ or $c_1 \neq c_2 = c_3$ or $c_1 = c_3 \neq c_2$.
   Then, from the Definition 7.11, it easily follows that $c_1 \sim c_3$.

   (b) $c_1 \neq c_2$, $c_2 \neq c_3$, and $c_1 \neq c_3$.
   From the Definition 7.11 we have

$$c_1 \sim c_2 \Rightarrow \left\{ \begin{array}{rcll} \exists \pi \in Iso(GR(c_1), GR(c_2)) & : & \pi(c_1) = c_2 \wedge \\ \forall v_1 \in V_1 : & & \\ attr(v_1) & = & attr(\pi(v_1)) \wedge \\ in(v_1) & = & in(\pi(v_1)) \wedge \\ isa(v_1) & = & isa(\pi(v_1)) \wedge \\ \forall o \in objs(v_1) : \ names(o, v_1) & = & names(o, \pi(v_1)) \end{array} \right. \tag{7.1}$$

$$c_2 \sim c_3 \Rightarrow \left\{ \begin{array}{rcll} \exists \phi \in Iso(GR(c_2), GR(c_3)) & : & \phi(c_2) = c_3 \wedge \\ \forall v_2 \in V_2 : & & \\ attr(v_2) & = & attr(\phi(v_2)) \wedge \\ in(v_2) & = & in(\phi(v_2)) \wedge \\ isa(v_2) & = & isa(\phi(v_2)) \wedge \\ \forall o \in objs(v_2) : \ names(o, v_2) & = & names(o, \phi(v_2)) \end{array} \right. \tag{7.2}$$

$$\begin{array}{r} \text{Eq. (7.1)} \\ \text{Eq. (7.2)} \\ \text{Lemma 7.1} \end{array} \Rightarrow \left\{ \begin{array}{rcll} \exists \rho = \pi \circ \phi \in Iso(GR(c_1), GR(c_3)) & : & \rho(c_1) = c_3 \wedge \\ \forall v_1 \in V_1 : & & \\ attr(v_1) & = & attr(\rho(v_1)) \wedge \\ in(v_1) & = & in(\rho(v_1)) \wedge \\ isa(v_1) & = & isa(\rho(v_1)) \wedge \\ \forall o \in objs(v_1) : \ names(o, v_1) & = & names(o, \rho(v_1)) \end{array} \right. \tag{7.3}$$

   From the Definition 7.11 and Equation (7.3) we have: $c_1 \sim c_3$. □

7.7.3 The refinement relation

We now define the refinement relation between two contexts.

Definition 7.12 Refinement.
The *refinement* relation between two contexts, denoted by $\precsim$, is recursively defined as follows:

$$\begin{array}{rcll} c \precsim c' & \Leftrightarrow & (c' = NIL) & \\ & & \vee & \\ & & (objs(c') & \subseteq \ objs(c) \ \wedge \\ & & attr(c') & \subseteq \ attr(c) \ \wedge \\ & & in(c') & \subseteq \ in(c) \ \wedge \\ & & isa(c') & \subseteq \ isa(c) \ \wedge \\ & & (\forall o \in objs(c') : & names(o, c') \subseteq names(o, c) \wedge \\ & & & ref(o, c) \precsim ref(o, c'))). \diamond \end{array}$$

The recursion in the definition of $c \precsim c'$ stops when c' is *NIL*, or $c \precsim c'$ is called again. Intuitively, a context c refines a context c' ($c \precsim c'$), if the contents of c' except references are subset of the contents of c, and the reference of an object o in c' is either undefined or refined by the reference of o in c. For example, in Figure 7.2 contexts c_9 and c_{10} refine context c_8, and contexts c_4, c_5 refine context c_3.

If both refinements $c \precsim c'$ and $c' \precsim c$ hold, then the contents of c and c' are intuitively the same. Indeed the following theorem shows that the refinement relation is partial order up to the equivalence relation.

Theorem 7.1 .

The refinement relation on the set of contexts $\mathcal{CXT}$ is a partial order up to the equivalence relation $\sim$, i.e., it is reflexive, transitive, and antisymmetric. $\diamond$

Proof: From the Definition 7.12 it follows immediately that $\precsim$ is reflexive and transitive.

We will prove that $\precsim$ is antisymmetric, that is:

$$\forall c, c' \in \mathcal{CXT}: \ (c \precsim c') \ \wedge \ (c' \precsim c) \Rightarrow c \sim c')$$

Assume, on the contrary, that: $c \not\sim c'$.

According to the Definition 7.11 of relation $\sim$ we have:

$$\begin{array}{lll} c \not\sim c' & \Leftrightarrow & (c \neq NIL \ \vee \ c' \neq NIL) \\ & & \wedge \\ & & (\forall \pi \in Iso(GR(c), GR(c')): \ \pi(c) \neq c' \ \vee \\ & & \quad \exists v \in V: \\ & & \quad attr(v) \ \neq \ attr(\pi(v)) \ \vee \\ & & \quad in(v) \ \neq \ in(\pi(v)) \ \vee \\ & & \quad isa(v) \ \neq \ isa(\pi(v)) \ \vee \\ & & \quad (\exists o \in objs(v): \ names(o, v) \neq names(o, \pi(v)))) \diamond \end{array}$$

We distinguishe the following cases:

1. $(c' \neq NIL)$ and $(c = NIL)$
   Then, from our assumptions, we have:
   $NIL \neq c \precsim c' = NIL$, which is true according to the Definition 7.12, and
   $NIL = c' \precsim c \neq NIL$, which is false according to the Definition 7.12.

2. $(c \neq NIL)$ and $(c' = NIL)$
   It is proved similar to the previous case.

3. $$\begin{array}{l} (c \neq NIL \ \wedge \ c' \neq NIL) \\ \wedge \\ (\ \nexists \pi \in Iso(GR(c), GR(c')): \ \pi(c) = c' \wedge \\ \quad \forall v \in V: \\ \quad attr(v) \ = \ attr(\pi(v)) \ \wedge \\ \quad in(v) \ = \ in(\pi(v)) \ \wedge \\ \quad isa(v) \ = \ isa(\pi(v)) \ \wedge \\ \quad (\forall o \in objs(v): \ names(o, v) = names(o, \pi(v)))) \end{array}$$

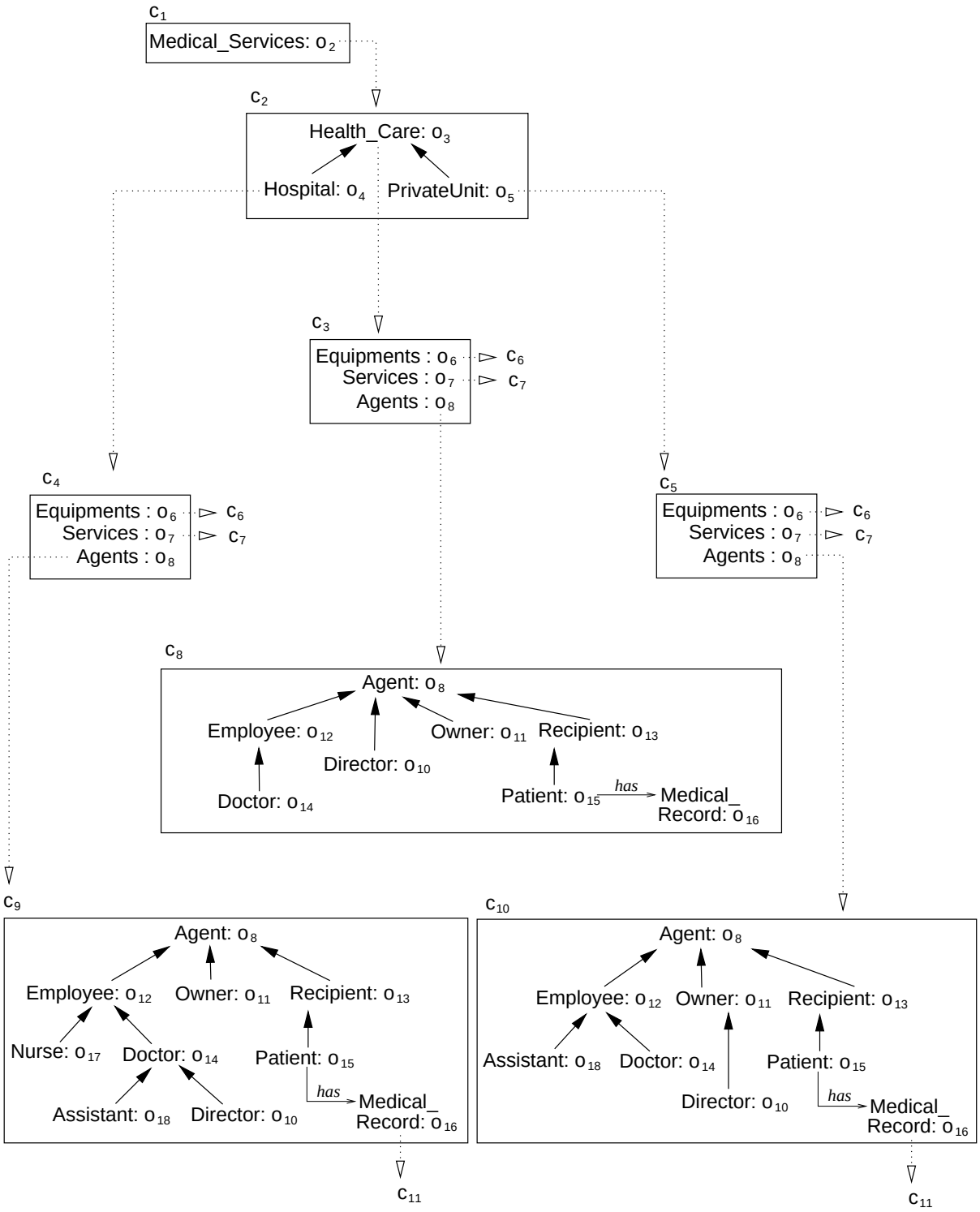


Figure 7.2: An example of refinement relation: contexts c_9 and c_{10} refine context c_8, and contexts c_4, c_5 refine context c_3.

We will show that there exists an isomorphism like this. From our assumption we have:

$$c \precsim c' \Rightarrow \begin{cases} objs(c') & \subseteq & objs(c) \wedge \\ attr(c') & \subseteq & attr(c) \wedge \\ in(c') & \subseteq & in(c) \wedge \\ isa(c') & \subseteq & isa(c) \wedge \\ (\forall o \in objs(c') & : & \\ names(o,c') & \subseteq & names(o,c) \wedge \\ ref(o,c) & \precsim & ref(o,c')) \end{cases} \tag{7.4}$$

$$c' \precsim c \Rightarrow \begin{cases} objs(c) & \subseteq & objs(c') \wedge \\ attr(c) & \subseteq & attr(c') \wedge \\ in(c) & \subseteq & in(c') \wedge \\ isa(c) & \subseteq & isa(c') \wedge \\ (\forall o \in objs(c) & : & \\ names(o,c) & \subseteq & names(o,c') \wedge \\ ref(o,c') & \precsim & ref(o,c)) \end{cases} \tag{7.5}$$

$$\begin{matrix} \text{Eq. } (7.4) \\ \text{Eq. } (7.5) \end{matrix} \Rightarrow \begin{cases} objs(c) & = & objs(c') \wedge \\ attr(c) & = & attr(c') \wedge \\ in(c) & = & in(c') \wedge \\ isa(c) & = & isa(c') \wedge \\ (\forall o \in objs(c) & : & \\ names(o,c) & = & names(o,c') \wedge \\ ref(o,c) & \precsim & ref(o,c') \wedge \\ ref(o,c') & \precsim & ref(o,c)) \end{cases} \tag{7.6}$$

This applied recursively to all subcontexts of c and c'. Let $GR(c) = \langle V, E, L \rangle$ and $GR(c') = \langle V, E, L \rangle$. The function $\phi : V \to V'$ which is defined as follows:

$$\begin{aligned} \phi(c) &= c' \\ \phi(ref(o,v)) &= ref(o,v'), \quad \forall v \in V, v' \in V' : \ \phi(v) = v' \ \wedge \ o \in objs(v) \end{aligned}$$

is a bijection, and from Equation 7.6 it is easy to see that:

$$\begin{aligned} &\forall v \in V : \\ &\quad objs(v) = objs(\phi(v)) \wedge \\ &\quad attr(v) = attr(\phi(v)) \wedge \\ &\quad in(v) = in(\phi(v)) \wedge \\ &\quad isa(v) = isa(\phi(v)) \wedge \\ &\quad (\forall o \in objs(v) : \\ &\quad\quad names(o,v) = names(o,\phi(v))) \end{aligned}$$

Thus, our original assumption is false. □

A contextualized information base is an incomplete model of the real world. This implies that the conditions of Definition 7.12 may hold in the information base but not in the real world, and conversely, they may hold in the real world but not in the information base. Thus, we cannot use Definition 7.12 directly to derive that $c \precsim c'$, for contexts c and c'. Indeed, the refinement relation is derived through the Core Axioms. The following theorem shows that the Core Axioms provide a sound and complete set of inference rules for the determination of all the refinement relations that are valid according to Definition 7.12.

Theorem 7.2 Soundness and completeness of refinement.
Core Axioms are sound and complete for the derivation of the refinement relation $\precsim$ as defined in Definition 7.12. $\diamond$

Proof: The soundness of the rules can be easily verified. With regard to the completeness, we show that every refinement relationship that is not derivable from the axioms of a contextualized information base by means of the above rules is falsified by some model of the rules.
Let T be a contextualized information base and $c \leq c'$ be a refinement relationship of T that is not derivable.
We now show a model I of T that falsifies the above relationship.
I contains a single object o, and for each derivable formula of the form $c \leq d$ such that $d \leq c'$ is not derivable, assigns o to d.
Clearly, o belongs to c and o does not belong to c'.
In order to verify that all the axioms of T are satisfied by I, consider the generic axiom $w \leq w'$.
If o does not belong to w then the axiom is clearly satisfied.
If o belongs to w, then there is a refinement $c \leq w$ that is derivable such that $w \leq c'$ is not derivable.
Therefore, by the Contextualized ISA Transitivity Axiom (Axiom 7.1), it holds that $c \leq w'$.
The refinement $w' \leq c'$ cannot be derived, as otherwise by the Contextualized ISA Transitivity Axiom $c \leq c'$ is derived.
Therefore o belongs also to w'.
It now follows from the Definition of refinement (Definition 7.12) that $w \leq w'$ holds. $\square$

7.8 Contextualization in Telos

As we mentioned in Section 6.7, our contextualization mechanism is generic and can be applied to any semantic data model that supports the traditional abstraction mechanisms of classification, generalization, and attribution. In this section, we show a specific data model can accommodate our contextualization mechanism. In particular, we choose for this purpose the Telos data model.

