Contextualization as an Abstraction Mechanism for Conceptual Modeling

Manos Theodorakis[1,2], Anastasia Analyti[1], Panos Constantopoulos[1,2], Nicolas Spyratos[3]

[1] *Institute of Computer Science, FORTH, P.O.Box 1385, GR 711 10 Heraklion, Crete, Greece*

[2] *Department of Computer Science, University of Crete, Heraklion, Greece*

[3] *Universite de Paris-Sud, LRI-Bat 490, 91405 Orsay Cedex, France*

E-mail: {*etheodor* | *analyti* | *panos*}*@ics.forth.gr, spyratos@lri.fr*

Abstract

The notion of context appears in several disciplines, including computer science, under various forms. In this paper, we are concerned with a notion of context in the area of conceptual modeling. First, we present a simple definition whereby a context is seen as a set of objects, within which each object has a set of names and possibly a reference: the reference of the object is another context which "hides" detailed information about the object. Then, we enhance our simple 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. Finally, we study the interactions between contextualization and the traditional abstraction mechanisms as well as the constraints that govern such interactions.

1 Introduction

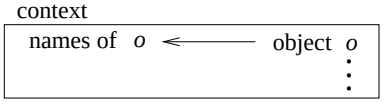
The notion of context is of fundamental importance in cognitive psychology, linguistics, and computer science. In computer science, a number of formal or informal definitions of some notion of context have appeared in several areas, such as artificial intelligence [15, 24, 13], software development [31, 12, 34, 35, 36, 19, 20], databases [3, 11, 14, 1, 7, 18, 30], machine learning [25, 44, 23], and knowledge representation [28, 42, 40, 46, 8, 38, 6, 4]. See also [26] 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 [3, 11, 14, 33, 1], aspects [31], and roles [12, 34], for dealing with data from different perspectives, or even in the form of workspaces which are used to support cooperative work [19]. In machine learning, context is treated as environmental information for concept classification [25, 44, 23]. In so called "multi bases", context appears as a collection of meta-attributes for capturing class semantics [18]. In artificial intelligence the notion of context appears as a means of partitioning knowledge into manageable sets [16], or as a logical construct that facilitates reasoning activities [24, 13]. 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 [28, 41, 42], or for dividing the global schema of a database into clusters in order to deal with schema complexity [46, 8, 38, 6, 4].

Our objective in this paper is to establish a formal notion of context to support the development and effective use of large information bases in various application areas.

A *context* in an information bases can be seen as a higher order conceptual entity that groups together other conceptual entities from a particular standpoint. Contexts allow one to focus on the objects of interest, as well as to name each of these objects using one or more convenient names [29, 41].

Roughly speaking, each object of a context is associated with a set of names as 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.

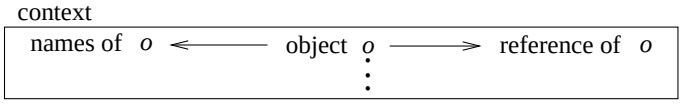
It is important to note that the same object can belong to different contexts with different names in each of them. Thus, to the extent that names convey meaning, this is one way of providing context-dependent interpretation of objects. As we shall see, there are more ways in which the notion of context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

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

1. 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.
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.

Our notion of context 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.
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 paper, we only 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 do not consider methodological issues such as criteria for context formation.

The remainder of the paper is organized as follows: In section 2, we define the notion of context *without* structuring of its objects, and we discuss some of its modeling capabilities. In section 3, we enhance the notion of context by *adding* structure among the objects through the traditional abstraction mechanisms. In section 4, we discuss context-based information bases, i.e. systems that support contextualization in addition to the traditional abstraction mechanisms. In section 5, we study the ways in which contexts interact with each other in the presence of the traditional abstraction mechanisms. In section 6, we compare our framework with those of related work. Finally, in section 7, we make some concluding remarks and suggestions for further research.

[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 [17].

2 The notion of context

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 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 1.

