## Title

ERDRAW: A Graphical Schema Specification Tool Reference Manual Draft 2.2



## Authors

Szeto, E

Markowitz, V M

## Publication Date

1991-05-01

PUB-3084

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

Information and Computing Sciences Division

# ERDRAW: A Graphical Schema Specification Tool

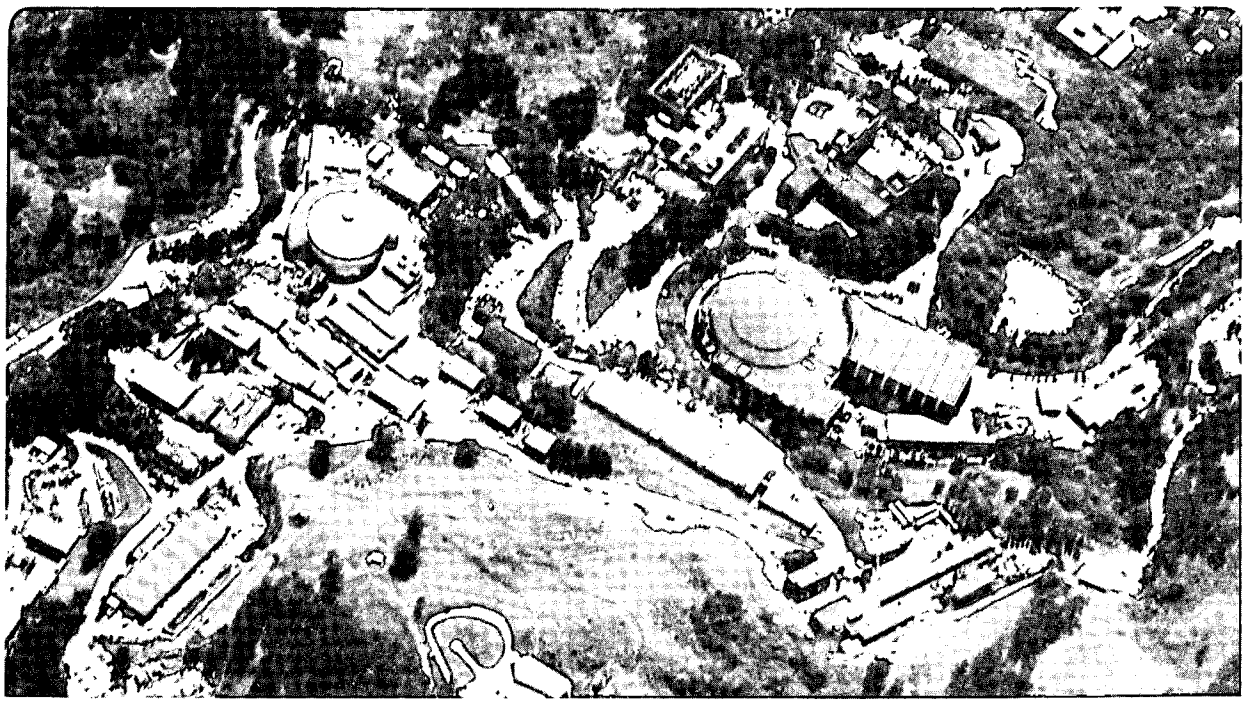
## Reference Manual

## Draft 2.2

E. Szeto and V.M. Markowitz

May 1991

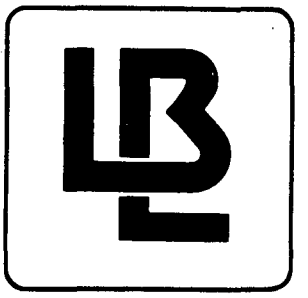






Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098

## DISCLAIMER



# *ERDRAW*

# A GRAPHICAL SCHEMA SPECIFICATION TOOL*

**Reference Manual**

DRAFT 2.2

**Ernest Szeto**†

**Victor M. Markowitz**‡

Data Management Group

Information and Computing Sciences Division

Lawrence Berkeley Laboratory

1 Cyclotron Road

Berkeley, CA 94720

May 1, 1991



* Issued as Technical Report LBL--PUB--3084. This work is supported by the Office of Health and Environmental Research Program and the Applied Mathematical Sciences Research Program, of the Office of Energy Research, U.S. Department of Energy, under Contract DE-AC03-76SF00098.

† Author's E-mail address: *szeto@rtsg.ee.lbl.gov*

‡ Author's E-mail address: *V_Markowitz@lbl.gov* Office phone number:(415) 486-6835

§ E-mail address: *A_Shoshani@lbl.gov* Office phone number:(415) 486-5171

# OVERVIEW

This document describes an X-window based tool called ***ERDRAW*** that supports graphical specifications of conceptual database schemas. ***ERDRAW*** is based on an *Extended Entity–Relationship* (EER) model; an overview of this model is provided in appendix B. ***ERDRAW*** generates an output file that can be used as input to a database schema design and translation tool, called ***SDT***. A full description of ***SDT*** can be found in the ***SDT*** reference manual issued as technical report LBL-27843.

***ERDRAW*** was implemented using C, and the X11 based Xlib and Xview toolkit, on Sun 3 and Sun 4 workstations under under Sun Unix OS 4.0.3. and Sun Unix OS 4.1.

This document consists of the following sections:

1. **User's Manual**, written following the format of Unix manual entries.
2. Screen images illustrating the work with ***ERDRAW***.
3. A pictorial tutorial on how to use ***ERDRAW***.
4. A sample postscript output.
5. The ***SDT*** command, input format, and a sample input file.
6. A sample report listing attributes.
7. A sample report listing subject terms.

This document also contains two appendices:

A. **Technical Documentation** on the data structures and algorithms underlying ***ERDRAW***.

B. An overview of the EER model.

## NEW FEATURES

Compared with ***ERDRAW* 1.3**, ***ERDRAW* 2.2** has the following additional features:

1. Description fields (up to 255 characters) have been added for descriptions of EER objects as well as their attributes.
2. *Instance Subject Terms* (IST) can be described for an entity set.
3. *Object Set Subject Terms* (OST) can be defined for a given schema; these terms can be associated with either entity-sets or relationship-sets.
4. *Attribute Subject Terms* (AST) can be defined globally for a given schema; these terms can be associated with attributes of EER object-sets.
5. A new report for printing out the information on subject terms has been added to the **File** pulldown menu. This ascii file report contains all the subject terms, their associations with EER objects and attributes, and their associations with each other. A diagnostic section of the report also shows all the immediate and remote parents (more generic terms) for each subject term. Loop detection is included in this predecessor report. Other submenu options such as **Load, Save, To SDT,** and, **Attributes List** have been revised accordingly, to incorporate the new changes.
6. In **Insert Attributes** mode, ***ERDRAW* 2.2** displays the additional flags **IST, OST,** and **AST** for instance, object, and attribute subject terms associated with an EER object or its attributes.
7. In addition to *Sybase 4.0* datatypes, ***ERDRAW* 2.2** allows specifying *Ingres 6.3* datatypes.

## CONTENTS

# 1. USER'S MANUAL

NAME

***ERDRAW*** - Extended Entity Relationship diagram Drawing tool.

USAGE

**erdraw** [LoadFile]

DESCRIPTION

***ERDRAW*** is an X11 based drawing program for drawing Extended Entity Relationship diagrams. Drawings can be outputted to postscript files and printed out on a postscript printer. Drawings can also be outputted to files that can be read by the Schema Design Tool (***SDT***).

The program has been used with olwm (the Open Look Window Manager) and twm (Tab/Tom's Window Manager) running X11 Release 4. Note that the procedure for sizing and relocating the ***ERDRAW*** window is dependent upon the window manager you happen to be using at the time. Consult your X11 manuals.

***ERDRAW*** was written using the Xview toolkit and Xlib. In using the Xview toolkit, it follows the Open Look Interface conventions on most usages. The right mouse button generally connotes "menu". It can used to activate the pop-up menu on the drawing canvas or the menus in the menu buttons, "File...", "Edit...", and "Add...". The left mouse button is used to "select" items, such as buttons, canvas EER objects to be moved, deleted, or for inserting attributes and labels. ***ERDRAW*** has various "drawing modes", such as for adding entity sets, relationship sets, and various kinds of arcs. Besides adding various graphical objects, drawing modes include moving objects from their original location and deleting objects. The following is a brief description of how to perform certain tasks using in the various drawing modes:

(1) Adding EER objects. Use either the "Add..." menu button or the canvas pop-up menu the select the mode "Add Entity Set" or "Add Relationship Set". To instantiate a new EER object with one of these two modes, press down the left mouse button at the desired canvas location. Hold down the left mouse button and drag the EER object to the desired location. Finalize this location by releasing the left mouse button.

(2) Adding Arcs. You can add arcs to connect already existing EER objects. Go into one of the drawing modes for adding arcs such as "Add Arc 1", "Add Arc M Mandatory", or "Add Arc ISA", etc. Draw a line from the source EER object to the target EER object. You can draw a line by pressing down the left mouse button to anchor a starting point, holding down the mouse button and dragging out the line, and releasing the mouse button at it's target point. The arc, represented by a line with an arc label in the middle, will be set between these two points. An arrowhead will indicate the ending point. The starting and ending points of the arc should be near (i.e., touching or slightly inside the EER object) one of the "standard" North, East, West, or South connecting points on the EER object. The program will "snap" the drawn line into it's correct placement.

(3) Adding Arcs with Roles. To add arcs with roles, you need to go through a three step process. Using the drawing mode "Add Role Label", you add a role label just as you add an entity or relationship object. The role label will have some dummy value such as "Role_Label". You then go into the "Add Role Line" drawing mode to draw a line connecting the starting EER object to the role label, then another line from the role label to the target EER object. EER object 1 → Role Label → EER object 2, indicates the connections. Role labels have the same standard North, West, East, and South connecting points.

(4) Adding Text Objects. "Text objects" is a crude facility for allowing users to add text on a page. It follows the same paradigm for adding role labels, entity, and relationship sets. Typical "text objects" to annotate a page are "Page 1", "Fig. 1a", and "~ denotes duplicate entity". Note that since cardinality for arcs with role labels are not shown, these arcs can be annotated with the cardinality using "text objects". Key attributes for EER objects can also be shown on the drawing using the same mechanism.

(5) Graphical Editing. After adding your graphical objects, entity sets, relationship sets, role labels, arcs, and text objects, you will probably want edit these objects, such as nudge them to different locations, or delete them entirely. The drawing modes "Move Objects" and "Delete Objects" allow you to do this kind of editing. You can move an object by going into the "Move Objects" drawing mode, and dragging the EER object (role label, or "text object") to its new location using the left mouse button. The connected arcs will stretch and shrink accordingly to remain connected to the objects. To delete objects, go into the "Delete Objects" drawing mode, and click on the object targeted for deletion. It will disappear, along with all its connected arcs and attributes. Arcs can be individually

deleted by clicking directly on the arc label in the middle of the arc. Note that moving an EER object is oftentimes a good way to get more nicely appearing lines in arcs. Due to lack of screen resolution, lines oftentimes look "jagged" when drawn at certain angles. Moving an EER object adjusts the connected lines and the angles in which they are drawn. Note that the canvas picture can be damaged. For e.g., "text objects" can leave pixel droppings, or erase parts of an underlying object when the user moves the objects around, esp. at times when they overlap each other. (Text objects cause the most damage, due to the way they are drawn by the program.) You can repair a damaged drawing by hitting the REFRESH button. The refresh button will clear the canvas, and redraw everything from the graphical parameters in the program's data structure.

(6) Adding Labels and Attributes. To label and add attributes to your EER objects, go into the "Insert Attributes" drawing mode. Click on the EER object, role label, or "text object", for inserting the desired text label. The appropriate form will pop up to allow you to type in the new value. For EER objects and Role Labels consisting of more than one word, a single token will be created by the program using the words with underscores substituted for spaces. Note that everything is centered. If you want to left justify text objects, you might need to move it to its correct location later on. Labels for EER objects, entered as separate words (tokens separated by spaces/underscores) will automatically be broken up to as much as three lines, if a single line cannot contain all the words. (Avoid mixed case letters without spaces or underscores to indicate separate words. The program does not know how to "wrap" these words.) EER objects have an additional form besides the "object label" form. If you push the "Attributes" button, it will pop up another form for entering attributes (name, datatype, length, attribute type, etc.). You can fill out the form, then hit the "append" button to append it to the EER object. You can browse the added attributes with the "begin", "end", "next", and "previous" buttons. The "insert" button allows you to insert an attribute before the current one. The "scroll list" button opens up yet another form for the user to see a list of attribute names already entered. In the scroll list form, the user can scroll down to a desired attribute name, and select that attribute as the "current" one. Be sure to exit these various layers of forms with the "last opened, first closed" discipline. (The program will force much of this discipline. However, X allows you to iconify a window, so there is a possibility for confusion.)

(7) Adding a new page and duplicate EER objects. Databases can get quite large and arc connections quite entangled. Drawing EER objects on separate pages allows the user to reduce this clutter. To add a new page, select "New Page" under the "Add..." menu button. Use "First Page", "Last Page", "Next Page", and "Previous Page" in the "Page" menu button to select the current page. An EER object can be represented on more than one page. An "original" EER object should be instantiated in an earlier page. "Duplicate" EER objects, usually on another page, have the same name as the original EER object except their names are preceded by a tag such as '~'. For example, an original entity "project" might appear on page 1, then reappear again on page 2 as a duplicate entity labeled "~project". Original and duplicate EER objects represent essentially the same EER object. They share the same attributes. They are outputted as a single EER specification for *SDT*.

(8) Outputting to External Files. Under the "File..." menu button, one can "Save" a drawing to an external file, "Quit" the utility, startup the utility and "Load" the saved file again. One can output the drawing to a postscript file intended for printing by selecting "To Postscript". The user can output either in portrait (default) or landscape page orientation. (The program will center and scale your pages accordingly.) One can output to a file intended as the input to *SDT* by selecting "To SDT". An ascii attributes report was also added to this menu list. The following filename extension conventions are suggested: *.erd for save/load *ERDRAW* files and *.ps for postscript files. *.sdt extension can be used for input file to SDT. The root (everything before the first dot) in the filename will be taken as the database name by SDT. In v2.x, ERDRAW will automatically suggest an output filename by taking the root of the input (loaded) filename and adding the proper extensions.

(9) Undo. An UNDO command has been added to the "Edit ..." menu. It will undo adding, deleting, and the moving of graphical objects for one level. It will not undo operations on individual attributes.

(10) Index. The button "ER Index" has been added. It allows the user to see all the entity and relationship set names in alphabetical order in a popup window. The user can select a single name, then go to the page on which it is drawn. The user can also edit attributes based on the selected name.

(11) Use with SDT. After an SDT file has been generated, *SDT* can be executed for generating

DBMS database definitions. These definitions are contained in three files: (1) *_relations.* file contains the table definitions (you may need to edit the beginning of this file if you manually create an empty database first with customized options); (2) *_keys.* file contains the index and key definitions; (3) *_refint.* file contains the referential integrity (e.g. Sybase trigger) procedures. Other two additional files are: (4) *_check.* file contains procedures for verifying the consistency of an existing (already loaded) database with regard to a set of referential integrity constraints. All these files can be loaded by redirecting the standard input to an ascii based SQL utility, such as isql for Sybase.

(12) Metadatabase. An ***ERDRAW*** metaschema file called Metaschema.erd is provided for setting up the metadatabase. Copy this template file to a new name, one that associates the metadatabase name with the database name. Use ***ERDRAW*** for creating an ***SDT*** file for it. *Note*: Make sure you do NOT change the metaschema, since ***SDT*** assumes a *standard* predefined metaschema. Execute ***SDT*** for generating the database definition for the metadatabase, and load these files for creating the metadatabase. The metadata can be loaded using the *_meta.* file mentioned above. *Note*: You may need to edit the beginning of this file if the metadatabase name generated by ***SDT*** does not match the intended metadatabase.

(13) Changing Startup Defaults. The user can change the startup defaults by editing the ascii startup defaults file "erdraw.defaults". The most common changes pertain to changing the dimensions of the entity rectangle (which are all the same within one drawing application) or relationship diamond, or arc labels. Be careful about changing the "duplicate" EER object marker, '~', since old applications created using the old convention will no longer be compatible with the new one. Old EER names represented by "~<name>" will no longer mean the "duplicate" EER object. The program will not work correctly for inserting attributes. Other changes can also cause the program to work incorrectly, so be careful!!! (You can comment out the original line(s) or backup the original erdraw.defaults file before experimenting.)

(14) Using Subject Terms. This is an optional feature, but useful for users who want to use query utilities that allow for searching for objects and attributes based on subject terms.

(14.a) Adding and Deleting Instance Subject Terms.

