Implementing OKBC Knowledge Model Using Object Relational Capabilities of Oracle 8

Gang Luo
Department of Computer Sciences
University of Wisconsin-Madison
1210 West Dayton Street
Madison, WI 53706
gangluo@cs.wisc.edu

Vinay K. Chaudhri
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
vinay@ai.sri.com

Abstract. PERK is a storage system that loads knowledge bases (KB) from the Oracle DBMS. Its current limitation is the inability to do client-side memory flushing out.

In this paper, we present a new architecture, named PERK-II, which utilizes the object relational capabilities provided by Oracle 8. It can do the client-side buffer management automatically, use object view and complex object retrieval (COR) to reduce network traffic, and push a lot of database functionality from PERK to the Oracle DBMS. So, the system is greatly simplified without sacrificing efficiency. Our basic design is based on open knowledge base connectivity (OKBC), which is an application programming interface (API) for accessing knowledge representation systems (KRS). By far, we have implemented the read-only OKBC operations using the Oracle Call Interface (OCI). Our test results show that the use of the Oracle client-side object cache does not substantially degrade the performance of the system as compared to using a "home grown" in-memory client cache.

Keywords: KB, OKBC, OCI, object view, object reference, ORDMBS.

1. Introduction

To efficiently store and load large knowledge bases (KB), we have previously developed a system called PERK that incrementally stores and loads KBs from the Oracle DBMS. The current limitation of PERK is that once a KB object is loaded into the memory of a knowledge representation system (KRS), it cannot be flushed out. Thus, PERK can't deal with KBs whose sizes exceed the virtual memory. We have been investigating an alternative architecture of PERK to address this limitation.

In the proposed new architecture, called PERK-II, we use the client-side object cache supported by the Oracle 8 ORDBMS. The objects of a KB are never defined in the address space of a KRS, and are always managed by the Oracle client-side object cache. The KRS operations are supported by manipulating the objects resident in the object cache. Since Oracle manages its object cache by flushing out objects as needed automatically, the problem of flushing the KB objects from the client-side memory is delegated from PERK to Oracle. In addition to this ability, this new architecture has several other benefits: a lot of the database functionality is pushed into the DBMS to which it belongs, which simplifies the storage system a lot. In the current architecture of PERK, the objects are defined in the memory of the KRS, because of which we are forced to reproduce many of the DBMS functionality in the KRS, for example, query optimization and indexing, and make the system very complex.

Our basic design of PERK-II is based around Open Knowledge Base Connectivity (OKBC) operations. For each of the mandatory OKBC operations, we would implement it by making direct Oracle Call Interface (OCI) calls to Oracle. OCI is an application programming interface (API) that allows us to create applications that use the native function calls of a third-generation language like C to access an Oracle database server and control all phases of SQL statement execution. It supports the data types, calling conventions, syntax, and semantics of C language. Thus, PERK-II acts like an OKBC wrapper for the Oracle DBMS.

In the new architecture, the cost of supporting OKBC operations on the objects resident in the Oracle client-side object cache is likely to be a little greater than that of the corresponding operations which manipulate objects resident in the memory of the KRS. For the case when the inference level of the OKBC operations is direct, this is basically due to the many OCI calls made for iterating collections and pinning objects. When the inference level is taxonomic, since PERK-II can utilize the complex object retrieval (COR) to minimize the number of network trips, the overhead of the OKBC operations on PERK-II is reduced significantly. We undertook some experiments to judge the relative difference in the cost of performing several basic OKBC operations. According to our experimental results, when the users are browsing a KB interactively with the GKB-Editor, the time difference between PERK-II and the "home grown" cache storage system is negligible.

Next, we describe the system architecture, the base tables and the object views used, and the various implementation issues of the OKBC operations. The performance results are also reported. The get-class-superclasses operation which queries the class-subclass structure in a KB, the get-frame-slots operation which queries the slots of a frame, and the get-slot-values operation which queries the values of the slot with the given slot type are used as examples in our description.

2. System Architecture

The system architectures of PERK and PERK-II are shown in Figure 1 and Figure 2 respectively.

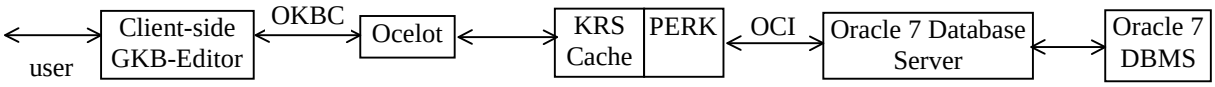

Figure 1. System architecture of PERK.

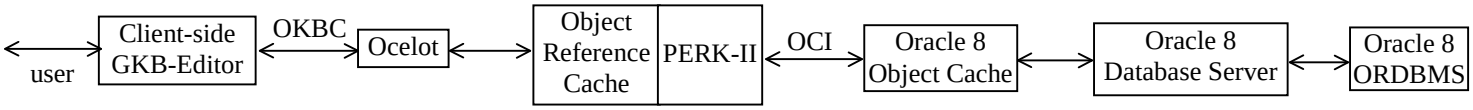

Figure 2. System architecture of PERK-II.

The components of the two systems are explained as follows:

- The GKB-Editor is a graphical user interface (GUI) for interactive KB editing and browsing.
- The Generic Frame Protocol (GFP), which is OKBC in our case, is an API for accessing KBs.
- Ocelot is our frame representation system (FRS).
- The storage system PERK allows KBs to be stored within the Oracle 7 DBMS.
- The storage system PERK-II allows KBs to be stored within the Oracle 8 ORDBMS, and cached in the Oracle 8 client-side object cache.

3. Base Tables

In the existing design of PERK, in order to store KBs in the Oracle DBMS, we define a generic mapping from the knowledge model of an FRS to the data model of a DBMS. This mapping contains a relational schema with three base tables or_kbs, or_frames, and or_slotvalues, as shown in Figure 3. These base tables are also utilized in PERK-II.

or_kbs	
kbname	id
CMCP-99	100

or_frames		
kb	frame	type
100	PEOPLE	1
100	STUDENT	1
100	KAREN	2
100	INSTANCE-OF	3

100	SUBCLASS-OF	3
100	HABIT	3

or_slotvalues				
kb	frame	slot	slottype	slotvalue
100	KAREN	INSTANCE-OF	OWN	STUDENT
100	STUDENT	SUBCLASS-OF	OWN	PEOPLE
100	KAREN	HABIT	OWN	TRAVELLING

Figure 3. Relational schema used to stored KBs in the Oracle DBMS, with sample data.

The or_kbs table contains two fields: kbname and kbid, corresponding to the name and the id of the KB respectively.

The or_frames table contains three fields: kb, frame, and type, corresponding to the KB id, the name, and the type of a certain frame respectively. Values 1, 2, and 3 of the type field correspond to class, common individual, and slot respectively.