Suppose now that we want to talk about Greek geography 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:

$$\text{Islands : o} \cdots\triangleright\ \text{c}_1$$

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 1. Then, Greek geography can be represented by an object identifier o'' and by associating it with context c, as shown in Figure 1.

Figure 1: Bottom-up modeling

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. Formally, we have the following definition.

Definition 2.1 Context. A context c is defined as a set of objects, denoted by $objs(c)$, such that each object o is associated with

1. a set of names, called *the names of o in c*, denoted by $names(o, c)$;
2. zero or one context, called *the reference of o in c*, denoted by $ref(o, c)$.

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*.

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

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 left to right in Figure 1). Clearly, we could have followed the opposite construction, i.e. a top-down approach ("moving" from right to left in Figure 1). In fact, as we shall see in other examples, we can also follow a mixed approach.

This flexibility is important in conceptual modeling and implies (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).

Continuing with our examples, 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 2.

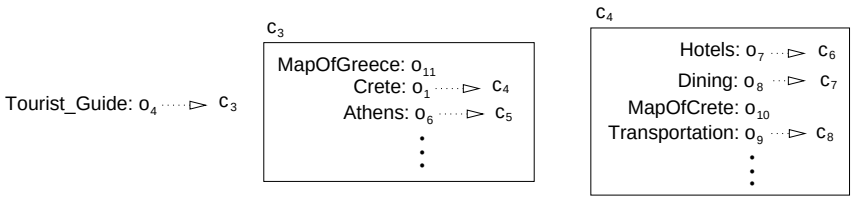


Figure 2: Top-down modeling

Continuing our top-down design, we now have to define the contexts c_4, c_5, and so on. Context c_4 is shown in Figure 2, whereas the remaining contexts are not shown. 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 context c_3 shares the object o_1 with context c_1 that we have seen earlier (see Figure 1). Also note that, in context c_3, object o_1 has a reference (context c_4), whereas in context c_1, the same object o_1 has no reference.

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 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.

Several remarks are in order here, concerning our definition of context:

- An object can belong to one or more different contexts. This feature is useful when we want to view an object under different perspectives.
- 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.
- Two different objects can have the same name within a context. As a result, ambiguities may occur when one tries to designate an object of a context using a name shared with a different object of the same context. We address this problem in [41, 42].
- The same object can have different references within different contexts. In other words, references are context-dependent.
- 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.
- 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).

Roughly speaking, the modeling power of contexts lies in 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 provides 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.

3 Structuring the contents of a context

Our definition of context allows for some simple relations between objects, for example, two different objects of a context can have the same name and/or the same reference. However, we would like to allow the objects of a context to be related in more complex ways.

For the purposes of this paper, we shall assume that the objects of a context can be structured as in a Telos information base [27, 21].

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 [27] objects are also 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 [17, 37, 27, 21].

The classification dimension calls for each object to be an *instance of* one or more *classes*. Classes are themselves objects, and 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 $\texttt{L}_0\texttt{\_Class}$, all simple classes under the class $\texttt{L}_1\texttt{\_Class}$, all metaclasses under the class $\texttt{L}_2\texttt{\_Class}$, and so on, where $\texttt{L}_0$, $\texttt{L}_1$, $\texttt{L}_2$, etc. are seen as abstraction *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 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 class, 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.

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

Returning now to our notion of context, from now on we shall use the following "enhanced" definition: A context consists of a set of objects, each object having a set of names and zero or one reference (as before), with the following *new* features:

- each object is either an individual or an attribute;
- each object can be related to other objects through instance-of or ISA links[3];

[3]Every object of a context is assumed to belong to one and only one level of the classification 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 instance-of or ISA link can have zero or one reference.

As a consequence of this enhanced definition of context, each context is assumed to be equipped with

- a predicate for defining the objects that are attribute links (the remaining objects being individuals),
- a predicate for defining instance-of links, and
- a predicate for defining ISA links.

More precisely, we assume each context to be equipped with three predicates as follows:

1. $attr(obj, from, to)$, declaring that object obj is an attribute link with source object $from$ and destination object to.
2. $in(from, to)$, declaring an instance-of link in which the object $from$ is an instance of the object to.
3. $isa(from, to)$, declaring an ISA link in which the class $from$ is a subclass of class to.

