Conceptual Graphs as a Framework for Deductive Object-Oriented Databases

Bikash C. Ghosh and Vilas Wuwongse

Division of Computer Science, Asian Institute of Technology
G.P.O. Box 2754, Bangkok 10501, Thailand
Fax: (66-2) 5245721, e-mail: vw@ait.th

Abstract. The idea of deductive object-oriented databases (DOODBs) is to combine the concepts of deductive databases (DDBs) and object-oriented databases (OODBs) into a single database system in order to gain the advantages offered by each of them. This kind of databases is suitable for knowledge bases and many advanced database applications. The formalism of conceptual graphs (CGs), a knowledge representation scheme in AI, is equipped with some useful constructs that are suitable for the requirements of DOODBs. A groundwork for DOODBs based on conceptual graphs has been carried out in this research. The DOODBs are characterized by data abstraction through objects, object identifiers, object types, type hierarchy, property inheritance, methods and message passing and a logical formalism with a sound inference system. Some restrictions and extensions are proposed for the general conceptual graphs so that they can be used to represent the DOODB concepts. These extended conceptual graphs are called deductive object-oriented conceptual graphs (DOOCGs). The object types, individual objects and object identifiers of DOODBs map into concept types, individual CGs and individual referents, respectively. Methods are defined using conceptual schema graphs with bound actors and interpreted in a success/failure paradigm. A set of extended derived rules of inference has been formulated for DOOCGs which are proved to be sound.

1 Introduction

Most of the currently popular "record-oriented" database systems have limited representational abilities that are considered particularly essential for some advanced application areas like CAD/CAM databases, VLSI design databases, office automation, engineering design and CASE databases [16], and knowledge bases. A detailed description of the major limitations of record-oriented data models can be found in [10]. The two of the major areas of extensions over the conventional databases are: i) capturing more semantics in the represention of data [3], and ii) storing intensional rules and performing reasoning over the database states [11]. The first extension leads to the research on *object-oriented database* (OODBs) systems [16], while the second one is handled by the *deductive databases* (DDBs) [12, 16].

Deductive databases integrate ideas from logic programming and relational databases [11] while object-oriented databases emerge from the integration of database concepts with object-oriented programming paradigm. Since each of the DDBs and ODDBs offers some advantages over conventional databases, there have been efforts to merge the two approaches to obtain the advantages of both the systems, and these efforts represent a challenge [1]. The notion of *deductive object-oriented database* (DOODBs) comes from some of these efforts. DOODBs can also serve as a starting point for a future *knowledge based management system* (KBMS) that integrates the ideas of artificial intelligence and databases [2].

The formalism of *conceptual graphs* (CGs), introduced by SOWA [13], offers some useful constructs which make it a likely platform for an integration of ideas from DDBs and OODBs. It is a powerful knowledge representation language in AI with a well-defined theoretical basis and a close mapping to both natural language and first-order logic [9, 13, 15]. The objective of this work is to develop a notion of a deductive object-oriented database system based on the formalism of CGs. Some restrictions and extensions to CG formalism are introduced and a theoretical basis for a DOODB system based on the extended CG formalism is developed.

2 Deductive Object-Oriented Database Systems

Two general approaches towards research in DOODBs have been mentioned in [17], namely DOODBs as an extension of DDBs and DOODBs as an extension of ODDBs. We propose a third approach which is to integrate the concepts of DDBs and OODBs and map them into a third framework. In this work, the formalism of CGs is used to represent the features of DOODB systems whose characteristics are outlined as follows.

(a) *Objects and object properties*: A DOODB should allow all conceptual entities to be uniformly modeled as objects. Objects should be the unit of access and control. An object has properties which could be static or dynamic. Static properties represent states (or structures) of the object. Dynamic properties (called behaviors) are represented by a set of methods. A DOODB system should provide mechanisms for generating and maintaining unique object identifiers for individual objects.

(b) *Types (or classes), type hierarchy and property inheritance*: Objects are instances of types. A type describes the structure and behavior of all of its instances. Types should form a type hierarchy based on the subtype/supertype relationships. Objects of one type inherit properties from the type and its supertypes. A DOODB system should have mechanisms for:

(i) resolving the problem of multiple inheritance, and
(ii) handling exceptions to property inheritance.

In general, the definition of an object type in a DOODB should include:

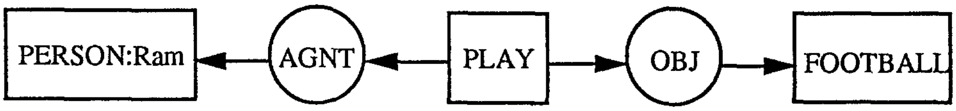
- Specification of the structure of the objects that are instances of this type,
- Specification of the position of the type in the type hierarchy,
- Specification of the set of methods that are applicable to the instances or states of the instances of that type.