The or_slotvalues table contains five fields: kb, frame, slot, slottype, and slotvalue, corresponding to the KB id, the frame, the slot, the type of the slot, and the value of the slot respectively. If the slot is "SUBCLASS-OF" and slottype is "OWN", then the frame is a sub-class of the slotvalue. If the slot is "INSTANCE-OF" and slottype is "OWN", then the frame is an instance of the slotvalue.

4. Object Views

In the proposed new design of PERK-II, we create several object views. The Oracle 8 client-side object cache uses the data structures defined by these object views to hold information as the objects are brought into the object cache.

4.1. class_supers_subs_view

Object view class_supers_subs_view is used for the get-class-subclasses, get-class-superclasses, and get-kb-roots operations. It has four fields: kb, class, superclasses, and subclasses, corresponding to the KB id, the name, the collection of the object references to the super-classes, and the collection of the object references to the sub-classes of a certain class respectively. The definition of the object view class_supers_subs_view is listed as follows:

```
create type classes_ref_ty as table of ref class_ty;

create type class_ty as object
(kb number(5),
class varchar2(100),
superclasses classes_ref_ty,
subclasses classes_ref_ty
);

create view class_supers_subs_view of class_ty
                with object identifier(kb, class)
as select c1.kb, c1.class,
        cast(multiset(
                select distinct make_ref(class_supers_subs_view, c1.kb, c2.slotvalue)
                from or_slotvalues c2
                where c1.kb=c2.kb and c1.class=c2.frame and c2.slot='SUBCLASS-OF' and c2.slottype='OWN')
        as classes_ref_ty),
        cast(multiset(
                select distinct make_ref(class_supers_subs_view, c1.kb, c3.frame)
                from or_slotvalues c3
                where c1.kb=c3.kb and c1.class=c3.slotvalue and c3.slot='SUBCLASS-OF' and c3.slottype='OWN')
   from (select distinct kb, frame as class
```

```
        from or_frames
        where type=1) c1;
```

4.2. frame_slots_view

Object view frame_slots_view is used for the get-frame-slots operation. It has four fields: kb, frame, slottype, and slots, corresponding to the KB id, the frame, the type of the slot, and the collection of the slots of the frame respectively. The definition of the object view slotvalue_view is listed as follows:

```
create type slots_type as table of varchar2(100);

create type frame_slots_ty as object
(kb number(5),
 frame varchar2(100),
 slottype varchar2(20),
 slots slots_type
);

create view frame_slots_view of frame_slots_ty
        with object identifier(kb, frame, slottype)
as select c1.kb, c1.frame, c1.slottype,
   cast(multiset(
          select distinct c2.slot
          from or_slotvalues c2
          where c2.kb=c1.kb and c2.frame=c1.frame and c2.slottype=c1.slottype)
        as slots_type)
   from (select distinct kb, frame, slottype
          from or_slotvalues) c1;
```

4.3. slotvalue_view

Object view slotvalue_view is used for the get-slot-values and get-frame-prettyname operations. It has seven fields: kb, frame, slot, slottype, slotvalues, classes, and superclasses, corresponding to the KB id, the frame, the slot, the type of the slot, the collection of the values of the slot, the collection of the object references to the types of the frame if the frame is an instance, and the collection of the object references to the super-classes of the frame if the frame is a class, respectively. The definition of the object view slotvalue_view is listed as follows:

```
create type slotvalues_ty as table of varchar2(500);

create type slotvalues_ref_ty as table of ref slotvalue_ty;

create type slotvalue_ty as object
(kb number(5),
 frame varchar2(100),
 slot varchar2(100),
 slottype varchar2(20),
 slotvalues slotvalues_ty,
 classes slotvalues_ref_ty,
 superclasses slotvalues_ref_ty
);

create view slot_view1(kb, frame, slot, slottype)
as select distinct kb, frame, slot, slottype
    from or_slotvalues;

create view slotvalue_view of slotvalue_ty
```

```
    with object identifier(kb, frame, slot, slottype)
as select c1.kb, c1.frame, c1.slot, c1.slottype,
          cast(multiset(
                    select distinct c2.slotvalue
                    from or_slotvalues c2
                    where c2.kb=c1.kb and c2.frame=c1.frame and c2.slot=c1.slot and c2.slottype=c1.slottype)
          as slotvalues_ty),
          cast(multiset(
                    select distinct make_ref(slotvalue_view, c1.kb, c4.frame, c1.slot, c1.slottype)
                    from or_slotvalues c3, slot_view1 c4
                    where c1.kb=c3.kb and c1.frame=c3.frame and c3.slot='INSTANCE-OF' and c3.slottype='OWN'
                              and c4.kb=c1.kb and c4.frame=c3.slotvalue and c4.slot=c1.slot and c4.slottype=c1.slottype)
          as slotvalues_ref_ty),
          cast(multiset(
                    select distinct make_ref(slotvalue_view, c1.kb, c6.frame, c1.slot, c1.slottype)
                    from or_slotvalues c5, slot_view1 c6
                    where c1.kb=c5.kb and c1.frame=c5.frame and c5.slot='SUBCLASS-OF' and c5.slottype='OWN'
                              and c6.kb=c1.kb and c6.frame=c5.slotvalue and c6.slot=c1.slot and c6.slottype=c1.slottype)
          as slotvalues_ref_ty)
  from slot_view1 c1;
```

4.4. advantages of object views

There are several advantages for using object views in our implementation:

- Object view changes the tuples of the usual relational tables into objects, then they can be cached in the client-side object cache and reduce network traffic.
- Each object in the object view has an object reference, which can be utilized to do complex object retrieval and reduce network traffic.
- Object view can group several tuples in the base table into a collection, then the collection can be fetched from the database server in one network trip, rather than fetching those tuples one by one, each with a separate network trip.
- Object view provides the flexibility of looking at the same relational data in more than one way. Thus we can use different in-memory object representations for different OKBC operations without changing the way the data are stored in the database.

5. Foreign Function Interface Issues between OCI and Lisp

5.1. pointer and content

OCI calls are C functions while OKBC calls are Lisp functions. Since the implementation of the OKBC calls uses OCI calls to access the Oracle database, the Lisp foreign function interface is needed to load the compiled OCI foreign code dynamically into the running Lisp. C language supports the get address operator &, as well as the get content operators * and ->, while Lisp doesn't. So we have to use some tricks when passing parameters from Lisp to C.

We define three Lisp functions, getpointer, getcontent, and getstring, as follows:

```
(defun getpointer(value) ; make a static array whose only element is value
   (make-array 1
     :element-type 'fixnum
     :initial-element value
     :allocation :lispstatic-reclaimable))

(defun getcontent(pointer)
   (aref pointer 0))
```

```
(defun getstring(len) ; make a static string whose length is len
  (make-array len
    :element-type 'character
    :initial-element #\null
    :allocation :lispstatic-reclaimable))
```

The first function is used to create an array whose only element is its parameter value. Each time the address of a variable var is needed, we create another variable var-pointer equal to (getpointer var), and use it as the address of the variable var. Then, when we want to get the content of the pointer var-pointer, that is, the value of var, we use (getcontent var-pointer).