Adding IST's requires performing the stages described below.

(i) Press down the right-most mouse menu button on the drawing canvas. A popup

menu will show up. Select the **Insert Attributes/Labels** drawing mode.

(ii) Select an entity set (represented by rectangles) by clicking on it.

(iii) A window will pop up. It will have the button **Instance Subject Terms**; click on this button.

(iv) A second window containing a list of IST's (if any exist for this entity set) will show up. This window also has an **Add** button; click on the **Add** button.

(v) A third window for data entry will appear. Enter the new subject term. Words separated by spaces will automatically get underscores to form a legal database identifier. Enter the list of broader terms. Broader terms should be separated by commas, such as in "organization, authority". For the first IST of a given entity set, the name of the entity set is presented as the initial broader term. Click on the *description* subwindow, and type in the description. You can break up long description lines with *carriage-returns* for data entry readability. *Note* : user *carriage-returns* are not stored; the description words are automatically wrapped as the program sees fit next time it is displayed.

(vi) Hit the **Add** button in this window to save the entered material. Note that you can type in broader terms that do not yet exist, or even broader terms that form a loop. The program does not enforce integrity, but will warn you of these conditions when you hit the **Add** button. These anomalies will also show up when you later select **Subject Terms List** submenu option under the **File** menu button in a diagnostic report. (This design was selected so as to make the programmer's job easier as well as for the person doing data entry. Maintaining integrity for a hierarchy of nodes is not an easy job, especially when some node in the middle is changed. The user should also be allowed to enter and edit subject terms "out of sequence" without having to be blocked by dependent conditions, some of which could be quite involved.)

(vii) After the whole list of *Instance Subject Terms* has been added, you can click on the **Done** button to exit the current window and return to the previous one.

For deleting IST's you should have a scrollable list of IST's available to you in a popup window from the steps described above. Note that you can move the mouse pointer near the terms and see the first few parents (broader terms) of each subject term in the list. You can also select one or more subject terms (or none) and hit the **Show** button to see a more detailed specification on each subject term (i.e.,

including all its broader terms and description) in another popup display-only window. To delete subject terms, you need to (i) select the subject term(s) to be deleted by clicking on the term(s); and (ii) hit the **Delete** button and the selected terms will disappear.

(14.c) Adding and Deleting Object Set Subject Terms.

This procedure is very similar to adding and deleting IST's (the same popup windows used for IST is also used for OST's), except for the first couple of steps. It is assumed that you have exited from all popup windows and the menu buttons at the top are active and selectable.

(i) Select the **Subject Terms** menu button at the top with the right mouse button.
(ii) Select submenu option **Define for Object Sets.** You should now be a similar situation for adding and deleting subject terms as described previously for IST's. Note, however, that these terms are not attached to any EER objects yet. They belong to a global pool of OST's that can be associated or disassociated from EER objects in the steps described below.

(14.d) Adding and Deleting Attribute Subject Terms.

This procedure, again is very similar to adding and deleting IST's and OST's. As with OST's these terms are global and can be associated or disassociated with the attribute(s) of a given EER object in a later step.

(i) Select the **Subject Terms** menu button at the top with the right mouse button.
(ii) Select submenu option **Define for Attributes.** The remaining steps should be similar to IST and OST additions and deletions.

(14.e) Associating Object Sets with Subject Terms.

(i) Select the **Subject Terms** menu button at the top with the right mouse button.
(ii) Select submenu option **Associate for Object Sets.** A popup window showing a list of subject terms on the left, and a list of object sets on the right will appear. The **Help** button provides online help on for adding and deleting associations for this window.

(iii) Select one or more subject terms in the list on the left. Select one or more object set in the list on the right.

(iv) Hit the **Add Association** button. Each selected object set on the right will be associated with all the selected subject terms on the left. You can use the **Show**

button to get a popup display-only window showing the associations of selected object sets. (If no object sets are selected, all the object sets and their associations, or lack thereof, are shown.)

(14.f) Deleting Associations of Object Sets with Subject Terms.

You should have the window showing the subject terms and object sets available to you from the steps described previously.

(i) Hit the BDelete Associations button. A popup window will appear showing a list of *(subjectTerm, objectSet)* pairs indicating existing associations.

(ii) Select on one or more of these associations.

(iii) Hit the **Delete** button and the selected associations will disappear from the list.

(iv) Hit the **Done** button to return to the previous window containing the list of subject terms on the left and object sets on the right.

(14.g) Associating and Disassociating Attributes with Subject Terms.

Associating and disassociating attribute subject terms is very similar to that of object terms, except for one additional step. You will be asked to select one object set from which to get a list of attributes.

(i) Select the **Subject Terms** menu button at the top with the right mouse button.

(ii) Select submenu option **Associate for Attributes**. A popup window listing available object sets will appear.

(iii) Select one object set.

(iv) Hit the **Attributes** button.

(v) A window containing a list of subject terms (attribute subject terms, in this case) on the left and attributes for the selected object set on the right will appear. The situation is similar to associating and disassociating subject terms for object sets described previously. The **Help** button provides online help. The **Show** button will show more detail on attributes and their associations. The **Add Associations** button will add associations for selected subject terms and attributes. The **Delete Associations** will pop up a window of existing associations for selection and deletion. The **Done** button will return you to the previous window.

(15) Generating Subject Terms for *SDT*. Before generating a file to be used as the input to SDT under the **Files** button menu, be sure to run the **Subject Terms List** report to check your subject terms. Note any problems pertaining to integrity in the diagnostic section.

Note also that the IST's are shown globally in the diagnostic section even though they were entered locally by clicking on the entity set. IST's are outputted to ***SDT*** as specialization entity sets. No two entity sets (whether specialized or generalized) can have the same name as far as the DBMS (Database Management System) is concerned, hence, a limitation from possible name conflict arises in a global context. In other words, the user is not permitted to enter an IST that has already been defined for another entity set, even though IST's are suppose to be "local" to a given entity set. With this restriction in place, showing IST's globally in a report should not present any problem since they are all unique. (An analogous situation could be mentioned with relationship sets, where it is perfectly natural to reuse a verb for the relationship name to relate entities. Reusing a relationship name not permitted in actual practice because of restrictions by ***ERDRAW*** and ***SDT*** as well as the DBMS.)

FILES

"erdraw.defaults" contains the startup resource specification defaults for this application. The environmental variable ERDRAW_DEFAULTS should be set to this file with the full pathname. E.g. in csh,

"setenv ERDRAW_DEFAULTS
/home/dm/disk1/szeto/erdraw/erdraw.defaults".

To output to postscript, one needs to set the environmental variable ERDRAW_PSPROLOG to the postscript prolog file "erprolog_ps" with the full pathname. E.g. in csh,

"setenv ERDRAW_PSPROLOG /home/dm/disk1/szeto/erdraw/erprolog_ps".

The prolog file contains the postscript subroutines for drawing generic EER graphical objects.

"setenv ERDRAW_HELPDIR /home/dm/disk1/szeto/erdraw".

This variable sets the directory locating the copyright message and help files.

SEE ALSO

"***SDT*** Reference Manual", by Victor M. Markowitz and Weiping Fang.

AUTHOR(S)

Ernest Szeto

BUGS/DEFICIENCIES

"Text objects" (including "role labels" and arc labels), unlike the other graphical objects, are not drawn in exclusive-OR raster op mode. (Other problems are created when done in this mode.) The result is that "text objects" will erase parts of another object when it overlaps that object. Use the REFRESH button to repair damages. Be careful about placing text objects too close together, esp. on the same line. Their "boundaries" (rectangular area that determines whether the mouse pointer is "inside" or "outside" an object) might interfere with each other.

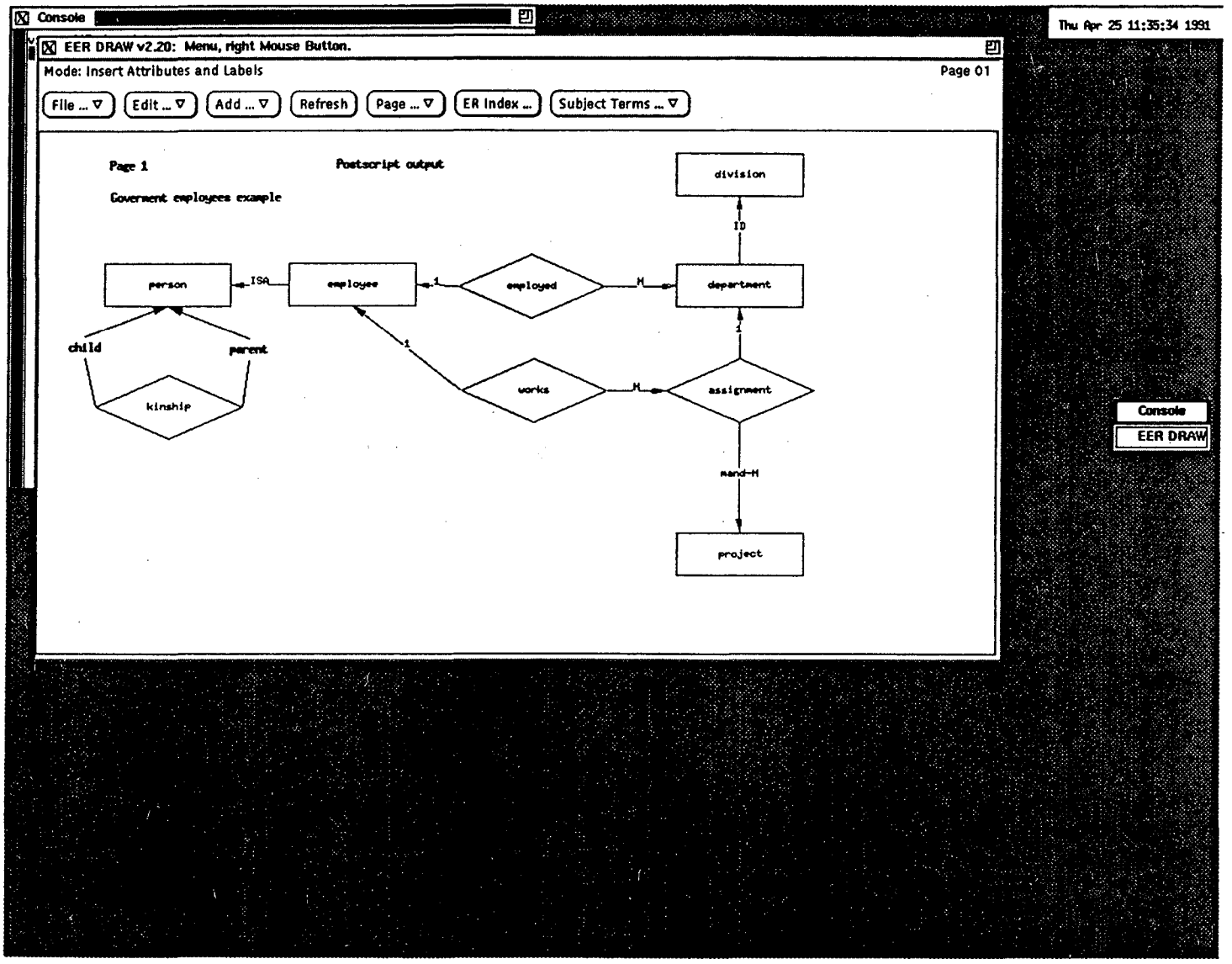
Holding down the left and right mouse button simultaneously can also damage your drawing by adding "rubber banded" images that are no longer valid for the new draw mode selected in the canvas pop-up menu. (Note, this is a non-sensory operation. Unfortunately, the program is not as robust as one would like.) To fix any damaged drawing, hit the REFRESH button.

***ERDRAW*** depends on SDT to do integrity checking of the schema, after the SDT file is outputted. A future version of ***ERDRAW*** might include integrity checks of the drawing at the press of a button during a drawing session.

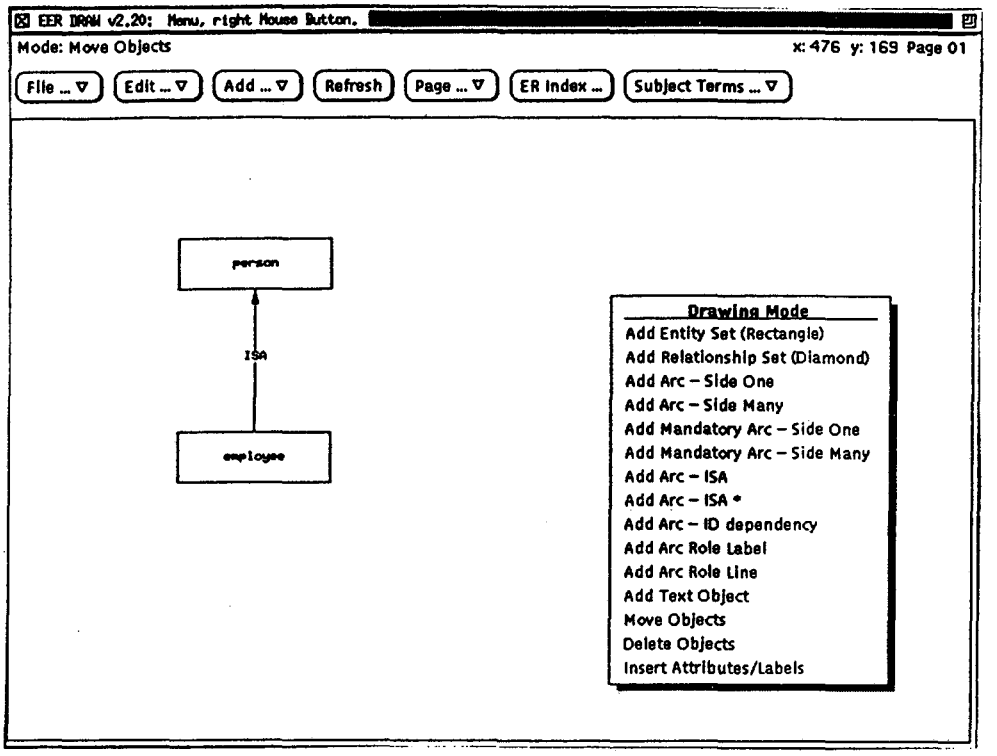
Key attributes and the cardinality of arcs with role labels are not shown automatically. The user would have to manually annotate the drawing using "text objects".