(c) A DOODB system should have a logical formalism to represent objects, methods, intensional rules or (deductive) laws, integrity constraints and queries, with a well-defined semantics of all of these components as well as answers to queries. The logical system should have a sound (deductive) inference mechanism.

(d) Some of the other desirable properties are : declarative expressions for databases, queries and answers to queries, and single language for expressing databases, queries and integrity constraints.

3 Conceptual Graphs

The primitives of the conceptual graph (CG) theory are : *concept instances* or *referents* and *conceptual relations*. A *conceptual graph* is formally defined [13] as a finite, connected, bipartite graph where the two kinds of nodes are concepts and conceptual relations. Every conceptual relation has one or more arcs, each of which must be linked to some concept. A single concept by itself may form a conceptual graph, but every conceptual relation must be linked to some concept. There are two forms of *display* for conceptual graphs: *graphical form* (Fig. 3.1(a)) and *linear form* (Fig. 3.1(b)).



(a) Graphical form of display

[PERSON:Ram]<--(AGNT)<--[PLAY]-->(OBJ)-->[FOOTBALL]
(b) Linear form of display

Fig. 3.1 Two forms of display for a conceptual graph that may be read as "The person Ram plays football"

The set of type labels T forms a lattice, called type *type lattice* based on the partial order relation $\leq_T$. The top ($\top$) element is called UNIV and the bottom ($\perp$) element is called ABSURD. There is a predefined set I = {#1,#2,#3,...} whose elements are called *individual markers*. The function *referent* maps a concept c into $I \cup \{*\}$, where * is called the generic marker. Every conceptual graph makes an assertion. An operator ϕ has been defined that maps conceptual graphs into formulas of first-order predicate calculus. For example, the graph in Fig. 3.1 makes the following assertion,

$\exists x, \exists y\ (PERSON(Ram) \land AGNT(x,Ram) \land PLAY(x) \land OBJ(x,y) \land FOOTBALL(y))$

A *canonical graph* is a CG that specifics the constraints on the pattern of concepts and

A *canonical graph* is a CG that specifies the constraints on the pattern of concepts and relations that may be linked to a concept and relation type. There are some primitive concept types and conceptual relation types which have no explicit definitions. New concept types and relation types are defined using lambda abstractions adapted to CGs. There are four *canonical formation rules* for deriving new CGs from existing CGs: *copy, restrict, join* and *simplify*. A *maximal join* operation [13] is defined on CGs and can be considered as a generalization of unification under suitable mapping [9]. If a CG u is *canonically derivable* from another CG v, then u is called a *specialization* of v and v is called a *generalization* of u. Generalization defines a partial ordering ($\leq_G$) of conceptual graphs called the generalization hierarchy. For any conceptual graphs u and v, if $u \leq_G v$, then $\phi u \supset \phi v$ and there exists a mapping $\pi: v \rightarrow u$, where πv, a subgraph of u is called *projection of v in u*.

Procedural information can be attached to CGs using *actors*. An actor is described as a process that responds to messages by performing some services and then generating messages that it passes to other actors. Actors are attached to CGs by input and output concepts linked to it. A *dataflow* graph consists of actors and concept nodes. It is defined as a finite, connected, bipartite graph with one set of nodes called actors and another set of nodes called concepts.

SOWA [13] developed a set of *first-order rules of inference* for CGs represented in terms of nested negative contexts and coreference links. A context is a special type of concept node with a type label called PROPOSITION and a set of CGs as referent. A negative context is a context with a monadic relation (NEG) attached to it. The coreference link is used to connect identical concepts and it is shown using a dotted line in the graphic form and using the same concept variables in the linear form. A game-theoretic semantics of CGs is given in [13] and the first-order rules are proved to be sound and complete. A set of derived rules of inference are also defined which are essentially sound and complete. Details about general CG formalism can be found in [13].

4 Deductive Object-Oriented Databases Using Conceptual Graphs

The features of DOODBs are to be represented by the *deductive object-oriented conceptual graphs* (DOOCGs) [7]. The major features of DOOCGs are explained in this section. More details about DOOCGs can be found in [7].

