Contexts and Composite Objects

FALQUET, Gilles, LEONARD, Michel Paul, AL-JADIR, Lina

Abstract

Many applications require the capability to define and manipulate a collection of related objects as a single unit. Object-oriented database models and systems usually provide the concept of composite object for that purpose. The concept of context that we propose here provides another, more general, way to view and manipulate collections of objects. We compare the two concepts both in their formal definitions and in their implementation in object-oriented database system.

Reference


Available at:
http://archive-ouverte.unige.ch/unige:46598

Disclaimer: layout of this document may differ from the published version.
Contexts and Composite Objects

Lina Al-Jadir, Gilles Falquet, Michel Léonard

Université de Genève
Centre Universitaire d'Informatique (CUI)
24 rue Général-Dufour, 1211 Genève 4, Switzerland
e-mail: {aljadour, falquet, leonard}@cui.unige.ch

Abstract

Many applications require the capability to define and manipulate a collection of related objects as a single unit. Object-oriented database models and systems usually provide the concept of composite object for that purpose. The concept of context that we propose here provides another, more general, way to view and manipulate collections of objects. We compare the two concepts both in their formal definitions and in their implementation in object-oriented database system.

Keywords

Object-oriented model; complex object; composite object; context.

1 Introduction

Data representation in object-oriented database models is based on the concepts of classification, instantiation, aggregation, and generalization/specialization. Each object is an instance of a class, which can be either atomic or non-atomic. An atomic object (an instance of an atomic class) has a simple value (an integer number, a character, a real number, etc.) while the value of a non-atomic object is determined by the value of its attributes (attributes are sometimes called instance variables). An attribute can be considered as a function that maps instances of a class to (sets of) instances of another class, called the domain class. This function establishes an aggregation relationship between the two classes.

If the value of attribute A on an object o is equal to or contains object o', we say that o references o' through attribute A. If o' is a non-atomic object then o is called a nested object (Kim et al, 1989). Object-oriented data models allow the definition of nested objects of arbitrary depth.

Many applications in such domains as computer-aided design require the capability to define and manipulate a nested object and all the objects it references directly or indirectly as a single unit (Kim et al, 1987). The concept of complex object has been proposed for that purpose. There are several different definitions of complex object (Atkinson et al, 1989; Kim et al, 1989; Nguyen and Rieu, 1989). Due to space limitations, we consider only the definition given in the Orion database model. 1

---

2. This research was partially supported by the Swiss National Fund for Scientific Research (FNRS) under grant no. 20-34056.92.
system (Kim et al, 1989). Orion’s model distinguishes two types of reference from one object to another: standard and composite. A composite reference is the standard reference augmented with the “part-of” relationship. A set of components objects linked through composite references is called a composite object or a complex object. A composite reference may be of two types: exclusive and shared. An exclusive composite reference from o’ to o means that o is part of only o’, while a shared reference from o’ to o means that o is part of o’ and of eventual other objects. A composite reference may be dependent or independent. A dependent composite reference from o’ to o means that the existence of o depends on the existence of o’ and that the deletion of o’ will trigger recursive deletion of o. The objects related through composite references form a part hierarchy. The classes to which the objects in the part hierarchy belong are also organized in a hierarchy called a composite class hierarchy. The root of a composite object is a special object.

We propose a different approach based on a generalization of the universal relation paradigm (Maier and Ullman, 1983). We define a context as an abstraction of the schema which allows to handle objects associations. Since composite objects and contexts are formally defined and implemented in a database system, we will compare them at conceptual and implementation levels.

The paper is organized as follows. Section 2 defines the context concept. Section 3 gives the fundamental differences between a context and a composite object. Section 4 discusses the differences of their implementation. Section 5 is a conclusion.

2 Contexts

We have introduced contexts in the framework of the F2 data model. However, the concept is general and may be applied to other object-oriented data models. F2 is an object-oriented database system developed at CUI (Estier and Falquet, 1992). Its data model is based on the concepts of object, class, attribute and specialization. It supports multiple instantiation and automated classification of objects. It includes integrity constraints like existential dependencies and keys. An attribute in F2 has a minimal (min) and a maximal (max) cardinality. The cardinalities of an attribute A specify that an object must take on A at least min values and at most max values. If min is greater than zero and the domain of the attribute is a non-atomic class, then it expresses an existential dependency between objects of the origin and domain classes of that attribute.