Telos [83, 64] is a knowledge representation language originally designed for information system development applications. Its data model is an object-oriented semantic network, where nodes and links are uniformly treated, and multiple instantiation, multiple specialization, and virtually unlimited instantiation levels are supported. Its assertional component includes facilities for declaring constraints and deductive rules (we do not deal with this component here).

In original Telos, an object is an individual or an attribute that belongs to a unique instantiation level. Though, in our theory, these characterizations of an object do not hold globally,

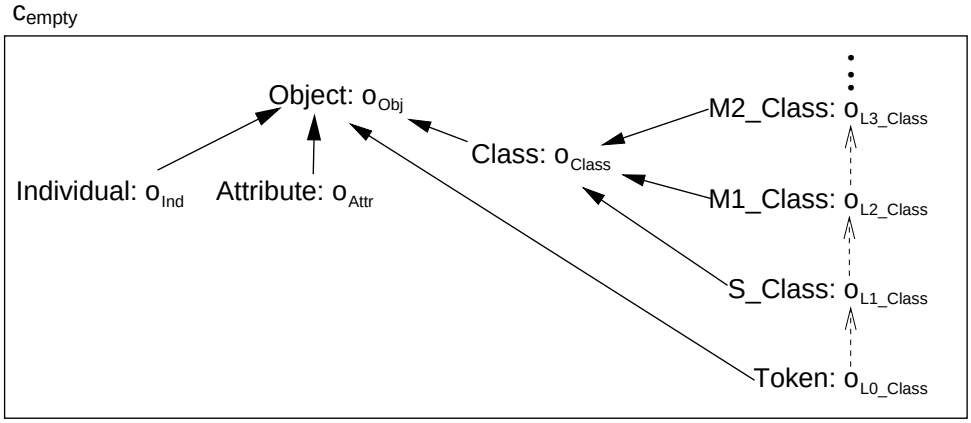


Figure 7.3: Context C_{empty}.

they do hold within a context. In other words, these characterizations are context-dependent. We will see below how the Telos features are incorporated into our contextualized framework. For our discussion, we consider a variant of Telos, where instance-of and ISA links are also objects.[6]

To support Telos, the special context C_{empty} contains all built-in objects of Telos and their relationships. Specifically, the contents of C_{empty} include[7] (see Figure 7.3)

- Objects o_{Ind}, o_{Attr}, named `Individual` and `Attribute`, respectively. These objects represent the classes of individuals and attribute objects, respectively.

- Objects o_{L_0}, o_{L_1}, o_{L_2}, and so on, named `Token`, `S_Class`, `M1_Class`, and so on, respectively. These objects represent the instantiation levels of *Tokens*, *Simple Classes*, *Metaclasses*, and so on, respectively.

- Objects o_{Obj} and o_{Class}, named `Object` and `Class`, respectively. These objects represent the class of objects, and class of classes, respectively.

- Instance-of links and ISA links that represent relationships between the objects of C_{empty}.

By definition the contents of the empty context are shared by any context. Therefore, the objects of C_{empty} are also objects of any context c. We say that an object o is *individual in context* c, iff there is an instance-of link in c from o to o_{Ind}. Similarly, an object o is *attribute in context* c, iff there is an instance-of link in c from o to o_{Attr}. A reference path p may be associated with a *instantiation level* i *in a context* c, iff there is an instance-of link in c from p to object o_{L_i}. The function $Level_c(p)$ returns the instantiation level of a reference path p in context c.

$$\forall c \in \mathcal{CXT}, p \in \mathcal{RP}_c : \quad Level_c(p) = i \quad \Leftrightarrow \quad \exists o \in \mathcal{O}\ In_c(o, p, o_{L_i})$$

[6] O-Telos is such a variant of Telos, used in the deductive object base ConceptBase [54].

[7] In the figure, it is not shown that objects o_{Ind}, o_{Attr}, o_{Obj}, and o_{Class} are instances of the object o_{Class}.

7.9 Telos-dependent axioms

We now give a number of axioms, called Telos-dependent axioms, which together with the core axioms should be satisfied by any Telos contextualized information base. Indeed the following axioms support embedding of Telos into our contextualized framework, and are contextualized versions of basic axioms of Telos [54].

Axiom 7.10 Membership to the Attribute built-in object.

If a is an attribute link in c from a reference path p_f to a reference path p_t, then there is an instance-of link in c from a to o_{Attr}.

$\forall c \in \mathcal{CXT},\ a \in \mathcal{O},\ p_f, p_t \in \mathcal{RP}$:

$Attr_c(a, p_f, p_t) \ \Rightarrow \ \exists o \in \mathcal{O}\ In_c(o, a, o_{Attr}).\ \diamond$

Axiom 7.11 Object kind uniqueness constraint.

An object cannot be both individual and attribute within a context.

$\forall c \in \mathcal{CXT}\ o_1, o_2, o_1', o_2' \in \mathcal{O}$:

$In_c(o_1', o_1, o_{Ind}) \ \wedge \ In_c(o_2', o_2, o_{Attr}) \ \Rightarrow \ o_1 \neq o_2.\ \diamond$

Note that, it is possible for an object to be individual in one context and attribute in another.

Axiom 7.12 Level uniqueness constraint.

A reference path cannot have more than one instantiation levels within a context, that is, it can be classified within a context under only one of the classes $\mathtt{L_i}$, $i \geq 0$.

$\forall c \in \mathcal{CXT},\ p \in \mathcal{RP}\ i_1, i_2 \in I\!N$:

$Level_c(p) = i_1 \ \wedge \ Level_c(p) = i_2 \ \Rightarrow \ i_1 = i_2.\ \diamond$

Axiom 7.13 Source and destination uniqueness constraints.

1.The source of a link in a context c is unique.

$\forall c \in \mathcal{CXT}, o \in \mathcal{O}, p_1, p_2 \in \mathcal{RP}$:

$From_c(o) = p_1 \ \wedge \ From_c(o) = p_2 \ \Rightarrow \ p_1 = p_2.$

2.The destination of a link in a context c is unique.

$\forall c \in \mathcal{CXT}, o \in \mathcal{O}, p_1, p_2 \in \mathcal{RP}$:

$To_c(o) = p_1 \ \wedge \ To_c(o) = p_2 \ \Rightarrow \ p_1 = p_2.\ \diamond$

Axiom 7.14 Instance-of constraints.

1.Attribute instantiation constraint.

Let a and a' be two attribute links in c from p_f to p_t, and p_f' to p_t', respectively. If there is an instance-of link in c from a to a', then there is an instance-of link in c from p_f to p_t, and an instance-of link in c from p_f' to p_t'.

$\forall c \in \mathcal{CXT},\ o, a, a' \in \mathcal{O},\ p_f, p_t, p_f', p_t' \in \mathcal{RP},\ \ \exists o', o'' \in \mathcal{O}$:

$In_c(o, a, a') \ \wedge \ Attr_c(a, p_f, p_t) \ \wedge \ Attr_c(a', p_f', p_t') \ \Rightarrow$

$In_c(o', p_f, p_f') \ \wedge \ In_c(o'', p_t, p_t')$

2.Instance-of level constraint.

If there is an instance-of link in c from p_f to p_t, then the instantiation level of p_t in c equals the instantiation level of p_f in c plus one.

$$\forall c \in \mathcal{CXT},\ o, \in \mathcal{O},\ p_f, p_t \in \mathcal{RP}:$$

$$In_c(o, p_f, p_t) \quad \Rightarrow \quad Level_c(p_t) = Level_c(p_f) + 1. \diamond$$

Axiom 7.15 ISA constraints.

1.Attribute generalization constraint.

Let a and a' be two attribute links in c from p_f to p_t, and p'_f to p'_t, respectively. If there is an ISA link in c from a to a', then there is an ISA link in c from p_f to p_t, and an ISA link in c from p'_f to p'_t.

$$\forall c \in \mathcal{CXT},\ o, a, a' \in \mathcal{O},\ p_f, p_t, p'_f, p'_t \in \mathcal{RP},\quad \exists o', o'' \in \mathcal{O}:$$

$$Isa_c(o, a, a')\ \wedge\ Attr_c(a, p_f, p_t)\ \wedge\ Attr_c(a', p'_f, p'_t)\ \Rightarrow$$

$$Isa_c(o', p_f, p'_f)\ \wedge\ Isa_c(o'', p_t, p'_t)$$

2.ISA level constraint.

If there is an ISA link in c from p_f to p_t, then the level of p_t in c equals the level of p_f in c.

$$\forall c \in \mathcal{CXT},\ o, \in \mathcal{O},\ p_f, p_t \in \mathcal{RP}:$$

$$Isa_c(o, p_f, p_t) \quad \Rightarrow \quad Level_c(p_t) = Level_c(p_f). \diamond$$

Chapter 8

Querying Contextualized Information Bases

A query system has to deal with two tasks, the maintenance of an information base and the answering of queries. In this chapter, we focus on the second task, and we propose a general framework for querying information bases which supports contextualization. There is a number of desiderata for querying information bases. A list of them (not complete) follows:

- **Expressive power**

  One can write down an informal list of the kind of operations that a query language should express.

- **Semantics**:

  Without this one cannot start to discuss query transformation or optimization. An interesting issue is what, if anything, the semantics adds to (or loses from) the syntax in which data is expressed.

- **Compositionality**:

  This requirement states that the output of a query can be used as the input of another query and is essential, among others, for constructing views. The requirement has both semantic and syntactic consequences. Semantically, if states that our queries must stay within the same data model. Syntactically, it requires that our language is referentially transparent.

A prominent feature of our contextualization mechanism is that it allows users to focus on a specific context at a time (call it *current context*), thus delimiting a portion of interest in the information base. As a consequence, the scope of user queries is localized to that portion, i.e., to the set of objects and contexts that are accessible from the current context. In turn, query evaluation is performed with respect to that portion of interest — and *not* with respect to the whole information base. As a result, users can find speedily the needed information.

In this chapter, we address issues of (i) accessing information in the context structure using paths of names, or paths of references, and (ii) retrieving contextualized information. To this end we define basic query operations on contexts such as select, project, path select, and construct. We illustrate the usefulness of our contextualization mechanism by presenting

higher level query operations, including the use of wild cards, that enable users to explore a contextualized information base. These higher level operations include focusing on a context of interest, searching the context structure for specific information, and making cross references of an object from one context to another in order to obtain alternative representations of that object. Work of this chapter published in [124].

8.1 Accessing information through paths

Accessing information in an information base often involves *navigating* from one object to another. Navigation is based on the notion of *path*. We distinguish two kinds of paths:

1. *object path*, which is a sequence of dot-separated objects,
2. *name path*, which is a sequence of dot-separated names.

In a contextualized information base, navigation is based on the notion of *reference path*. Indeed, as the reference of an object within a context is also a context, references provide a means to traverse from an object o of a context c to the objects of another context via the reference of o in c.

Definition 8.1 Reference path.

An object path $o_1.o_2.\ldots.o_k$ is a *reference path* in a context c_0 if the following conditions hold:

1. $o_1, \ldots, o_k$ are objects;
2. Object o_1 is contained in context c_0;
3. Let c_i be the reference of object o_i w.r.t. c_{i-1}, for $i = 1, \ldots, k-1$. Then, object o_{i+1} is contained in context c_i. $\diamond$

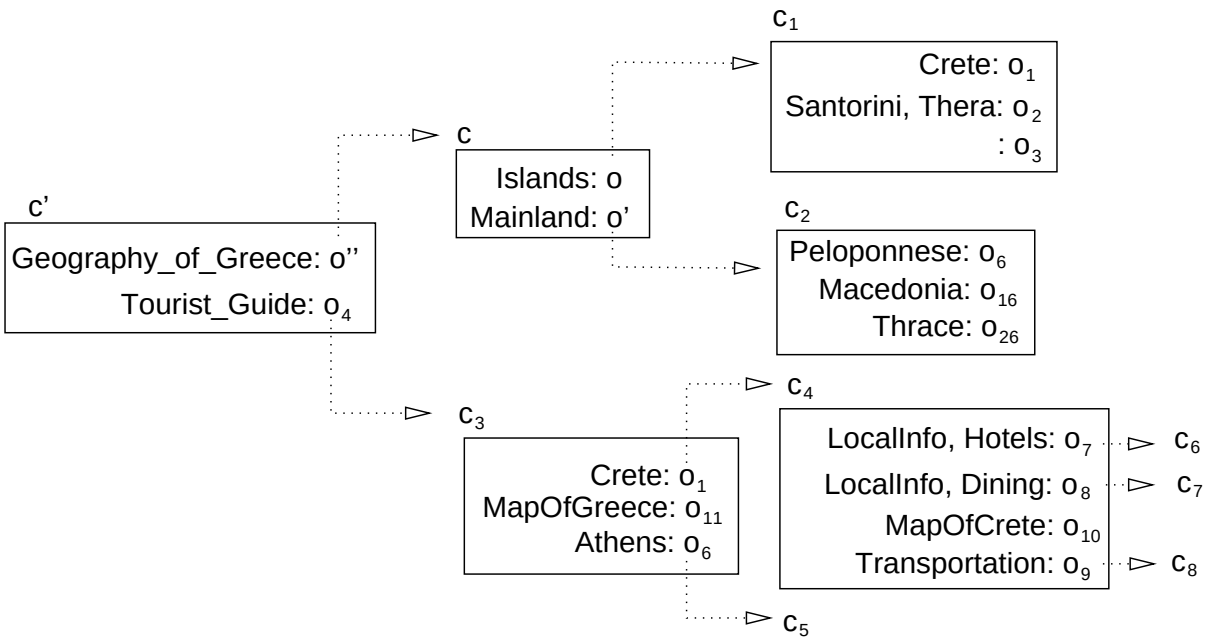


Figure 8.1: Accessing information through paths.

Reference paths form the basis for reaching objects in a context navigating through the references of objects. For example, in Figure 8.1, $o.o_1$ is a reference path that provides one way

to reach object o_1 starting from context c, namely, through the context c_1. Moreover, $o''.o.o_1$ and $o_4.o_1$ are reference paths leading to object o_1, through different contexts.

To make the specification of a reference path user-friendly, we can use a *name path*, i.e., a sequence of object names. We denote such a sequence using dots to separate names, e.g. `Geography_of_Greece.Islands.Crete`. Clearly, not every name path necessarily *matches* a reference path, e.g. `Geography_of_Greece.Crete` does not match any reference path (as the object designated by `Geography_of_Greece` in the context c' does not have as reference the context to which belongs an object designated by `Crete`). Hence, the following definition:

Definition 8.2 Matching.