4.1 Primitives of Deductive Object-Oriented Conceptual Graphs

Object types in DOODBs are represented by concept types in CGs, *individual objects* of a type are represented by individual CGs of the corresponding concept type. For example, an object type STUDENT can be defined as a subtype of PERSON as shown in Fig. 4.1, where all other concept types and conceptual relations are assumed to be defined already. An individual object of type STUDENT is shown in Fig. 4.2. The object type hierarchy is represented by the concept type hierarchy that is a complete lattice based on the subsumption relation $\leq_T$. The type definition specifies the position of the newly defined type in the hierarchy. Individual referents for the concepts that are parameters of the type definitions are used as *object identifiers*. For example, the individual referent #112 in Fig. 4.2 is an object identifier. Note that the individual referents include the individual markers as well as other externally communicable objects like strings and numbers.

```
STUDENT = (λ x)
        [PERSON:*x] -
                (CHRC)-->[ID]
                (OBJ)<--[ENROLLMENT] -
                                (PTIM)-->[DATE]
                                (CHRC)-->[DEG]
                                (LOC)-->[DIVISION],
                (ADVISOR)-->[FACULTY].
```

Fig. 4.1 Definition of STUDENT type as a subtype of PERSON

```
STUDENT(#112) =
        [PERSON:#112] -
                (CHRC)-->[ID:3421]
                (OBJ)<--[ENROLLMENT:#321] -
                                (PTIM)-->[DATE:01/01/1990]
                                (CHRC)-->[DEGREE:"M Engg."]
                                (LOC)-->[DIVISION:"CS"],
                (ADVISOR)-->[FACULTY:#311].
```

Fig. 4.2 An individual object of type STUDENT

Formulation of *methods* and *message passing* is one of the important problems in merging OODB and DDB concepts. For our representation, the major requirements are as follows:

- a mechanism for defining methods and associating the method definitions for a type with the corresponding type definition,
- a format for method calling or message passing,
- a backtrackable method evaluation mechanism (so that all possible values are returned) with a success/failure interpretation [6] of method evaluation,
- a mechanism that allows individual objects of one type to share the methods defined for its supertypes.

A method is defined using a *conceptual schema graph* [13] with bound actors. A method call or message is represented by a *message graph*, that is a simple CG with an appropriate actor bound to it. An example method definition for the age of a person is shown in Fig. 4.3, where <TODAY> and <DIFF_DT> are two actors. The actor <TODAY> is a system defined actor that asserts the current date in the referent field of its output concept. The actor <DIFF_DT> is a primitive actor (or can be defined by a functional dataflow graph) that asserts the difference of two of its input dates into the referent field of its output concept. An example message graph is shown in Fig. 4.4, where the type label in the actor box before the name of the actor specifies the receiving object type. The graph in Fig. 4.5 is the same as that in Fig. 4.4 except that the output concept of the actor <PERSON:DIFF_DT> contain a request mark ("?") which signifies the triggering of the execution of the corresponding method. In general, if m is a message graph then ?m is used to denote the corresponding method call. Note that the projection of a message graph is defined as the projection of the CG obtained from the message graph by erasing the actor bound to it.

A new CG operation is necessary in formulating the rules for method execution. The main purpose of this operation is to find out the *maximal common overlap (modulo restrictions)* between two CGs.

Definition 4.1 Let two CGs u_1 and u_2 have a common generalization v with *maximally extended compatible projections* [13] π_1:v→u_1 and π_2:v→u_2. A join of the two graphs π_1v and π_2v on these maximally extended compatible projections is called a *constrained join*.

Example 4.1 Suppose we have two CGs u_1 and u_2 as shown in Fig. 4.6. The graph v is a common generalization of the two graphs u_1 and u_2 with two maximally extended compatible projections π_1 and π_2, where π_1 = {([PERSON],[STUDENT]),(AGNT,AGNT),([PLAY])} and π_2 = {([PERSON],[PERSON:Sam]),(AGNT,AGNT),([PLAY],[PLAY])}. The graph π_1v is a subgraph of u_1 and π_2v is a subgraph of u_2. A join of π_1v and π_2v (on the projections π_1 and π_2) yields the graph w. The operation is called the *constrained join* and the graph w is the graph obtained by constrained join of u_1 and u_2.

Rules for method execution

Let S be a set of CGs and m be a message graph which contains an actor e. the message is "passed" to the type t, i.e., the method call ?m is issued. Steps taken in executing the method with respect to S are as follows:

1) Search for a definition of method corresponding to the message starts at the schematic cluster [13] of type t. If a schema is not found for the type t, then the search proceeds upwards along the hierarchy according to the rules of method inheritance. The search stops with one of the following conditions:

   - An appropriate method definition is found and a schema graph is selected. Suppose that the schema graph u is selected. The method evaluation proceeds.
   - The topmost concept in the hierarchy has been reached but no