The third function is used to create a string whose length is its parameter len. Each time a string variable of length len is needed, we use (getstring len) to create the corresponding character array.

5.2. type conversion between Lisp and C

When we use the Lisp function ff:defforeign to define OCI foreign functions to Lisp, the C types are converted into the Lisp types as follows:

- The C pointer type text * is defined as the Lisp type string.
- Other C pointer types, such as dvoid *, when used as pointers in Lisp, are defined as the Lisp type array. When used as ordinary values in Lisp, they are defined as the Lisp type integer.
- All the other C types are defined as the Lisp type integer.
- All the return types of the OCI functions are defined as the Lisp type integer.

If some parameters of an OCI call can be used both as Lisp pointers and ordinary values, several foreign functions with different Lisp parameter types need to be defined for the same OCI call.

5.3. garbage collection

When we create Lisp arrays and strings, and pass them as parameters to the OCI calls, they have to be created as static arrays using the getpointer and getstring function defined above, with the allocation parameter of the Lisp make-array function set to :lispstatic-reclaimable. This is because the data in a static array is placed in an area called "static space" that is not garbage collected, which means that the data is never moved and, therefore, pointers to it can be safely stored in the foreign code. Otherwise, these arrays and strings will be moved about by the garbage collector. Their pointers, passed as parameters to the OCI foreign code, will no longer be valid after the garbage collection (which occurs from time to time), and errors will occur.

The allocation parameter is set to :lispstatic-reclaimable rather than :static, then the array allocated will be automatically freed by Lisp when the last reference from a Lisp object to it disappears, and memory leaks are prevented from happening. So, besides passing the pointers of the Lisp arrays and strings to the OCI calls, we need to store their values in some Lisp variables until the program ends.

5.4. constant

All the enumerated types of OCI are treated as the Lisp type integer. The enumerated values and the OCI constants are defined as Lisp constants using the Lisp function defconstant.

5.5. ff:def-c-type

When the objects are pinned in the client-side object cache, they are stored in C structures whose members are of OCI types. When we use the Lisp function ff:def-c-type to define the corresponding Lisp structures, the OCI types are converted into Lisp types as follows:

- The OCI type OCIString * is defined as the Lisp type * :char.
- All the other OCI pointer types are defined as the Lisp type * :int.
- The OCI type OCINumber is defined as the Lisp type 22 :char.

6. Connection Operations

In our design, there is an establish-connection function that does all the connection establishing operations when the user logs into the system, right before the OKBC operations are executed. There is another end-connection function that does all the connection ending operations when the user exits the system after executing the OKBC operations.

6.1. why they are needed

The connection operations are used to allocate/deallocate the necessary handles (which are used to store information about the context, the connection, and other OCI functions or data), attach/detach the server, and begin/end the session.

6.2. why they are executed only once

Allocating/deallocating handles, attaching/detaching the server, and beginning/ending a session are all very time-consuming. So, instead of executing all these steps each time an OKBC call is made, all of them are executed only once before/after all the OKBC calls.

6.3. steps of the establish-connection function

- Allocates the environment handle, the error handle, and the server handle.
- Attaches the server.
- Allocates the user session handle and the service handle.
- Authenticates and begins the session.
- Does all the necessary initialization operations for the individual OKBC operations.

6.4. steps of the end-connection function

- Does all the necessary finalization operations for the individual OKBC operations.
- Ends the session.
- Detaches the server.
- Deallocates the handles.

There are totally about twenty read-only OKBC operations, which have been implemented and thoroughly tested with the Allegro Common Lisp environment. In the following, we select the most typical three operations: get-class-superclasses, get-frame-slots, and get-slot-values, as examples, showing how to implement the OKBC operations using the object relational capabilities of Oracle 8.

7. Implementation of the get-class-superclasses Operation

7.1. tricks used to save time

*7.1.1. hash table *class-ref-hash-table**

We maintain a global hash table *class-ref-hash-table*, which associates a class in a kb with its corresponding object reference. Given a class name and a kb number, we look up the hash table for the object reference to the class. If it exists, the object reference is retrieved from the hash table directly. Otherwise, a select SQL statement is executed to get it, and the returned object reference is inserted into the hash table. This prevents the select SQL statement from being executed when retrieving the super-classes of a previously used class.

Usually, the object references to the objects in the object views are very long, with variable lengths depending on the object identifiers used. So, instead of directly storing the object references in the hash table, we store the pointers to them there.

7.1.2. get-class-superclasses-initialize/finalize function

We have a get-class-superclasses-initialize function that is executed directly after connecting to the database. All the hash table initializing, output variable defining, input variables binding, handle allocation, descriptor allocation, attribute setting, parameter setting, and SQL statement preparation operations for the get-class-superclasses function are done there. We have another get-class-superclasses-finalize function that is executed directly before disconnecting from the database. It does all of the hash table emptying, handle deallocation, and descriptor deallocation operations. Then, these operations do not need to be repeated every time the OKBC functions are called.

*7.1.3. hash table *stmthp-hash-table**

We maintain a global hash table *stmthp-hash-table* which associates an SQL statement with its corresponding statement handle. Given an SQL statement, we look up the hash table for its statement handle. If it doesn't exist, a new statement handle is allocated for the SQL statement and inserted into the hash table, and the SQL statement is prepared for execution. Otherwise, it is fetched from the hash table directly. This prevents allocating statement handles repeatedly for the same SQL statement.

7.1.4. get-itr-pointer function

We have a get-itr-pointer function that is used to get the iterator for a collection. If the iterator doesn't exist, we use OCIIterCreate to create it in the object cache, and initialize it to point to the beginning of the collection. Otherwise, we use OCIIterInit to let it point to the beginning of the collection without creating it again.

7.1.5. pinning object

When we use the OCI call OCIObjectPin to pin an object, we set the pin_option parameter to OCI_PIN_ANY. If the object has already been in the object cache, it is returned directly without being retrieved from the database server.

7.1.6. complex object retrieval

Oracle 8's OCI supports COR, which allows an application to follow object references to a certain depth level, and pre-fetch other objects while fetching the root object. If the super-classes of all levels of a certain class are wanted (the inference-level parameter of the get-class-superclasses function is :taxonomic), the depth level of the COR descriptor is set to UB4MAXVAL for the maximum possible depth level. Otherwise, if only the direct level super-classes of a certain class are required (the inference-level parameter is :direct), the depth level of the COR descriptor is set to 1. Then, all the super-classes of a certain class can be fetched from the database server in one network trip, instead of fetching each super-class with a separate network trip.