## 2. *ERDRAW* SCREEN IMAGES

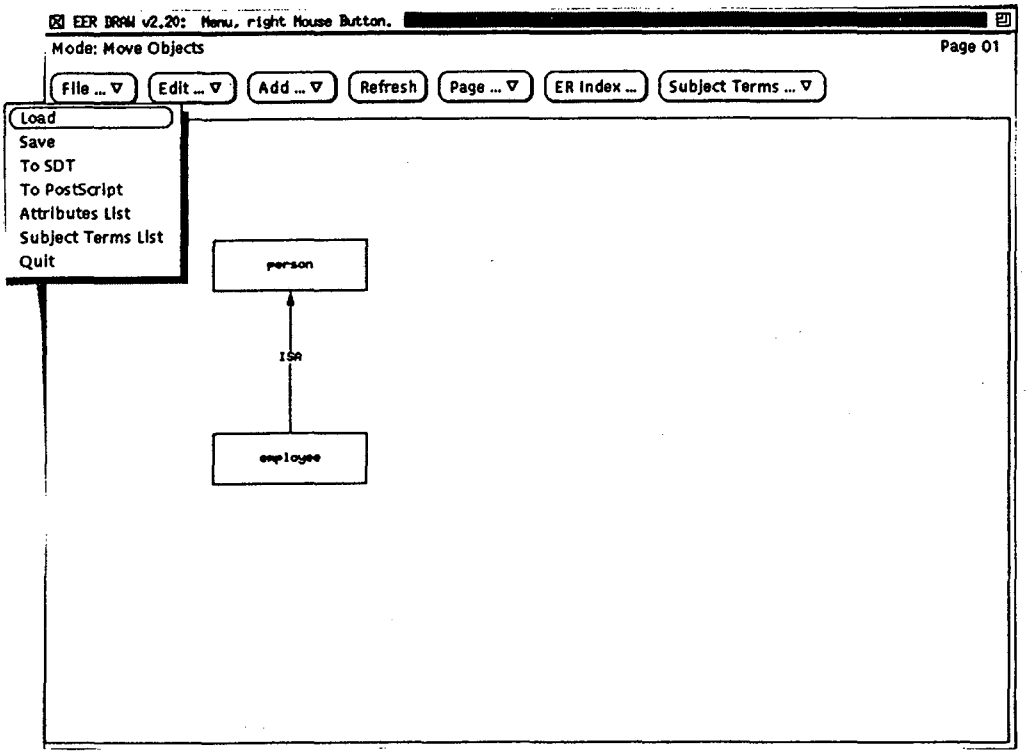
### *ERDRAW* Canvas



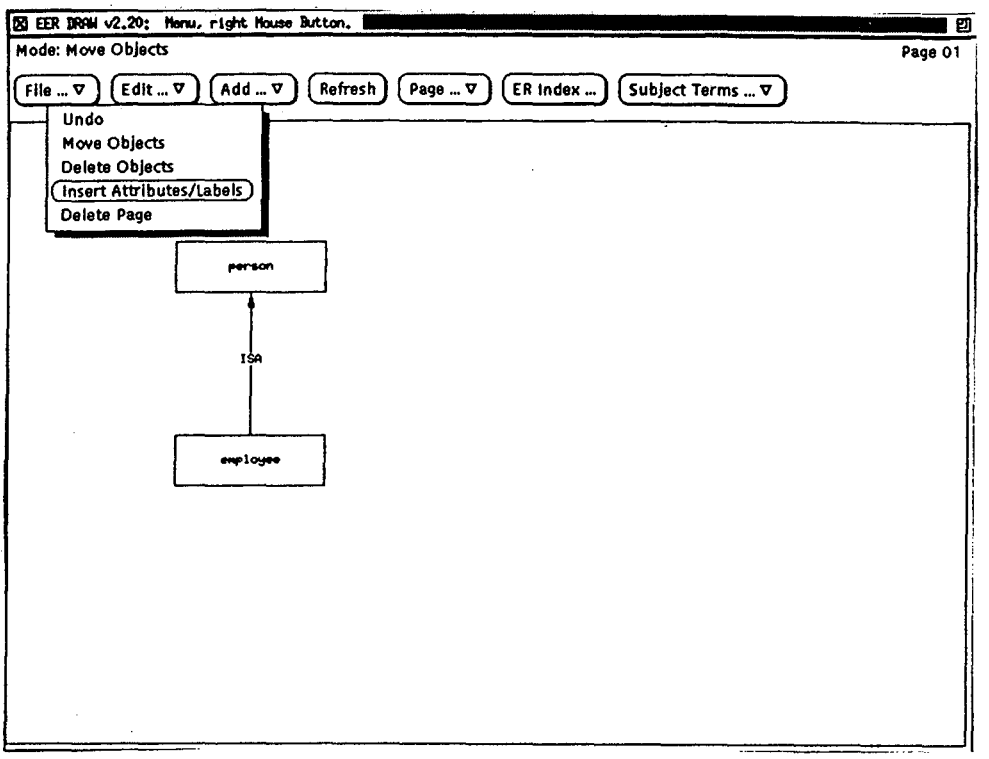
## *ERDRAW* : Canvas Pop–Up Menu



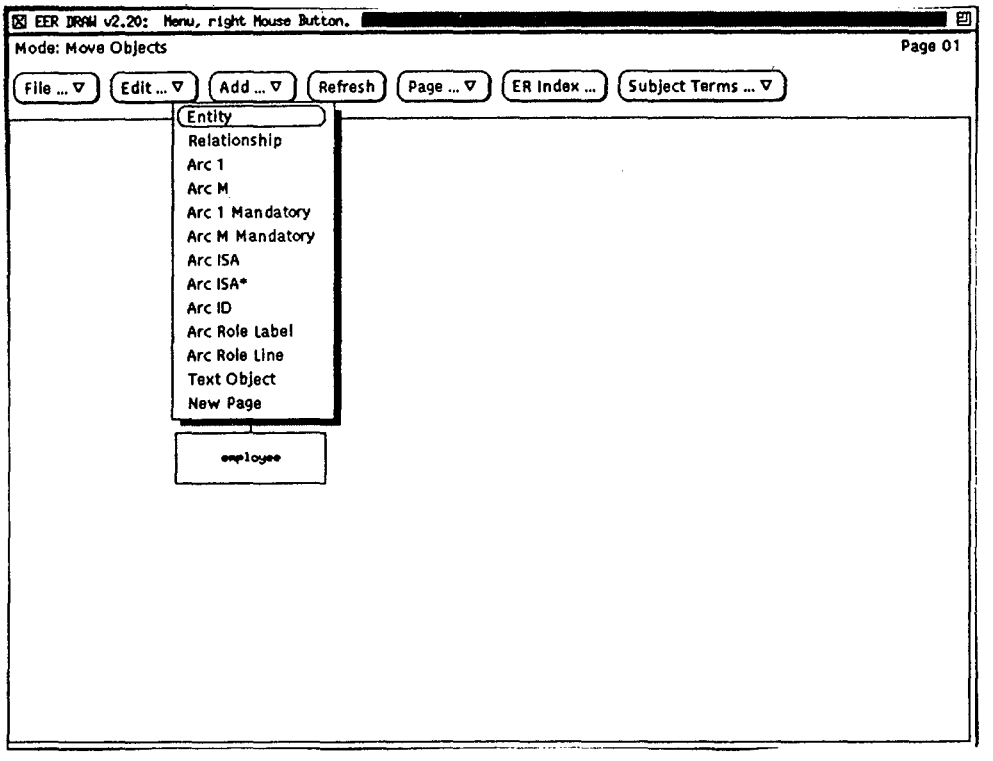

## *ERDRAW* : File Pull–Down Menu (use right mouse button)



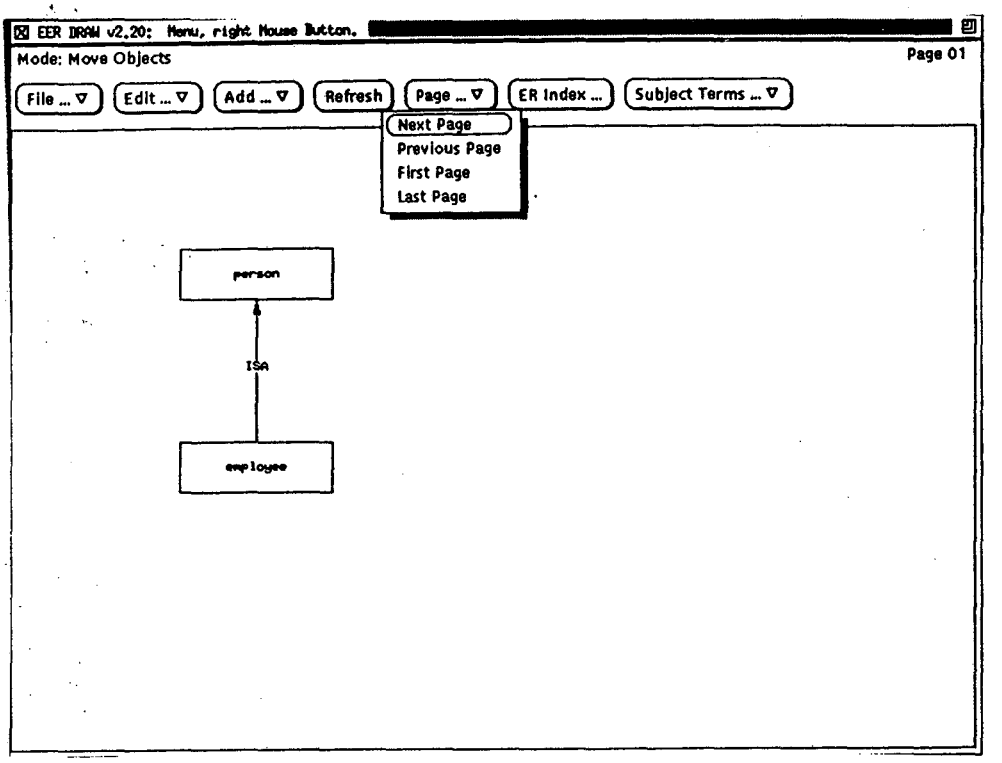
## *ERDRAW* : Edit Pull–Down Menu (use right mouse button)



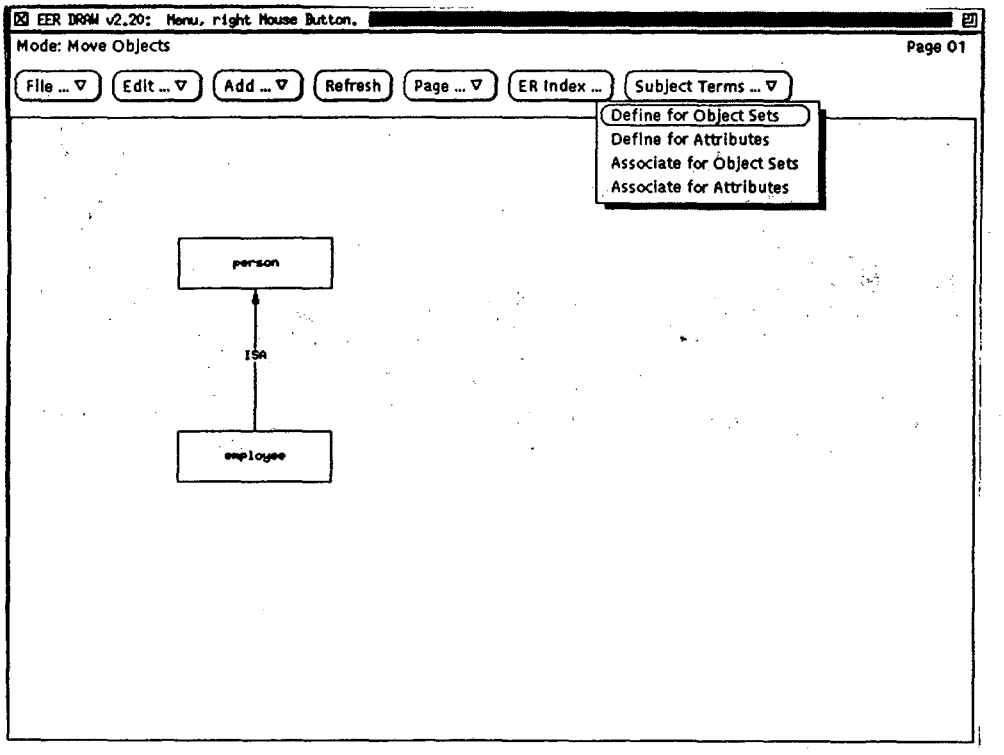

## *ERDRAW* : Add Pull–Down Menu (use right mouse button)



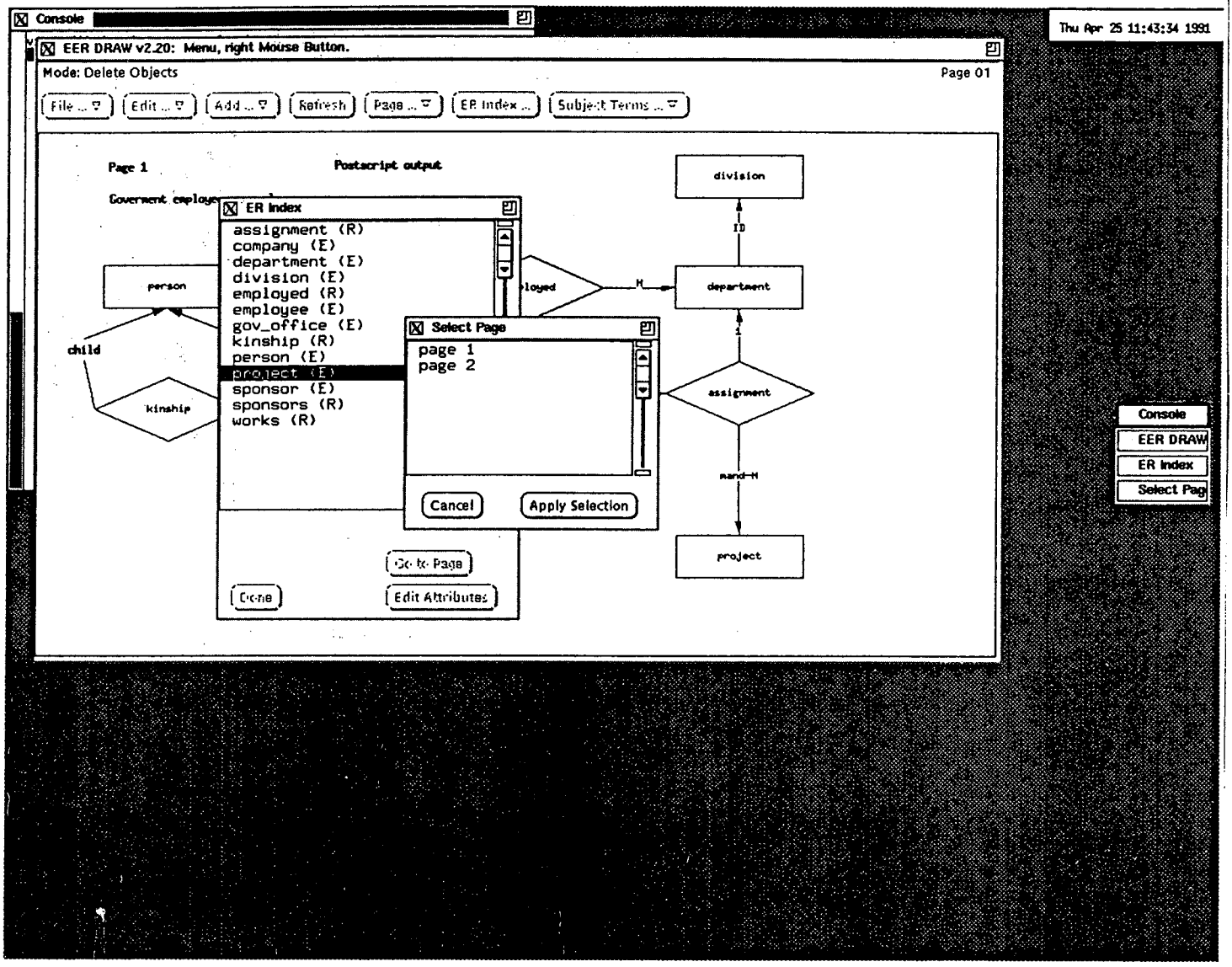
***ERDRAW* : Page Pull–Down Menu** (use right mouse button)