Note that, as attributes are objects, an attribute 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 3(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 3(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, being 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 [41, 40][4].

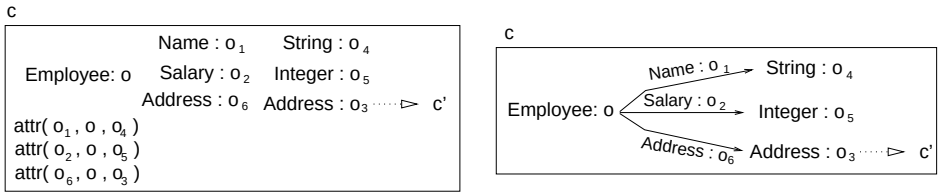


Figure 3: Modeling an employee using attributes

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 modeling an employee is the following: the employee class references a context containing the attribute information. This is shown in Figure 4(a), where the ISA declarations in context c define that `Name` is a subclass of `String` and `Salary` is a subclass of `Integer`. Figure 4(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.

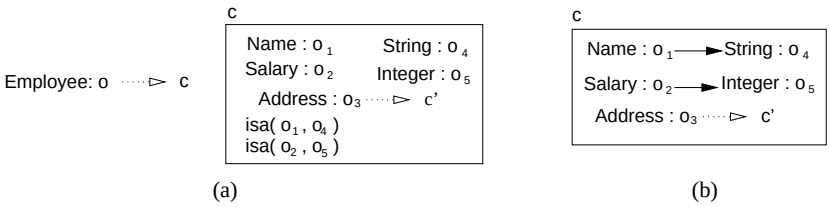


Figure 4: Modeling an employee using contexts

At first glance, the modeling of Figure 4 looks simpler than that of Figure 3. However, depending on the application, the one or the other could be preferred. The important point here is that the contextualization mechanism offers several alternatives for modeling a given application.

[4] 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 5 shows another example of context with structured contents, this time using all three abstraction mechanisms, i.e. classification, generalization, and aggregation.

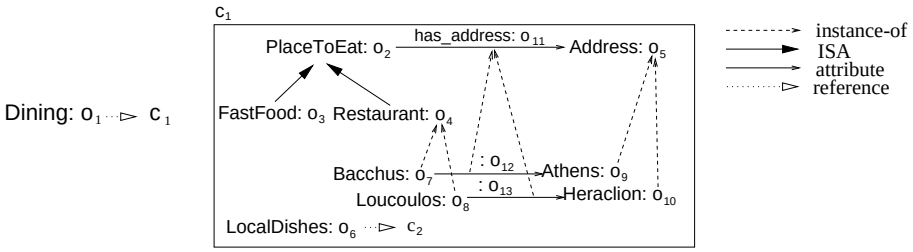


Figure 5: Structured contents of a context

We emphasize once again, that all items in the definition of a context (including the predicate declarations) are context-dependent. 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. 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.

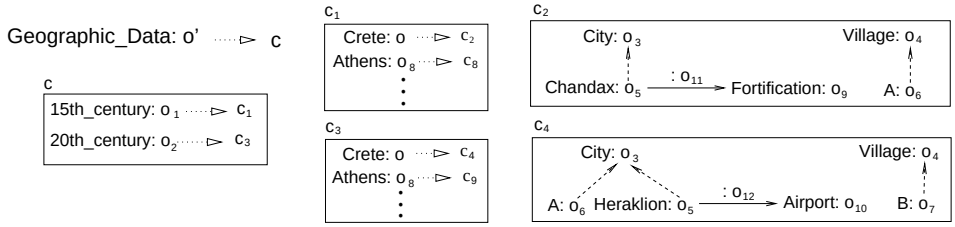


Figure 6: Context-dependent description

Finally, it is important to note that contextualization is orthogonal to the other abstraction mechanisms.