7.1.7. fetching result

The iters parameter of the OCI call OCIStmtExecute is set to 1, which lets the execution of the select SQL statement fetch the result into the predefined buffer directly, and save the OCI call OCIStmtFetch.

7.1.8. argument checking

When we use the Lisp function ff:defforeign to define OCI foreign functions to Lisp, the arg-checking parameter is set to nil, which turns off the argument checking operations when passing parameters from Lisp to the OCI foreign functions.

7.1.9. global pointer variables

The main operation in the getpointer function is make-array, which is very time-consuming. In order to avoid using the getpointer function to allocate new arrays for the local pointer variables of the OKBC functions, we have several global pointer variables whose initial values are (getpointer 0). Then, whenever the OKBC functions are called, their local pointer variables are assigned the values of those global pointer variables, and the getpointer function isn't needed in the OKBC functions any more.

7.1.10. getcontent

The getcontent function uses the time-consuming Lisp function aref. So, whenever we need to use the getcontent function for the same parameter pointer-var several times, we just assign the value of (getcontent pointer-var) to some variable var, and use var for this value when it is needed.

*7.1.11. ff:string-to-char**

When we use the Lisp function ff:defforeign to define C foreign functions to Lisp, the C type text * is defined as the Lisp type string rather than integer, which saves the Lisp function call ff:string-to-char* when passing string parameters from Lisp to C. This is because Lisp strings can be converted to C strings correctly, though not vice versa.

7.1.12. iteration

In order to do iteration on the elements of a collection, we can use two sets of OCI calls: the set of collection functions and the set of iterator functions. According to our test result, the latter are faster than the former. So we choose the iterator functions OCIIterCreate, OCIIterInit, and OCIIterNext.

7.1.13. OCIRefIsEqual

When the inference-level parameter of the get-class-superclasses function is :taxonomic, we need to check whether or not a super-class of the given class has been fetched before, since the class-subclass structure may be a cyclic graph without consideration of the direction of the edges. There are two ways to do this: the first one is to use the OCI call OCIRefIsEqual to compare the equality of the object references; the second one is to use string comparison to compare the names of the classes. Since OCI calls are time-consuming, we choose the latter.

7.2. interface issues

7.2.1. fixnum

In the definition of the getpointer function, the element-type parameter of the Lisp make-array function is set the value of 'fixnum rather than 'integer, or else the Lisp program will produce segmentation fault. This is because in Lisp, integers can be bignums or fixnums. C integers may be larger than all possible Lisp fixnums and smaller than most (but not all) bignums. If a fixnum is passed to the C foreign code, it is always correctly represented, while a bignum won't be so unless it is small enough. So we have to use fixnum as the value of the element-type parameter rather than integer.

7.2.2. strcpy

When we get the input value for a bind string variable, we can't simply use setq to assign it to that string variable. Instead, we define the C function strcpy as a Lisp foreign function, and use this function to do so. This is because the value of the bind string variable, that is, the pointer to a character array, has already been stored in the bind handle, and we need to keep its validity.

7.3. steps of the functions

7.3.1. steps of the get-class-superclasses (:inference-level :direct) function

- Uses the get-object-reference-class function to get the object reference to the given class object in the object view class_supers_subs_view.
- If the object reference is zero, which means that the class doesn't exist, returns nil. Otherwise, pins the class object in the object cache using COR on the type CLASS_TY, with the depth level set to 1, which pre-fetches the direct level super-classes of the class.
- Gets the iterator for the collection of object references in the superclasses field of the class object.
- Does iteration on the collection. For each element in the collection, retrieves the object reference to the super-class in that element, pins the super-class in the object cache, and gets its name.
- Returns the names of the super-classes.

7.3.2. steps of the get-class-superclasses (:inference-level :taxonomic) function

- Uses the get-object-reference-class function to get the object reference to the given class object in the object view class_supers_subs_view.
- If the object reference is zero, which means that the class doesn't exist, return nil. Otherwise, pins the class object in the object cache using COR on the CLASS_TY type, with the depth level set to UB4MAXVAL, which pre-fetches the super-classes of the class of all levels.
- Pushes the object reference into the working stack. Sets the first-time variable to T.
- While the working set isn't empty, does the following:
    - Pops up an object reference from the working stack.
    - Pins the class object in the object cache.
    - Checks the name of the class. If the class has been fetched before, that is, its name has already existed in the superclass stack, forgets about it. Otherwise, does the following:
        - If the first-time variable is T, sets it to nil. Otherwise, gets the name of the class and pushes it into the superclass stack.
        - Gets the iterator for the collection of object references in the superclasses field of the class object.
        - Does iteration on the collection. For each element in the collection, retrieves the object reference to the super-class in that element, and pushes the object reference into the working stack.
- Returns the names of the super-classes in the superclass stack.

7.3.3. steps of the get-object-reference-class function

- Looks up the hash table *class-ref-hash-table* for the object reference to the given class in the kb.
- If the object reference exists, it is directly retrieved from the hash table and returned. Otherwise, does the following:
    - Binds the input variables to the select SQL statement

      select ref(v)
      from class_supers_subs_view v
      where v.class=:1 and v.kb=:2.
    - Executes the select SQL statement.
    - If the select SQL statement isn't executed successfully, returns 0. Otherwise, does the following:
        - Fetches the object reference from the resulting tuple.
        - Inserts the object reference into the hash table *class-ref-hash-table*.
        - Returns the object reference.

7.3.4. steps of the get-class-superclasses-initialize function

- Allocates the statement handle for the select SQL statement that queries the object reference to a class object.
- Prepares the select SQL statement for execution.
- Binds the input variables to the select SQL statement.
- Defines the output variable for the select SQL statement.
- Allocates the COR handle.
- Allocates the COR descriptor.
- Specifies the type of reference to be followed while constructing the complex object in the COR descriptor.
- Specifies the depth level in the COR descriptor.
- Puts the COR descriptor in the COR handle.
- Initializes the hash table *class-ref-hash-table* with zero entry.

7.3.5. steps of the get-class-superclasses-finalize function

- Frees the COR handle and descriptor.
- Frees the statement handle.
- Removes all the entries from the *class-ref-hash-table* hash table.
- Deletes the collection iterator.

8. Implementation of the get-frame-slots Operation

8.1. steps of the get-frame-slots function

- Gets the object reference to the given frame with the given slot-type in the object view frame_slots_view.
- If the object reference is zero, which means that the frame doesn't exist, returns nil. Otherwise, pins the frame object in the object cache.
- Gets the iterator for the collection of slots in the slots field of the frame object.
- Does iteration on the collection. For each element in the collection, retrieves the slot in that element.
- Returns the slots of the given frame.