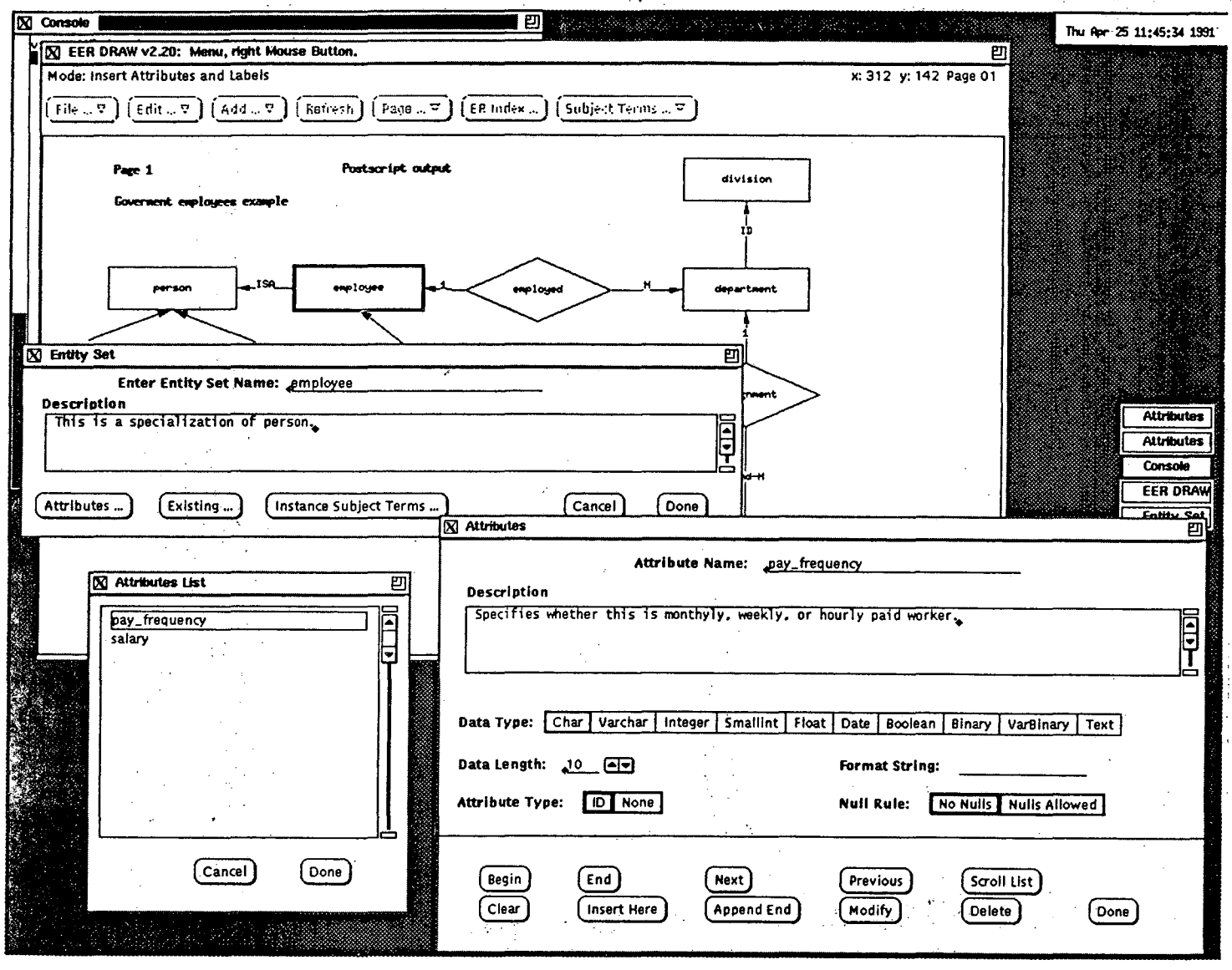

***ERDRAW* : Subject Terms Pull–Down Menu** (use right mouse button)



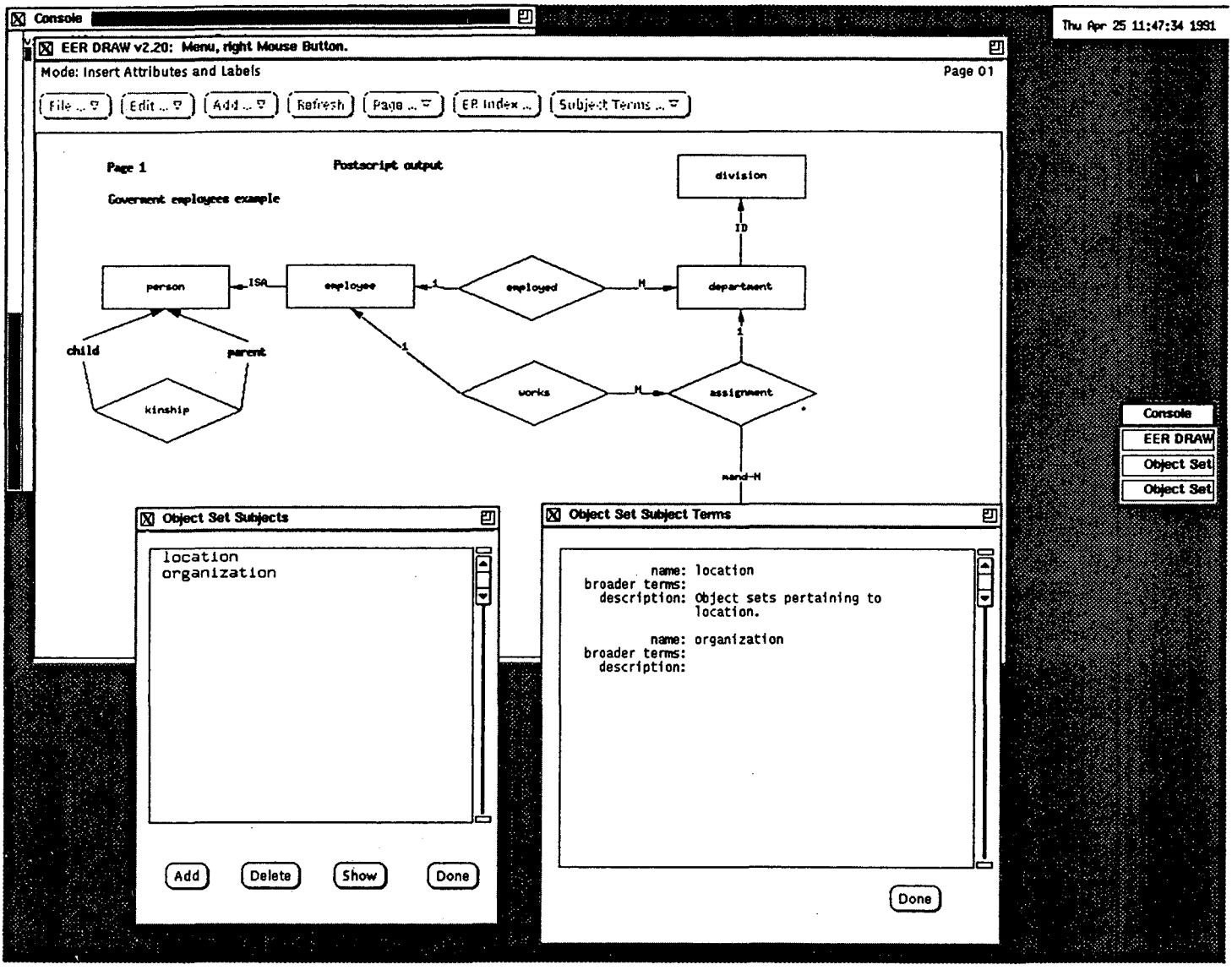
## *ERDRAW* : Index of EER Names



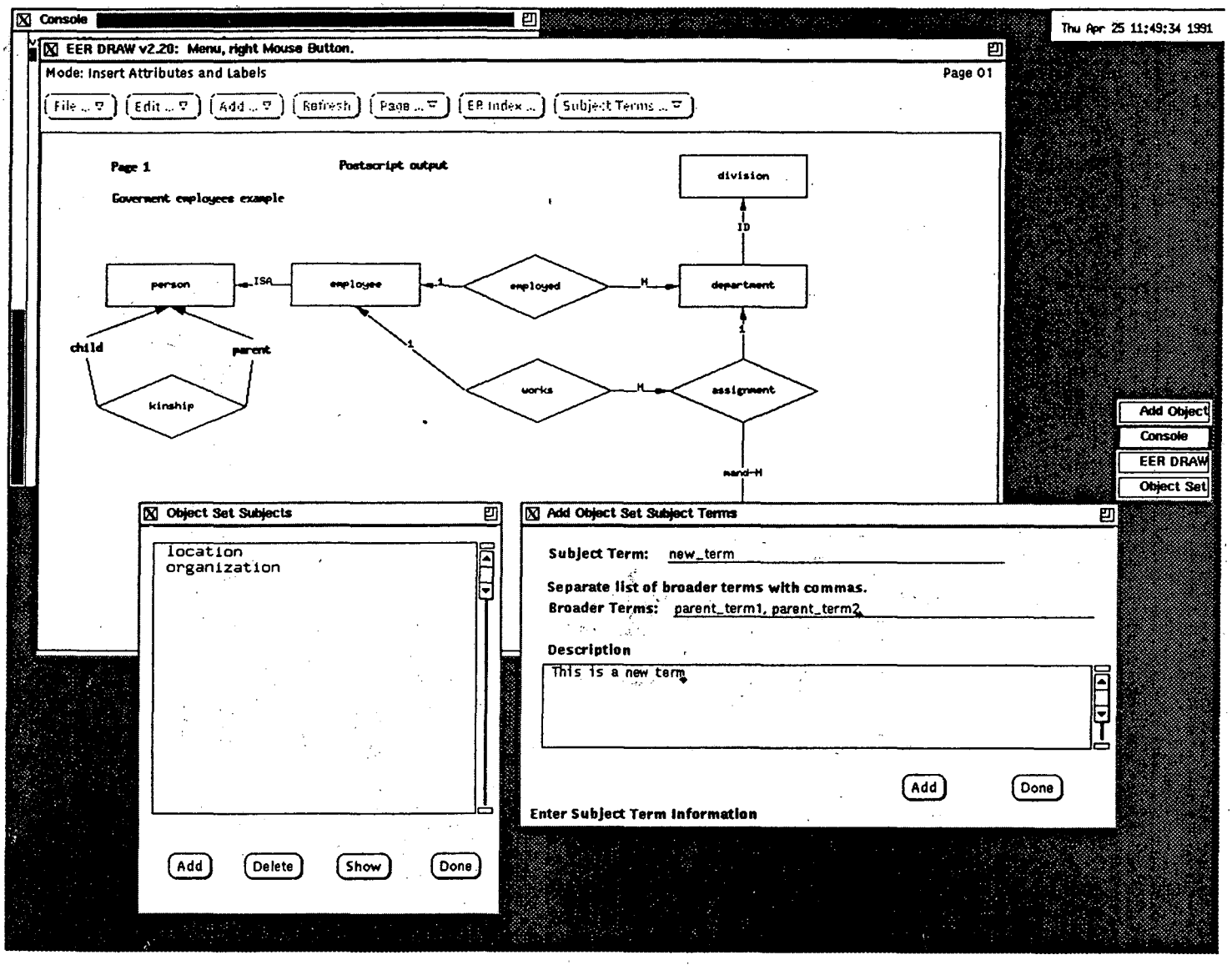
## *ERDRAW* : Pop–up Windows for Inserting Attributes



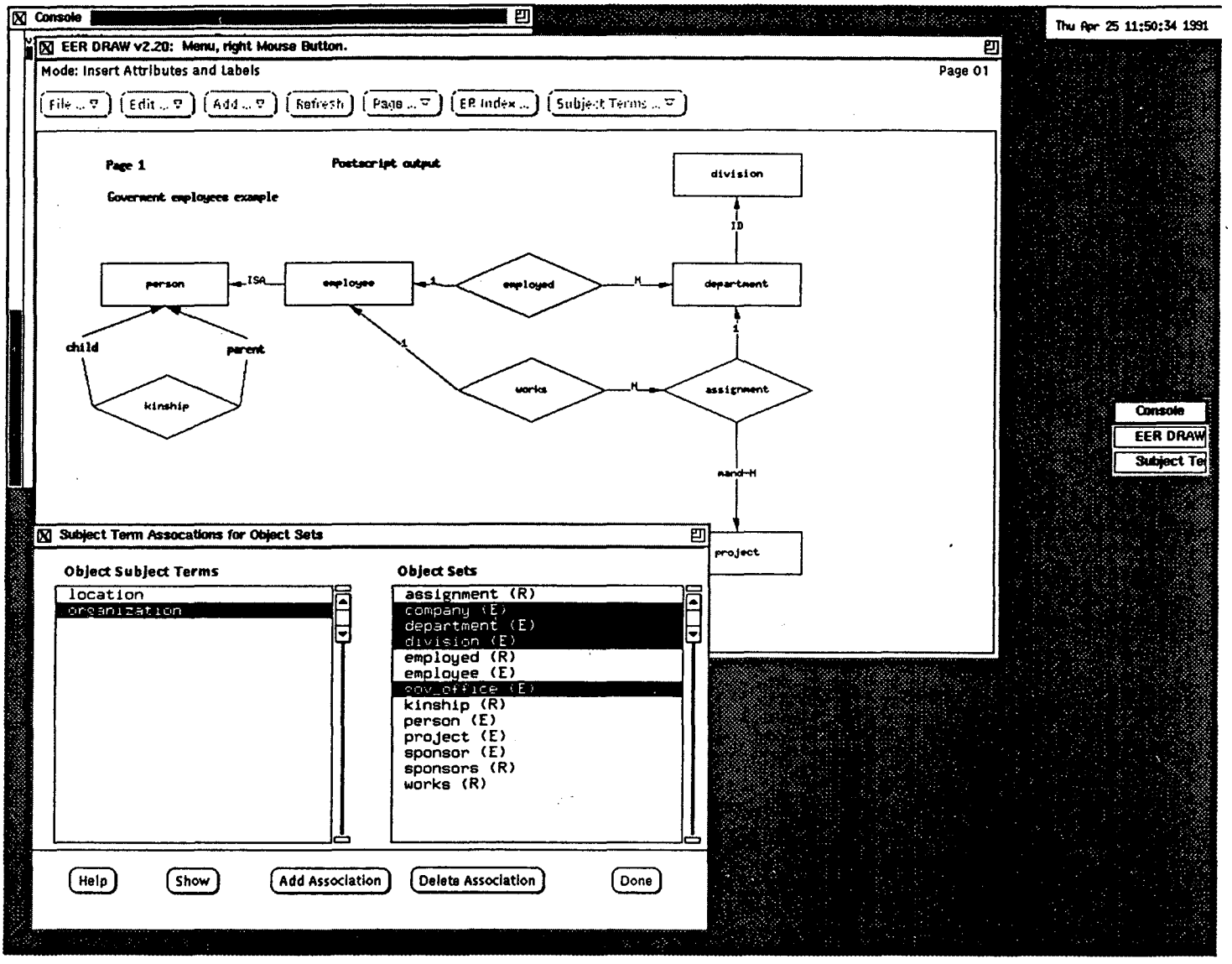
## *ERDRAW* : Subject Terms Definition List and Details Window