4 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 has many advantages, 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 7(a)): (i) the information base environment, and (ii) a special context, which contains the contents of the information base.

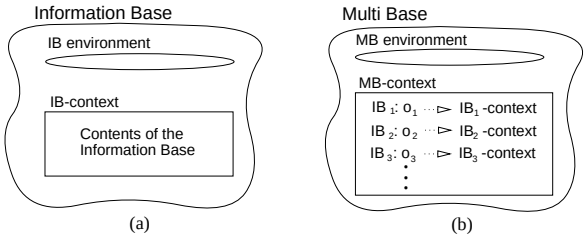


Figure 7: Context-based information bases

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 7(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 7(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.

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 [17, 37] for a survey). In this section, we study the interaction between the traditional abstraction mechanisms and the mechanism of contextualization.

5.1 Attribution and contextualization

In our enhanced definition of context, each object can be either an individual or an attribute, 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[5]. This can be done as shown in Figure 8. Let o be an attribute from object o_1 to object o_2 with references c_1 and c_2, respectively. Then, its 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 8. 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.

Of course, apart from the minimal necessary information shown in Figure 8, 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*

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

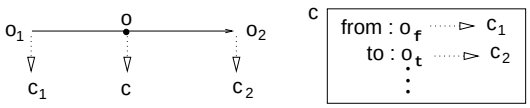


Figure 8: The reference of an attribute

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

Figure 9 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 9(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 9(b).

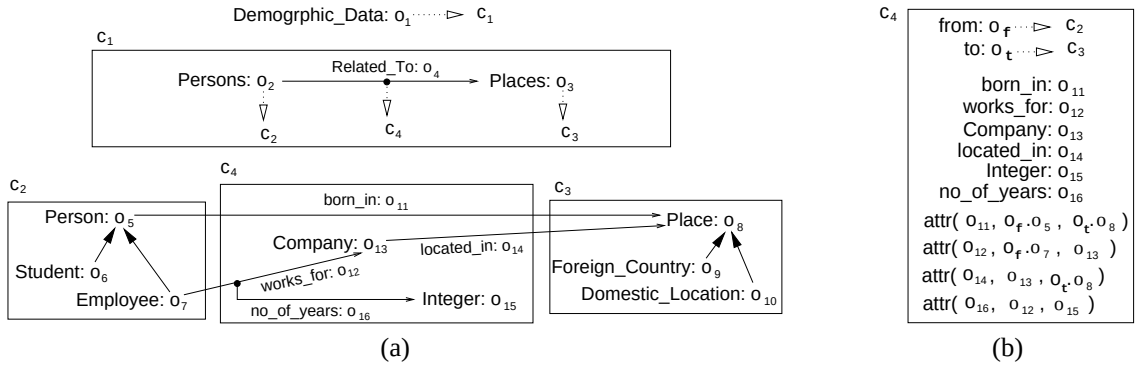


Figure 9: Top-down modeling of demographic data

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 instance-of links from objects in the source reference of l to objects in the destination reference of l.

Constraint 5.2 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 10(a), object o (`Instance`) is instance-of object o' (`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'.

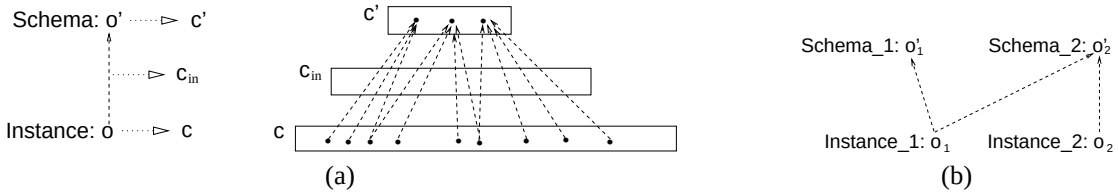


Figure 10: The reference of an instance-of link

Intuitively, within c_{in}, objects of c are instances of objects of c'. For example, if c contains a set of relational tuples and c' contains a relational database schema then the instance-of links relate tuples in c with tables in c'. 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 set of objects to share the same schema, and the same set of objects to be classified under different schemas. For example, in Figure 10(b), 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 11, 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.