Let $N = n_1.n_2.\ldots.n_k$ be a name path, and $P = o_1.o_2.\ldots.o_k$ be a reference path, in a context c_0. We say that N *matches* P(or that P *matches* N) in context c_0 if the following conditions hold:

1. n_1 is a name for o_1 in c_0, and
2. If c_i is the reference of o_i w.r.t. context c_{i-1}, then n_{i+1} is a name for o_{i+1} in c_i, for $i = 1, \ldots, k$.

The boolean operation $match(c, P, N)$ returns $true$ is P matches N in context c, and $false$ otherwise. $\diamond$

For example, in Figure 8.1, the name path `Islands.Crete` matches the reference path $o.o_1$, and vice versa, in context c.

Clearly, in general, a name path may match several reference paths, and conversely. For example, in Figure 8.1, the name path `Crete.LocalInfo` matches the reference paths $o_1.o_7$ and $o_1.o_8$, while the reference path $o_1.o_7$ matches the name paths `Crete.LocalInfo` and `Crete.Hotels`, in context c_3.

We use two primitive operations on paths, $first(P)$ and $rest(P)$, where P is a path of any kind (i.e., reference path or name path). The operation *first* returns the first element of P and the operation $rest$ returns the path resulting from P after removing its first element[1].

Based on these primitive operations, we can define other useful operations. In this paper we shall make use of the operations $length(P)$ and $elem(i, P)$, that return the length of P and the ith element of P, respectively (assuming $1 \leq i \leq length(P)$). We shall also use the operation $last(P)$ that returns the last element of P, assuming that P is not empty (note that $last(P) = elem(length(P), P)$).

In the rest of the chapter, whenever we refer to a name path (starting in a context c), we will mean a name path that matches a reference path (starting in c). Reference paths and name paths form the basis for our query language. A more powerful form of name paths based on wildcards and regular expressions is described in Section 8.2.

8.2 Generalized path expressions

In this section, we extend the notion of name path to a more powerful syntax for name paths, called *generalized name paths*. Generalized name paths allow both regular expressions and wildcards to be used in name paths, which provide a powerful mechanism for finding objects in the information base.

[1]These are the well-known operations car and cdr used in LISP.

8.2.1 Path composition

Let $p_\emptyset$ be the symbol for the *empty path* (i.e., objects path or name path). Let p_1, p_2 be two paths of the same kind, and S_1, S_2 be two sets of paths of the same kind. The *composition* of paths, denoted by the symbol $\oplus$, is defined as follows:

$$p_1 \oplus p_2 = \begin{cases} p_1, & \text{if } p_2 = p_\emptyset, \\ p_2, & \text{if } p_1 = p_\emptyset, \\ p_1.p_2, & \text{otherwise.} \end{cases} \tag{8.1}$$

$$p_1 \oplus S_1 = \begin{cases} \emptyset, & \text{if } S_1 = \emptyset, \\ \{p_1.p \mid \forall p \in S_1\}, & \text{otherwise.} \end{cases} \tag{8.2}$$

$$S_1 \oplus S_2 = \begin{cases} \emptyset, & \text{if } S_1 = \emptyset \text{ or } S_2 = \emptyset, \\ \{p_1.p_2 \mid \forall p_1 \in S_1,\ p_2 \in S_2\}, & \text{otherwise.} \end{cases} \tag{8.3}$$

Note that if $S = \{p_\emptyset\}$ then $S \neq \emptyset$.

8.2.2 Generalized name paths

A user may not be aware of the structure of the information base but may simply know a few keywords concerning the objects of interest. Therefore, a user may not be expected to give a complete name path to those objects, but only an incomplete path made up of the keywords he knows plus special symbols indicating how the path may be completed. Moreover, one often does not know the whole name of an object precisely, but only knows a part of it. It is therefore useful to have a concept of "wildcards". The first wild card is "%", which matches zero or more characters in a name. Hence, we introduce the concept of *generalized name path* (g.n.p. for short) which is similar to the generalized path expressions of [3, 1].

Definition 8.3 Generalized name paths.

1. If α is a name, then $.\alpha$ is a generalized name path.
2. If β and γ are strings (possibly empty), then $.\beta\%\gamma$ is a generalized name path.
3. If g and g' are generalized name paths, then the following are also generalized name paths:
   gg', $g|g'$, $g\&g'$, (g), $(g)?$, $(g)+$, $(g)*$, $\neg g$

The symbol $\%$ means any arbitrary number of letters, $|$ is used for disjunction, $\&$ is used for conjunction, $\neg$ is used for negation, $?$ means zero or one occurrences, $+$ means one or more, and $*$ means zero or more. $\diamond$

Examples of generalized name paths are:

`.Employees.M`$\%$

`(.Employees|.Students).address(.zipcode)`$?$

`.Employees(.nearby.address)`$*$`.cityname`

The first expression specifies the name paths starting from `Employees` following a name beginning with letter "M". The second expression specifies the name paths starting either from `Employees` or from `Students`, following the name `address` with an optional `zipcode` at the

end. The last expression specifies all name paths starting from `Employees`, following by and arbitrary number of `nearby.address` and ending with name `cityname`.

The only difficulty with our use of regular expressions here is that because of the Kleene closure ($*$), a g.n.p. may match an infinite number of reference or name paths if there is a cycle. This is resolved if we impose the contraint of well-definedness on context, introduced in [121, 125], but this is beyond the scope of this paper.

8.2.3 Retrieving reference and name paths from generalized name paths

The set of name paths and the set of reference paths that start in context c and "match" a g.n.p. g are denoted by $npaths_c(g)$ and $rpaths_c(g)$, respectively, and are defined as follows:

Definition 8.4 Retrieve reference paths: $rpaths_c(g)$.
This operation takes as input a g.n.p g and a context c and returns the set of reference paths that start in context c and "match" g.

$$rpaths_c(g) = \{p \mid \ \exists n \in npaths_c(g) : \ match(c, p, n)\}$$

Definition 8.5 Retrieve name paths: $npaths_c(g)$.
This operation takes as input a g.n.p. g and a context c and returns the set of name paths that start in c and "match" g. Let α be a name, β and γ be strings, and g' be a g.n.p., then

$$\begin{array}{lcl}
npaths_c(.\alpha) & = & \{\alpha\} \cap \bigcup_{o \in objs(c)} names(o, c) \\
npaths_c(.\beta\%\gamma) & = & \{n \in \bigcup_{o \in objs(c)} names(o, c) : \ n \text{ begin with } \beta \text{ and end with } \gamma\} \\
npaths_c(gg') & = & npaths_c(g) \oplus \bigcup_{\forall n \in npaths_c(g) \ \wedge \ \forall p: \ match(c,p,n)} npaths_{Ref_c(p)}(g') \\
npaths_c(g|g') & = & npaths_c(g) \ \cup \ npaths_c(g') \\
npaths_c(g\&g') & = & \{r, r' \mid r \in npaths_c(g) \ \wedge \ r' \in npaths_c(g') \ \wedge \ \exists o \in \mathcal{O} : \ r, r' \in npaths_c(o)\} \\
npaths_c((g)?) & = & p_\emptyset \ \cup \ npaths_c(g) \\
npaths_c((g)+) & = & \begin{cases} \emptyset, & \text{if } \ npaths_c(g) = \emptyset, \\ npaths_c(g(g)+), & \text{otherwise.} \end{cases} \\
npaths_c((g)*) & = & p_\emptyset \ \cup \ npaths_c((g)+) \\
npaths_c(\neg(g)) & = & npaths_c((.\%)*) - npaths_c(g) \ \diamond
\end{array}$$

Function $Ref_c(p)$, which appears in the evaluation of $npaths_c(gg')$, is formally defined in Section 7.4.2 (see Item 3). We recall that this function accesses the last context that appears when navigating through reference path p starting in context c.

8.3 Basic operations on contexts

An information system has to deal with two tasks, the maintenance of an information base and the answering of queries. In this paper, we focus on the second task, and we define the main operations for querying information bases that support contextualization.

To simplify the presentation we do not take into account the structuring of contexts. To this end, we view the contents of a context c as a set of triplets of the form $\langle o, names(o, c), ref(o, c)\rangle$, where o is an object of c. Thus, given a context c, we have:

$$cnts(c) = \{\langle o, names(o, c), ref(o, c)\rangle \mid o \in objs(c)\}.$$

For referring to context c_4 of Figure 8.1, we have:

$$cnts(c) = \{\ \langle o_7, \{\texttt{LocalInfo}, \texttt{Hotels}\}, c_6\rangle,\ \langle o_8, \{\texttt{LocalInfo}, \texttt{Dinning}\}, c_7\rangle,$$
$$\langle o_{10}, \{\texttt{MapOfCrete}\}, NIL\rangle,\ \langle o_9, \{\texttt{Transportation}\}, c_8\rangle\ \}.$$

In this section, we introduce three kinds of operations for querying a contextualized information base:

primitive operations on contexts and paths;

operations for querying a single context; and

traversal operations for obtaining paths to desired objects.

Then, in the following section, we demonstrate the expressive power of these operations by defining higher level operations based on them.

The fundamental operations in our query language are *select*, *project*, *construct*, and *path-select*. A brief description of these operations follows:

1. The *select* operation selects from the contents of a context these triplets that satisfy a given predicate. The output of this operation is a new context which contains the selected triplets. Intuitively, this operation selects from a given context the information of interest.

2. The *project* operation isolates certain elements of a context's contents, such as the objects from the context, the set of names bound to the objects, or the references of the objects.

3. The *construct* operation takes as input one or more contexts and returns as output a new context which is constructed by combining information of the input context(s). This operation allows restructuring of the information base customizing it to individual's needs.

4. The *path-select* operation selects paths originating from a given context. Paths can be of any kind, i.e., object paths, or name paths. This operation guides the navigation of the user in the database in order to reach objects of interest.

Both the select and construct operations return a new context as output. However, the select operation preserves the initial triplets, i.e., names and references of objects, whereas the construct operation creates new triplets. The output contexts of these query operations may be further queried or inserted in the information base.

8.3.1 Primitive operations

We refer to the functions that are used to define query predicates, as *primitive operations*. We distinguish two kinds of primitive operations, namely *context primitives* and *path primitives*.

1. *Context primitives* are the functions used in the definition of a context. These are: $names(o, c)$, $ref(o, c)$, $objs(c)$, and $cnts(c)$.

2. *Path primitives* are the functions used to extract parts of a path, to derive information about a path, or to match a reference path and a name path. These functions are: $first(p)$, $rest(p)$, $match(c, p, n)$.

8.3.2 Querying a single context: select, project

8.3.2.1 The select operation (Σ)

Often, given a context c, we want to select those triplets in the context's contents that satisfy a given predicate. We can do this through the *select* operation Σ, which takes as input a context c and a predicate $pred$ and returns a context denoted by $\Sigma(c, pred)$. The contents of the output context consist of all triples in the contents of c that satisfy the predicate.

Definition 8.6 Select.

Let $Q = \Sigma(c, pred)$. Then:

$$cnts(Q) = \{\langle o, names(o, c), ref(o, c)\rangle \in cnts(c) \mid pred(o, c)\}$$

where c is the input context, Q is the output context, and $pred$ is the query predicate. $\diamond$

Roughly, the equation $Q = \Sigma(c, pred)$ in the above definition should be seen as a query definition and $cnts(Q)$ as its answer.

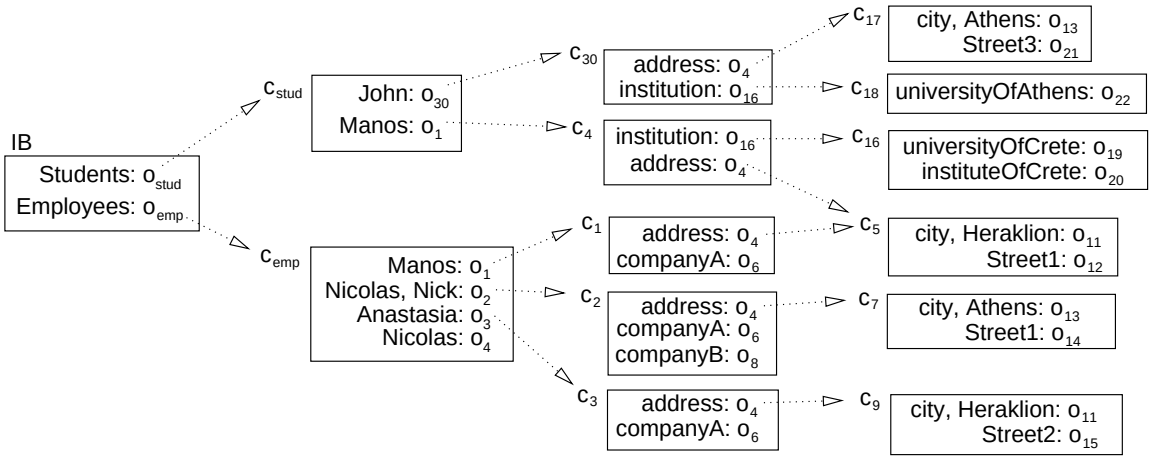
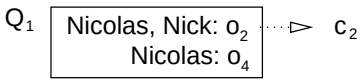


Figure 8.2: An information structure for students and employees

For example, assume that context c_{emp} in Figure 8.2 contains employees. Then the operation

$$Q_1 = \Sigma(c_{emp}, \lambda o_e \texttt{ Nicolas} \in names(o_e, c_{emp}))$$

selects all employees of context c_{emp} with the name `Nicolas`. In the query predicate, the symbol λo_e denotes that o_e is a free variable. According to the definition of the select operation, this free variable ranges over the objects of the context c_{emp}, namely, o_1, o_2, o_3, o_4. The answer to this query is the context Q_1, shown in the following figure:



The predicate $pred$ is a first-order predicate whose terms are constants, variables (representing objects, contexts, names, and paths) and primitive operations. In general, we allow comparisons using boolean operators, such as $=, \neq, <, \leq, >, \geq, \subseteq, \subset, \in$. Furthermore, several formulas may be combined into a larger formula using the boolean connectives *and* ($\wedge$), *or* ($\vee$) and *not* ($\neg$). Predicate variables other than o are quantified by *for all* ($\forall$) and *there exists* ($\exists$)

quantifiers.