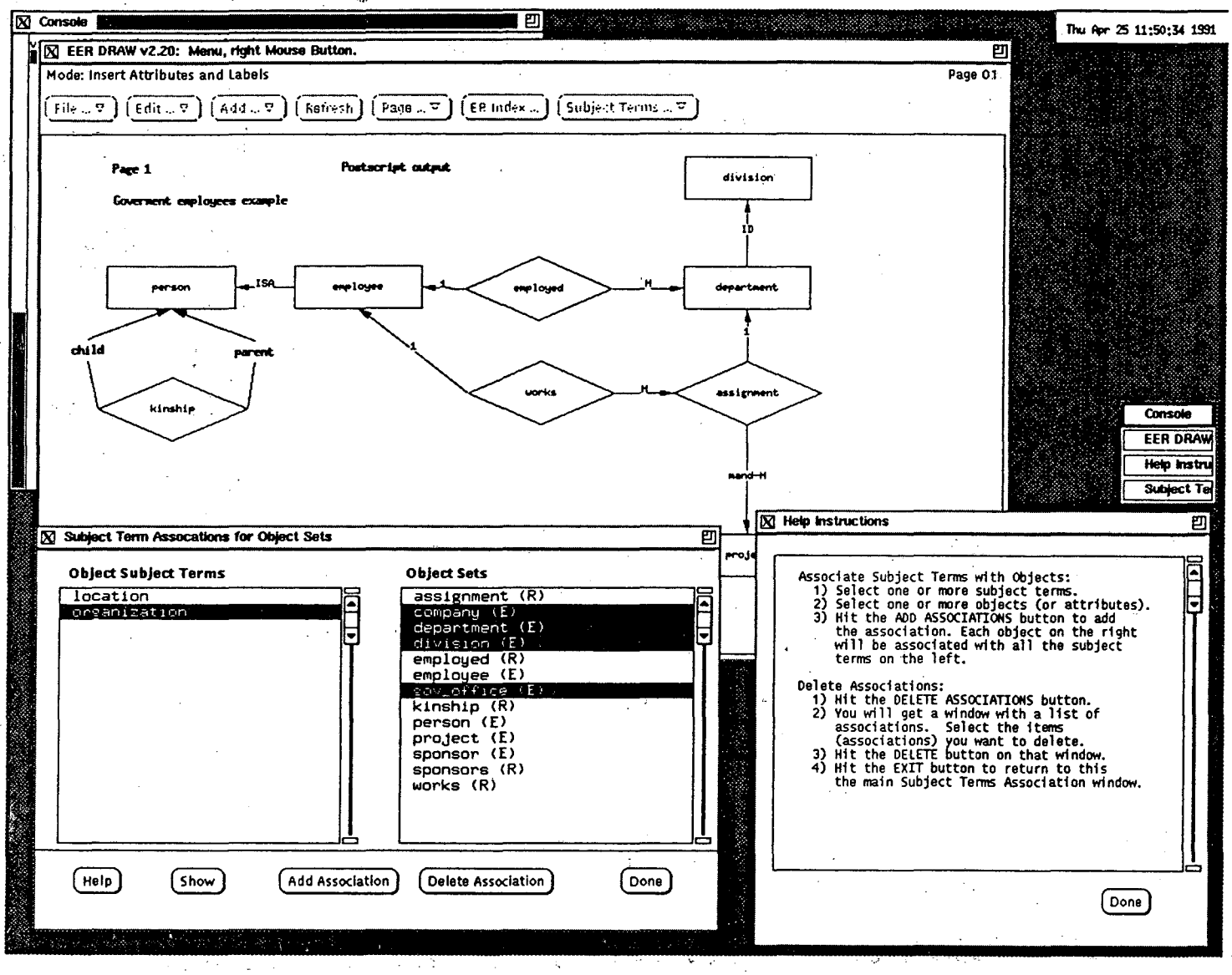
## *ERDRAW* : Adding New Subject Terms



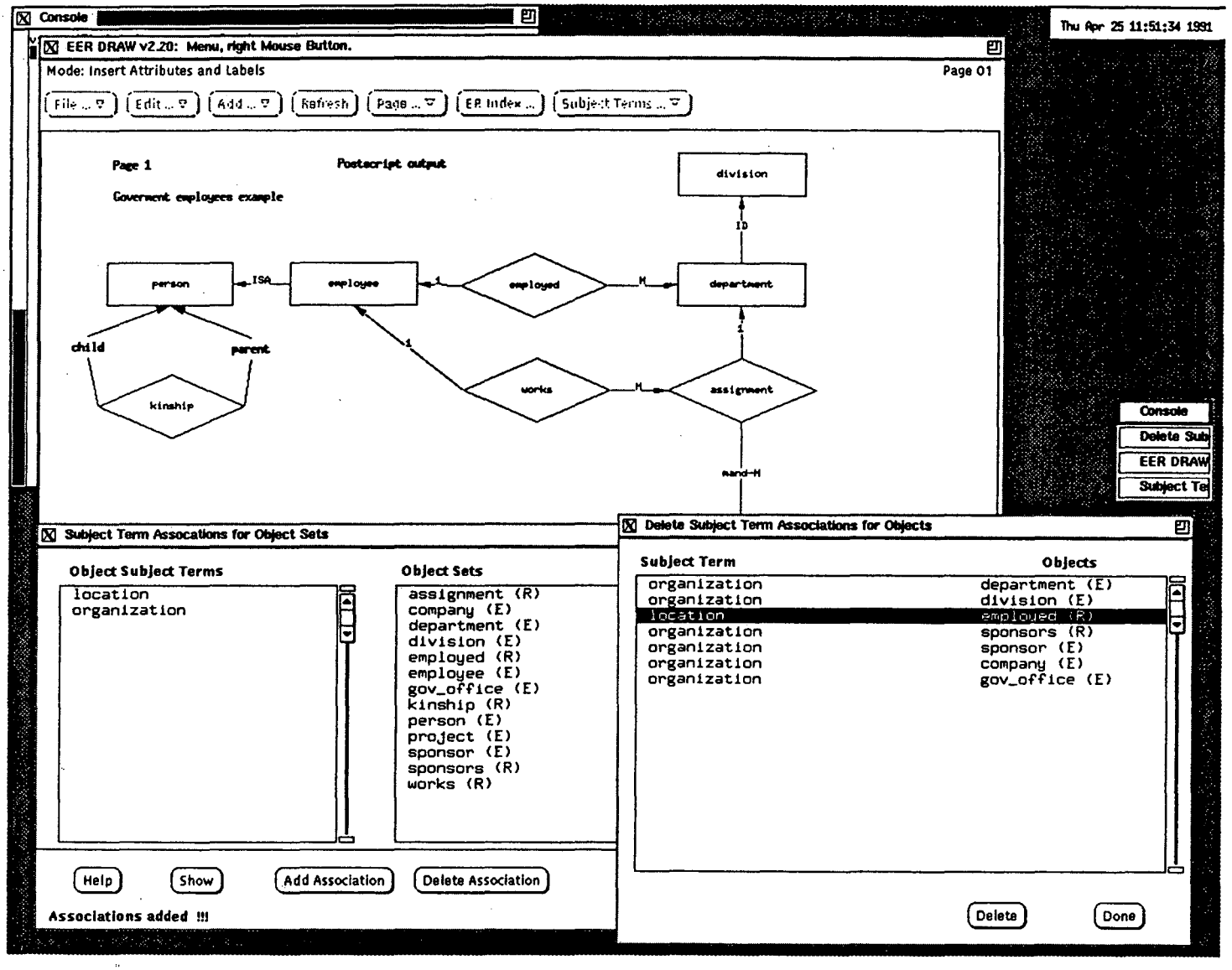
## *ERDRAW* : Associating Subject Terms with Object–Sets



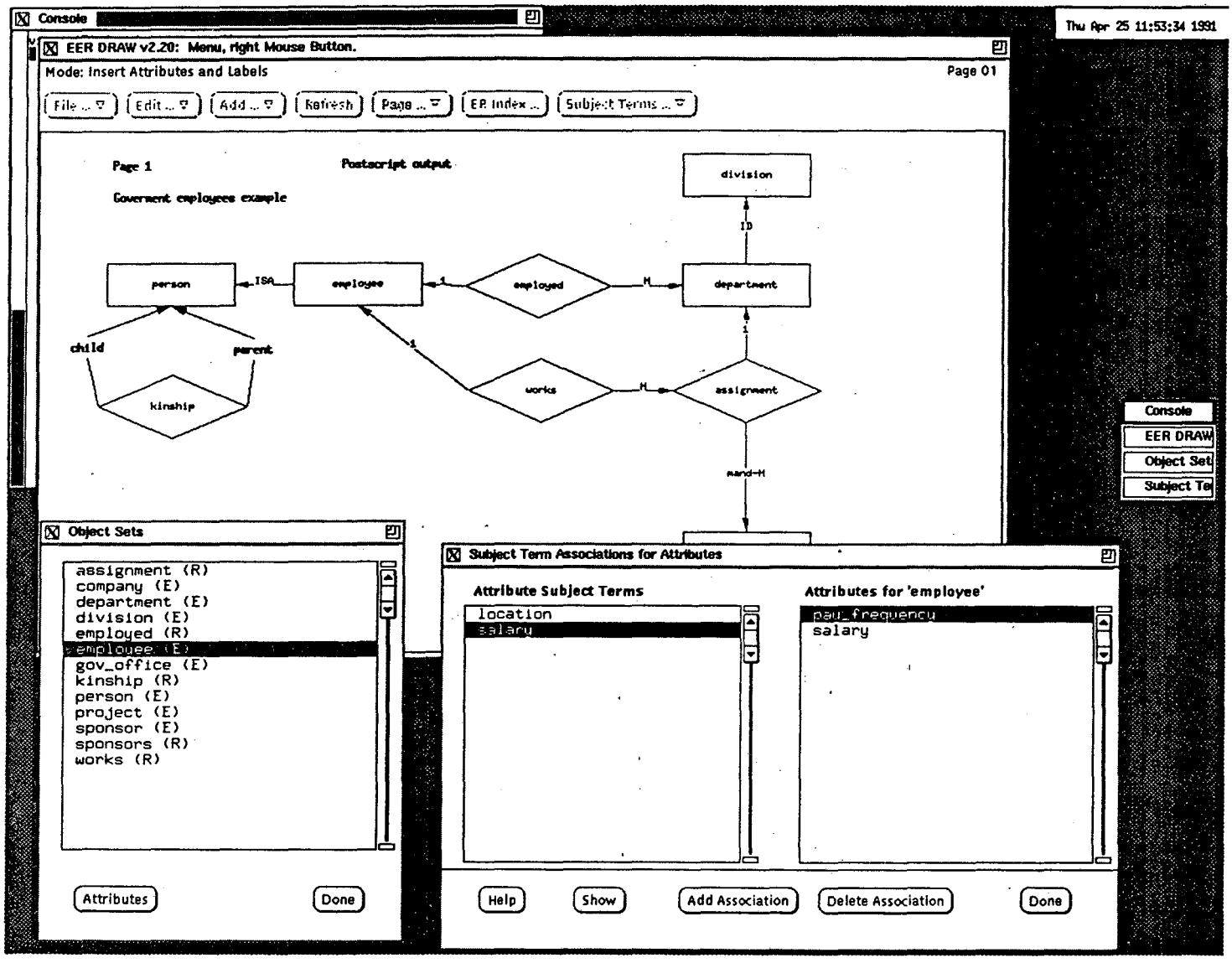
## *ERDRAW* : Associating Subject Terms / Help Window



## *ERDRAW* : Removing Associations of Subject Terms



## *ERDRAW* : Associating Subject Terms with Attributes

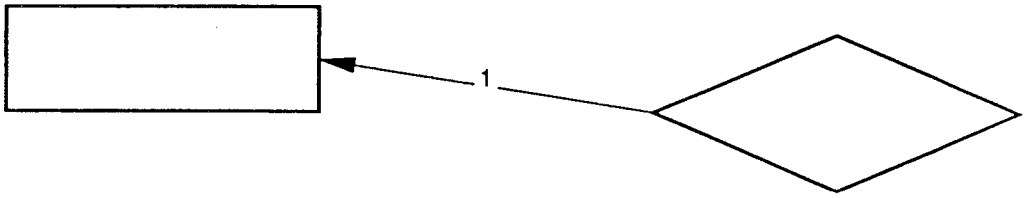


## 3. *ERDRAW* PICTORIAL TUTORIAL

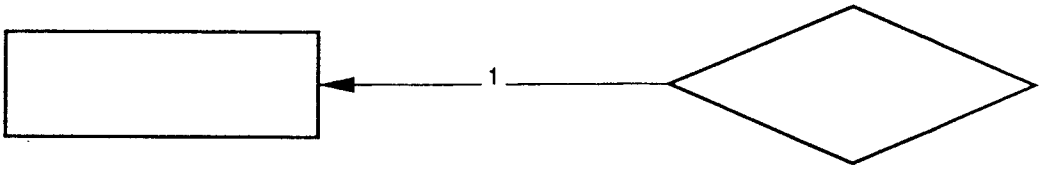
### Add and Move Objects

#### 1) Add ER objects

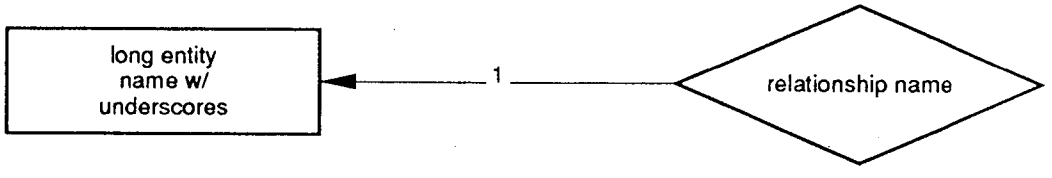
#### 2) Add Arc(s)




#### 3) Move Object(s)

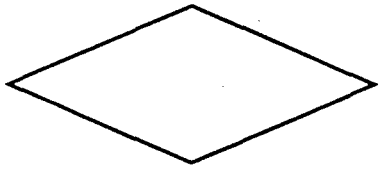



#### 4) Insert Labels/Attributes



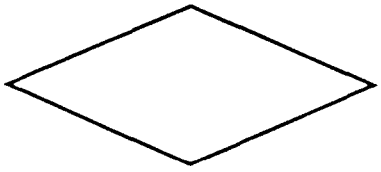
## Add Arcs with Roles

**1) Add ER objects**



**2) Add Role Labels**

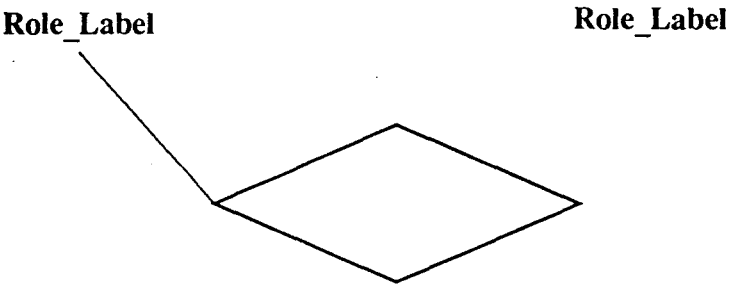
Role_Label Role_Label



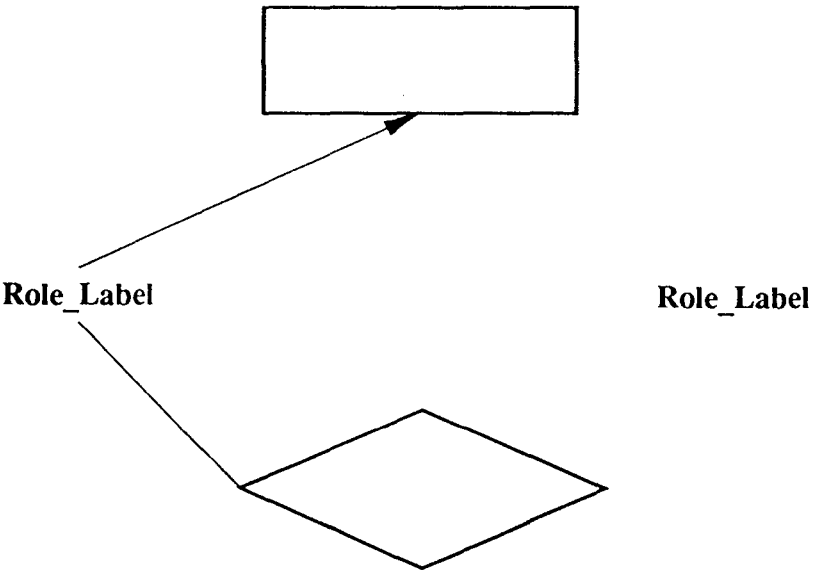
## Add Arcs with Roles (cont).