A context is an abstraction of the schema (Falquet et al, 1988) (Falquet, 1989). It includes structural and behavioural features. It consists of a connected graph, a set of constraints and a set of methods. Each node N of the graph corresponds to one class C(N) and each edge from N₁ to N₂ corresponds to one attribute of C(N₁) with C(N₂) as domain.

The context definition statement in the F2 data definition language has the following format:

```
context contextName
    nodes { nodeName : className [ with ( { attributeName } ) ] }
    edges { nodeName₁ attributeName nodeName₂ }
    constraints { constraint }
    methods { methodName parameters implementation }
```

A context has a unique name (contextName). The parameters associated with each keyword are described as follows:
− a node (nodeName) is defined on a class of the schema (className). Without the keyword with, all the attributes of this class with an atomic domain belong to the context. Else, only those attributes which are specified (attributeName) are included in the context. Several nodes of the same context may be defined on the same class.
− an edge links two nodes of the context (nodeName1, nodeName2). It uses an attribute (attributeName) whose origin is the class of nodeName1 and domain is the class of nodeName2.
− a constraint is a predicate whose free variables are context nodes.
− a method (methodName, parameters) with its code (implementation) is defined on a context.

Figure 2 shows a context named “Publishing” defined over the database schema given in figure 1.
Since nodes are a parameter of the connection function, we may have various connections. Figure 3 shows a database state $\sigma$ and different connections on the context “Publishing” in the state $\sigma$.

A context can be updated with the use of two functions: assert and retract. Updating a context has repercussions on the underlying database. The function assert has three parameters: a context $K$, a tuple of objects $t = \{N_1: o_1, N_2: o_2, ..., N_p: o_p\}$, and a database state $\sigma$. It links in $\sigma$ the objects of the tuple $t$ through all the paths which link the nodes $\{N_1, N_2, ..., N_p\}$ in the context $K$. The function retract($K, t, \sigma$) “unlinks” the objects of the tuple $t$ in $\sigma$ through all the paths which link the nodes $\{N_1, N_2, ..., N_p\}$ in the context $K$. For instance, assert(Publishing, $\{S: s_1, E: e_1\}$, $\sigma$) links the objects $s_1$ and $e_1$ by creating a new object $d_1$ in the class “document” such that $d_1$ is linked to $e_1$ and $s_1$. Retract(Publishing, $\{S: s_1, E: e_2\}$, $\sigma$) removes the links between the objects $s_1$ and $e_2$ by setting the value of the attributes “authors” and “describes” of $d_1$ to null. Formal properties of these operations are discussed in detail in (Falquet, 1994).

3 Fundamental differences between context and composite object

The concepts of context and composite object are different. We list below their fundamental differences.

**Definition level.** A composite class hierarchy and a context are defined at different levels. The first one is a fragment of the schema, while the second one is defined over the schema and is not part of it. As a consequence, a context may be created, deleted and modified without affecting the underlying schema. Moreover, the content of a context is derived from stored data through a connection function. The context approach is based on the universal relation paradigm instead of the navigation paradigm. A connection function returns objects which are associated without giving the paths through which they are linked. Association between objects is non-ambiguous according to the design of the context.

**Structure.** A context is not hierarchical. It has no root class playing a central role. All its nodes play the same role.

“Part-of” links. Context edges are defined on attributes which do not have necessarily the “part-of” semantics. The connection function returns objects associations and not only components objects.

**Constraints.** Constraints may be added on context nodes to allow the definition of subsets of classes without introducing new subclasses in the schema. For example, we can add the constraint <D.title contains “expert system”> to the context “Publishing” to access only the documents on expert systems through this context.
There are other differences between a context and a composite object which result from the implementation of these two concepts in the database systems F2 and Orion. These differences may disappear with the database systems evolution. We illustrate them by the following example (Kim et al, 1989). A document consists of a title, authors and a number of sections. A section in turn is composed of paragraphs. A document may share entire sections or section paragraphs with other documents. Annotations may be added to documents; however, they are not shared among different documents. Furthermore, documents may contain images that are extracted from files. Figure 4a shows the composite class hierarchy “Document” in Orion.

**Composite attributes.** A composite attribute from class C’ to class C can be modelled in F2 by an attribute from C to C’. If the composite attribute is exclusive, then in F2 its maximal cardinality must be equal to one. If the composite attribute is dependent, then in F2 its minimal cardinality is greater than zero. For example, the attribute “annotations” (fig. 4a) is modelled in F2 by the attribute “annotationOf” of the class “paragraph” (fig. 1) with maximal cardinality 1 and minimal cardinality 1. A paragraph is an annotation in only one document but it can be at the same time a section paragraph (different from Orion). Figure 4 shows two contexts “Special document” and “Document” which are defined over the schema given in figure 1 to represent a document.