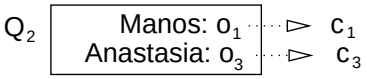
As another example, the operation

$$Q_2 = \Sigma(c_{emp}, \lambda o_e \;\; \exists c_e : \;\; c_e = ref(o_e, c_{emp}) \wedge \\ \exists p_e : \;\; match(c_e, p_e, \texttt{address.city}) \wedge \\ match(c_e, p_e, \texttt{address.Heraklion}))$$

selects all employees who live in the city of Heraklion. Note that c_e and p_e are bound variables that take values in the set of contexts and reference paths, respectively. Function $ref(o_e, c_{emp})$ returns the reference of free variable o_e w.r.t. context c_{emp}, i.e., it returns one of the contexts c_1, c_2, c_3 (call this reference c_e). Context c_e contains more information about the particular employee represented by the object o_e (such as his/her address, the company he/she works for). Operation $match$ now returns $true$ if there is a reference path p_e within context c_e that matches the name paths `address.Heraklion` and `address.city`, i.e. if c_e contains an object named `address` and the reference of that object contains an object with both names `city` and `Heraklion`. This can be also expressed using generalized name paths as:

$$npaths_{c_e}(\texttt{.address(.city\&.Heraklion)}) \neq \emptyset$$

The answer of this query is the context Q_2 shown in the following figure:



Assume now that context c_{stud} shown in Figure 8.2 contains students. Then the operation

$$Q_3 = \Sigma(c_{emp}, \lambda o_e \;\; \exists p_e : \;\; match(ref(o_e, c_{emp}), p_e, \texttt{address.Heraklion}) \;\wedge\; o_e \in objs(c_{stud}))$$

selects all employees that live in Heraklion and they are students. The first condition is checked through the formula $\exists p_e : \;\; match(ref(o, c_{emp}), p_e, \texttt{address.Heraklion})$, and the second through the formula $o_e \in objs(c_{stud})$. So, context Q_3 contains the triplet $\langle\{\texttt{Manos}\}, o_1, c_1\rangle$.

As a final example, the operation

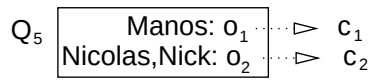
$$Q_4 = \Sigma(c_{emp}, \lambda o_e \;\; \exists p_e : \;\; match(ref(o_e, c_{emp}), p_e, \texttt{address.Heraklion}) \wedge \\ o_e \in objs(\Sigma(c_{stud}, \lambda o_s \\ \exists p_s : match(ref(o_s, c_{stud}), p_s, \texttt{institution.universityOfCrete}))))$$

selects all employees that live in Heraklion and are students at the university of Crete. So, context Q_4 contains the triplet $\langle\{\texttt{Manos}\}, o_1, c_1\rangle$.

Let us see an example demostrating the use of generalized name paths in the select operation. If we wanted to find all employees whose names start either with letter "M" or with letter "N" and work for a company (we assume that the name of a company starts with the substring "company") we would issue the operation:

$$Q_5 = \Sigma(\; c_{emp}, \lambda o_e \; (\; \exists rp_1 \in rpaths_{c_{emp}}((.M\%)|(.N\%)) : \;\; o_e = last(rp_1)) \wedge \\ (\; \exists rp_2 \in rpaths_{c_{emp}}(.\%.company\%) : \;\; o_e = first(rp_1)))$$

The answer of this query is the context Q_5 shown in the following figure:



Note that the *select* operation can be expressed in a more user-friendly manner, using an SQL-like syntax as follows:

Q = **select** (<name path>|<g.n.p.>) [<triplet variable>]
[**within context of** <context id> | <context variable> | <name path>]
[**range of** <triplet variable> **in** <name path> | <context variable>]
[**range of** <context variable> **is** <name path>]
[**where** <condition>]

where the **select** clause determines the triplets, which are selected from the context defined in the **within context of** clause and which satisfy the condition described in the **where** clause. In order to give more flexibility to the user we allow the use of variables. There are two kinds of variables which can be declared and used within a select operation (all variables are prefixed by the symbol "@"):

1. triplet variables, declared in the **select** clause, or in the **range of - in** clause, and
2. context variables, delcared in the **range of - is** clause. For example, **range of** @S **is** `Student`, define the context variable S over the reference of objects with name `Students`.

In this syntax, users work with the names of name paths to determine the objects or the contexts of interest. Names and name paths used in select operation are resolved w.r.t. the context defined in **within context of** clause. The context in that clause can be determined either by a context identifier, or by a context variable, or by a name path w.r.t. the current focuced context.

Using this syntax, our previous examples can be expressed as follows (we assume that the current focused context is the context IB):

Q_1 = **select** `Nicolas`
within context of `Employees`

Q_1 = **select** `Employees.Nicolas`

Q_2 = **select** @X
within context of `Employees`
where @X.`address.city` **and** @X.`address.Heraklion`

The declaration "**within context of** `Employees`" express that we focus the query on the context c_{emp}, because this is the reference of the object o_{emp} which is named `Employees` in the current context (i.e., the context IB). Note that the name path is ommited in **select** clause and in this case variable @X is defined over the triplets of context c_{emp}. If a triplet variable is used as the beginning of a name path the name path is resolved w.r.t. the reference of the triplet.

Q_3 = **select** @X
within context of `Employees`
range of @S **is** `Student`
where @X.`address.Heraklion` **and** @X **in** $objs(@S)$

Q_4 = **select** @X
within context of Employees
where @X.address.Heraklion **and**
Student.@X.institution.universityOfCrete

Q_5 = **select** $(.M\%)|(.N\%)$ @X
within context of Employees
where .@X.company%

8.3.2.2 The project operation (Π)

Sometimes we want to isolate certain elements of a context's contents, such as the objects of the context, the set of names bound to the objects, or the references of the objects. We can do this through the *project* operations Π_O, Π_N, Π_R, that take as input a context c and return the set of objects, the set of names, and the set of references of c, respectively.

Definition 8.7 Project.
Let c be a context, then

- $\Pi_O(c) = objs(c)$
- $\Pi_N(c) = \bigcup_{o \in objs(c)} names(o, c)$
- $\Pi_R(c) = \{c' \in \mathcal{CXT} \mid \exists o \in objs(c) : \ c' = ref(o, c)\}$

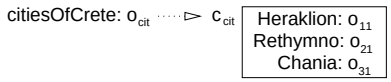
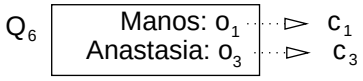


Figure 8.3: Cities located in the island of Crete

For example, assume that context c_{cit} in Figure 8.3 contains cities located in the island of Crete. We represent three of the cities, namely, Heraklion, Rethymno and Chania, by the objects o_{11}, o_{21}, o_{31}, respectively. Then, the operation

$$Q_6 = \Sigma(c_{emp}, \lambda o_e \ \ \exists n_c : \ n_c \in \Pi_N(c_{cit}) \wedge \exists p_e : \ match(ref(o_e, c_{emp}), p_e, \texttt{address}.n_c))$$

selects the employees that live in a city on the island of Crete.
Note that the operation $\Pi_N(c_{cit})$ returns a set which contains the names of all cities located in Crete, i.e. $\Pi_N(c_{cit}) = \{Heraklion, Rethymno, Chania\}$. Additionally, note that the name path address.n_c contains the name variable n_c. The answer of this query is the context Q_6 shown in the following figure:



The *project* operation can be expressed in SQL-like syntax as follows:

project$_O$ <context>
project$_N$ <context>
project$_R$ <context>

8.3.2.3 Combining Contexts (Γ)

The construct operation combines a set of contexts to create a new context. It allows the user to "customize" information by selecting portions of information from different contexts and putting them together into a new context.

The *construct* operation takes as input one or more contexts and a predicate, and returns as output a new context, denoted by $\Gamma(c_1, \ldots, c_n, pred)$, whose contents satisfy the input predicate. In fact, the contents of the output context consist of new triplets which are constructed by combining the contents of the input context(s).

Definition 8.8 Construct.

Let $Q = \Gamma(c_1, \ldots, c_n, pred)$. Then

$$cnts(Q) = \{\langle o, N, r\rangle \mid pred(o, N, r, c_1, \ldots, c_n)\}$$

where c_i are the input contexts, Q is the output context, $pred$ is the query predicate, and o, N, r are free variables ranging over the set of objects, the power set of names, and the set of contexts, respectively. $\diamond$

For example, referring to Figure 8.2, the operation

$$\begin{aligned} Q_7 = \Gamma(c_1, \lambda o_a\ \lambda N_a\ \lambda r_e & \\ o_a = o_4\ \wedge\ N_a = \{&\texttt{Addresses\_of\_Employees}\} \wedge \\ r_e = \Gamma(c_{emp},\ \ & \lambda o_e\ \lambda N_e\ \lambda r_a \\ & o_e :\ \ o_e \in \Pi_O(c_{emp})\ \wedge\ \ \exists c_e :\ \ c_e = ref(o_e, c_{emp}) \wedge \\ & \exists o'_a \in \Pi_O(\Sigma(c_e, \lambda o_x\ \texttt{address} \in names(o_x, c_e))) \wedge \\ & N_e = names(o_e, c_{emp})\ \wedge\ r_a = ref(o'_a, c_e))) \end{aligned}$$

creates a context which contains the addresses of the employees, each labeled by the names of the corresponding employee. In particular, the following hold for the nested construct operation:

1. the operation takes as input the context c_{emp} which contains employees;
2. o_e is a bound variable which ranges in the objects of context c_{emp}; this is expressed by

   $$\exists o_e :\ \ o_e \in \Pi_O(c_{emp}).$$

3. c_e is the reference of object o_e in context c_{emp}, which contains more information about the employee represented by o_e;
4. free variable o_x ranges over the objects of context c_e with name `address`. This is expressed by:

   $$o'_a \in \Pi_O(\Sigma(c_e, \lambda o_x\ \texttt{address} \in names(o_x, c_e)));$$

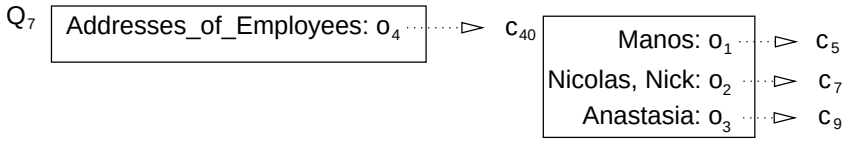
5. free variable N_e is the set of names bound to object o_e in context c_{emp}. This is expressed by:

   $$N_e = names(o_e, c_{emp});$$

6. free variable r_a is the reference of bound variable o'_a in context c_e, which contains more information about the address of the employee o_e;

7. the nested construct operation creates a new context c_{40} which contains triplets of the form $\langle o_a, N_e, r_a \rangle$.

Finally, this query operation creates the context Q_7, which contains the object o_4 under the nam `Addresses_of_Employees` and refers to context c_{40}, i.e., the context created by the nested construct operation. The result is shown in the following figure:



We often need to restructure the contents of contexts either by eliminating useless information or by adding new information. The construct operation can be used for this purpose. For example, assume that the information we want to view for each employee is his/her address and from the address only the city he/she lives in. To select the desired information and eliminate the rest we perform the following query:

$$
\begin{aligned}
Q_8 = \Gamma(c_{emp}, \lambda o_e\ \lambda N_e\ \lambda r \\
& o_e \in \Pi_O(c_{emp}) \wedge \\
& N_e = names(o_e, c_{emp}) \ \wedge \ \exists c_e : \ c_e = ref(o_e, c_{emp}) \wedge \\
& r = \Gamma(c_e, \ \lambda o_a\ \lambda N_a\ \lambda r_a \\
& \qquad o_a \in \Pi_O(\Sigma(c_e, \lambda o_x\ \texttt{address} \in names(o_x, c_e))) \wedge \\
& \qquad N_a = names(o_a, c_e) \ \wedge \ \exists c_a : \ c_a = ref(o_a, c_e) \wedge \\
& \qquad r_a = \Sigma(c_a, \lambda o_y\ , \texttt{city} \in names(o_y, c_a)))
\end{aligned}
$$

This operation creates a new context (context Q_8) for employees by selecting from the information available for each employee that regarding his/her city of resedence and by eliminating all other information. The selected information is placed into a new context structure, as shown in Figure 8.4. Note that contexts c_{31}, c_{32}, c_{33}, c_{35}, c_{37}, and c_{39} are new.

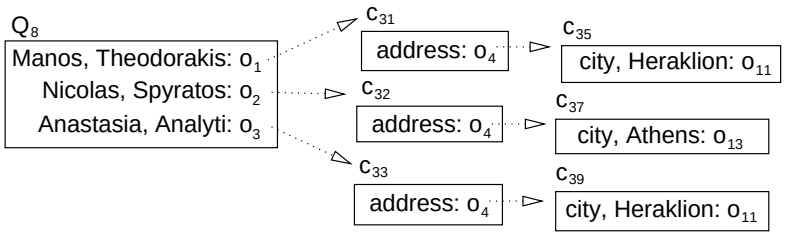


Figure 8.4: A new information structure for employees

This feature could be useful for web-based information bases, where there is a tremendous amount of available information. It provides a means for selecting and organizing information of interest.

We must stress that the construct operation allows users to reorganize the context structure of an information base. This reorganization may involve the creation of new contexts by (i) eliminating useless information from existing contexts, and (ii) combining information found in different contexts. Thus, users can customize the information base according to their needs.

Note that the *construct* operation can be expressed in a more user-friendly manner, using an SQL-like syntax as follows:

Q = **construct** <object variable, set of names variable, reference variable>
[**range of** <triplet variable> **in** <name path> | <context variable>]
[**range of** <context variable> **is** <name path>]
[**where** <condition>]

Using this syntax, query Q_7 of our previous example can be expressed as follows (we assume that the current focused context is the context IB):

Q_7 = **construct** <@O,$names(@O, @C_e)$,$ref\,(@O, @C_e)$>
range of @C$_{emp}$ **is** `Employees` @C$_{emp}$
where
@O **in project**$_O$ @C$_{emp}$
and
(@O **in project**$_O$
(**select** `address`
within context of @C$_{emp}$)
and
@C$_e$ = $ref\,(@O, @C_{emp})$

8.3.3 Traversal operations: path select

We often are interested not only in selecting objects satisfying some criteria but also in being aware of the path we can follow to reach those objects. This is useful in order to (i) choose the desired path to reach an object of interest, (ii) see in which paths the desired information is embedded, and (iii) find different representations of the same object.

For example, if we want to reach an object o starting from a context c, we can select all reference paths that start in c and whose last element is object o.

In order to perform selections of paths that satisfy a query predicate, we introduce two *path select* operations: the *reference path select*, denoted by $RP\Sigma$, which returns a set of reference paths, and the *name path select*, denoted by $NP\Sigma$, which returns a set of name paths. Intuitively, we want this operation to take as input a context c and a predicate, and to return those reference paths or name paths in c that satisfy the input predicate.

Definition 8.9 Path select.
Let c be a context and $pred$ a predicate. We define the following path select operations:

1. Reference path select: $RP\Sigma(c, pred) = \{rp \mid pred(c, rp)\}$ where rp is a reference path starting in c.
2. Name path select: $NP\Sigma(c, pred) = \{np \mid pred(c, np)\}$, where np is a name path starting in c. $\diamond$

For example, refering to Figure 6.1, the operation

$RP\Sigma(c', \lambda p\ last(p) = o_1)$

selects all reference paths starting in c' that leading to object o_1. The answer to this query is the following set of reference paths: $\{\ o_4.o_1,\ o''.o.o_1\ \}$. So, if we want to choose the shortest path to reach object o_1, we must choose the first of these paths. Apart from the shortest path, this query shows us how many different representations exist for the same object and where they are located in the context hierarchy. In our example, following the path $o_4.o_1$ we find object o_1 contained in context c_3, with name `Crete` and with a reference c_4 (this is a representation of object o_1 w.r.t. context c_4). On the other hand, following the path $o''.o.o_1$, we find object o_1 contained in context c_1, with the same name `Crete`, but without any reference (this is another representation of object o_1, in this case w.r.t. context c_1).