**3) Add role line**

**(Rel diamond --> Role Label)**

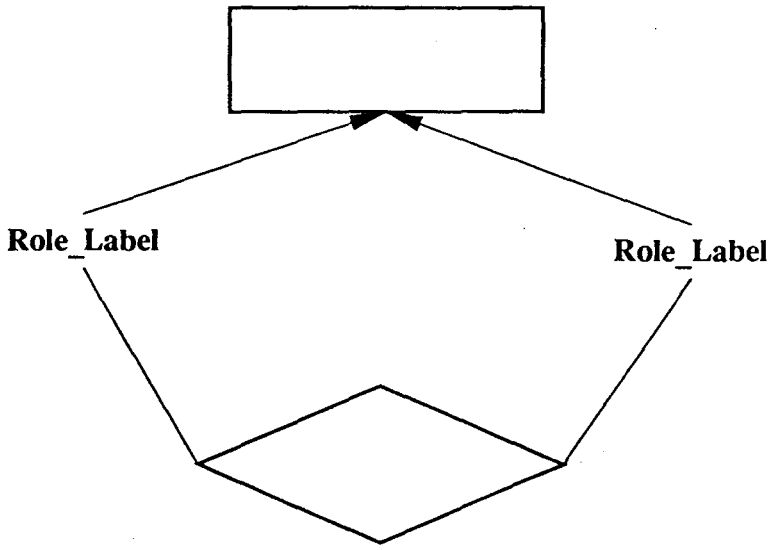



**4) Add other role line.**

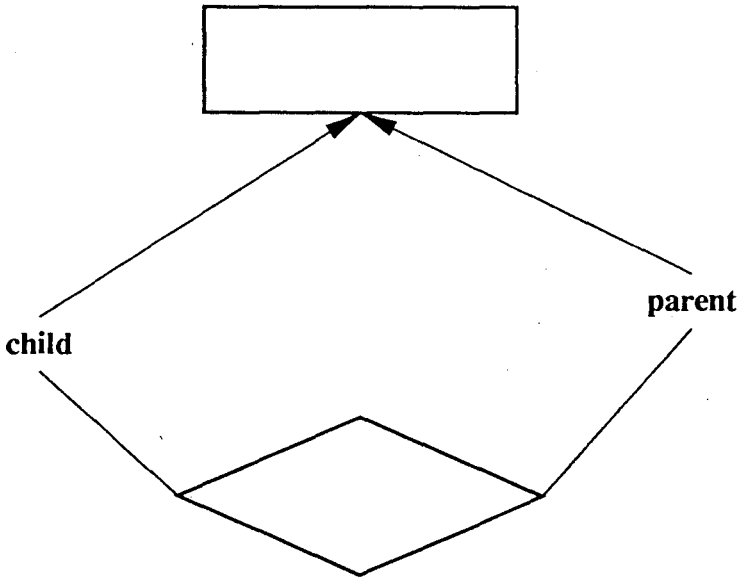


**Add Arcs with Roles (cont).**

**5) Do the same for other Role Label**

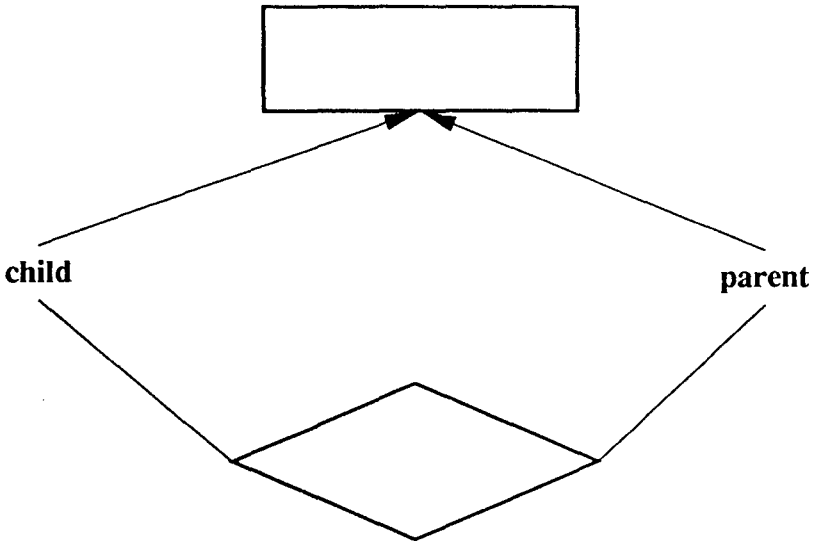



**6) Use "Insert Attributes" to change role labels.**

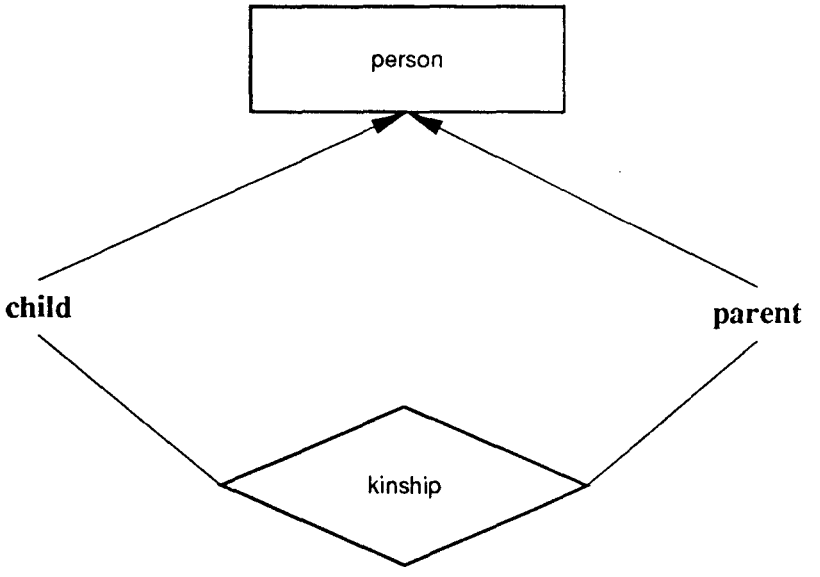


**Insert Attributes for EER Objects.**

**7) Move Objects (Role Labels)**

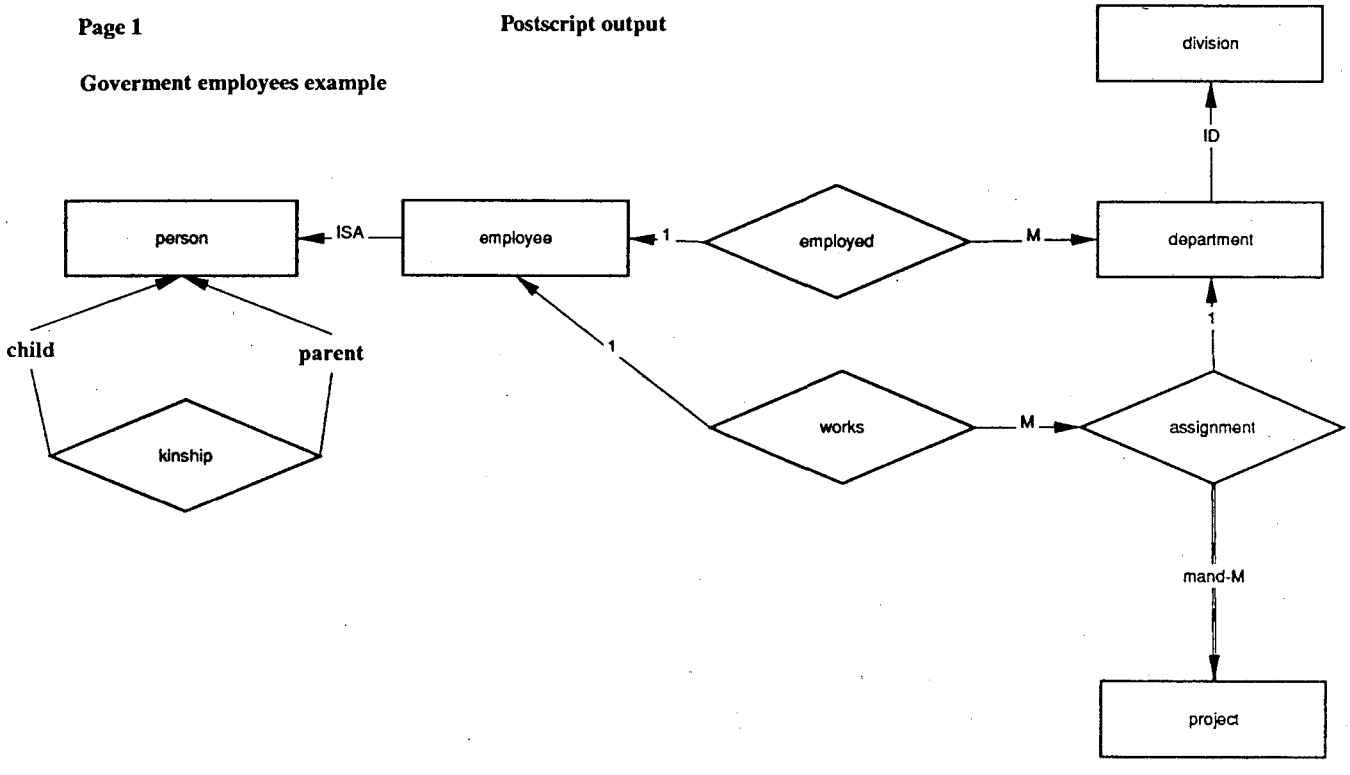



**8) Insert Attributes** **(for ER objects)**



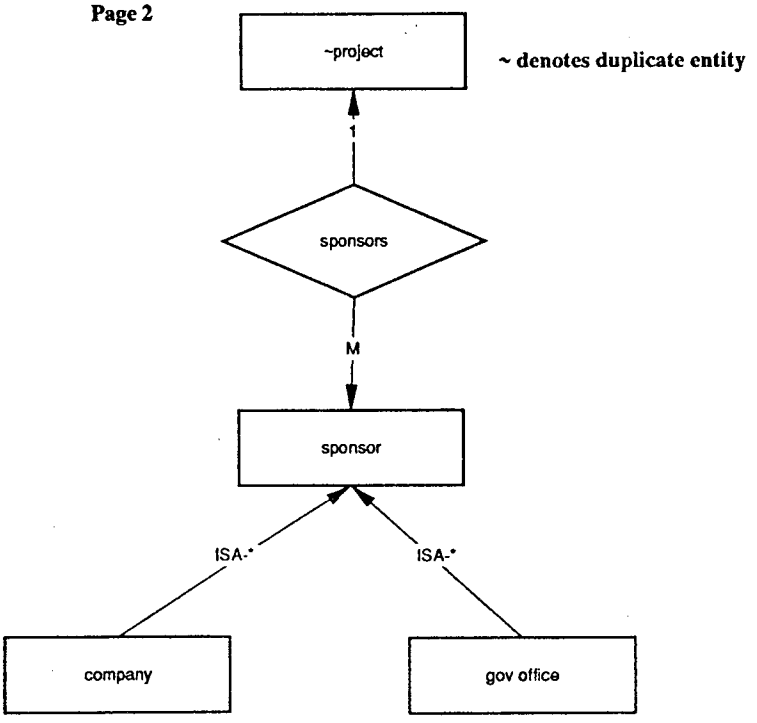
## 4. *ERDRAW* POSTCRIPT OUTPUT

### The Employee Record Example



## The Employee Record Example (cont.)

**Page 2**



# 5. *SDT* COMMAND AND INPUT

## 5.1 Command

*sdt* [ *–sT* ] [ *–cX* ] [ *–mY* ] [ *–tZ* ] *file*

where

*T* can be either *e* (for *EER*) or *r* (for *relational*), and specifies the type of input schema for ***SDT***; parameters *X*, *Y*, and *Z* below are ignored when *T* = *r*. If the *–s* option is not specified, *EER* schema is assumed by default.

*X* can be either *a* (for *association*) or *i* (for *involvement*) and specifies the type of relationship cardinality used in the EER schema. If the *–c* option is not specified, *association* cardinality is assumed by default.

*Y* can be either *r* (for *restricted*) or *n* (for *no merging*) and specifies the type of merging to be performed. If the *–m* option is not specified, the *no merging* is assumed by default.

*Z* can be either *s* (for SYBASE), *i* (for INGRES 6.3), or *x* (for INFORMIX 4.0), and specifies the target RDBMS. If the *–t* option is not specified, SYBASE is assumed by default.

*file* is the input file containing an EER schema specification following the syntax given in Figure 5.1.

## 5.2 Input Format for EER Schemas (Figure 5.1)

The syntax for specifying EER schemas is given in figure 5.1. A BNF-like notation is used in order to describe this syntax. Words in italic lower case letters denote non-terminals, while words in italic upper case letters and roman lower case letters denote terminals. Single-quoted characters are terminal delimiters whereas the rest are meta characters.

Notes:

1. A *number* must be in the syntax for a constant integer in C.
2. *size* is an upper bound on the number of objects in EER schema.
3. A *domain* must be in the form accepted by SYBASE/SQL, INGRES/SQL, or INFORMIX/SQL, respectively; the correct specification of the *domain* is the responsibility of the user.
4. An *identifier* is a letter or an underscore ('_'), possibly followed by a combined string of letters, underscores, and digits. Keywords are reserved identifiers.

*specification* ::= *size object_subject_list*
*size* ::= number
*object_subject_list* ::= *object_subject* | *object_subject_list object_subject*
*object_subject* ::= *object* | *subject*
*object* ::= *obj_head obj_tail* ';'
*obj_head* ::= *obj_name* '(' *obj_type* ')'
*obj_name* ::= identifier
*obj_type* ::= *entity* | *relationship*
*entity* ::= *E* | *ENTITY*
*relationship* ::= *R* | *RELATIONSHIP*
*obj_tail* ::= *attr_clause arc_clause descr_clause*
*attr_clause* ::= *ATTRS* ':' *attr_list* | empty string
*attr_list* ::= *attr* | *attr_list* ',' *attr*
*attr* ::= *attr_name* '(' *attr_type* ',' *descr* ',' *attr_subjects* ',' *domain null_rule*')'
*attr_name* ::= identifier
*attr_type* ::= *ID* | empty string
*attr_subjects* ::= *subj_name* | *attr_subjects subj_name*
*domain* ::= *data_type* | *data_type* '(' number ')'
*data_type* ::= identifier
*null_rule* ::= *NO NULLS* | *NULLS ALLOWED* | empty string
*arc_clause* ::= *ARCS* ':' *arc_list* | empty string
*arc_list* ::= *arc* | *arc_list* ',' arc
*arc* ::= *obj_name* '(' *arc_type* ',' *role* ')' | *subj_name* '(' *ST* ')'
*arc_type* ::= *ID* | *ISA* | *ISA** | *ONE* | *M* | *D*1 | *DM*
*role* ::= identifier | empty string
*descr_clause* ::= *DESCR* ':' *descr*
*descr* ::= '"'text'"'
*subject* ::= *subj_head subj_tail* ';'
*subj_head* ::= *subj_name* '(' *subj_type* ')'
*subj_name* ::= identifier
*subj_type* ::= *SO* | *SA*
*subj_tail* ::= *broader_terms descr_clause*
*broader_terms* ::= *ARCS* ':' *subject_list* | empty string
*subject_list* ::= *subj_name* (*ISA*,) | *subject_list* ',' *subj_name* (*ISA*,)

Figure 5.1 The Syntax for EER Schemas.

5. The default for the *null_ru 0e* when it is not specified, is *NO NULLS.*

6. For *arc_type* : *ID, ISA,* and *ISA**, represent the arc types exactly as they appear in the EER schema; *ONE* represents a relationship cardinality of *one* and *M* represents a relationship cardinality of *many*; *D* 1 represents both a relationship cardinality of *one* and *mandatory* involvement, and *DM* represents both a relationship cardinality of *many* and *mandatory* involvement.

### 5.3 Example of Input File

The following input file for ***SDT*** is generated by ***ERDRAW*** for the EER diagram presented in section 4:

```
23
person(E)
        ATTRS: ssn(ID, "", , varchar(10) NO NULLS), name(, "", , char(10) NO NULLS),
                bdate(, "Should also include a birth date for each person.", , datetime NULLS ALLOWED)
        DESCR: "This is a description for a person.";
employee(E)
        ATTRS: pay_frequency(, "Specifies whether this is monthly, weekly, or hourly paid worker.", salary,
                                char(10) NULLS ALLOWED), salary(,"", salary, int NULLS ALLOWED)
        ARCS: person(ISA, )
        DESCR: "This is a specialization of person.";
department(E)
        ATTRS: name(ID, "", , char(10) NO NULLS)
        ARCS: division(ID, ), organization(ST,)
        DESCR: "";
division(E)
        ATTRS: name(ID, "", , char(10) NO NULLS)
        ARCS: organization(ST,)
        DESCR: "";
employed(R)
        ARCS: employee(ONE, ), department(M, ), location(ST,)
        DESCR: "";
assignment(R)
        ARCS: project(DM, ), department(ONE, )
        DESCR: "";
works(R)
        ARCS: employee(ONE, ), assignment(M, )
        DESCR: "";
project(E)
        ATTRS: number(ID, "", , int NO NULLS),
                name(, "Project name should also be included.", , varchar(50) NULLS ALLOWED)
        DESCR: "";
```

```
kinship(R)
        ATTRS: kinship_attr(ID, "For associating kinship attribute.", , char(10) NO NULLS)
        ARCS:  person(M, parent), person(M, child)
        DESCR: "";
sponsors(R)
        ARCS:  project(ONE, ), sponsor(M, ), organization(ST,)
        DESCR: "";
sponsor(E)
        ATTRS: name(ID, "", , char(10) NO NULLS)
        ARCS:  organization(ST,)
        DESCR: "Sponsors might have terms that are specializations of itself.";
company(E)
        ATTRS: name(ID, "", , char(10) NO NULLS)
        ARCS:  sponsor(ISA*, ), organization(ST,)
        DESCR: "";
gov_office(E)
        ATTRS: name(ID, "", , char(10) NO NULLS)
        ARCS:  sponsor(ISA*, )
        DESCR: "";
location (SO)
        DESCR: "Object sets pertaining to location.";
organization (SO)
        DESCR: "";
location (SA)
        DESCR: "Attributes pertaining to location.";
salary (SA)
        DESCR: "Attributes pertaining to salary.";
sponsor_sub_1 (E)
        ARCS: sponsor(ISA,)
        DESCR: "Specialization of sponsor.";
sponsor_sub_2 (E)
        ARCS: sponsor_sub_1(ISA,)
        DESCR: "Add another child to indicate specialization.";
worker (E)
        ARCS: employee(ISA,)
        DESCR: "This should be an instance term for of employee.";
rtsg (E)
        ARCS: department(ISA,)
        DESCR: "";
engineering (E)
        ARCS: division(ISA,)
        DESCR: "";
icsd (E)
        ARCS: division(ISA,)
        DESCR: "";
```

## 6. *ERDRAW* ATTRIBUTES REPORT

```
------------------------
ERDRAW PAGE 1
------------------------
 person (E)
  [This is a description for a person.]
    ssn              varchar(10)    NO NULLS       ID
    name              char(10)      NO NULLS
    bdate             datetime      NULLS ALLOWED
     [Should also include a birth date for each person.]
 employee (E)
  [This is a specialization of person.]
    pay_frequency      char(10)      NULLS ALLOWED
     [Specifies whether this is monthyly, weekly, or hourly paid
     worker.]
    salary             int           NULLS ALLOWED
 department (E)
    name               char(10)      NO NULLS       ID
 division (E)
    name               char(10)      NO NULLS       ID
 employed (R)
 assignment (R)
 works (R)
 project (E)
    number             int           NO NULLS       ID
    name               varchar(50)   NULLS ALLOWED
     [Project name should also be included.]
 kinship (R)
    kinship_attr       char(10)      NO NULLS       ID
     [For associating kinship attribute.]

--------------
ERDRAW PAGE 2
--------------
 sponsors (R)
 sponsor (E)
  [Sponsors might have terms that are specializations of itself.]
    name               char(10)      NO NULLS       ID
 company (E)
    name               char(10)      NO NULLS       ID
 gov_office (E)
    name               char(10)      NO NULLS       ID
```

## 7. *ERDRAW* SUBJECT TERMS REPORT

```
==============================
 Object Set Subject Terms
==============================
      name: location
 broader terms:
  description: Object sets pertaining to location.

      name: organization
 broader terms:
  description:

================================
 Attribute Set Subject Terms
================================
      name: location
 broader terms:
  description: Attributes pertaining to location.

      name: salary
 broader terms:
  description: Attributes pertaining to salary.

========================================
 Object Instance Subject Terms Section
========================================
--------------------------------------------------
 Instance Terms for Object 'assignment (R)'
--------------------------------------------------
 (No object instance subject terms)

--------------------------------------------------
 Instance Terms for Object 'company (E)'
--------------------------------------------------
 (No object instance subject terms)

--------------------------------------------------
 Instance Terms for Object 'department (E)'
--------------------------------------------------
      name: rtsg
 broader terms: department
  description:

--------------------------------------------------
 Instance Terms for Object 'division (E)'
--------------------------------------------------
      name: engineering
 broader terms: division
  description:
```

name: icsd
broader terms: division
description:

---

Instance Terms for Object 'employed (R)'

---

(No object instance subject terms)

---

Instance Terms for Object 'employee (E)'

---

name: worker
broader terms: employee
description: This should be an instance term for of employee.