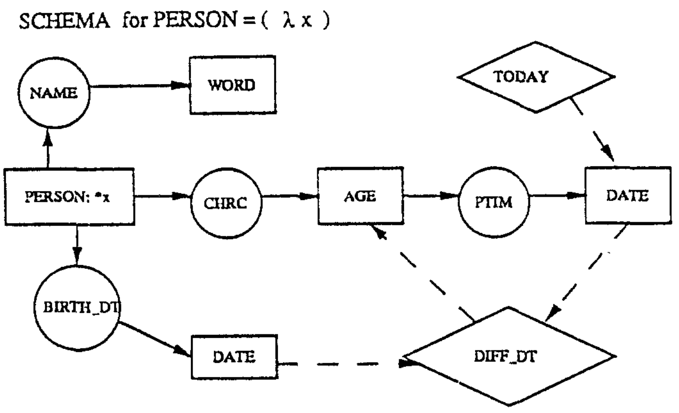


Figure 4.3: An example of method definition for the type PERSON

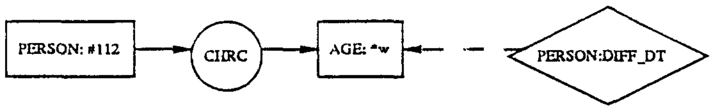


Figure 4.4: An example message graph

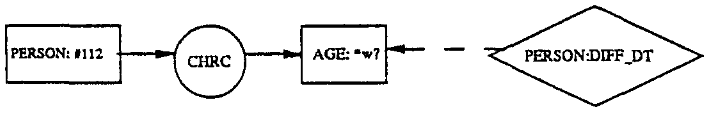


Figure 4.5: The message of Fig. 4.4 is being "passed"

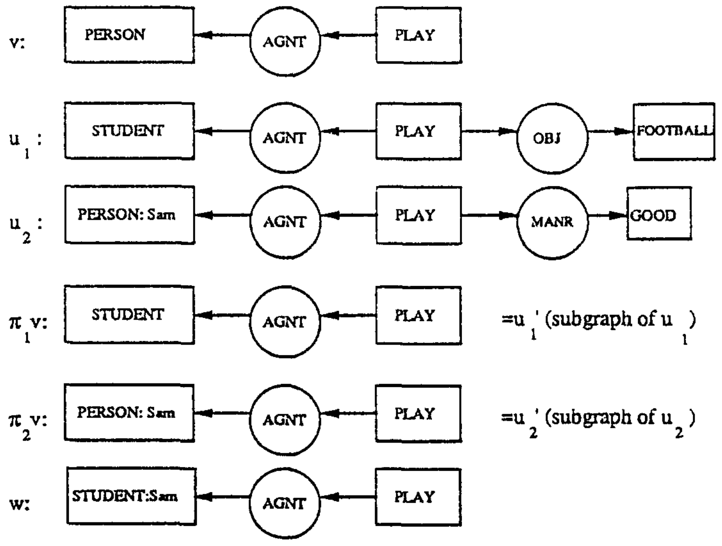


Figure 4.6: An example of constrained join - the graph w is obtained by constrained join of u_1 and u_2

appropriate method definition has been found. In this case, the method evaluation stops and the original method call fails.

2) The message graph ?m is joined with the selected schema graph u. Let u' be the resulting graph after joining m with u. After the join, the output concept in u' of the actor e will be marked with request ("?") marker. This will trigger a goal directed computation. For all input concept i, with referent (i) = *, mark (i) is set to "?" [13].

3) The next action will depend on the nature of the input concept nodes that are marked with "?" marker. According to the rules for method definition [7], each of the input concept of the actor e is either an output concept of another actor or a message to another type, or an "independent property" that is a part of the type defining graph.

- For each input concept that is an output concept of a message to another type, the message is passed to the corresponding type. If the evaluation of any of such message fails, then the actor e is blocked [13] and the original method call (?m) fails.
- For each input concept that is an output concept of another actor, the "?" maker is propagated in the backward direction.

The system would search for an individual graph to evaluate those concepts that represent independent property and are marked with request marker. This search can be performed in the following way:

- Suppose the corresponding type has been defined with an abstraction of the form λxv. The graph v is joined with u' using the constrained join. Let v' be the resulting graph. If v' does not contain all those concepts that are marked with "?" marker but are not output concepts of any actor or message, then the search stops and the original method call fails. Otherwise, the graph v' is to be projected on the set of individual CGs in S. Two possible cases are:

  (i) The graph v' has no projection on the set of individual CGs in S. The original message call fails.

  (ii) The graph v' has a projection π' on the set of individual CGs in S.

- If the previous step succeeds, then the graph π'v' is joined (using maximal join) with u' yielding w for each projection π' of v' on the individual graphs in S. As a result of this, individual referents would be asserted into some or all the concept nodes that were marked with "?" marker.

4) If all previous steps succeed, then the actor e in w must be enabled and fired.

This firing would assert a referent in the output concept of the actor. Let w' be the final working graph after the actor has fired. The original method call succeeds with the answer <true,π> where π:m→w'. In general, there may be more than one return value depending on the number of distinct projections of v' on the set of individual graphs in S. Suppose m' is the CG obtained from m by erasing actor(s) bound to it. Then πm' is also true, i.e., the assertion made by the message graph m is true.