Similarly, the operation

$$NP\Sigma(c', \lambda n\ last(n) = \texttt{Crete})$$

selects all name paths in c' which end with the name `Crete`, i.e. the set:
$\{$ `Tourist_Guide.Crete`, `Geography_Of_Greece.Islands.Crete` $\}$.

8.4 Additional operations on contexts: copy, deepcopy

The *copy* operation takes as input a context c, and returns as output a new context which contains all objects of c with the same names and the same reference.

Definition 8.10 Copy.

$$Copy(c) = \Sigma(c, true)$$

The *deep copy* operation takes as input a context c, and returns as output a new context which contains all objects of c with the same names and deep copies of their reference. The output context is equivalent to c.

Definition 8.11 Deep copy.

$$DeepCopy(c) = \begin{cases} NIL, & \text{if } c = NIL \\ \Gamma(c, \lambda o\ \lambda N\ \lambda r & \\ \quad o \in \Pi_O(c)\ \wedge & \\ \quad N = names(o, c)\ \wedge & \\ \quad r = DeepCopy(ref(o, c))), & \text{otherwise} \end{cases} \tag{8.4}$$

As the deep copy of a context is based on deep copies of object references, the deep copy operation is recursive. The recursion stops when the reference of an object is NIL.

These operations correspond to the copy and deep copy operations defined in Subsection 4.2.4.

8.5 High-level operations

In this section, we give examples of high-level operations which are defined using the basic operations seen so far.

In our examples, we assume that the user interacts with the information base by (i) focusing on a context (called *the current context* and denoted by CC), and (ii) searching for information

starting from that context. During this interaction the user specifies objects of interest through name paths, and indicates the appropriate actions to be performed on these objects. Here we introduce two high-level operations: $lookup$ and $cross\text{-}reference$.

Open(♯N):

The user can focus on a context of interest using the operation $Open(\sharp N)$, where N is a name path (starting in the current context and reaching an object of interest), and $\sharp N$ denotes the reference of that object w.r.t. the last accessed context in the matched reference path. For example, in Figure 6.1, assume that the current context is context c'. Then the operation $Open(\sharp$`Tourist_Guide.Crete`$)$ focuses on context c_4. In visual user interfaces, the operation $Open(\sharp N)$ can be implemented by clicking on a visual representation of N.

It is possible that N matches more than one reference paths, so $\sharp N$ is ambiguous. In this case the system returns the set of name paths that match each of these reference paths. Based on these name paths, the user can select the desired reference path which determines the value of $\sharp N$.

Lookup:

The $lookup$ operation takes as input a name n and returns the set of name paths starting in the current context and whose last name is n.

Definition 8.12 Lookup.

Let np be a name. Then

$$lookup(n) = NP\Sigma(CC, \lambda N_i\ last(N_i) = n)$$

This operation is especially useful in environments such as the Web, where users seeking information follow a three step approach: (i) they focus on a context c, i.e., a search engine, (ii) they give a name n, i.e., a keyword they know, and (iii) they use $lookup(n)$ to decide on which context to focus next. Roughly speaking, the name n is a keyword that the user has in mind and that describes the information he is looking for, and the result of the operation is the set of all name paths reaching that information. Each of these name paths describes a different way to reach the desired information.

For an example, refer to Figure 8.5. The information base context (context IB) contains four objects o_5, o_9, o_{13} and o_1, representing cooking, botany, diseases and languages, respectively. The references of these objects, i.e., the contexts c_5, c_9, c_{13} and c_1, give furter information about these topics.

Assume now that we have focused on the context IB (i.e., on the whole information base) and we want to find information about the word Fennel, but we don't know exactly where to look. A solution is to issue the operation $lookup($`Fennel`$)$. The result is a set of four name paths starting from the current context and whose last element is `Fennel`. This set consists of four name paths (as shown below), partitioned into two blocks according to the object they reach: the first three paths reach the same object o, thus constitute the first block, while the fourth name path reaches the object o_{17} and constitutes (alone) the second block.

$\Rightarrow$ `Cooking.Recipes.Fish_in_oven.Ingredients.Fennel`

`Botany.Herbs.Fennel`

`Languages.English.Fennel`

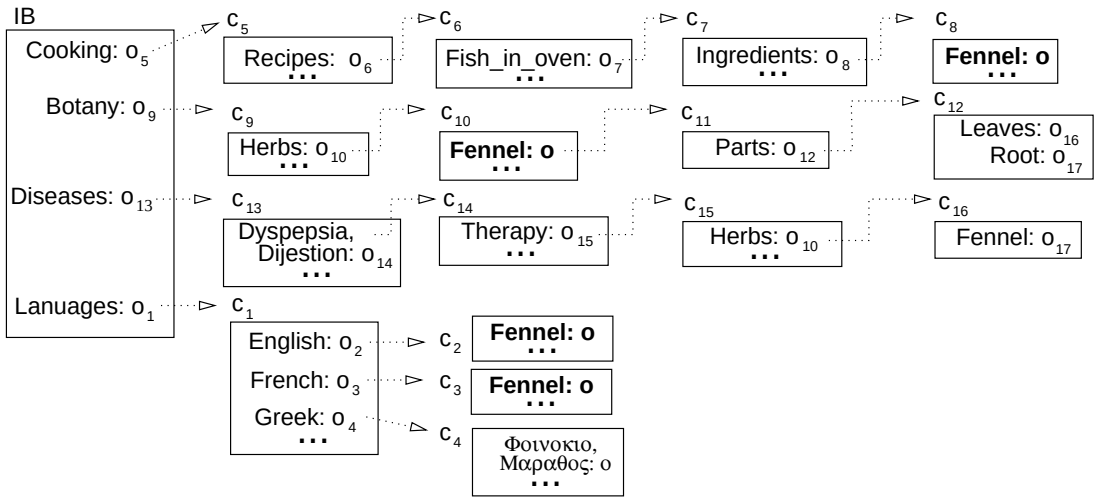


Figure 8.5: Example of high-level operations

$\Rightarrow$ `Diseases.Dyspepsia.Therapy.Herbs.Fennel`

`Diseases.Digestion.Therapy.Herbs.Fennel`

Note that the above name paths can be used in further explorations concerning the word Fennel. This can be done by "feeding" any of the above paths into the $Open$ operation, seen earlier. For example, if we wanted to find more information about Fennel as a plant, then we would use the path starting from `Botany`, i.e., we would feed the path `Botany.Herbs.Fennel` into $Open$, issuing the operation $Open(\sharp$`Botany.Herbs.Fennel`$)$. As a result we would focus on the context c_{11} would then become the current context. Using this new current context we could then explore first its contents (using *select* or *project* operations) and then those of c_{12} (using again $lookup$, for example). In general, $Open$ and $lookup$ provide a powerful combination for exploring a contextualized information base.

Cross Reference:

Suppose that an object o is reachable from the current context through a name path N. For example, in Figure 8.5, the object o is reachable from the context c_9 through the name path `Herb.Fennel`. Suppose now that we would like to know the name paths through which that same object o is reachable from another given context c. For example, in Figure 8.5, the object o is reachable from context c_5, through the name path `Fish_in_oven.Ingredients`. The operation that helps us to do this is the *cross-reference* operation, that takes as input an object o and a context c and returns the name paths starting in c and reaching o. In fact, the definition that we give below is more general in that:

- the object o is specified as the last object of a reference path that matches a given name path N;
- the context c is specified as the last reference of a reference path that matches a given name path N_{cr}.

As a result, N may specify more than one object and N_{cr} more than one context.

Definition 8.13 Cross Reference.

Let n, n_{cr} be two name paths. We define the following cross reference operation:

$$CrossRef\,(N, \sharp N_{cr}) = NP\Sigma(\sharp N_{cr}, \lambda N_x \quad \exists p_x : \ match(\sharp N_{cr}, p_x, N_x) \wedge \\ \exists p_y : \ match(CC, p_y, N) \ \wedge \ last(p_x) = last(p_y))$$

An application of cross-reference is for translating from one language to another. For example, refer to Figure 8.5 and suppose that c_1 is the current context. Consider now the English word Fennel and suppose that we would like its translation in French. This can be expressed as $CrossRef\,(\texttt{English.Fennel}, \sharp\texttt{French})$ which returns the name `Fenouil`. Similarly, suppose that we would like to find the translation of `Fennel` into all languages contained in `Languages`. This would be expressed as $CrossRef\,(\texttt{English.Fennel}, CC)$ which returns

`English.Fennel`,

`French.Fenouil`,

`Greek.`Μάραθος,

`Greek.`Φοινόκιο.

Note that there are two words in Greek for the english word Fennel, therefore the operation

$CrossRef\,(\texttt{Greek.}$Μάραθος$, \texttt{Greek})$

would return Μάραθος and Φοινόκιο.

Continuing our example, assume that we would like to know how the real world entity expressed by the english world Fennel (i.e., object o) is being used in Cooking or Botany. To do this, we issue the operations

$CrossRef\,(\texttt{English.Fennel}, \sharp{::}\ \texttt{Cooking})$

$CrossRef\,(\texttt{English.Fennel}, \sharp{::}\ \texttt{Botany})$.

The symbol "::" is a scope resolution operator, and it means that the name path it follows is resolved w.r.t. the information base context (context IB). The first operation would return

`Recipes.Fish_in_oven.Ingredients.Fennel`

and the second

`Herbs.Fennel`

Note that context c_8 sees Fennel as an ingredient of a specific recipe, whereas, context c_{10} sees Fennel as a herb from the Botany point of view.

Recall that the operation $lookup(\texttt{Fennel})$ found two different objects with name `Fennel`, where the second can be reached by the name path `Diseases.Dyspepsia.Therapy.Herbs.Fennel`. To find out more about the meaning of that second object, we may try to find alternative paths leading to that object in the information base. To do this, we focus on the information base context IB and then, we issue the operation:

$CrossRef\,(\texttt{Diseases.Dyspepsia.Therapy.Herbs.Fennel}, CC)$

which returns the input path:

```
Diseases.Dyspepsia.Therapy.Herbs.Fennel
```

and the paths:

```
Diseases.Dijestion.Therapy.Herbs.Fennel
```

```
Botany.Herbs.Fennel.Parts.Root.
```

The latter gives the information that the root of fennel is used in the therapy of dyspepsia.

In summary, the high-level operations that we have seen can be used for exploring contextualized information bases by providing support for (i) focusing on a context of interest and navigating along refererce paths, (ii) searching in a context, and (iii) switching to different contexts based on cross reference information. Similarly to these, other high-level operations can be defined, providing the basis for effective information searching and correlation.

8.6 Summary

In this chapter, we discuss how to access information in a contextualized information base. This is done through reference and name paths. These paths provide a means to traverse from an object to another through object references. Then, we presented the basis for querying such information bases by defining four fundamental query operations on contexts: *select*, *project*, *construct* and *path select*. Finally, we present three high-level operations which allow the user to (i) focus on the context of interest, (ii) search in a context using keywords, and (iii) switching to different contexts based on cross reference information.

Chapter 9

Applications

This chapter discusses how the contextualization mechanism presented in this thesis can support applications which rely on partitionings of an information base. Such well-known partitions are database views, software engineering workspaces, versions, configurations, thesauri fields and guided terms.

9.1 Views

Relational views. A view in the relational data model is defined as the (virtual) relation derived from a query. Similarly to base relations, views consist of tuples. However, all information contained in a view is derived from base relations or other views.

In our contextualization mechanism, the notion of view is fully supported by the notion of context as follows: A view context is defined as the (virtual) context derived from a query. A view context, like base contexts, consists of triplets (the contents of the context) that are derived from base contexts or other view contexts. Recall that query operations (except for *project* and *path select*) take as input one or more contexts and return as output a new context. Examples of view contexts are given in subsection 8.3.2.

Object-oriented views. Object-oriented views [98, 2, 19, 94] (see [81] for a survey) are supposed to be "upward compatible" with relational views, i.e., object-oriented views extend the functionality of the relational views. The additional functionality of object-oriented views is induced by the higher expressive power of the object-oriented approach.

Our contextualization mechanism supports object-oriented views[1] as

1. It offers object-oriented structuring capabilities, i.e., classification, generalization, and attribution abstraction mechanisms;
2. The object preservation property holds for the querying operations, i.e., objects contained in the result of a query already exist in the input contexts, and no new objects are created.

Thus, the proposed framework can deal with issues such as

[1] We support only the structural part of the object-oriented views, whereas the functional part (methods) is not supported yet.

- Updates on base objects through context views (note that as the object preservation property holds, we do not have to cope with the view update problem [44]).
- Integration of heterogeneous databases by interpreting them as contexts.
- Different interpretations of the same object, or in other words, *multiple interfaces* to the same object. This is because an object may be viewed differently in various contexts.
- Reorganization of the class hierarchy.

Note that the view organization model presented in [98] can also be applied in our framework.

As our framework proposes a new abstraction mechanism (in addition to traditional ones), it can support more powerful and expressive view definitions. However, restructuring of objects and contexts, as well as schema reorganization becomes more complex. View support in our framework is still an open issue under consideration.

9.2 System decomposition

The decomposition of a system [134] is a process that results in a set of subsystems, where every element in the system is included in at least one of the subsystems.

This definition of system decomposition carries a notion of context as (i) subsystems are meaningful partitions of the system, and (ii) different sets of subsystems may result from decompositions taken under different perspectives.

A system decomposition leads naturally into the notion of a *level structure* over a system. A level structure formalizes the idea that sets of subsystems are "nested" within particular systems which in turn are nested within other systems (this is similar to nesting of contexts).