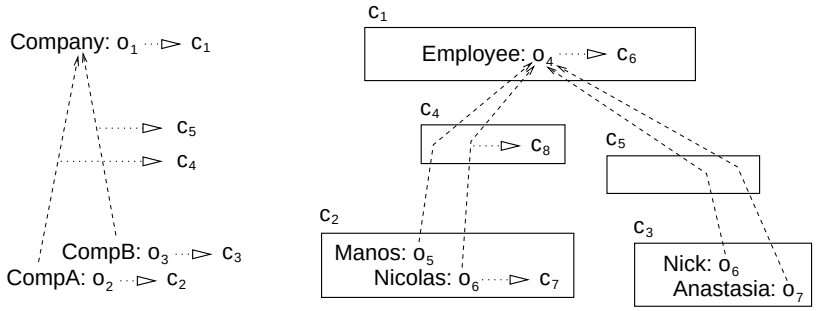


Figure 11: Interaction between classification and contextualization

Continuing with the example of Figure 11, 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.

5.3 Generalization and contextualization

Generalization establish a subclass-superclass relation between classes and it 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 called *reference inheritance* related to the interaction between generalization and contextualization. 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 5.1 Context refinement. We say that context c *refines* context c', or that context c is a refinement of c', iff (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 predicate of c' is also a predicate of c, and (iv) the reference of every object of c' refines the reference of the object in c.

Note that the above definition of context refinement is recursive and that every context is a refinement of itself. We show in [39] 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.[6].

The following constraint expresses the application of reference inheritance on ISA links.

Constraint 5.3 Reference Inheritance Constraint. The source reference of an ISA link refines the destination reference of the link.

For example, in Figure 12, 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.

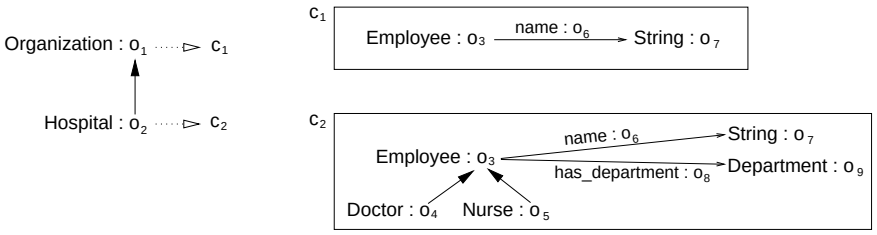


Figure 12: Interaction between generalization and contextualization

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. This, however, is an implementation issue that lies beyond the scope of this paper. A mechanism for eliminating duplications is proposed in [39].

Context refinement can be achieved in stages through the repetitive application of the following operations on the contents of a context: (i) the addition of a new object (possibly an attribute), (ii) the specialization, generalization, or classification of an object (possibly an attribute), (iii) the addition of a name to an object, (iv) the addition of a reference to an object or link, and (v) 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).

We now give a more involved example. 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

[6]Notice the similarity between context equivalence and deep object equality in object oriented databases [2].