---

Instance Terms for Object 'gov_office (E)'

---

(No object instance subject terms)

---

Instance Terms for Object 'kinship (R)'

---

(No object instance subject terms)

---

Instance Terms for Object 'person (E)'

---

(No object instance subject terms)

---

Instance Terms for Object 'project (E)'

---

(No object instance subject terms)

---

Instance Terms for Object 'sponsor (E)'

---

name: sponsor_sub_1
broader terms: sponsor
description: Specialization of sponsor.

name: sponsor_sub_2
broader terms: sponsor_sub_1
description: Add another child to indicate specialization.

---

Instance Terms for Object 'sponsors (R)'

---

(No object instance subject terms)

---

Instance Terms for Object 'works (R)'

---

(No object instance subject terms)

==========

Object and Subject Term Associations

==========

assignment (R)
(No associated objext set subject terms)

company (E)
organization

department (E)
organization

division (E)
organization

employed (R)
location

employee (E)
(No associated objext set subject terms)

gov_office (E)
(No associated objext set subject terms)

kinship (R)
(No associated objext set subject terms)

person (E)
(No associated objext set subject terms)

project (E)
(No associated objext set subject terms)

sponsor (E)
organization

sponsors (R)
organization

works (R)
(No associated objext set subject terms)

==========

Attribute and Subject Term Associations

==========

company.name

(No associated subject terms)

department.name
(No associated subject terms)

division.name
(No associated subject terms)

employee.pay_frequency
salary

employee.salary
salary

gov_office.name
(No associated subject terms)

kinship.kinship_attr
(No associated subject terms)

person.ssn
(No associated subject terms)

person.name
(No associated subject terms)

person.bdate
(No associated subject terms)

project.number
(No associated subject terms)

project.name
(No associated subject terms)

sponsor.name
(No associated subject terms)

==========

Diagnostic Section

==========

---

Subject Terms pointing to Non-Existant Parent Terms

---

---

Entities with unconnected subject terms

---

==========

Immediate/Remote Parents of a given Subject Term

(Loop detection also indicated)

==========

---

All Parents of Object Set Subject Term

---

---

All Parents of Attribute Set Subject Term

---

---

All parents of Instance of Subject Term (shown globally as opposed to being shown winthin one Entity Set)

---

engineering
    division
icsd
    division
rtsg
    department
sponsor_sub_1
    sponsor
sponsor_sub_2
    sponsor_sub_1
    sponsor
worker
    employee

# APPENDIX A. TECHNICAL DOCUMENTATION FOR *ERDRAW* V2.2 COMMENTARY ON CODE

**Ernest Szeto**

***ERDRAW*** is an X11 based Extended Entity Relationship diagram drawing tool that outputs postscript and EER specifications to ***SDT*** (Schema Design and Translation Tool). The tool was written in C using the Xview toolkit and Xlib. This document will briefly discuss the techniques used in ***ERDRAW*** and give an overview of the data structures and code organization.

## A.1 Source Code File Organization

The following is a revised description of the source files (with some new additions). Most of these *.c source files have a corresponding *.h header file.

erdraw.c — This file contains the main() routine. It also contains the routine to load the resource specifications. The rest of the code consists of event handlers and supporting routines for the Xview toolkit. These routines and handlers are responsible for ***ERDRAW***'s graphical editing capabilities, such as moving objects around, deleting them, and the associated behavior of "attached" or "affected" objects such as arcs. This file also contains routines for setting up pop-up forms for the user to insert attributes.

erwidgets.c — This file contains Xview widget declarations. These widgets include the base frame, control panels, the drawing canvas, pop-up forms, buttons, menus, scroll lists, etc. After declaring the widgets, it goes into the main Xview event loop. It consists of one (very long) routine: defineAndRunWidgets().

ertables.c — This file contains the internal data structure and supporting routines for ***ERDRAW***. The programmer has adopted the philosophy of not allowing code in other files to directly access the data structure. (The exceptions to this access rule include routines in erfiles.c and erlistmgr.c.) Access routines in this file allow mediated access to the data structure. The data structure consists mainly of tables for (1) EER and "text" objects, (2) arcs, (3) attributes, and (4) subject terms. The EER table is supplemented by lists to optimize search Efficiency.

erlistmgr.c — This file contains list management routines to service the lists in ertables.c. It also contains access routines so that external routines can access data in the lists.

erfiles.c — this file contains routines for communicating with external files. It has routines for (1) saving and loading a drawing to/from a file, (2) outputting the drawing into postscript code, to be printed on a postscript printer, (3) outputting the EER schema in a format that can be read by ***SDT***. In version 1.3, an ascii attributes list report was also added. In version 2.0, a subject terms report was also added.

erutil.c — This file contains miscellaneous, but generic routines, that can be used not only by ***ERDRAW***, but other applications.

pred.c — This file contains generalized routines for reporting all immediate and remote predecessors. Predecessors are inserted into this subsystem as *(item, itemPredecessor)* string pairs. It will also report the existence of any loops detected somewhere along the predecessor path. It is used by the subject terms subsystem for reporting and checking parent ("broader") terms.

canvlist.c — This file contains code for a customized scroll list widget created using Xview's canvas widget and vertical scrollbar.

idxui.c — This file contains user interface component declarations used to create a popup window for an index list of EER names. The file was originally generated by Sun's GUIDE (Graphical User Interface Development Editor) and GXV (GUIDE to Xview converter).

idxproc.c — This file contains event handling procedures for the index list widget mentioned above. This was also originally generated by GUIDE and GXV and adapted for the ***ERDRAW*** application.

erpnlui.c — This user interface components file was generated by GUIDE/GXV for the new *description* field as well as the **Instance Subject Terms** button. The code formerly in erpnlproc.c was relocated to erwidgets.c and erdraw.c to conform to old source code organization. This file contains the window displaying the selected EER object. It allows the user to hit another button for selecting attributes, or existing EER objects, as well as *Instance Subject Terms*.

erattrui.c — This user interface components file was generated by GUIDE/GXV for the new *description* field for lengthier explanation of an attribute field. Most of the code in erattrproc.c has been relocated to erdraw.c and erwidgets.c to conform to earlier source code conventions.

sdt[ui|proc].c — (These files were generated by GUIDE/GXV. sdtui.c contains the user interface component declarations, and sdtproc.c contains the event handling procedures. The event

handling procedures needed to be expanded and customized for this application.) This *Subject Terms Definition* (STD) window shows a list of subject term definitions, used for adding or deleting IST's, OST's, and AST's (Instance, Object, and Attribute subject terms, respectively).

stdadd[ui|proc].c (Similar comments as above on GUIDE/GXV generated files and customization.) This window allows the user to enter the subject term, its broader terms, and description. The user hits the **Add** button to add the entry. The window is used for adding IST's, OST's, and AST's.

stobj[ui|proc].c This window presents a list of object sets. Only one can be selected. The object set selected will have its attributes displayed in the *Subject Terms Association* window.

sta[ui|proc].c This window presents the a list of subject terms on the left and a list of object sets (or attributes) on the right. The user can select items on the left and right for association.

stadel[ui|proc].c This window is invoked from the sta*.c window. It presents the user with *(subjTerm,objSet)* or *(subjTerm,attribute)* pairs for deletion.

det[ui|proc].c This is a "details" display-only window. It appears whenever the user hits the **Show Details** button, displaying existing subject terms, or displaying object sets (or attributes) and their associations. It is also used to display the online help text.

start[ui|proc].c Startup window with copyright message and setting for DBMS datatypes. (Currently, Sybase and Ingres are allowed.)

quit[ui|proc].c Window to prompt user for filename for saving and quitting.

erprolog_ps This file contains the generic postscript subroutines for drawing EER and text objects in postscript. The environmental variable ERDRAW_PSPROLOG should be set with the full pathname to this file.

erdraw.defaults (This is not really part of the code.) This ascii file, editable by the user (though it should be done with caution), specifies the resources required by ***ERDRAW***. These resource parameters are read at startup time by the program. The environmental variable ERDRAW_DEFAULTS should be set with the full pathname to this file.

sta.help Ascii help file for the sta window. ***ERDRAW*** reads this file to give the user instructions on how to use the *Subject Terms Association* window.

erdraw.copyright (See also, erdraw.copyright.nroff.) Ascii file containing copyright message and abstract, read by ***ERDRAW*** upon startup.

### A.2 Event Driven Programming

X11 applications tend to be "event driven". Event driven applications tend to have the following structure:

1. Set up the widgets and declare the event handlers for certain events occurring within a widget.
2. Go into the main event loop.
3. Activate the appropriate event handler when a specific event occurs on a specific widget. Return to the main event loop when done.

For example, one may declare a button with the Xview toolkit. Two types of events can occur on this button: 1) the button is pushed. 2) the button is released. In declaring the button widget, the programmer also specifies the routine (handler) to be activated when the button is pushed/released. See example below:

```
/* Declarations */
 ...
refreshButton = xv_create(panel, PANEL_BUTTON,
                          PANEL_LABEL_STRING,    "Refresh",
                          PANEL_NOTIFY_PROC,     refreshButtonProc,
                          NULL);
 ...
/* Main event loop */
window_main_loop(baseFrame);
 ...

/* Event Handlers */
 ...
refreshButtonProc()
 {
 ...
 }
 ...
```

The same paradigm applies for the canvas widget. However, the canvas widget accepts many more events, such as "mouse pointer moved to such and such a location", "left mouse button is pushed at x,y location", "right mouse button is released at x,y location", "mouse pointer is dragged to x,y location", etc. The event handling is much more complex for a drawing canvas where objects a drawn and react under runtime user control.

### A.3 Exclusive–OR Raster Op Mode

X has 16 drawing modes called "raster ops" (raster operations). A standard technique to simulate graphical objects stretching, such as resizing a box or stretching a "rubber band" line, is to set the Graphics Context to the exclusive-OR raster op mode. The Graphics Context determines how an object is drawn when a drawing routine is called, such as XDrawLine() or XDrawRectangle(). In the exclusive-OR mode, the new destination pixel is produced by an exclusive-OR of the old destination pixel with the source pixel.

newCanvasPixel = xor(oldCanvasPixel, sourcePixel)

In this mode, the programmer can easily draw and erase a figure on canvas by drawing on it once to have the figure appear, then drawing over it again to erase it. A sequence of these operations at slightly different locations in response to the mouse pointer motion events can simulate a graphical object being moved or "dragged" to a new location.

The exclusive-OR mode has the added advantage that graphical objects can overlap without damaging each other. (This is generally true, but not completely true, esp. in the case of text leaving "pixel droppings".) One can move a graphical object on top of another, then move it out again without damaging the underlying object. For example, one can move a line on top of a box, then back out again, and the box will retain its original image.

Note, however, for ***ERDRAW***, that one should not move objects on top of each other, especially text on top of another object. The program has the added task of deciding whether a mouse pointer is "inside" (or affecting) a certain object. The situation is ambiguous when two objects occupy roughly the same rectangular boundaries that define the mouse as being "inside" an object. The program will behave erratically when overlapping objects are moved.

### A.4 Drawing Text

Because the exclusive-OR mode does not completely erase text completely, leaving "pixel droppings", the programmer of ***ERDRAW*** has resorted to blanking out the background with spaces at the old location and redrawing text by blanking out the background in the new location, and redrawing text in the new (clean) location, to simulate the movement of text. The disadvantage with this technique is that text overlapping another object will "damage" or erase portions of the overlapping region. The only way to fix such a damage is to hit the REFRESH button. The REFRESH button clears the canvas, and redraws everything from graphical parameters in the data structure. In future revisions of ***ERDRAW***, the exclusive-OR mode will be reevaluated, and other alternatives explored.

## A.5 Data Structures

*ERDRAW* maintains three tables:

1. A table (**erTable**) for EER objects and "text objects". (The term "text object", text that acts like other EER graphical objects when it comes to graphical editing, is used ambiguously in the code comments. It can either mean, more generally, BOTH "text objects" to annotate a page and "role labels" for arcs, or more specifically "text objects" to annotate page as distinct from "role labels".)
2. A second table (**arcTable**) with arc entries connecting two objects in the EER table.
3. An attributes table (**attrTable**) containing all the attributes for the EER entries in the EER table.
4. A subject terms table ((fBsubjTable) implements storage for various types of subject terms.

erTable and arcTable have "methods" (a term taken from object oriented programming) that are associated with the record definition. These "methods" are implemented as function pointers in C. The tables are implemented as arrays with memory dynamically allocated at startup time. The array boundaries can be increased by changing the values "erdraw.maxNoERObjs", "erdraw.maxNoArcObjs", and "erdaw.maxNoERAttrs" in the ascii resource specification startup file "erdraw.defaults". (The environmental variable ERDRAW_DEFAULTS should be set to this file with the full path name.)

**erTable** has the following definition in pseudo-code:

| | |
|---|---|
| deleted | A field to mark whether this record is deleted (and, hence, its space is reusable). A '*' marks the record as deleted. |
| type | Type of record. Entity (E), Relationship (R), Text Object (T), or Role Label (L). |
| name | Name of ER object, text object, or role label. |
| descr | Description for this ER object. |
| cx, cy | Center x, y coordinates of graphical object. |
| pageNo | Page number to which this object belongs. |
| arcList | List of arc indexes tied to this ER object. |
| attrList | List of attribute indexes for this ER object. |
| hasIST | Flag to indicate this ER object (Entity) has *Instance Subject Terms*. |
| osubjList | Object set subject list of indexes of subject terms associated with this ER object. |
| listBac | Backup string list for "undo" operation. |

*Fields used by 'role labels' only:*

| | |
|---|---|
| fromER | The "from ER" object attached to this role label. |
| toER | The "to ER" object attached to this role label. |
| arcType | Arc type for this role label. |

*Methods:*

| | |
|---|---|
| insideMethod | Determines whether mouse pointer is inside this object. |
| redrawMethod | Redraw this graphical ER object on a Xview canvas. |
| psDrawMethod | Draw this graphical ER object in postscript. |
| outlineMethod | Highlight the graphical ER object by "outlining" it when the mouse pointer is near or "inside" the object. |
| invertMethod | Invert the image (white to black and vice versa) of the graphical object when it is selected. |

**arcTable** has the following definition in pseudo-code:

| | |
|---|---|
| deleted | A '*' indicates this record is marked for deletion and is reusable. |
| type | Type of arc (e.g., 1, Many, Mandatory-1, ISA, ID, etc.). |
| x1, y1 | Starting point of arc. |
| x2, y2 | Ending point of arc. |
| pageNo | Page number in which this arc appears. |
| fromObj | ER object from which the starting point is attached. |
| toObj | ER object to which the ending point is attached. |
| conn1 | Connecting point (North, West, East, South) of "from ER object". |
| conn2 | Connecting point (North, West, East, South) of "to ER object". *Methods:* |
| affectedMethod | Determine whether this arc is affected and needs to move and stretch. For e.g., if an attached ER object moves, it needs to change with it. |
| redrawMethod | Redraw this arc on an Xview canvas. |
| psDraw | Draw this arc in postscript. |

**attrTable** is an array of pointers. Since there can be many attributes, and each taking a sizable chunk of memory, memory for attributes are dynamically allocated/freed with the insertion/deletion of attributes. **attrTable** has the following definition in pseudo-code:

| | |
|---|---|
| erIdx | Index to ER object to which this attribute belongs. |
| attrSeq | Ordinal position of this attribute in the ER object. |
| name | Name of attribute. |
| descr | Description of attribute. |
| type | Type of attribute (e.g., integer, char, date, binary, etc.). |
| length | Length in bytes for this attribute. (Required for specifying char(), varchar(), varbinary(), etc.). |
| format | Display or data entry format for attribute (e.g, "(999) 999-9999" for telephone numbers.) Currently not used. |
| null | Accepts nulls or not. |
| keyType | Whether it is an ID type of attribute. |
| listBac | Backup string list used in "undo" operation. |

**subjTable** is the most recent and major addition, and has the following pseudo-code:

| | |
|---|---|
| deleted | Indicates whether this record is marked deleted. |
| type | Type of subject term. 'I' for IST, 'O' for OST, or 'A' for AST. |
| name | Name of subject term. This is a single identifier, perhaps containing words separated by underscores. |
| broaderTerms | A string list of broader terms separated by commas. |
| descr | Long description of subject term. |
| erIdx | For IST's, the entity set to which this subject term belongs. |

***ERDRAW*** saves and loads drawings to/from an ascii file. An ascii file is used (as opposed to binary in the prototype version) to make the program easier to trace and debug. Also, ascii has the advantage of not being machine dependent. ***ERDRAW*** also outputs to SDT by traversing the data structure and outputting information from these structures. For the most part, the EER specifications for SDT pretty closely corresponds to the graphical data stored in ***ERDRAW***. ***ERDRAW*** also outputs to postscript. When traversing through its data structure, it outputs postscript commands at a very high level, such as "400 600 entity", and "30 500 500 100 (ISA) drawArrowHeadArc". The postscript subroutines for these high level commands are stored in a file called erprolog_ps. (Postscript code consists usually of at least two sections: 1) the prolog and 2) the script.) The environmental variable ERDRAW_PSPROLOG should be set to the file erprolog_ps with the full pathname for ***ERDRAW*** to work correctly in outputting to postscript.