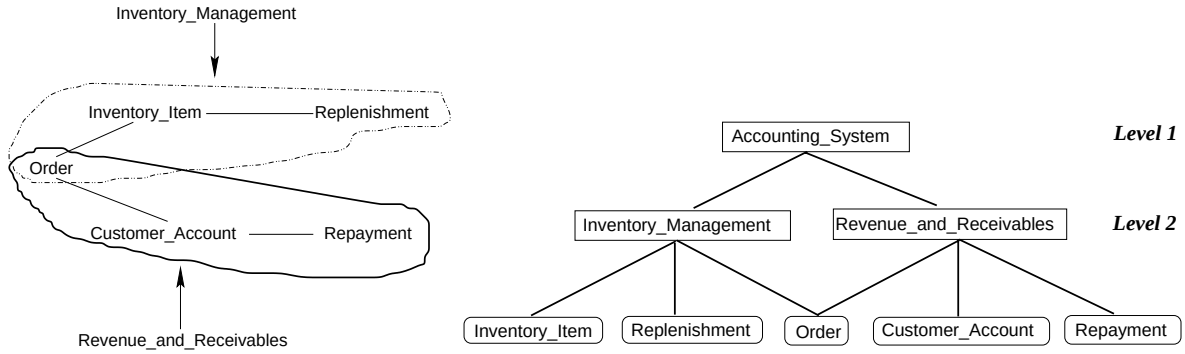
Thus, the structure of system decomposition could be represented through the context mechanism as follows:

1. A system is represented by an object (called *system object*);
2. Different decomposition perspectives are described by different contexts (called *decomposition contexts*).
3. Subsystems are described in the reference of the system object with respect to a decomposition context.

The decomposition of a system under a particular perspective can be represented by the context-subcontext structure. However, the decomposition hierarchy conveys some additional semantics. This is because all the information contained in the subsystem must be part of the system as well. The system-subsystem relation is actually a special kind of context-subcontext relation.

For example, the system of Figure 9.1(a) has been decomposed into two subsystems: an inventory management subsystem, and a revenue and receivables subsystem. Figure 9.1(b) shows the level structure for the system decomposition shown in Figure 9.1(a). The decomposition comprises three (sub)systems: the accounting system itself, the inventory management,

and the revenue and receivables subsystems. The level structure comprises two partitions: one, *Level 1*, contains the accounting system itself; and the other, *Level 2*, contains the two subsystems. Each system in the second partition is a subsystem of some system in the higher-level partition. Figure 9.2 illustrates how the decomposition of Figure 9.1 is supported by our context model.



(a): Decomposition of a system (b): Level structure for decomposition

Figure 9.1: System Decomposition

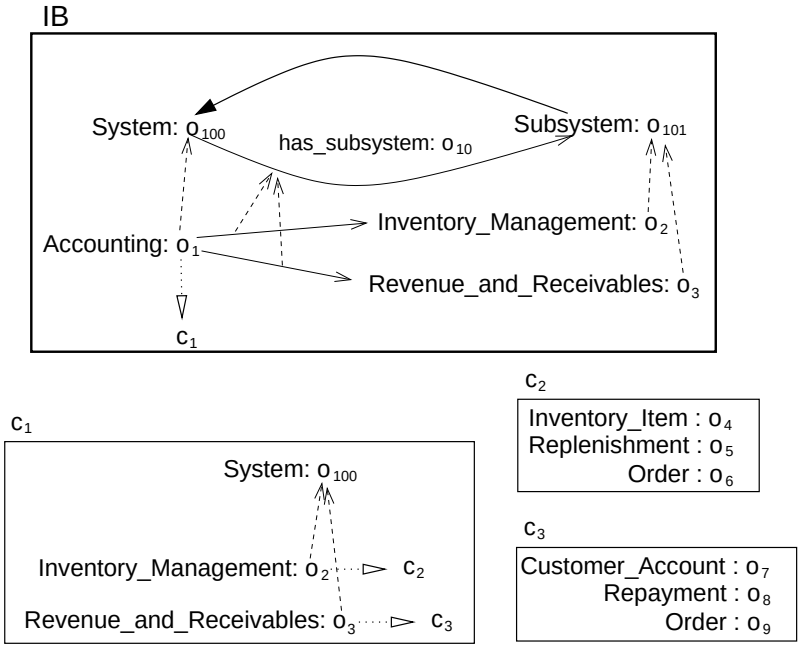


Figure 9.2: Decomposition of a system using contexts

9.2.1 Communication-based applications

This category includes applications that need a kind of communication to exchange information. Examples of this kind of applications are workspaces, multi-databases, and modules. Components of these applications exchange information by (i) establishing a communication channel among them, and (ii) importing from or exporting to the communication channel the desired information.

In order to support such applications using our framework, we define the following high level operations:

1. $c_{comm} = SetUpCommunicationChannel_c(o_1, o_2)$: sets up a communication channel among the input objects o_1 and o_2 of a context c. This is done as shown in Figure 9.3. Specifically, this operation establishes a link in c from o_1 to o_2, as well as a reference, c_{comm}, of that link in c. The reference c_{comm} constitutes the context through which information is exchanged between o_1 and o_2, and is returned by the operation.

2. $Export_c(o, c_{comm})$: object o of a context c exports information from the reference of o in c to the communication channel c_{comm}.

3. $Import_c(o, c_{comm})$: object o of a context c imports information from the communication channel to the reference of o in c.

An example of establishing communication channels between objects is shown in Figure 9.3.

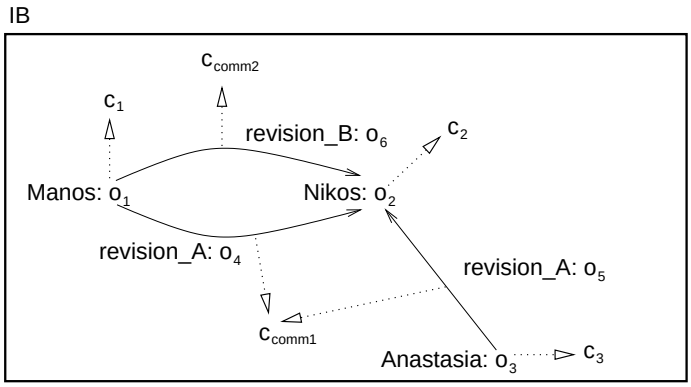


Figure 9.3: Communication channel

9.2.2 Thesauri management systems

In thesaurus management systems, two general approaches can be adopted to the organization of the thesaurus terms [34, 115]:

1. organization into fields or disciplines;

2. organization by facets.

In practice these techniques are frequently combined: that is, a thesaurus may be organized primarily into fields, and facets may then be introduced as the basis for organizing concepts within a given field.

Fields Categories of concepts are initially grouped to reflect the various fields of interests of its users. Terms which are usually associated with a given field, such as “art”, are brought together, and are also effectively separated from those belonging to different areas of interest, such as “economics” or “physics”.

It is clear that the notion of field in thesaurus management systems can be modelled using contexts as follows: (i) fields are represented by objects whose references contain detailed information about them, and (ii) subfields are contained in the reference of the fields. For example, consider a thesaurus devoted to "medicine", which is organized into subfields such as "injuries", "diseases", and "treatments", but also includes a number of terms from other disciplines, such as "management", "law", and "data processing". This thesaurus can be described using contextualization, as shown in Figure 9.4.

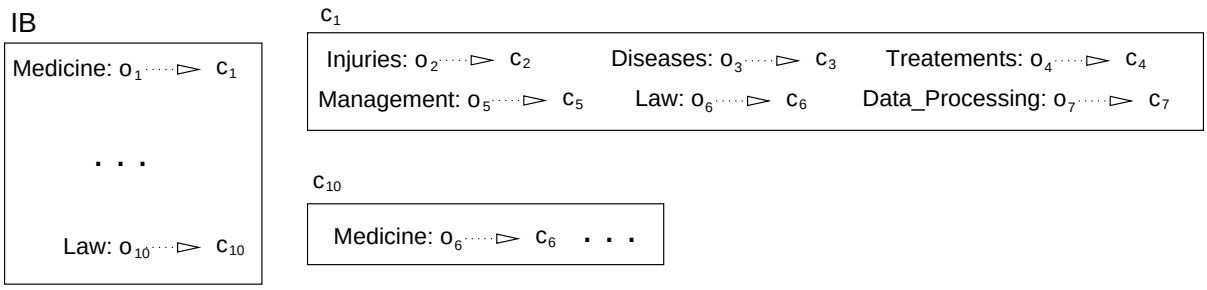


Figure 9.4: Representation of thesauri fields with contexts

Note that object o_6 represents the part of the law disciplines used for medicine, and thus it is shared by the fields "medicine" and "laws".

Apart from modeling fields, the context mechanism can solve ambiguities that appear when the same term is used in different fields to convey different meanings (homonyms).

Facets The notion of facet is more abstract than the concept of a field. When applying this technique, terms are organized into classes or sets according to the basic kind of concept they represent, with no initial regard for the field or fields with which a given concept is usually associated. For example, the facet "concrete things" might be divided into "natural occurring things" and "artificial things". The facet "artificial things" might further distinguished into "product" and "tools", and the facet "tools" might further be distinguished into "hammer", "screwdriver", etc.

In our model, a facet organization can be represented through a context hierarchy as follows: (i) an abstract term is represented by a class, whose reference contains its divisions, and (ii) concrete terms are classified into abstract terms through instance-of links. For example, "concrete things" is an object whose reference contains the divisions "natural occurring things" and "artificial things"; the reference of "artificial things" contains divisions "product" and "tools"; whereas objects "hammer" and "screwdriver" are instances-of the class "tools". Moreover, the facet classification carries ISA semantics, e.g. "tools" is a subclass of "artificial things".

Node labels *Node labels* (also called *guide terms*) shows the point of view according to which an abstract term in the facet hierarchy is divided into less abstract terms. For example, the term "aircrafts" is divided into "freight aircrafts" and "passenger aircrafts", according to the "payload" point of view, whereas the same term is divided into "civil aircrafts" and "military aircrafts", according to the "user" point of view.

Node labels can be represented through a context hierarchy as follows: (i) the abstract term is an object whose reference contains the node labels according to which it is divided, and (ii) each node label has a reference that contains the divisions of the abstract term.

9.2.3 Workspaces, versions, multiversion objects, configurations

A *version* can be thought of as an object whose reference contains more information about the version. A *multiversion object* can be thought of as an object whose reference contains all versions of the object, as well as a version history, i.e., derivation links (attribute links that are instances of the attribute class `derived_from`) from one version to another that indicate version derivation. A *configuration* can be thought of as an object whose reference contains versions of its components.

A *workspace* can be thought of as an object whose reference contains information about the workspace. For example, the public workspace might contain multiversion objects, whose reference with respect to the public workspace may contain all objects that represent fully verified object versions, as well as its version history.

A detailed example about workspaces, versions, multiversion objects, and configurations is presented in Chapter 5.

Chapter 10

Conclusions and Future Work

In this thesis we deal with *contextualization*, an abstraction mechanism for information modeling, which allows partitioning and packaging of information added in an information base. In particular, we define a notion of context and we address issues such as

1. contextualized naming,
2. operations on contexts,
3. properties of the operations on contexts,
4. interaction between contextualization and traditional abstraction mechanisms,
5. establishing a formal model for contextualization in information bases,
6. querying contextualized information bases,
7. applying context in areas requiring partitionings such as views, workspaces, versions, thesauri fields and facets.

The aim of this thesis is to establish a formal notion of context in order to support the development of applications which rely on partitionings of an information base, such as views, workspaces, versions, fields and facets of thesauri management systems.

The modeling capabilities provided by contextualization abstraction mechanism are the following:

1. **Naming capabilities.**
   (a) An object can have zero, one, or more names within a context. If an object has no name within a context is called *anonymous*. This feature is useful in the case that we are not interested in naming the anonymous object or in the case that we don't know its name yet. If an object has more than one names within a context then these names are *synonymous* for that object. We view synonyms as an alternative way to externally identifying the same object.
   (b) Two different objects can have the same name within a context. In other words, our mechanism support *homonymous* objects. Ambiguities that may occur in referencing homonyms are resolved by more names (unique in the context of reference), that is by synonyms.

(c) The same object can have different names in different contexts (in which it belongs). In other words, names are context-dependent. This is very convenient, because a name which has a clearly understood meaning in one context may not do so in another.

2. **Structuring capabilities.**

   (a) An object can be related to other objects through attribute, instance-of, or ISA links within a context, i.e., attribution, classification, and generalization hierarchies are context-dependent.

   (b) Nesting of contexts. An object of a context can be associated with another context, that we call its *reference*. The reference of an object within a given context can be seen as a nested subcontext within that context.

3. **Sharing capabilities.**

   (a) An object can belong to one or more different contexts, i.e., contexts may overlap. This feature is useful when we want to view an object under different perspectives.

   (b) The same object can have different references within different contexts. In other words, references are context-dependent.

   (c) Two different objects, whether or not they belong to the same context, or to different contexts, can have the same reference. This is convenient, as a given context can be reachable through different object paths. In application that needs communication, this the context which is referenced by two or more different objects can be seen as the shared space used for object communication and interoperability.

Our proposed mechanism offers several operations for maintaining contextualized information bases such as operations for updating, copying, combining, and comparing contexts. Moreover, our mechanism offers a general framework for querying contextualized information bases by

1. accessing information in the context structure using name path expressions or paths of objects (paths are formed by following the references of objects),

2. defining useful fundamental query operations on contexts such as select, project, generate (which allows the reorganization of the context structure), and path select.

Adding contextualization to an information base provides several advantages, including the following:

1. Reduction of modeling complexity: providing modularity we make the contents of an information base more understood.

2. Expressiveness: providing context-dependent semantics, advanced naming capabilities (e.g. homonyms, synonyms, anonyms, frugal names) handling of inconsistent information, and interaction between contextualization and the traditional abstraction mechanisms of classification, generalization, and attribution.

3. Reusability: reducing development time, by affecting several reusability aspects: (i) *specification*: partitioning of information makes information of each part to be more understandable, and then makes it more likely to be reused; (ii) *composition*: allowing restructuring of information; (iii) *extensibility*: a context or a copy of it can be easily extended. Moreover, reusability is supported by abstract objects and refinement relation between contexts we provide.

4. Focused information access: a context delimits the parts of an information base that are accessible in a given way.

5. Flexibility: providing top-down, bottom-up, or mixed modeling techniques,

Further work on context mechanism includes dealing with

1. The incorporation of context mechanism, as prescribed by our model, in specific data models, such as relational data model.

2. The application of context mechanism in heterogeneous information bases, where each information base can be seen as a context. Research on this topic can use ideas of presented in subsection 9.2.1, for support communication-based applications.

3. Restructuring of objects and contexts, as well as schema reorganization. Contextualization contextualized information bases is more complex in restructuring of information than this in conventional object-oriented frameworks.

4. Supporting types of partitioning the domain of discourse. Intuitively, these types correspond to different criteria for decomposing (or partitioning) the domain of discourse. For example, temporal criteria can be used to partition the domain of discourse into centuries, each century into decades, each decade into year and so on.

5. Making the reference of an object as first class object (similar to attribute link) and allowing an object to have more than one reference in a context. This offers more modeling expressiveness and flexibility as it allows objects to have more than one reference in a context being classified under different classes of references. Roughly speaking, these reference classes can be seen as the criteria for partitioning the domain of discourse.