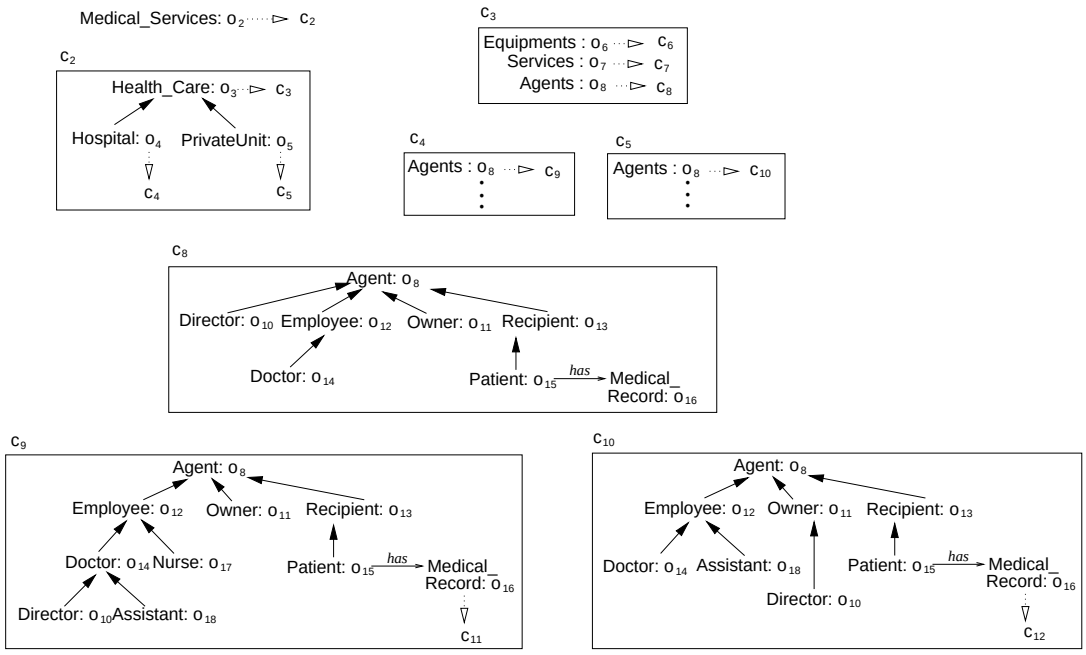


Figure 13: Interaction between generalization and contextualization

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.

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 14. 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.

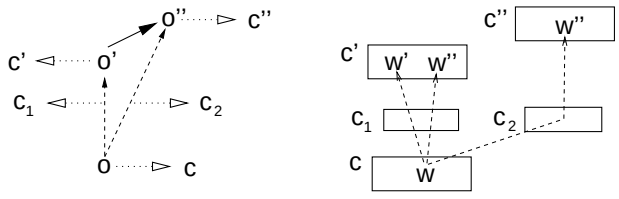


Figure 14: The interaction between classification, generalization, and contextualization

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 5.3), it should hold that if an object w in c is classified into a class w'' in c'' then, w should also be classified into 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 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 5.4 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'.

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 [39].

6 Related work

Various forms of contextualization have appeared in the area of computer science. However, these forms are very diverse and serve different purposes. We choose for comparison with our work the more closely related approaches which we classify in three topics: general contextualization frameworks, semantic model clustering, and nested associations.

6.1 General contextualization frameworks

In this subsection, we compare the present work with our own previous works [41, 42], as well as with the work of Mylopoulos and Motschnig-Pitrik [28, 29]. These works attempt to introduce a general framework for contextualization in information bases.

In [42], a naming mechanism based on the concept of context is proposed, in order to resolve naming problems that arise in information bases 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. 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-overlapping contexts.

In [28, 29, 41], 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. These works support nesting of contexts, context overlapping, relative naming, and define operations on contexts, such as context union, intersection, and difference. In addition, [41] establishes properties of operations on contexts, such as commutativity, associativity and distributivity. On the other hand, [28] considers issues such as authorization and transaction execution.

The notion of context introduced in this work, still supports nesting of contexts, context overlapping, and relative naming, as described in our previous works [40, 41], yet it advances with respect to [28, 29, 41, 42] mainly along two 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.

- We support the interaction of our contextualization mechanism with the traditional abstraction mechanisms. Works in [28, 29, 41, 42] lack this interaction. In [29], 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.

6.2 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* [8, 38, 6, 43, 4, 5, 10, 38, 46, 32, 45].

In [38], several kinds of clustering are defined, all of which are supported by our framework:

1. *Dominance grouping*: Here, 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. For example, in Figure 11, the reference of the object o_8 (`Agents`) in context c_3, contains the object itself (under the name (`Agent`)).
2. *Abstraction grouping*: Here, 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*: Here, 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 [6], 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, with the advantages described in the previous subsection,.
- 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 [6] 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 [6], 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.