8.2. tricks used

Besides the tricks used in the get-class-superclasses function, the following one is used. We know that both the :own and :template type slots of a frame are frequently retrieved together. So, when the slots of one type are found but no slots of the other type are found, we still set the object reference to the frame with the second slot type to 0 and insert it into the hash table *frame_slots-ref-hash-table*. Then, the next time the frame is retrieved, the object references to both the frames with the :own and :template slot types can be retrieved from the hash table *frame_slots-ref-hash-table* directly without executing the select SQL statement.

9. Implementation of the get-slot-values Operation

9.1. steps of the get-slot-values (:inference-level :direct) function

- Gets the object reference to the given slot of the frame (with the given slot type) in the object view slotvalue_view.
- If the object reference is zero, which means that the slot of the frame doesn't exist, returns nil. Otherwise, pins the slot object in the object cache.
- Gets the iterator for the collection of slot values in the slotvalues field of the slot object.
- Does iteration on the collection. For each element in the collection, retrieves the slot value in that element.
- Returns the slot values of the given slot of the frame.

9.2. steps of the get-slot-values (:inference-level :taxonomic) function

- Gets the object reference to the given slot of the frame (with the given slot type) in the object view slotvalue_view.
- If the object reference is zero, which means that the slot of the frame doesn't exist, return nil. Otherwise, pins the slot object in the object cache using COR on the type SLOTVALUE_TY, with the depth level set to UB4MAXVAL, which pre-fetches its slot values of all levels.
- Pushes the object reference into the working stack. Sets the first-time variable to T.
- While the working set isn't empty, does the following:
    - Pops up an object reference from the working stack.
    - Pins the slot object in the object cache.
    - Checks the frame of the slot. If the slot has been fetched before, that is, its frame name has already existed in the frames stack, forgets about it. Otherwise, does the following:
        - If the first-time variable is nil, gets the frame name of the slot and pushes it into the frames stack.
        - Gets the iterator for the collection of slot values in the slotvalues field of the slot object.
        - Does iteration on the collection. For each element in the collection, retrieves the slot value in that element, and pushes it into the slotvalues stack.
        - If the first-time variable is T, sets it to nil. Otherwise, does the following:
            - Gets the iterator for the collection of object references to slots in the classes field of the slot object.
            - Does iteration on the collection. For each element in the collection, retrieves the object reference to the slot in that element, and pushes the object reference into the working stack.

- Gets the iterator for the collection of object references to slots in the superclasses field of the slot object.
- Does iteration on the collection. For each element in the collection, retrieves the object reference to the slot in that element, and pushes the object reference into the working stack.

- Returns the names of the slot values in the slotvalues stack.

9.3. tricks used

Besides the tricks used in the get-class-superclasses function, the following one is used. We know that both the slotvalues of the :own and :template type slots of a frame are frequently retrieved together. So, when a slot of one type is found but no slot of the other type is found, we still set the object reference to the second type of slot to 0 and insert it
into the hash table *slot-ref-hash-table*. Then, the next time the slot is retrieved, the object references to both the :own and :template type slots can be retrieved from the hash table *slot-ref-hash-table* directly without executing the select SQL statement.

10. Performance

The experimental setup for comparing the performance of the "home grown" cache storage system and that of PERK-II consists of a test KB and the software used to conduct the experiments. The KB we considered covered the risk analysis domain, which has 1141 classes, 1649 instances, and 74 slots. Frames averaged 2.6 slots apiece, with an average of 1.1 fillers per slot.

Experiments were running on Allegro Common Lisp 5.0, using OCELOT 1.5. Both the FRS and the Oracle server were running on the same SUN SPARC workstation, with the operating system Solaris 5.2. Lisp was restarted before each trial to avoid caching effects, and the garbage collection time was not included. All the experiments assume a warm start for the DBMS server. Overall elapsed times were measured using the Lisp time function. Each trial was repeated enough times so that the confidence intervals on the mean elapsed time were less than 5% of its length.

10.1. Experiment 1: Interactive Browsing

The first experiment considers a scenario where users are interactively browsing a KB using the Generic Knowledge Base Editor (GKB-Editor). The users in this scenario perform the class hierarchy browsing operation frequently. Therefore, we measured the time to perform this operation for our test KB. Since PERK-II utilizes the Oracle client-side object cache, so we need to differentiate between the browsing operation in the first browse and in subsequent browses for the caching effects.

	“home grown” cache	PERK-II
first browse	1176	3855
subsequent browses	1176	1407

Table 1. Interactive browsing time (milliseconds)

In Table 1, we show the time to do a typical browse that starts from all the root classes in the KB and expands their direct level subclasses and instances. We can see that in the first browse, since the client-side object cache hasn't been utilized, the browsing time required by PERK-II is 2.5 times longer than that of the "home grown" cache storage system. That is natural, because PERK-II needs to fetch the necessary objects from the database server into the object cache, while for the "home grown" cache storage system everything is in memory. However, in subsequent browses, when the necessary objects have already been cached in the object cache, the browsing times of the two storage systems are basically the same, which is of more importance to us.

In this scenario, the operations of searching for a node, expanding it, and displaying its contents are also frequently executed. Their execution time is not reported here, but it follows a characteristic similar to the browsing time.

10.2. Experiment 2: Operation-level Comparison

In the second experiment, we compare the times to execute individual OKBC operations using the "home grown" in-memory cache storage system and PERK-II respectively.

We test four sets of operations:
(1) OKBC operations on the "home grown" in-memory cache storage system,
(2) our own operations on PERK-II utilizing the object cache,
(3) OKBC operations on PERK-II utilizing the object cache,
(4) OKBC operations on PERK-II without utilizing the object cache.
The execution time of the first three sets of operations is shown in Figure 4, 5, 6, 7, and 8, while the execution time of the last set of operations is too long to be shown in the figures.

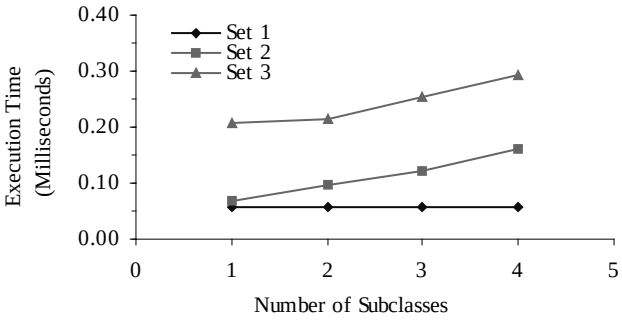

Figure 4. Execution time of get-class-subclasses (:inference-level :direct) operation.