6. Investigation of additional high level query operations.

7. Investigation of additional properties of context operations.

Appendix A

Operations on Versioning

In this Appendix, we give the detailed algorithms of the operations presented in Chapter 5.

Operation A.1 Check-out.
check-out(**Input** $r : \mathcal{NP},\ n : \mathcal{N}$).

$/*$ With this operation, a designer checks-out a new version of the version o (referred to as r) from the public workspace into his / her private workspace. The new version is a copy of o and is given the name n w.r.t. the private workspace. This operation also copies the history context that contains o from the public to the private workspace. $*/$

1. `CurrentWorkingContext` $=$ `CC`.
2. SCC(@.`Public`).
3. $o = lookupOne(r)$.
4. $o\_copy = copy(o)$.
5. $hc = whereContainedIn(o, HISTORY)$.
6. $hc\_copy = copy(hc)$.
7. $insert(hc\_copy, names(hc, HISTORY), Home)$.
8. $insert(o\_copy, \{n\}, hc\_copy)$.
9. $updateHistory(o\_copy, hc\_copy)$.
10. If $o \in objs$(`CurrentWorkingContext`) then $deleteObj(o,$ `CurrentWorkingContext`).
11. $insert(o\_copy, \{n\},$ `CurrentWorkingContext`).
12. `CC` $=$ `CurrentWorkingContext`.
13. End.

Operation A.2 Check-in.
check-in(**Input** $r, h : \mathcal{NP},\ n : \mathcal{N}$).

/* With this operation, a designer checks-in his own new version (referred to as r) into the public workspace with a name n. This operation also inserts the new version into the history context referred to as h and will update the version history hierarchy. */

1. $o = lookup(r)$.
2. $hc = lookup(h)$.
3. $v = copy(o)$.
4. $insert(v, \{n\}, hc)$.
5. $updateHistory(ver\_o, hc)$.
6. End.

Operation A.3 Export to group.
export(**Input** $exportedListOfContexts : \mathcal{P}(\mathcal{NP}),\ exportedCxtName : \mathcal{N}$).

/* With this operation, the designer creates a context c which contains a copy of the context referenced by each name path $\mathtt{r_i}$ contained in the input set $exportedListOfContexts$. Then, it inserts the context c into the group workspace, under the name $exportedCxtName$. */

1. $toBeExportedCxt = createCxt(\{\})$.
2. For each $r_i \in exportedListOfContexts$ do
3. $\quad insert(lookupOne(r_i), \{str(r_i)\}, toBeExportedCxt)$.
4. $insert(toBeExportedCxt, \{exportedCxtName\}, GROUP)$.
5. End.

Operation A.4 Import from group.
import(**Input** $r : \mathcal{NP},\ n : \mathcal{N}$).

/* With this operation, a designer imports the context referred to by r from the group workspace into his / her private workspace. */

1. $SCC(Group)$.
2. $c = lookupOne(r)$.
3. $insert(c, \{n\}, \mathtt{Home})$.
4. End.

Operation A.5 Update history.
updateHistory(**Input** $v : \mathcal{O},\ c : \mathcal{CXT}$).

/* This operation creates a link object (named "derived-from") from the version named $Current$ w.r.t. context c to the version v. Then moves the name $Current$ from the version currently named $Current$ to version v. */

1. $ccxt\_old = \mathsf{CC}$.
2. $SCC(c)$.
3. $curr = lookupOne(Current)$.
4. Create a link object l from object $curr$ to object v.
5. Insert the pair $l : \{derived\text{-}from\}$ in $lex(c)$.
6. $deleteName(curr, Current, c)$.
7. $addName(v, Current, c)$.
8. $SCC(ccxt\_old)$.
9. End.

The operation $copy(Input\ o : \mathcal{O};\ Output\ o' : \mathcal{O})$ calls $copyCxt(o, o')$ in the case that o is a context, or copies the simple object o to a new one o'.

References

[1] Serge Abiteboul. Querying semi-structured data. In Foto N. Afrati and Phokion Kolaitis, editors, *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, volume 1186 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 1997.

[2] Serge Abiteboul and Anthony Bonner. Objects and Views. In *Proceedings of ACM-SIGMOD Conference*, pages 238–247, February 1991.

[3] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[4] Serge Abiteboul and Jan Van Den Bussche. Deep Equality Revisited. In Tok Wang Ling, Alberto O. Mendelzon, and Laurent Vieille, editors, *Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases (DOOD'95)*, volume 1013 of *Lecture Notes in Computer Science*, pages 213–228, Singapore, December 1995. Springer.

[5] Antonio Albano, R. Bergamini, Giorgio Ghelli, and Renzo Orsini. An Object Data Model with Roles. In *Proceedings of the International Conference on Very Large Data Bases - VLDB'93*, pages 39–51, Dublin, Ireland, 1993.

[6] Giuseppe Attardi and Maria Simi. A Formalization of Viewpoints. *Fundamenta Informaticae*, 23(2-4):149–174, 1995. Also Technical Report, International Computer Science Institute, Berkeley, TR-93-062, October, 1993.

[7] François Bancilhon and Nicolas Spyratos. Update Semantics of Relational Views. *ACM Transactions on Database Systems*, 6(4):557–575, December 1981.

[8] Jon Barwise and Jerry Seligman. The Rights and Wrongs of Natural Regularity. In J. Tomberlin, editor, *Philosophical Perspectives*, volume 8, pages 331–365. Ridgeview, California, 1994.

[9] Carlo Batini and Maurizio Lenserini. A Methodology for Data Schema Integration in the Entity Relationship Model. *IEEE Transactions on Software Engineering*, SE-10(6):650–663, November 1984.

[10] Philip A. Bernstein and Umeshwar Dayal. An Overview of Repository Technology. In *Proceedings of VLDB'94*, pages 705–713, Santiago, Chile, 1994.

[11] Hemant K. Bhargava, Steven O. Kimbrough, and Ramayya Krishnan. Unique Names Violations, a Problem for Model Integration or You Say Tomato, I Say Tomahto. *ORSA Journal on Computing*, 3(2):107–120, Spring 1991.

[12] Ravi Boppana, Johan Hastad, and Stathis Zachos. Does co-NP Have Short Interactive Proofs? *Information Processing Letters*, 25:127–132, May 1987.

[13] Michael Brodie, John Mylopoulos, and Joachim W. Schmidt. *On Conceptual Modeling*. Springer Verlag, New York, NY, 1984.

[14] Harry Bunt. Context and Dialogue Control. *THINK Quarterly*, 3(1):19–31, 1994.

[15] Saša Buvač and Richard Fikes. A Declarative Formalization of Knowledge Translation. In *Proc. of the ACM CIKMK: The 4th Int. Conf. on Information and Knowledge management*, 1995.

[16] Saša Buvač and John McCarthy. Combining Planning Contexts. In Austin Tate, editor, *Advanced Planning Technology - Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*, pages 84–88. AAAI Press, 1996.

[17] Brad Campbell and Joseph M. Goodman. HAM: A General Purpose Hypertext Abstruct Machine. *Communications of the ACM*, 31(7):856–861, July 1988.

[18] L.J. Campbell, T.A Halping, and H.A. Proper. Conceptual Schemas with Abstractions: Making Flat Conceptual Schemas More Comprehensible. *Data & Knowledge Engineering*, 20(1):39–85, June 1996.

[19] Serge Abiteboul Cassio Souza dos Santos and Claude Delobel. Virtual Schemas and Bases. In Janis Bubenko Matthias Jarke and Keith Jeffery, editors, *Advances in Database Technology - EDBT'94. Proceedings of the 4th International Conference on Extending Database Technology*, pages 81–94. Springer,, 1994.

[20] Wojciech Cellary and Genevieve Jomier. Consistency of Versions in Object-Oriented Databases. In *Proceedings of the 16th International Conference on Very Large Data Bases - VLDB'90*, pages 432–441, Brisbane, Australia, 1990.

[21] Hong-Tai Chou and Won Kim. A Unified Framework for Version Control in a CAD Environment. In *Proceedings of the 12th International Conference on Very Large Data Bases - VLDB'86*, pages 275–281, Kyoto, August 1986.

[22] Vassilis Christophides, Martin Dörr, and Irini Fundulaki. A Semantic Network Approach to Semi-Structured Documents Repositories. In *To appear in Proceedings of the First European Conference on Research and Advanced Technology for Digital Libraries*, Lecture Notes in Computer Science, Pisa, Italy, September 1997. Springer-Verlag.

[23] Panos Constantopoulos and Yannis Tzitzikas. Context-Driven Information Base Update. In *Proceedings of CAiSE'96*, Lecture Notes in CS 1080, pages 319–344, Heraklion, Crete, Greece, May 1996. Springer.

[24] George P. Copeland and Satrag N. Khoshafian. Object Identity. In *OOPSLA'86 Proceedings*, pages 406–416. ACM, September 1986.

[25] Peter N. Creasy and H.A. Proper. A Generic Model for 3-Dimensional Conceptual Modelling. *Data & Knowledge Engineering*, 20(2):119–161, October 1996.

[26] Bogdan Czejdo and David Embley. View Specification and Manipulation for a Semantic Data Model. *Information Systems*, 16(6):585–612, 1991.

[27] Norman Delisle and Mayer Schwartz. Contexts: A Partitioning Concept for Hypertext. *ACM Transactions on Office Information Systems*, 5(2):168–186, April 1987.

[28] Steve Easterbrook and Bashar Nuseibeh. Using ViewPoints for Inconsistency Management. *Software Engineering Journal*, 11(1):31–43, January 1996.

[29] David Embley. *Object Database Development: Concepts and Principles*. Addison-Wesley, 1998.

[30] Adam Farquhar, Angela Dappert, Richard Fikes, and Wanda Pratt. Integrating Information Sources Using Context Logic. In *Proc. of AAAI Spring Symposium on Information Gathering from Distributed Heterogeneous Environments*, 1995. Also Technical Report, Knowledge Systems Laboratory, KSL-95-12, January 1995.

[31] P. Feldman and D. Miller. Entity Model Clustering: Structuring a Data Model by Abstraction. *The Computer Journal*, 29(4):348–360, April 1986.

[32] Anthony Finkelstein, Dov Gabbay, Anthony Hunter, and Bashar Nuseibeh. Inconsistency Handling in Multi-Perspective Specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, August 1994.

[33] Anthony Finkelstein and Ian Sommerville. The Viewpoints FAQ. *Software Engineering Journal*, 11(1):2–4, January 1996.

[34] International Organization for Standardization. ISO 2788-1986. Guidelines for the Establishment and Development of Monolingual Thesauri, 1986.

[35] Scott Fortin. The Graph Isomorphism Problem. Technical Report 96-20, University of Alberta, Edomonton, Alberta, Canada, 1996.

[36] Peter Freeman and Julio Cesar Leite. Requirements Validation Through Viewpoints Resolution. *IEEE Transactions on Software Engineering*, 17(12):1253–1269, December 1991.

[37] Dov Gabbay and Anthony Hunter. Making Inconsistency Respectable: Part 1 — A logical framework for inconsistency in reasoning. In Ph. Jorrand and J. Kelemen, editors, *Foundations of Artificial Intelligence Research*, volume 535 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 1991.

[38] Dov Gabbay and Anthony Hunter. Making Inconsistency Respectable: Part 2 — Meta-level handling of inconsistency. In M. Clarke, R. Kruse, and S. Moral, editors, *Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU'93)*, volume 747 of *Lecture Notes in Computer Science*, pages 129–136. Springer, 1993.

[39] Munish Gandhi, Edward L. Robertson, and Dirk Van Gucht. Levelled Entity Relationship Model. In *Proceedings of 13th International Conference on the Entity-Relationship Approach - ER'94*, pages 420–436, Manchester, U.K., December 1994. Springer-Verlag.

[40] Pankaj K. Garg. Abstraction Mechanisms in Hypertext. *Communications of the ACM*, 31(7):862–870, July 1988.

[41] Fausto Giunchiglia. Contextual Reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, XVI:345–364, 1993.

[42] Eric J. Glover, Steve Lawrence, Michael D. Gordon, William P. Birmingham, and C. Lee Giles. Web Search – Your Way. *Communications of the ACM*, 1999. accepted for publication.

[43] Cheng Goh, Stuart Madnick, and Michael Siegel. Context Interchange: Overcoming the Challenges of Large-Scale Interoperable Database Systems in a Dynamic Environment. In *Proc. of the 3rd Int. Conference on Information and Knowledge Management (CIKM-94)*, pages 337–346, Gaithersburge, Maryland, 1994.

[44] Georg Gottlob, Paolo Paolini, and Roberto Zicari. Properties and Update Semantics of Consistent Views. *ACM Transactions on Database Systems*, 13(4):486–524, December 1988.

[45] Georg Gottlob, Michael Schrefl, and Brigitte Röck. Extending Object - Oriented Systems with Roles. *ACM Transactions on Information Systems*, 14(3):268–296, July 1996.

[46] Ramanathan V. Guha. *Contexts: A Formalization and Some Applications*. PhD thesis, Stanford University, 1991. Also published as Technical Report STAN-CS-91-1399-Thesis, and MCC Technical Report Number ACT-CYC-423-91.

[47] Miclael Hammer and Dennis McLeod. Database Description with Semantic Data Model: A Semantic Database Model. In Alfonso F.Carderas and Dennis McLeod, editors, *Research Foundation in Object-Oriented and Semantic Database Systems*, pages 34–69. Prentice-Hall, 1990.

[48] Bernard M. Hauzeur. A Model for Naming, Addressing, and Routing. *ACM Transactions on Office Information Systems*, 4(4):293–311, 1986.

[49] Stephn J. Hegner. Unique complements and decompositions of database schemata. *Journal of Computer and System Sciences*, 48(1):9–57, February 1994.

[50] Garry Hendrix. Expanded the Utility of Semantic Networks through Partitioning. In *Advance Papers of the IJCAI'75, Int. Joint Conf. on AI*, pages 115–121, 1975.

[51] Garry Hendrix. Encoding Knowledge in Partitioned Networks. In Nicolas Findler, editor, *Associative Networks*. New York: Academic Press, 1979.

[52] Richard Hull and Roger King. Semantic Database Modeling. *ACM Computing Surveys*, 19(3):202–260, September 1987.

[53] Lucja Iwanska. Reasoning with Intentional Native Adjectivals: Semantics, Pragmatics and Context. *Computational Intelligence*, 13(3):348–390, 1997.

[54] Matthias Jarke, Stefan Eherer, Rainer Gallersdorfer, Manfred A. Jeusfeld, and Martin Staudt. ConceptBase - A Deductive Object Base Manager. *Journal of Intelligent Information Systems (JIIS)*, 4(2):167–192, 1995.

[55] Vipul Kashyap and Amit Sheth. Semantic and Schematic Similarities between Database Objects: A Context-Based Approach. *VLDB Journal*, 5(4):276–304, December 1996.