Success and failure of a method call

The value of the method execution ?m will be <false,{}> only if the method evaluation fails. The method evaluation fails if any of the following is true:

- The search for an appropriate schema fails in the first step, i.e., their no method definition corresponding to the message.
- The firing of the actor is blocked in step 4 for any reason whatsoever.

If the method does not fail, then it succeeds with the answer <true,π> where π:m→w', where w' is the final working graph. Suppose m' is the conceptual graph obtained from m by erasing actor bound to it. Then πm' also makes a true assertion.

Property inheritance

In CG notations, a branch of the type defining graph consisting of a conceptual relation and a concept type label corresponds to an attribute in the DOODB concepts and an individual referent in the concept node of that branch is the value of that attribute. The inheritance of properties and the exception rules are "naturally" handled by the *type expansion* [13] operation of CGs that use the maximal join and replaces a single concept node with the graph that defines that concept type. Methods defined for a particular type are also inherited by its subtypes. The search for a method definition starts at the type that is specified in the actor box with the actor name. If a method is called with the type label t, then the search starts at the schematic cluster of type t. If the method definition is not found, then it is searched in the set of method definitions for each of its direct supertypes. The search proceeds upward along the chains of the hierarchy until an appropriate method definition is not found, then it is searched in the set of method definitions for each of its direct supertypes. The search proceeds upward along the chains of the hierarchy until an appropriate method is found or the topmost type UNIV is reached. The first method encountered in the search is the one that is executed.

4.2 Conceptual Basis

The meanings of conceptual graphs in CG formalism are implicitly related to some basic CG constructs. These are collectively referred to as a conceptual basis, which is defined as follows.

Definition 4.2 A conceptual basis is defined as

CB = $<(R,\leq_R), (T,\leq_T), D, SC, P, A>$, where

CB	:	Conceptual basis,
R	:	A set of conceptual relation labels,
$\leq_R$	:	Subsumption relation between conceptual relation labels,
$(R,\leq_R)$	:	Conceptual relation hierarchy,
T	:	A set of concept type labels,
$\leq_T$	:	Subsumption relation between concept type labels,
$(T,\leq_T)$	:	Concept type hierarchy - the type lattice,
D	:	A set of concept type definitions for types in T,
SC	:	A set of conceptual schema graphs for types in T,
P	:	A set of prototype graphs for types in T,
A	:	A set of actor definitions.

4.3 Deductive Object-Oriented Conceptual Graphs (DOOCGs)

Before defining DOOCGs, we define atomic conceptual graphs and compound graphs as follows.

Definition 4.3 An atomic conceptual graph (ACG) [4] is a conceptual graph that contains no logical connective and no quantifier other than the implicit existential quantifiers. An atomic conceptual graph containing only individual referents in all the concept nodes is termed as a ground atomic conceptual graph.

Definition 4.4 An conceptual graph is called a compound graph if any or all of the following conditions hold: (a) to contain contexts of depth [13] higher than 0, (b) it contains ACGs connected by coreference links.

Example 4.2 An example of atomic conceptual graphs and ground atomic conceptual graphs is shown in Fig. 4.7(a) and Fig. 4.7(b) respectively. A compound conceptual graph represented by nested negative contexts is shown in Fig. 4.7(c).

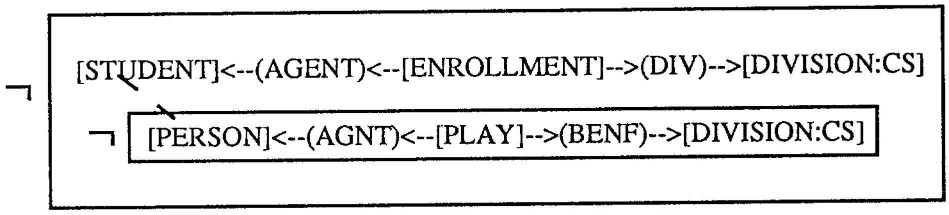
Now we define DOOCGs.

Definition 4.5 The deductive object-oriented conceptual graphs (DOOCGs) include the following graphs:

(a) Atomic conceptual graphs.
(b) Message graphs.
(c) Compound graphs; represented in nested negative contexts with the following constraints:
  - The graphs consist of only two levels of nesting.
  - The context at depth 1 contains one or more atomic conceptual graphs and/or message graphs in addition to one negative context.
  - The context at depth 2 contains exactly one atomic conceptual graph.