6.3 Nested associations

Work in [22] 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 [22], 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 [10], 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 [10], 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.

7 Conclusions

In this paper, we concerned with a notion of context in the area of conceptual modeling. In our approach, a context is seen as a structured set of objects, in which each object is associated with a set of names and (possibly) a reference: 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. One of the contribution of this paper is that we 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.

Future work includes the development of a general framework for querying information bases which support contextualization.

References

[1] S. Abiteboul and A. Bonner. Objects and Views. In *Proc. ACM-SIGMOD Conference*, pages 238–247, Feb. 1991.

[2] S. Abiteboul and J. Van Den Bussche. Deep Equality Revisited. In T. Ling, A. Mendelzon, and L. Vieille, editors, *Proc. of DOOD'95*, volume 1013, pages 213–228, Singapore, Dec. 1995. Springer.

[3] F. Bancilhon and N. Spyratos. Update Semantics of Relational Views. *ACM Trans. Database Syst.*, 6(4):557–575, Dec. 1981.

[4] L. Campbell, T. Halping, and H. Proper. Conceptual Schemas with Abstractions: Making Flat Conceptual Schemas More Comprehensible. *DKE*, 20(1):39–85, June 1996.

[5] P. Creasy and H. Proper. A Generic Model for 3-Dimensional Conceptual Modelling. *DKE*, 20(2):119–161, Oct. 1996.

[6] B. Czejdo and D. Embley. View Specification and Manipulation for a Semantic Data Model. *IS*, 16(6):585–612, 1991.

[7] N. Delisle and M. Schwartz. Contexts: A Partitioning Concept for Hypertext. *ACM Trans. Office Inf. Syst.*, 5(2):168–186, Apr. 1987.

[8] P. Feldman and D. Miller. Entity Model Clustering: Structuring a Data Model by Abstraction. *Computer J.*, 29(4):348–360, Apr. 1986.

[9] N. Findler. *Associative Networks*. New York: Academic Press, 1979.

[10] M. Gandhi, E. Robertson, and D. Gucht. Levelled Entity Relationship Model. In *Proceedings of 13th International Conference on the Entity-Relationship Approach - ER'94*, pages 420–436, Manchester, U.K., Dec. 1994. Springer-Verlag.

[11] G. Gottlob, P. Paolini, and R. Zicari. Properties and update semantics of consistent views. *ACM Transactions on Database Systems*, 13(4):486–524, Dec. 1988.

[12] G. Gottlob, M. Schrefl, and B. Röck. Extending Object - Oriented Systems with Roles. *ACM Trans. Inf. Syst.*, 14(3):268–296, July 1996.

[13] R. 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.

[14] S. J. Hegner. Unique complements and decompositions of database schemata. *Journal of Computer and System Sciences*, 48(1):9–57, Feb. 1994.

[15] G. Hendrix. Encoding Knowledge in Partitioned Networks. In N. Findler, editor, *Associative Networks*. New York: Academic Press, 1979.