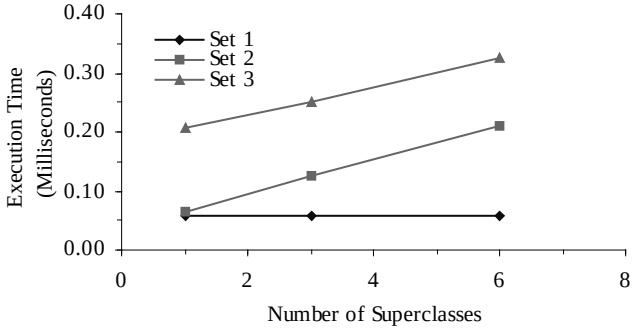

Figure 5. Execution time of get-class-superclasses (:inference-level :direct) operation.

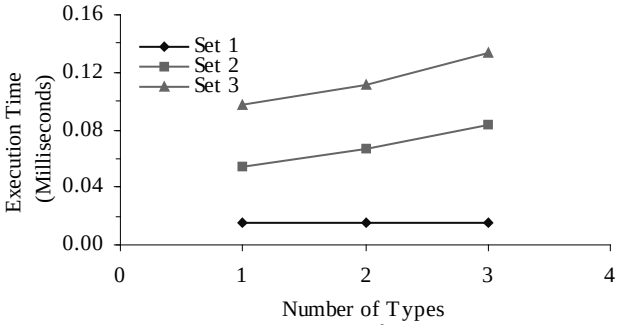

Figure 6. Execution time of get-instance-types (:inference-level :direct) operation.

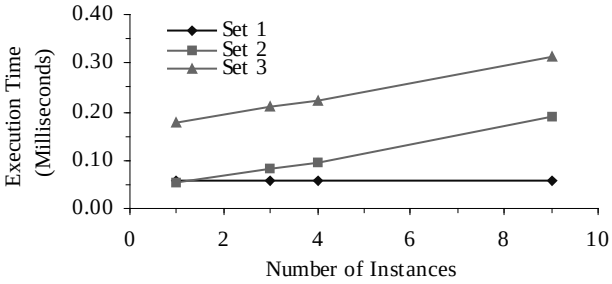

Figure 7. Execution time of get-class-instances (:inference-level :direct) operation.

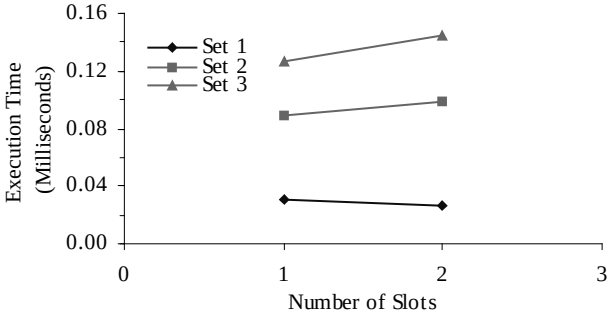

Figure 8. Execution time of get-frame-slots (:inference-level :direct) operation.

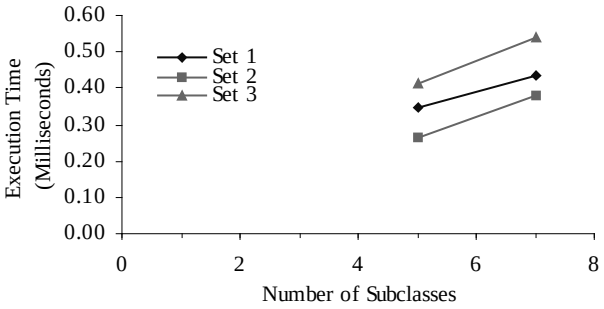

Figure 9. Execution time of get-class-subclasses (:inference-level :taxonomic) operation.

There isn't a very significant time difference between the OKBC operations on the "home grown" cache storage system and on PERK-II when the object cache is utilized. The reason the OKBC operations on PERK-II need a little more time is mainly due to the OKBC call overhead, and the many OCI calls made to do collection iteration and object pinning. Especially when the inference-level of the OKBC operations is taxonomic, since PERK-II can utilize COR to pre-fetch objects while fetching the root object, its OKBC operations don't need much more time than that of the "home grown" cache storage system. Most importantly, when we use the GKB-Editor to do these operations, to the user, the time difference between the two storage

systems is negligible. So, in practice, we can assume that the performance of PERK-II is as good as that of the "home grown" cache storage system, while the former is greatly simplified.

11. Comparison of the 007 Benchmark and our Performance Tests

The 007 Benchmark represents a comprehensive test of OODBMS performance, while we only test part of the Oracle 8 ORDBMS performance. The 007 Benchmark tests the speed of many different kinds of pointer traversals, including traversals over cached data, traversals over disk-resident data, sparse traversals, and dense traversals, while we only test traversals over cached data and traversals over disk-resident data. The 007 Benchmark tests the efficiency of many different kinds of updates. Since we haven't implemented the update OKBC operations, such test can't be done at present. The 007 Benchmark tests the performance of the query processor on several different types of queries, including exact match lookup, range query, and join query, while we only tested the exact match lookup.

12. Future Work

Thus far, we only implemented the OKBC read-only operations on PERK-II and integrated them into the GKB-Editor, and haven't dealt with the OKBC operations related to facets and some OKBC read-only operations whose inference-level is taxonomic. In the future, we will implement all the other OKBC operations that are missed at present and complete the whole OKBC knowledge model on PERK-II.

13. Summary and Conclusions

We have shown how to implement OKBC using the object relational capabilities provided by Oracle 8. Here, the Oracle object cache is utilized to do client-side buffer management automatically, and the DBMS functionality like query optimization and indexing are pushed into the DBMS where it belongs, which simplifies the system a lot. Also, object view and COR are used to reduce network traffic and improve efficiency. So, the new storage system PERK-II can achieve high efficiency without sacrificing simplicity. In practice, we can assume that PERK-II has as good performance as the "home grown" cache storage system.

In addition, since PERK-II utilizes a KB-independent DMBS schema, it can be used with different KBs in a fashion that is transparent to the user. Also, it is highly scalable for large KBs, and portable to different kinds of FRSs.

In conclusion, PERK-II overcomes the most significant shortcoming of PERK, keeps all the important advantages of PERK, and is greatly simplified.

Acknowledgments

This work was supported by DARPA under contract number N66001-97-C-8551. Gang Luo contributed to this research project during his stay at the Artificial Intelligence Center (AIC) of SRI International.

References

1. Peter D. Karp, Vinay K. Chaudhri, and Suzanne M. Paley. A Collaborative Environment for Authoring Large Knowledge Bases. Technical report, Submitted for publication, April 1997.
2. Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. OKBC: A Programmatic Foundation for Knowledge Base Interoperability. In Proceedings of the 1998 National Conference on Artificial Intelligence, Madison, Wisconsin, 1998.
3. Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. Open Knowledge Base Connectivity 2.0.3. Technical report, Submitted for publication, April 1998.
4. Michael J. Carey, David J. Dewitt, and Jeffrey F. Naughton. The 007 Benchmark. In Proceedings of the ACM SIGMOD Conference, Washington D.C., May 1993.
5. Oracle 8i Documentation, Release 8.1.5, Oracle Corporation, 1999.
6. Allegro CL User Guide, version 4.3, Franz Inc., 1996.
7. Geiger, K. Inside ODBC, Microsoft Press, 1995.