[PERSON]<--(AGNT)<--[SIT]-->(LOC)-->[PLACE]
(a) An atomic conceptual graph

[PERSON:Sam]<--(AGNT)<--[SIT:#543]-->(LOC)-->[PLACE:Cafeteria]
(b) A ground atomic conceptual graph -
all concept nodes contain individual referents


¬
[STUDENT]<--(AGENT)<--[ENROLLMENT]-->(DIV)-->[DIVISION:CS]
¬
[PERSON]<--(AGNT)<--[PLAY]-->(BENF)-->[DIVISION:CS]

(c) A compound graph that says "If a person is a student of CS division then he/she can play for CS division"

Fig. 4.7 Three conceptual graph examples

There may be coreference links among concepts of the conceptual graphs at various contexts.

These graphs have the general form: $\neg[v_1 \ldots v_n \neg[w]]$ where, v_i $(i=1,\ldots,n)$ is an atomic conceptual graph or message graph, w is an atomic conceptual graph. If $n=0$, the above graph becomes $\neg[\neg[w]]$ or simply w that represents a single atomic conceptual graph w in the outermost context.

4.4 Deductive Object-oriented Conceptual Graph Language (DOCL)

Definition 4.6 A *deductive object-oriented conceptual graph language* (DOCL) consists of the following:

(a) a *concept universe* consisting of,
- i) a conceptual basis CB,
- ii) a non-empty set of individual referents I_r that consists of the individual markers {#1,2,#3,...} as well as other externally communicable objects like strings, numbers and times,
- iii) the conformity relation :: that relates types in T to individual markers in I_r, the function ltype that relates concepts and conceptual relations to type labels in T and R respectively and the function referent that relates concepts to the elements of $I_r \cup \{*\}$, where * denote the generic marker,
- iv) the set of canonical formation rules together with the maximal join operation and the constrained join operation,

v) the rules for actors and dataflow graph definitions [7, 13],
vi) the rules for method definition, method call and method execution [7],
vii) a canonical basis B which is a finite set of conceptual graphs with all types in T and all referents either * or individuals in I_r, and

(b) a set of conceptual graphs formed from the concept universe.

Now we define some terms related to a DOCL.

Definition 4.7 A *conceptual program* CP is a finite set of deductive object-oriented conceptual graphs.

Definition 4.8 The *degree* of DOOCGs is a mapping from a set of DOOCGs into the set of integers. For a DOOCG C, the degree of C, denoted by DEG(C) is the number of conceptual graphs in the context at depth 1. For a DOOCG of the general form $C = \neg[v_1\ v_2\ \dots\ v_n\ \neg[u]]$, DEC(C) = n. For an atomic CG or message graph u in the outermost context, DEC(u) = 0.

Definition 4.9 A *query graph* is either an atomic conceptual graph or a message graph.

Definition 4.10 A *query* consists of a single query graph or a number of query graphs in conjunction.

Thee may be coreference links among concepts of the query graphs in a query. For example $Q = q_1 \dots q_n$ is a query where each of q_i is either an ACG or a message graph.

Definition 4.11 A *goal graph* is either an atomic conceptual graph or a message graph.

Definition 4.12 A *goal* consists of a set of goal graphs in conjunction. There may be coreference links among concepts of the goal graphs in a goal.

Definition 4.13 The goal G corresponding to a query $Q = q_1 \dots q_n$ is represented as $\{q_1, \dots, q_n\}$, which is the assertion made by the query graph $q_1, \dots, q_n$ in conjunction (together with the coreference links of Q).

Definition 4.14 Let u and v be two conceptual graphs. The graph u is a referent-specialization of v if u is a specialization of v with the following property: u is the same as v except that zero or more generic concepts in v are specialized to individual concepts in u. Let π be a projection of v. The referent-specialization operator (rso) ρ for the projection π consists of a set of pairs of the form (c,c''), which is obtained from π in the following way:

a) initially ρ is set equal to { },
b) for each pair (c,c') such that $\pi c = c'$ where referent(c) = * and referent(c') $\neq$ *, a pair (c,c'') is added to ρ, where type(c'') = type(c) and referent(c'') =

referent(c').

Note that, $\rho = \{\}$ represents the identity referent-specialization.

Theorem 4.1 [7] Let v be a conceptual graph, π be a projection of v and ρ be the referent-specialization operator for the projection π. Then, $\pi v \leq_G \rho v \leq_G v$.

Definition 4.15 Let CP be a conceptual program and $G = \{g_1,...,g_n\}$ be a goal. An answer for G from CP is $G' = \{g'_1,...,g'_n\}$, where each g'_i, called an answer graph for the goal graph g_i is a referent-specifialization of g_i. It is to be noted that if G contains only individual concepts then the only possible answer is obtained by the identity referent-specialization.