[16] G. Hendrix. Encoding Knowledge in Partitioned Networks, 1979. In [9].
[17] R. Hull and R. King. Semantic Database Modeling. *ACM Comput. Surv.*, 19(3):202–260, Sept. 1987.
[18] V. Kashyap and A. Sheth. Semantic and Schematic Similarities between Database Objects: A Context-Based Approach. *VLDB Journal*, 5(4):276–304, Dec. 1996.
[19] R. Katz. Towards a Unified Framework for Version Modeling in Engineering Databases. *ACM Comput. Surv.*, 22(4):375–408, Dec. 1990.
[20] G. Kotonya and I. Sommerville. Requirements Engineering with Viewpoints. *Software Engineering Journal*, pages 5–19, Jan. 1996.
[21] M. Koubarakis, J. Mylopoulos, M. Stanley, and A. Borgida. Telos: Features and Formalization. Technical Report 18, Institute of Computer Science Foundation for Research and Technology - Hellas, 1989.
[22] B. Kristensen. Complex Associations: Abstractions in Object-Oriented Modeling. In *Proc. OO Prog., Syst., Lang. and Appl. - OOPSLA*, volume 29:10 of *ACM Sigplan Notices*, pages 272–283, 1994.
[23] S. Matwin and M. 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.
[24] J. McCarthy. Notes on Formalizing Context. In *Proc. IJCAI-93*, pages 555–560, Chambery, France, 1993.
[25] R. 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.
[26] R. 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, Apr. 1995.
[27] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing Knowledge about Information Systems. *ACM Trans. Inf. Syst.*, 8(4), Oct. 1990. University of Toronto.
[28] J. Mylopoulos and R. Motschnig-Pitrik. Partitioning Information Bases with Contexts. In *Proceedings of CoopIS'95*, pages 44–55, Vienna, Austria, 1995.
[29] J. Mylopoulos and R. Motschnig-Pitrik. Semantics, Features, and Applications of the Viewpoint Abstraction. In *Proceedings of CAiSE'96*, pages 514–539, Heraklion, Greece, May 1996.
[30] A. Ouksel and C. Naiman. Coordinating Context Building in Heterogeneous Information Systems. *J. of Intelligent Inf. Systems*, 3(2):151–183, 1994.
[31] J. Richardson and P. Schwarz. Aspects: Extending Objects to Support Multiple, Independent Roles. In *Proc. ACM-SIGMOD Conference*, pages 298–307, Denver, Colorado, May 1991.
[32] U. Schiel, A. Furtado, E. Neuhold, and M. Casanova. Towards Multi-Level and Modular Conceptual Schema Specifications. *IS*, 9(1):43–57, 1984.
[33] M. Scholl, C. Laasch, and M. 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, Dec. 1991.
[34] E. Sciore. Object Specialization. *ACM Trans. Inf. Syst.*, 7(2):103–122, Apr. 1989.
[35] J. Shilling and P. Sweeney. Three Steps to Views: Extending the Object-Oriented Paradigm. In *Proc. OO Prog., Syst., Lang. and Appl. - OOPSLA*, pages 353–361, Oct. 1989.
[36] Y. Shyy and S. Su. K: A High-level Knowledge Base Programming Language for Advanced Dadabase Applications. In *Proc. ACM-SIGMOD Conference*, pages 338–347, Denver, Colorado, May 1991.
[37] V. Storey. Understanding Semantic Relationships. *VLDB Journal*, 2(4):455–488, Oct. 1993.
[38] T. Teorey, G. Wei, D. Bolton, and J. Koenig. ER Model Clustering as an Aid for User Communication and Documentation in Database Design. *Commun. ACM*, 32(8):975–987, Aug. 1989.
[39] M. Theodorakis. *Contextualization: An Abstraction Mechanism for Information Modeling*. PhD thesis, Department of Computer Science, University of Crete, Greece, 1998.
[40] M. Theodorakis, A. Analyti, P. Constantopoulos, and N. Spyratos. A Theory of Contexts in Information Bases. Technical Report 216, Institute of Computer Science Foundation for Research and Technology - Hellas, Mar. 1998.
[41] M. Theodorakis, A. Analyti, P. Constantopoulos, and N. 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, Aug. 1998. IEEE Computer Society.
[42] M. Theodorakis and P. Constantopoulos. Context-Based Naming in Information Bases. *International Journal of Cooperative Information Systems*, 6(3 & 4):269–292, 1997.
[43] O. Troyer and R. Janssen. On Modularity for Conceptual Data Models and the Consequences for Subtyping, Inheritance & Overriding. In *Proc. of Ninth Int. Conf. on Data Eng.*, pages 678–685, Vienna, Austria, Apr.

1993. IEEE Computer Society.

[44] P. 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.

[45] D. Vermeir. Semantic Hierarchies and Abstractions in Conceptual Schemata. *IS*, 8(2):117–124, 1983.

[46] H. Weber. Modularity in Data Base System Design: A Software Engineering View of Data Base Systems. *VLDB Surveys*, pages 65–91, 1978.