8. Sonya E. Keene. Object-Oriented Programming in Common Lisp, Addison-Wesley Publishing Company, 1988.
9. Robert Wilensky. Common Lispcraft, W.W.Norton & Company, Inc., 1986.
10. Guy L. Steele Jr. Common Lisp the Language, second edition, Butterworth-Heinemann, 1990.

Appendix

1. OKBC Operations Implemented By Far

- establish-connection
- close-connection
- get-class-superclasses (:inference-level :direct)
- get-class-superclasses (:inference-level :taxonomic)
- get-class-subclasses (:inference-level :direct)
- get-class-subclasses (:inference-level :taxonomic)
- get-kb-roots
- get-instance-types
- class-p
- individual-p
- coerce-to-frame
- get-frame-name
- slot-p
- coercible-to-frame-p
- get-slot-values (:inference-level :direct)
- get-slot-values (:inference-level :taxonomic)
- get-frame-pretty-name
- get-frame-slots
- get-kb-frames
- get-class-instances
- openable-kbs (using openable-kb-names and get-kb-id-from-name)

2. Base Tables and Indexes

In our design, we use the following base tables and indexes:

2.1. or_kbs

```
create table or_kbs
(kbname varchar2(20),
  kbid number(5),
 constraint or_kbs_key primary key(kbname, kbid)
);
create index or_kbs_index1 on or_kbs(kbname);
```

The or_kbs table contains two fields: kbname and kbid, corresponding to the name and the id of the KB.

2.2. or_frames

```
create table or_frames
(kb number(5),
 frame varchar2(100),
 type number(1),
 constraint or_frames_key primary key(kb, frame, type)
```

```
);
create index or_frame_index1 on or_frames(kb, frame);
create index or_frame_index2 on or_frames(kb);
```

The or_frames table contains three fields: kb, frame, and type, corresponding to the KB id, the name, and the type of a certain frame. Values 1, 2, and 3 of the type field correspond to class, common individual, and slot, respectively.

2.3. or_slotvalues

```
create table or_slotvalues
(kb number(5),
 frame varchar2(100),
 slot varchar2(100),
 slottype varchar2(20),
 slotvalue varchar2(500),
 constraint or_slotvalues_key primary key(kb, frame, slot, slottype, slotvalue)
);
create index or_slotvalues_index1 on or_slotvalues(kb, frame, slottype);
create index or_slotvalues_index2 on or_slotvalues(kb, frame, slot, slottype);
create index or_slotvalues_index3 on or_slotvalues(kb, slot, slottype, slotvalue);
```

The or_slotvalues table contains five fields: kb, frame, slot, slottype, and slotvalue, corresponding to the KB id, the frame, the slot, the type of the slot, and the value of the slot respectively. If the slot is "SUBCLASS-OF" and slottype is "OWN", then the frame is a sub-class of the slotvalue. If the slot is "INSTANCE-OF" and slottype is "OWN", then the frame is an instance of the slotvalue.

3. Object Views

We transform the base tables into object views as follows:

3.1. frame_view

Object view frame_view is used for the class-p, individual-p, slot-p, coerce-to-frame, and coercible-to-frame-p operations.

```
create type frame_ty as object
(frame varchar2(100),
 type number(1),
 kb number(5)
);

create view frame_view of frame_ty
     with object identifier(frame, type, kb)
as select distinct frame, type, kb
    from or_frames;
```

3.2. kb_view

Object view kb_view is used for the get-kb-frames operation.

```
create type kb_frames_ty as object
(kb number(5),
 frames frames_type
);

create view kb_view of kb_frames_ty
```

```
                with object identifier(kb)
as select c1.kb,
        cast(multiset(
                select distinct c2.frame
                from or_frames c2
                where c2.kb=c1.kb)
        as frames_type)
   from allkb c1;
```

3.3. class_supers_subs_view

It is described in section 4.1 of this paper.

3.4. class_instances_view

Object view class_instances_view is used for the get-class-instances operation.

```
create type class_instances_ty as object
(kb number(5),
 class varchar2(100),
 instances instances_type
);

create view class_instances_view of class_instances_ty
                with object identifier(kb, class)
as select c1.kb, c1.class,
        cast(multiset(
                select distinct c2.frame
                from or_slotvalues c2
                where c2.kb=c1.kb and c2.slotvalue=c1.class and c2.slot='INSTANCE-OF' and c2.slottype='OWN')
        as instances_type)
   from classes c1;
```

3.5. instance_view

Object view instance_view is used for the get-instance-types operation.

```
create type instance_ty as object
(kb number(5),
 instance varchar2(100),
 types types_ty
);

create view instance_view of instance_ty
        with object identifier(kb, instance)
as select c1.kb, c1.instance,
        cast(multiset(
                select distinct c2.slotvalue
                from or_slotvalues c2
                where c1.kb=c2.kb and c1.instance=c2.frame and c2.slot='INSTANCE-OF' and c2.slottype='OWN')
        as types_ty)
    from instances c1;
```

3.6. frame_slots_view

It is described in section 4.2 of this paper.

3.7. slotvalue_view

It is described in section 4.3 of this paper.

4. Unpin

In our current implementation of the OKBC functions using PERK-II, in order to avoid the time-consuming unpin operations in each OKBC call, we don't unpin the objects after we pin them in the object cache. However, if there are too many objects being pinned in the object cache, some of them which are not used at that moment have to be unpinned. Otherwise, the object cache will soon overflow. Our suggestion is to do (OCICacheUnpin *envhp* *errhp* *svchp*) after each group of several OKBC calls, which hasn't been implemented in our code so far.

5. Convention

In our own functions, all the kbs are treated as integers; all the frames, classes, individuals, instances, slots, slottypes, and slot values are treated as strings.

6. Implementation of the OKBC Operations

In the following, we describe the implementation of the OKBC operations one by one.

6.1. establish-connection

It is described in section 6.3 of this paper.

6.2. close-connection

It is described in section 6.4 of this paper.

6.3. get-class-superclasses (:inference-level :direct)

It is described in section 7 of this paper.

6.4. get-class-superclasses (:inference-level :taxonomic)

It is described in section 7 of this paper.

6.5. get-class-subclasses (:inference-level :direct)

It is basically the same as get-class-superclasses (:inference-level :direct).

6.6. get-class-subclasses (:inference-level :taxonomic)

It is basically the same as get-class-superclasses (:inference-level :taxonomic).

6.7. get-kb-roots

Steps of this function are described as follows:

- Executes the select SQL statement

```
select distinct ref(c1)
from class_supers_subs_view c1, table(c1.superclasses) (+) c2
where c1.kb=:1
group by c1.kb, c1.class
```

having count(c2.COLUMN_VALUE)=0.