**Node attributes.** A context node can hide some attributes of the class on which it is defined. For example, we can define in the context “Document” the node D with the attribute “title” and hide the attribute “number” (fig. 1).

**Points of view.** Contexts allow to show objects from several points of view while a composite object gives a unique representation of an object. A context K₁ may show a class C with a set of related classes RC₁ and another context K₂ may show the same class C with another set of related classes RC₂. An object in C may then be associated with objects of different classes in two different contexts. For example, in the context “Document” a document is associated with sections, paragraphs and images, while in the context “Publishing” it is associated with employees, systems and departments.
**Methods.** Methods can be defined on a context, while in Orion they are defined on component classes of a composite class hierarchy. Encapsulation in F2 is made at context level instead of class level.

**Objects associations.** The connection function returns connection tuples which express associations between objects. For example, in the context “Document” a connection tuple \([D: d, S: s_3, P: p_1]\) expresses that \(p_1\) is a paragraph in section \(s_3\) of document \(d\). The function “components-of” in Orion gives the components of a composite object without showing the associations between them. For example, components-of(d, \{paragraph, section\}) returns paragraphs \(p_1, p_2\) and sections \(s_3, s_4\) as components of document \(d\) without indicating which paragraph belongs to which section.

**Ambiguities.** A connection tuple is an association of objects which is not ambiguous. For example, in the context “Document” two nodes are defined on the class “paragraph” to play two different roles. In this context \(p\) can be a section paragraph of document \(d\) or an annotation of \(d\) or both. A connection tuple \([D: d, P: p]\) expresses that \(p\) is a section paragraph of \(d\), while a connection tuple \([D: d, A: p]\) expresses that \(p\) is an annotation paragraph of \(d\). In the context “Special document” a connection tuple \([D: d, P: p]\) expresses that \(p\) belongs to a section of \(d\) and it is an annotation of \(d\) (objects are associated if they are linked through all the paths that link their nodes). The function “components-of” in Orion may return ambiguous answers in the presence of a cycle. For example, components-of(d, \{paragraph\}) returns \(p\) which can be interpreted as a section or an annotation paragraph of \(d\).

**Updates.** Update operations on a context (assert, retract) allow to update connection tuples and consequently corresponding objects in the database. These operations are not supported on composite objects. In a context, we can associate two objects whose classes are not directly linked by attributes. For example, assert(Document, \([D: d, P: p], \sigma\) associates a paragraph with a document by creating a new object in the class “section”. The function “make” in Orion can only link an object to its parent objects.

5 Conclusion

A context is different from a composite object. Context is a powerful concept which can be used to manipulate objects associations in the database, to avoid ambiguities in database access, to support schema evolution (Andany et al, 1991), to express object dynamics (Al-Jadir et al, 1993) and to help in the conception and administration of an information system (Al-Jadir et al, 1994).

**References**

Al-Jadir, L., Falquet, G. and M. Léonard (1993); Context Versions in an Object-Oriented Model; Proc. 4th International Conference DEXA (pp. 24-35).


Atkinson, M., Bancilhon, F., Dewitt, D., Dittrich, K., Maier, D. and S. Zdonik (1989); The Object-Oriented Database System Manifesto; Proc. International Conference DOOD (pp. 223-240).

Estier, T. and G. Falquet (1992); Le petit manuel de Farandole 2; Technical report, Cahiers du CUI, no 54, Centre Universitaire d'lnformatique, Genève.

Falquet, G. (1989); Interrogation de bases de données à l'aide d'un modèle sémantique; Thèse de doctorat, Université de Genève.


Falquet, G., Guyot, J., Junet, M., Léonard, M., Bursens, R., Crausaz, P. and I. Prince (1988); Concept Integration as an Approach to Information Systems Design; Computerized Assistance during the information systems life cycle (eds T.W. Olle, A.A. Verrijn-Stuart and I. Bhabuta); IFIP, North-Holland (pp. 19-65).

Kim, W., Bertino, E. and J.F. Garza (1989); Composite Objects Revisited; Proc. International Conference ACM SIGMOD (pp. 337-347).


Nguyen G.T. and D. Rieu (1989); Schema evolution in object-oriented database systems; Data & Knowledge Engineering, 4 (pp. 43-67).