4.5 Extended Rules of Inference for DOOCGs

The propositional rules of inference or alpha rules, the first-order rules of inference or beta rules and a set of derived rules or delta rules are presented in [13]. Two fundamental properties of DOOCGs are stated in the form of rules (α0) and δ0). Let S be a set of DOOCGs.

(α0) Let $u \in S$ and v be any atomic conceptual graph. If u is a specialization of v, (i.e., v is a generalization of u), then the assertion made by v is true if the assertion made by u is true. That means, if $u \leq_G v$, then $\phi u \supset \phi v$ where ϕ is the formula operator defined in [13].

(δ0) Suppose m is message graph and the method call ?m succeeds with respect to S with a projection π. Let m' be the graph obtained by erasing the actors from the graph πm. The graph m' makes a true assertion in the outermost context.

Now a special case of the rule (α0) is stated here as (Δ0) as follows.

(Δ0) Suppose, u and v are two atomic conceptual graphs, and the graph u makes a true assertion. Then the assertion made by v must be true if any of the following statements holds:

(1) Subgraph: v is a subgraph of u.
(2) Subtypes: u is identical to v except that one or more type labels of v are restricted to subtypes of the corresponding concept types in u.
(3) Individuals: u is identical to v except that one or more generic concepts of v are restricted to individual concepts of the same type in u.

The extended derived rules of inference are formulated as follows.

Theorem 4.2 Let S be a set of DOOCGs and u and v be any conceptual graphs where $u \leq_G v$ with a projection $\pi: v \to u$. Let m be a message graph, and the method call ?m succeeds with respect to S with a projection π_m. Any graph derived from S by the following *extended derived rules of inference* is said to be provable from S.

(Δ1) In an oddly enclosed context, v may be replaced with πv where each coreference link to a concept c of v is transferred to the corresponding concept πc in πv. In an oddly enclosed context, m may be replaced with $\pi_m m$ and actors bound to $\pi_m m$ may be erased, where each coreference link to a concept c of m is transferred to the corresponding concept $\pi_m c$ in $\pi_m m$.

(Δ2) Let P be a graph ¬[u ¬[v]]. Then P itself is a theorem, and P with coreference links $<\pi c, c>$ for any c in v is also a theorem. The graph ¬[πv ¬[v]], and ¬[πv ¬[v]] with coreference links $<\pi c, c>$ for any c in v are also theorems.

(Δ3) In an evenly enclosed context, u may be replaced with v where each coreference link to a concept πc of u is transferred to the corresponding concept c of v and the other coreference links attached to u are erased.

(Δ4) Generalized modus ponents: If the outer context contains the graph u as well as a graph of the form ¬[v ¬[w]], possibly with some coreference links from v to w, then the graph w may be derived with each coreference link attached to a concept c in v reattached to the corresponding concept πc in u. If the outer context contains the graph ¬[m ¬[w]], possibly with some coreference links from m to w, then the graph w may be derived with each coreference link attached to a concept c in m reattached to the corresponding concept $\pi_m c$ in $\pi_m m$.

The following theorem holds [7].

Theorem 4.3 The extended derived rules of inference are sound.

As in linear logic, the notions of interpretations, models and logical consequence have been defined for DOOCGs [7]. Actors are interpreted as conceptual functions. Each conceptual function maps the referents of the input concepts of the corresponding actor into the set of referents of its output concept. The game theoretic semantics for general CGs with respect to a closed world [13] has been extended for the DOOCGs. A model-theoretic semantics and a fixpoint semantics of DOOCGs have been defined and they are proved to be equivalent for a conceptual program. A proof procedure, called direct derivation proof has been formulated for DOOCGs and it is proved to be sound and complete. Details about the semantics of DOOCGs and the proof procedure can be found in [7]. A notion of a DOOCG theory has been developed, and a DOODB is formally represented as a special DOOCG theory.

Definition 4.16 A DOOCG theory T_G consists of a DOCL L, a set of axioms and a set of inference rules,

- Logical axiom : {}, the empty set.
- Proper axioms : A set of DOOCGs.
- Set of inference rules : the set of first-order rules and the set of extended derived rule.
- Proof procedure : The direct derivation proof procedure [7].

Definition 4.17 A *deductive object-oriented database*, DOODB is a DOOCG theory whose proper axioms are:

- A conceptual program CP that consists of:
  - a set O of elementary facts or individual objects which are ground atomic conceptual graphs.
  - a set S that consists of deductive laws, which are compound DOOCGs and message graphs.
- A set of integrity constraints IC, which are DOOCGs.

5 Conclusions

A subset of general conceptual graphs has been extended to form deductive object-oriented conceptual graphs that includes simple graphs, message graphs and compound graphs with only two levels of nesting and exactly one simple graph in the context at depth 2.