- For each resulting tuple, does the following:
    - Gets the object reference to the class from the tuple.
    - Pins the class object in the object cache.
    - Gets the name of the class.
- Returns the names of the classes.

6.8. get-instance-types

Steps of this function are described as follows:

- Gets the object reference to the given instance in the object view instance_view.
- If the object reference is zero, which means that the instance doesn't exist, returns nil. Otherwise, pins the instance object in the object cache.
- Gets the iterator for the collection of instance types in the types field of the instance object.
- Does iteration on the collection. For each element in the collection, retrieves the instance type in that element.
- Returns the types of the given instance.

6.9. class-p

Steps of this function are described as follows:

- Gets the object reference to the given frame in the object view frame_view.
- If the object reference is zero, which means that the frame doesn't exist, returns nil. Otherwise, pins the frame object in the object cache.
- Gets the value in the type field of the class object.
- If the value is 1, then the frame is a class, and T is returned. Otherwise, nil is returned.

6.10. individual-p

6.10.1. steps of the individual-p function

- Gets the object reference to the given frame in the object view frame_view.
- If the object reference is zero, which means that the frame doesn't exist, returns nil. Otherwise, pins the frame object in the object cache.
- Gets the value in the type field of the class object.
- If the value isn't 1, then the frame is an individual, and T is returned. Otherwise, nil is returned.

6.10.2. trick used

We implement the individual-p function without using the class-p function. This is to deal with the case when the object reference to the frame is zero. In this case, the frame doesn't exist, and both the individual-p and class-p functions should return nil.

6.11. coerce-to-frame

Steps of this function are described as follows:

- Gets the object reference to the given frame in the object view frame_view.
- If the object reference is not zero, returns the frame. Otherwise, if the object reference is zero, which means that the frame doesn't exist, does the following:
    - If the error-p parameter is T, a not-coercible-to-frame-p error is signaled.
    - If the error-p parameter is nil, nil is returned.

6.12. get-frame-name

This function returns the symbol name of the frame.

6.13. slot-p

Steps of this function are described as follows:

- Gets the object reference to the given frame in the object view frame_view.
- If the object reference is zero, which means that the frame doesn't exist, returns nil. Otherwise, pins the frame object in the object cache.
- Gets the value in the type field of the class object.
- If the value is 3, then the frame is a slot, and T is returned. Otherwise, nil is returned.

6.14. coercible-to-frame-p

Steps of this function are described as follows:

- Gets the object reference to the given frame in the object view frame_view.
- If the object reference is zero, which means that the frame doesn't exist, returns nil. Otherwise, returns T.

6.15. get-slot-values (:inference-level :direct)

It is described in section 9 of this paper.

6.16. get-slot-values (:inference-level :taxonomic)

It is described in section 9 of this paper.

6.17. get-frame-pretty-name

Steps of this function are described as follows:

- Sets the slot to be "PRETTY-NAME".
- Sets the slot type to be :own.
- Sets the inference-level to be :direct.
- Calls the get-slot-values function to get the pretty name of the frame.

6.18. get-frame-slots

It is described in section 8 of this paper.

6.19. get-kb-frames

Steps of this function are described as follows:

- Gets the object reference to the given kb in the object view kb_frames_view.
- If the object reference is zero, which means that the kb doesn't exist, returns nil. Otherwise, pins the kb object in the object cache.
- Gets the iterator for the collection of frames in the frames field of the kb object.
- Does iteration on the collection. For each element in the collection, retrieves the frame in that element.
- Returns the frames of the given kb.

6.20. get-class-instances

Steps of this function are described as follows:

- Gets the object reference to the given class in the object view class_instances_view.
- If the object reference is zero, which means that the class doesn't exist, returns nil. Otherwise, pins the class object in the object cache.
- Gets the iterator for the collection of instances in the instances field of the class object.
- Does iteration on the collection. For each element in the collection, retrieves the instance in that element.
- Returns the instances of the given class.

6.21. openable-kbs (using openable-kb-names and get-kb-id-from-name)

6.21.1. steps of the openable-kb-names function

- Gets the object reference to the object in the object view kbnames_view.
- Pins the object in the object cache.
- Gets the iterator for the collection of kb names in the kbnames field of the object.
- Does iteration on the collection. For each element in the collection, retrieves the kb name in that element.
- Returns the kb names.

6.21.2. steps of the get-kb-id-from-name function

- Binds the kb name to the select SQL statement
  select kbid
  from or_kbs
  where kbname=:1.
- Executes the select SQL statement.
- Fetches the resulting tuple and gets the kb id.

7. Performance

The numerical execution time of the four sets of operations in experiment 2 is listed in Table 2 and 3 (the number in the braces represents how many values the operation returns).

	Set 1	Set 2	Set 3	Set 4
get-class-subclasses (1)	0.058	0.068	0.208	17.25
get-class-subclasses (2)	0.057	0.095	0.215	23.99
get-class-subclasses (3)	0.057	0.123	0.255	34.51
get-class-subclasses (4)	0.058	0.161	0.293	44.47
get-class-superclasses (1)	0.057	0.066	0.206	24.94
get-class-superclasses (3)	0.057	0.126	0.252	31.38
get-class-superclasses (6)	0.058	0.209	0.324	73.20
class-p	0.020	0.028	0.046	5.493
get-instance-types (1)	0.015	0.054	0.097	6.730
get-instance-types (2)	0.015	0.067	0.112	6.913
get-instance-types (3)	0.016	0.084	0.134	7.340
get-class-instances (1)	0.056	0.052	0.178	10.73
get-class-instances (3)	0.057	0.083	0.210	11.00
get-class-instances (4)	0.059	0.094	0.224	11.21
get-class-instances (9)	0.056	0.191	0.314	11.42
get-frame-slots (1)	0.030	0.089	0.127	7.030
get-frame-slots (2)	0.026	0.099	0.145	7.300
get-slot-value (1)	0.135	0.080	0.230	12.81
get-frame-pretty-name	0.174	0.064	0.101	7.940
get-frame-name	0.0078	0.010	0.038	0.040

coerce-to-frame	0.0074	0.008	0.015	1.600
get-kb-roots	1.532	0.007	0.009	489.5
get-kb-frames	0.003	38.51	38.25	683.8

Table 2. Execution time (milliseconds) of the direct inference-level OKBC operations.

	Set 1	Set 2	Set 3	Set 4
get-class-subclasses (5)	0.348	0.266	0.412	35.30
get-class-subclasses (7)	0.433	0.381	0.542	50.46
get-class-superclasses (4)	0.383	0.246	0.391	33.85
get-slot-values (2)	0.297	0.181	0.33	18.71

Table 3. Execution time (milliseconds) of the taxonomic inference-level OKBC operations.