[56] Randy Katz. Towards a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys*, 22(4):375–408, December 1990.

[57] Barbara Katzenberg and Peter Piela. Work Language Analysis and the Naming Problem. *Communications of the ACM*, 36(4):86–95, June 1993.

[58] Nikos Kazantzakis. *The Greek Passion*. Simon and Schuster, New York, 1953. Translated by Jonathan Griffin.

[59] Michael Kifer, Won Kim, and Yehoshua Sagiv. Querying Object-Oriented Databases. In Michael Stonebraker, editor, *Proceedings of ACM-SIGMOD Conference*, pages 393–402, San Diego, Clifornia, June 1992.

[60] Johannes Köbler, Uwe Schöning, and Jacobo Torán. Graph Isomorphism is Low for PP. *Journal of Computational Complexity*, 2:301–330, 1992.

[61] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser Boston Inc., Boston, MA,, 1993.

[62] Alfred Kobsa. Utilizing Knowledge: The Knowledge Representation Workbench. In John Sowa, editor, *Principles of Semantic Networks*. Morgan Kaufmann, 1979.

[63] Gerald Kotonya and Ian Sommerville. Requirements Engineering with Viewpoints. *Software Engineering Journal*, 11(1):5–18, January 1996.

[64] Manolis Koubarakis, John Mylopoulos, Martin Stanley, and Alexander Borgida. Telos: Features and Formalization. Technical Report 18, Institute of Computer Science Foundation for Research and Technology - Hellas, 1989.

[65] Bent Bruun Kristensen. Complex Associations: Abstractions in Object-Oriented Modeling. In *Proceedings of of Object-Oriented Programming, Systems, Languages and Applications - OOPSLA*, volume 29:10 of *ACM Sigplan Notices*, pages 272–283, 1994.

[66] Ronald W. Langacker. *Foundations of Cognitive Grammars*. Stanford University Press, 1987.

[67] Steve Lawrence. Context in Web Search. *IEEE Data Engineering Bulletin*, 23(3):25–32, 2000.

[68] David Lewis. General Semantics. In Donald Davidson and Gilbert Herman, editors, *Semantics of Natural Language*. Reidel, Dordrecht, 1972.

[69] John Lyons. *Semantics*, volume 2. Cambridge University Press, 1977.

[70] Rudolf Mathon. A Note on the Graph Isomorphism Counting Problem. *Information Processing Letters*, 8:131–132, March 1979.

[71] Stan Matwin and Miroslav Kubat. The role of Context in Concept Learning. In *Proceedings of the ICML-96, Workshop on Learning in Context-Sensitive Domains*, pages 1–5, Bari, Italy, July 1996.

[72] John McCarthy. Generality in Artificial Intelligence. *Communications of the ACM*, 30(12):1030–1035, December 1987.

[73] John McCarthy. Notes on Formalizing Context. In *Proc. IJCAI-93*, pages 555–560, Chambery, France, 1993.

[74] John McCarthy and Saša Buvač. Formalizing Context (Expanded Notes), 1994. Technical Note STAN-CS-TN-94-13, Stanford University, Available from http://sail.stanford.edu.

[75] John McCarthy and Pat Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1969.

[76] Ryszard Michalski. How to Learn Impressive Concepts: A Method Employing a Two-Tiered Knowledge Representation for Learning. In *Proceedings of the 4th International Workshope in Machine Learning*, pages 50–58, Irvine, CA, 1987.

[77] Ryszard Michalski. Beyond Prototypes and Frames: The Two-Tiered Concept Representation. In Iven Mechelen, James Hampton, Ryszard Michalski, and Peter Theuns, editors, *Categories and Concepts: Theoretical Views and Inductive Data Analysis*, pages 145–172. Academic Press, 1993.

[78] Gary L. Miller. Graph Isomorphism, General Remarks. *Journal of Computer and System Sciences*, 18(2):128–142, April 1979.

[79] Gail A. Mitchell. *Extensible Query Processing in an Object-Oriented Database*. PhD thesis, Brown University, May 1993. Technical Report CS TR 93-16.

[80] Renate Motschnig-Pitrik. An Integrated View on the Viewing Abstraction: Contexts and Perspectives in Software Develepment, AI, and Databases. *Journal of Systems Integration*, 5(1):23–60, April 1995.

[81] Renate Motschnig-Pitrik. Requirements and Comparison of View Mechanisms for Object-Oriented Databases. *Information Systems*, 21(3):229–252, 1996.

[82] John Mylopoulos. Information Modeling in the Time of the Revolution. *Information Systems*, 23(3/4):127–155, 1998.

[83] John Mylopoulos, Alexander Borgida, Matthias Jarke, and Manolis Koubarakis. Telos: Representing Knowledge about Information Systems. *ACM Transactions on Information Systems*, 8(4), October 1990. University of Toronto.

[84] John Mylopoulos and Renate Motschnig-Pitrik. Partitioning Information Bases with Contexts. In *Proceedings of CoopIS'95*, pages 44–55, Vienna, Austria, 1995.

[85] John Mylopoulos and Renate Motschnig-Pitrik. Semantics, Features, and Applications of the Viewpoint Abstraction. In *Proceedings of CAiSE'96*, pages 514–539, Heraklion, Greece, May 1996.

[86] Aris Ouksel and Channah Naiman. Coordinating Context Building in Heterogeneous Information Systems. *Journal of Intelligent Information Systems (JIIS)*, 3(2):151–183, 1994.

[87] Michael Papazoglou and Gunter Schlageter, editors. *Cooperative Information Systems*. Academic Press, 1998.

[88] Mike P. Papazoglou and Jeroen Hoppenbrouwers. Contextualizing the Information Space in Federated Digital Libraries. *SIGMOD Record*, 28(1):40–46, 1999.

[89] Barbara Pernici. Objects with Roles. In *Proceedings of the ACM Conference on Office Information Systems*, pages 205–215, New York, 1990.

[90] Grand E. Potter and Robert P. Trueblood. Traditional, Semantic, and Hyper-Semantic Approaches to Data Modeling. *IEEE Computer*, 21(6):53–63, June 1988.

[91] Vassilis Prevelakis and Dennis Tsichritzis. Perspectives on software development environments. In *Proceedings of CAiSE'93*, Lecture Notes in CS 685, pages 586–600, Paris, France, June 1993. Springer-Verlag.

[92] Joel Richardson and Peter Schwarz. Aspects: Extending Objects to Support Multiple, Independent Roles. In *Proceedings of ACM-SIGMOD Conference*, pages 298–307, Denver, Colorado, May 1991.

[93] Don D. Roberts. *The Existential Graphs of Charles S. Peirce*. The Hague, Mouton, 1973.

[94] Elke Rundensteiner. MultiViews: A Methodology for Supporting Multiple Views in Object-Oriented Databases. In *Proceedings of the 18th International Conference on Very Large Database (VLDB)*, pages 187–198, Vancouver, British Columbia, Canada, August 1992.

[95] H.J.J.H. Schepers, O.B.P. Rikkert de Koe, G.M.G. Havermans, and D.K. Hammer. Naming, addressing, routing and relaying in the OSI environment. *Computer Networks and ISDN Systems*, 23:241–251, 1992.

[96] Ulrich Schiel, Antonio L. Furtado, Erich J. Neuhold, and Marco A. Casanova. Towards Multi-Level and Modular Conceptual Schema Specifications. *Information Systems*, 9(1):43–57, 1984.

[97] P.F. Schneider. Contexts in PSN. In *Proceedings of the AI-CSCSI-CSEIO Conference*, pages 71–78, Victoria, B.C., May 1980.

[98] Marc H. Scholl, Cristian Laasch, and Markus Tresch. Updatable Views in Object-Oriented Databases. In *Proceedings of the 2nd Int. Conf. on Deductive and Object-Oriented Database Systems*, pages 189–207, Munich, December 1991.

[99] Edward Sciore. Object Specialization. *ACM Transactions on Information Systems*, 7(2):103–122, April 1989.

[100] Edward Sciore, Michael Siegel, and Arnon Rosenthal. Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. *ACM Transactions on Database Systems*, 19(2):254–290, 1994.

[101] Edward Sciore, Michael Siegel, and Arnon Rosenthal. Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. *ACM Transactions on Database Systems*, 19(2):254–290, 1994.

[102] Jerry Seligman. The Structure of Regularity, 1993. Draft Lecture Notes, Indiana University Logic Group, Bloomington, IN.

[103] Nirad Sharma. On Formalizing and Reasoning with Contexts. Technical Report 352, Department of Computer Science, The University of Queensland, September 1995.

[104] Gail M. Shaw and Stanley B. Zdonik. A Query Algebra for Object-Oriented Databases. Technical Report CS-89-19, Brown University, 1989.

[105] Mary Shaw. The Impact of Modeling and Abstraction Concerns on Moderm Programming Languages, 1984. In [13].

[106] John J. Shilling and Peter F. Sweeney. Three Steps to Views: Extending the Object-Oriented Paradigm. In *Proceedings of of Object-Oriented Programming, Systems, Languages and Applications - OOPSLA*, pages 353–361, October 1989.

[107] Yoav Shoham. Varieties of Context. In V. Lifschitz, editor, *AI and Mathematical Theory of Computation — Papers in Honor of John McCarthy*, pages 393–407. Academic Press, 1991.

[108] Yuh-Ming Shyy and Stanley Y.W. Su. K: A High-level Knowledge Base Programming Language for Advanced Dadabase Applications. In *Proceedings of ACM-SIGMOD Conference*, pages 338–347, Denver, Colorado, May 1991.

[109] John Miles Smith and Diane C.P. Smith. Database Abstractions: Aggregation and Generalization. In John Mylopoulos and Michael Brodie, editors, *Reading in Artificial Intelligence & Databases*, pages 138–152. Morgan Kaufmann, 1989.

[110] Luiz Fernando G. Soares, Noemi L. R. Rodriguez, and Marco A. Casanova. Nested Composite Nodes and Version Control in an Open Hypermedia System. *Information Systems*, 20(6):501–519, 1995.

[111] John F. Sowa. *Conceptual Structures.* Reading Mas: Addison Wesley, 1984.

[112] John F. Sowa. Syntax, Semantics, and Pragmatics of Context. In *Proc. 3rd Int. Conf. on Conceptual Structures, ICCS'95*, Lecture Notes in AI 954, pages 1–15, Santa Cruz, CA, USA, August 1995. Springer.

[113] Lynn Andrea Stein and Stanley B. Zdonik. Clovers: The Dynamic Behavior of Types and Instances. Technical Report CS-89-42, Brown University, 1989.

[114] Veda C. Storey. Understanding Semantic Relationships. *VLDB Journal*, 2(4):455–488, October 1993.

[115] Elaine Svenonius. Design of Controlled Vocabularies. In *The Encyclopedia of Library and Information Science*, volume K45, pages 82–109. Marcel Dekker, New York, 1990.

[116] Toby J. Teorey, Guangping Wei, Deborah L. Bolton, and John A. Koenig. ER Model Clustering as an Aid for User Communication and Documentation in Database Design. *Communications of the ACM*, 32(8):975–987, August 1989.

[117] Manos Theodorakis. Name Scope in Semantic Data Models. Master's thesis, Department of Computer Science, University of Crete, Greece, October 1995. (In Greek). Also Technical Report no. 141 [118].

[118] Manos Theodorakis. Name Scope in Semantic Data Models. Technical Report 141, Institute of Computer Science Foundation for Research and Technology - Hellas (ISC-FORTH), October 1995.

[119] Manos Theodorakis. Context-based Naming in Semantic Networks. In *Proceedings of the 3rd Doctoral Consortium on Advanced Information Systems Engineering, CAiSE'96*, pages 53–56, May 1996.

[120] Manos Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. A Theory of Contexts in Information Bases. Technical Report 216, Institute of Computer Science Foundation for Research and Technology - Hellas (ISC-FORTH), March 1998.

[121] Manos Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. Context in Information Bases. In *Proceedings of the 3rd International Conference on Cooperative Information Systems (CoopIS'98)*, pages 260–270, New York City, NY, USA, August 1998. IEEE Computer Society.

[122] Manos Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. Contextualization as an Abstraction Mechanism for Conceptual Modeling. In *Proceedings of the 18th International Conference on Conceptual Modeling (ER'99)*, pages 475–489, Paris, France, November 1999.

[123] Manos Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. Contextualization as an Abstraction Mechanism in Conceptual Modeling. Technical Report 255, Institute of Computer Science Foundation for Research and Technology - Hellas (ISC-FORTH), March 1999.

[124] Manos Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. Querying Contextualized Information Bases. In *Proceedings of the 24th International Workshop of Information and Communication Technologies and Programming (ICTP'99)*, Plovdiv, Bulgaria, June 1999.

[125] Manos Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. A Theory of Context in Information Bases. To be published in Information Systems Journal, 2001.

[126] Manos Theodorakis and Panos Constantopoulos. Context-Based Naming in Information Bases. *International Journal of Cooperative Information Systems*, 6(3 & 4):269–292, 1997.

[127] Manos Theodorakis and Panos Constantopoulos. On Context-Based Naming in Information Bases. In *Proceedings of the 2nd International Conference on Cooperative Information Systems (CoopIS'97)*, pages 140–149, Charleston, South Carolina, USA, June 1997. IEEE Computer Society.

[128] Olga De Troyer and René Janssen. On Modularity for Conceptual Data Models and the Consequences for Subtyping, Inheritance & Overriding. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 678–685, Vienna, Austria, April 1993. IEEE Computer Society.

[129] Jeremy Tucker. Naming and Addressing issues - OSI Network Addresses. *Computer Networks and ISDN Systems*, 13:149–153, 1992.

[130] Peter Turney. Robust Classification with Context-Sensitive Features. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-93*, pages 268–276, Edinburgh, 1993. Scotland: Gordon and Breach.

[131] Yannis Tzitzikas. View updates in knowledge bases. Master's thesis, Dept. of Computer Science, University of Crete, Greece, October 1995. Technical Report no. 142.

[132] Scott L. Vandenberg. Algebras for Object-Oriented Query Languages. Technical Report CS-TR-1993-1161, University of Wisconsin-Madison, 1993.

[133] Dirk Vermeir. Semantic Hierarchies and Abstractions in Conceptual Schemata. *Information Systems*, 8(2):117–124, 1983.

[134] Yair Wand and Ron Weber. An Ontological Model of an Information System. *IEEE Transactions on Software Engineering*, 16(11):1282–1292, November 1990.

[135] Herbert Weber. Modularity in Data Base System Design: A Software Engineering View of Data Base Systems. *VLDB Surveys*, pages 65–91, 1978.

[136] Roel Wieringa and Wiebren de Jonge. The Identification of Objects and Roles. Technical Report IR-267, Faculty of Mathematics and Computer Science, Vrije University, Amsterdam, March 1991.