The approach taken for this work towards representing DOODBs is fundamental in the sense that most of the other known approaches are either to extend DDBs to incorporate some of the constructs of OODBs or to extend OODBs with some of the features of DDBs [17]. Being a relatively new area of research, there is no generally accepted notion of DOODBs yet. A more concrete notion of DOODBs has been specified in this work in terms of characteristics of DOODBs. The use of conceptual graphs allows a large amount of "knowledge" to be statistically captured in the representation itself through the use of type definition, type hierarchy and property inheritance.

One of the major problems of integrating ideas of DDBs and OODBs is the representation and interpretation of methods [6]. The solution to this problem has been formulated with restrictive use of conceptual schema graphs, actors and dataflow graphs of conceptual graph formalism. This offers several advantages:

- Methods can be easily shared among different objects along a chain of type hierarchy without need for any additional cost, because conceptual schema graphs are associated to the type labels in the type hierarchy in the CG formalism by their definitions.

- The use of actors and dataflow graphs offers a natural means to merge the "procedural" notion of methods in OODBs with the declarative paradigm of DDBs, because actor and dataflow graphs in Cg formalism can be used declaratively when treated as a descriptions and procedurally when they are executed [13].

- The mechanism of method evaluation is formulated in a success/failure paradigm which is considered as one of the requirements for merging object-oriented concepts with deductive database concepts [6].

This work created a basis for a database system that would be useful for knowledge representation as well as many advanced database applications. However, much work has to be done over the basis founded by the results of this work to make it evolve into a complete operational database system.

References

1. S. Abiteboul, S. Grumbach: A Rule-Based Language with Functions and Sets, ACM Transactions on Database Systms, Vol. 16, No. 1, pp. 1-30, March 1991.
2. M.L. Brodie, J. Mylopoulos, eds,: On Knowledge Base Management Systems-Integrating Artificial Intelligence and Database Technologies, Springer-Verlag, 1986.
3. K.R. Dittrich: Object-Oriented Database Systems: The Notion and the Issues, Proceeding of International Workshop on Object-Oriented Database Systems, Dittrich, K.R. & Dayal, U. (eds.), IEEE Company Society Press, pp. 2-4, 1986.
4. G. Ellis: Compiled Hierarchical Retrieval, Proceeding of the Sixth Annual Workshop on Comceptual Structures, pp. 187-207, July 1991.
5. J. Fargues, Marie-Claude Landau, A. Dugourd, L. Catach: Conceptual Graphs for Semantic and Knowledge Processing, IBM Journal of Research and Development, Vol. 30, No. 1, pp. 70-79, January 1986.
6. H. Gallaire: Merging Objects and Logic Programming: Relational Semantics, Proceeding of Fifth National Conference on Artificial Intelligence, AAAI-85, Vol. 2, pp. 754-758, 1986.
7. B.C. Ghosh: Towards Deductive Object-Oriented Databases Based on Conceptual GRaphs, Masters Thesis, Division of Computer Science, Asian Institute of Technology, 1991.
8. T.R. Hines, J.C. Oh, M.L.A. Hines: Object-Oriented Conceptual Graphs, Proceeding of the Fifth Annual Workshop on Conceptual Structures, pp. 81-89, July 1990.
9. M. Jackman, C. Pavelin: Conceptual Graphs, Approaches to Knowledge Representation: An Introduction, Ringland, G.A. & Duce, d.A. (eds.), John-Wiley & Sons Inc., pp. 161-174, 1988.
10. W. Kent: Limitations of Record-Based Information Models, Readings in Artificial Intelligence and Databases, Mylopoulos, J. & Brodie, M.L., (eds.),

Morgan Kaufmann Publishers, Inc., pp. 85-97, 1989.

11. J.W. Lloyd: An Introduction to Deductive Database Systems, The Australian Computer Journal, Vol. 15, No. 2, pp. 52-57, May 1983.
12. J.W. Lloyd: Foundations of Logic Programming, Second Extended Edition, Springer-Verlag, 1987.
13. J.F. Sowa: Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley Publishing Company, Inc., 1984.
14. J.F. Sowa, E. Way: Implementing a Semantic Interpreter using Conceptual Graphs, IBM Journal of Research and Development, Vol. 30, No. 1, pp. 57-69, January 1986.
15. J.F. Sowa: Knowledge Representation in Databases, Expert Systems and Natural Language, Artificial intelligence in Databases and Expert System, Meersman, R.A., Shi, Zh. & Kung, C.H. (eds.), North-Holland Publishing Co., Amsterdam, pp. 17-50, 1990.
16. J.D. Ullman: Principles of Database and Knowledgebase Systems, Vol. 1, Computer Science Press, 1988.
17. K. Yokota: Outline of a Deductive and Object-Oriented Language: Juan. SIGDB No. 78, July 1990.