## A.6 Conclusive Remarks

***ERDRAW*** is a prototype. Many improvements can be made to it, such as online integrity checking of the EER schema and better handling of "text objects". Version 1.3 included additions such as "undo" and easier handling of attributes while retaining the same data structure, a data structure oriented primarily towards managing graphics. Version 2.x implements features for creating a metadatabase, including the use of subject terms.

# APPENDIX B. THE EXTENDED ENTITY–RELATIONSHIP MODEL

**Victor M. Markowitz**

The concepts of the *Entity–Relationship* model have been defined originally in [1] and have been repeatedly reviewed since then. The *Extended Entity–Relationship* model is surveyed in [2]. We follow, in general, the definitions in these references, with slight modifications. Thus, we represent Entity-Relationship structures by directed, rather than undirected, diagrams. The problem of representing EER structures using relational constructs is examined in [3].

## B.1 Fundamental Concepts

### B.1.1 Object–Sets

The first stage of **Entity-Relationship** (ER) modeling consists of determining the principal objects about which information is collected, called **entity–sets**. Entity-sets are qualified by **attributes**, that represent their descriptive properties. For instance, PERSON could be an entity-set with attributes SOCIAL–SECURITY–NUMBER, NAME, JOB–TITLE, and SALARY. Associations of entity-sets are represented by **relationship–sets**. For instance WORK could be a relationship-set associating entity-sets PERSON and PROJECT. A relationship-set may have attributes, just like an entity-set, such as the PERCENTAGE–OF–TIME a person WORKs on each project. Individual instances of entity-sets and relationship-sets are called entities and relationships, respectively. In the following we shall refer commonly to entities and relationships as **objects**, and to entity-sets and relationship-sets as **object–sets**.

### B.1.2 Value–Sets

Attributes take their values from underlying domains called **value–sets**. Examples of value-sets could be CHARACTER, INTEGER. Value-sets can be associated with a *format* describing the structure of their elements (e.g. six-digit character). Attributes provide an interpretation of a given value-set in the context of some object-set. For instance, attribute NAME gives the interpretation of value-set CHARACTER in the context of entity-set PERSON. The independent identity of attribute *values* is of no interest in the modeled environment, but only when coupled with some object. For instance a value of attribute

[1] P.P. Chen, "The entity-relationship model- towards a unified view of data", *ACM Trans. on Database Systems* **1**,1 (March 1976), pp. 9-36.

[2] T.J. Teorey, D. Yang, and J.P. Fry, "A logical design methodology for relational databases using the extended entity-relationship model", *Computing Surveys* **18**,2 (June 1986), pp. 197-222.

[3] V.M. Markowitz and A. Shoshani, "On the correctness of representing extended entity-relationship structures in the relational model", Proc. of *1989 SIGMOD Conference*, SIGMOD Record **18**, 2, June 1989, pp. 430-439.

SOCIAL–SECURITY–NUMBER is of interest only as characterizing an instance of entity-set PERSON. Value-sets are the basis of correlating attributes: attributes associated with the same value-set, are said to be *compatible*, that is, can be compared. When value-sets are *uninterpreted*, that is, devoid of any semantic meaning (e.g. sets of integers or characters), the attribute compatibility has no real significance. For instance, although two attributes, such as AGE and HEIGHT, could be based on a same value-set (e.g. numbers) their comparison could be meaningless. Value-sets can be *interpreted* by associating them with *units*. Then two attributes are said to be compatible only if the units of their underlying value-sets are the same or can be converted to a common unit. For instance the value-set underlying attributes AGE and HEIGHT could be associated with years and kilograms as units, respectively. Interpreted value-sets allow the specification of two kinds of constraints: (i) *value constraints* restrict the the values that an attribute can take from a value-set (e.g. the value-set of attribute AGE can be specified as consisting of integers between 13 and 65); and (ii) *operational constraints* restrict the operations allowed on the attribute values (e.g. AGE values could be added and subtracted, while NAMES values could be compared but not added). Generally attributes can be associated not only with single value-sets, but also with the cartesian product of several value-sets.

### B.1.3 Entity–Relationship Diagram

ER structures are expressible in a diagrammatic form called **ER Diagram** (ERD). Entity-sets, relationship-sets, and attributes, are represented graphically by rectangles, diamonds, and ellipses, respectively. Every vertex is labeled by the name of the object-set or attribute; entity, and relationship vertices are uniquely identified by their labels globally, while attribute vertices are uniquely identified by their labels only locally, with respect to their object-set (that is, within the set of attribute vertices connected to some object-set vertex). Edges in an ER diagram represent the interaction of the various object-sets and attributes. The ER diagram is a *directed graph*, that is, it has directed edges. In figure B.1 we present an example of an ER diagram consisting of the following main components : PERSON, PROJECT, DIVISION and DEPARTMENT are entity-sets, relationship-set EMPLOYED represents the employment of persons by departments, relationship-set ASSIGNMENT represents the assignment of projects to departments, and KINSHIP represents the kinship relation between persons.

### B.1.4 Entity–Identifier

A subset of the attributes associated with an entity-set is specified as the **entity–identifier**. Entity-identifiers are used to distinguish among the instances of an entity-set. For instance SOCIAL–SECURITY–NUMBER could be an identifier for entity-set PERSON, as shown in the ER diagram of figure B.1 where attributes belonging to identifiers are underlined. However, entity-identifiers are not

always enough to *uniquely* distinguish among the instances of an entity-set. For example, there may be a Service department in both the Appliance and Automotive divisions of some company. In that case, the entity-identifier NAME of entity-set DEPARTMENT is not enough to uniquely distinguish between the various instances of departments with the same name in different divisions. Such entity-sets are called **weak**, and said to depend for identification (**ID–dependent**) on other entity-sets. In ER diagrams, vertices that represent weak entity-sets are connected by directed edges, labeled *ID* , to the vertices representing the entity-sets on which the weak entity-sets depend. For instance, in the former example DEPARTMENT could be made ID-dependent on entity-set DIVISION, as shown in the ER diagram of figure B.1. We assume that there is a *single* identifier specified for every entity-set, although other *alternate* identifiers can be also specified.

### B.1.5 Existence Dependency

ER structures imply certain **existence dependencies** among interacting objects. An object-set $O_i$ is said to depend existentially on an object-set $O_j$ if any object of $O_i$ exists *only if* a related object of $O_j$ also exists. Accordingly, relationships depend on the existence of the associated entities. For example, an ASSIGNMENT relationship can be specified only if the corresponding involved DEPARTMENT and PROJECT entities also exist. Similarly, weak entities depend on the existence of the entities needed for their identification. For example, a DEPARTMENT entity can be specified only if the corresponding DIVISION entity needed for its identification, also exists. In ER diagrams edges represent not only the interaction of the various EER objects, but also their mutual existence dependencies. Thus, there will be directed edges (i) from relationship-sets to the entity-sets they associate; and (ii) from weak entity-sets to the entity-sets on which they depend for identification.

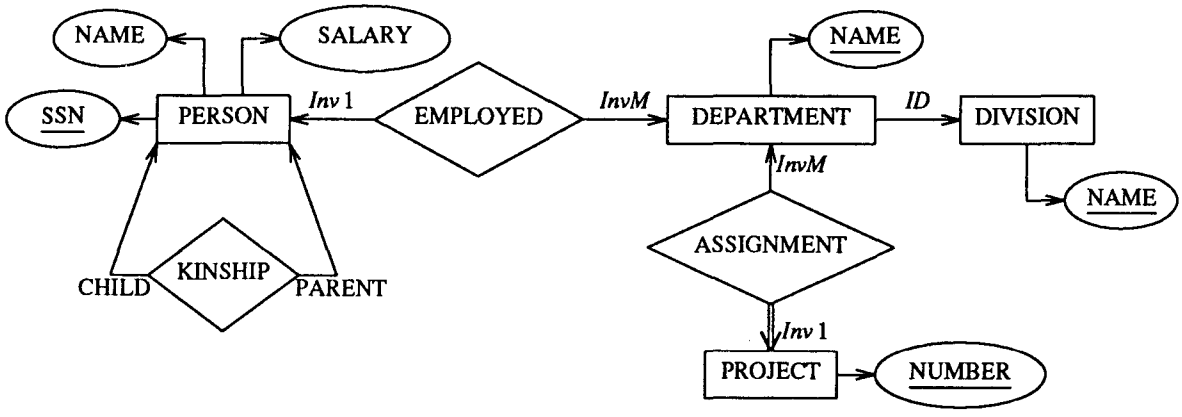



Figure B.1 An Entity-Relationship Diagram Example.

### B.1.6 Association and Involvement Cardinality

**Association–cardinality** and **involvement–cardinality** are restrictions placed on an entity-set with respect to a relationship-set. Association and involvement-cardinalities can be either *one* or *many*. For example, in the relationship-set EMPLOYED associating the DEPARTMENT and PERSON entity-sets, the DEPARTMENT entity-set would have an association-cardinality of *many* if each person is allowed to be employed in several departments, and of *one* if we wish to express the restriction of each person being employed in one department only. Conversely, the same restrictions are expressed by involvement-cardinalities of *many*, respectively *one*, of entity-set PERSON. Formally, if $R_k$ is a relationship-set that involves entity-set $E_i$, then (i) an association-cardinality of *one* for $E_i$ in $R_k$ means that, given any element of the cross-product of all the entity-sets involved in $R_k$ except $E_i$, there is *at most one* instance of $E_i$ that can be associated by $R_k$ with that element; and (ii) an involvement-cardinality of *one* for $E_i$ in $R_k$ means that an entity of $E_i$ can be involved in *at most one* relationship of $R_k$. This definition applies to any relationship-set, irrespective of the number of entity-sets it associates.

In ER diagrams association and involvement-cardinalities are represented by labels. Thus, if entity-set $E_i$ has an association (resp. involvement) cardinality of *one* with respect to $R_k$, then the edge connecting the vertices representing $E_i$ and $R_k$ is associated with label 1 (resp. *Inv* 1); and if entity-set $E_i$ has an association (resp. involvement) cardinality of *many* with respect to $R_k$, then the edge connecting the vertices representing $E_i$ and $R_k$ is associated with label *M* (resp. *InvM*). Edges that connect entity vertices with relationship vertices, and that are not associated with such a label, are assumed to correspond to cardinalities of *many*. In the ER structure represented in figure B.1, for example, the involvement-cardinalities of relationship-set EMPLOYED represent the restriction of a person being employed by at most one department, and the involvement-cardinalities of relationship-set ASSIGNMENT represent the restriction of a project being assigned to at most one department.

### B.1.7 Mandatory Involvement

The involvement of objects in relationships is, by default, optional. For example, the entities of entity-set PROJECT may or may not be involved in relationships of relationship-set ASSIGNMENT, which means that there could be projects that are not assigned to any department (e.g. because the department is not yet known). Conversely, the involvement of an object-set in a relationship-set can be specified as **mandatory**, which means that an object of that object-set must be involved, at any time, in *at least one* relationship of the respective relationship-set. Mandatory involvement of entity-sets in relationship-sets is represented graphically by *double–line* edges instead of the regular edges representing the non-mandatory (optional) involvements. For example, the mandatory involvement of entity-set PROJECT in

relationship-set ASSIGNMENT is represented as shown in the ER diagram of figure B.1, and means that each project must be assigned, at any time, to at least one department.

### B.1.8 Role

An entity-set involved in a relationship-set is said to have a *role* in that relationship-set. Roles are essential in distinguishing the multiple involvements of an entity-set in a relationship-set (represented in the corresponding ER diagram by *parallel* edges from the relationship-set vertex to the entity-set vertex). Roles are represented in ER diagrams by labels on the edges connecting the corresponding object-sets. For example, the two involvements of entity PERSON in relationship-set KINSHIP are characterized by distinct two roles, PARENT and CHILD, respectively, which are represented as shown in figure B.1.

## B.2 Extended Concepts

The concepts of entity-set, relationship-set, attribute, and value-set are fundamental in the ER model. Two abstraction capabilities that were not included in the original ER model and have been subsequently added are generalization and aggregation. The ER model extended with generalization and aggregation is called the **Extended ER** (EER) model.

### B.2.1 Generalization

**Generalization** emphasizes the similarities of entities, while abstracting away their differences. Thus, generalization views a set of entity-sets (e.g. employees, students, scientists, secretaries) as a single *generic* entity-set (e.g. persons). The attributes which are common to the entity-sets that are generalized (such as name and age) are then represented only once, associated with the generic entity-set. Similarly, relationship-sets that are common to the entity-sets that are generalized are associated with the generic entity-set. The entity-sets that are generalized can have additional attributes of their own (e.g. scientists can have degrees) and can be involved in relationship-sets in which the generic entity-set is not involved (e.g. scientists may be related to projects, while secretaries are not). The inverse of generalization is called *specialization*. A specialization entity-set *inherits* all the attributes of any of its generic entity-sets, including the entity-identifier.

### B.2.2 Types of Generalization

Generalization can abstract either homogeneous or heterogeneous entity-sets. In the first case generalization is called **homogeneous** generalization, and in the second case generalization is called **heterogeneous** kind. For homogeneous generalization, the type of the generic entity-set unifies and replaces the types of the specialization entity-sets, while for heterogeneous generalization the type of the generic entity-set is a

new *virtual type* and the types of the specialization entity-sets are preserved. While entity of any homogeneous-specialization entity-set is allowed to migrate to any other homogeneous-specialization of the same generic entity-set (that is, is allowed to change *roles*), entities of heterogeneous-specialization entity-sets are not allowed to migrate to any other entity-set. For instance, entity-sets STUDENT and EMPLOYEE can be homogeneous-generalized by generic entity-set PERSON; then a STUDENT entity is allowed to migrate to entity-set EMPLOYEE (i.e. a person can cease to be a student and become an employee, or be both a student and an employee). In contrast, entity-sets GOV.OFFICE and COMPANY can be heterogeneous-generalized by generic entity-set SPONSOR; then a COMPANY entity is not allowed to migrate to entity-set GOV.OFFICE (i.e. a company cannot 'become' a government office). Typically (but not necessarily), heterogeneous-generic entity-sets are required to be covered by their heterogeneous-specializations.

### B.2.3 Extended Entity–Relationship Diagram

We must extend the definition of the ER diagram in order to represent the new generalization construct; the extended ER diagram is called EER diagram. The vertices representing specializations are connected by directed edges labeled *ISA* to the vertices representing the corresponding generic entity-sets; for heterogeneous-generalizations the edges are double-shafted and the label is *ISA**.

The EER diagram of figure B.2 extends the ER diagram of figure B.1, with two generalization hierarchies, namely the PERSON homogeneous-generalization of entity-set EMPLOYEE and and the SPONSOR heterogeneous-generalization of entity-sets COMPANY and GOV. OFFICE. The second generalization allows the association of entity-set PROJECT with SPONSOR by relationship-set SPONSORS, which represents the sponsoring of projects by government offices and private companies. Without the generalization capability PROJECT would be associated by two different relationship-sets with the entity-sets GOV. OFFICE and COMPANY, respectively, although these relationship-sets express the same kind of association.

### B.2.4 Role Revisited

Homogeneous-generalization implies the specification of new roles for the homogeneous-generic entity-sets. Thus, if entity-set $E_i$ is a homogeneous-specialization of entity-set $E_j$, then $E_i$ and $E_j$ assume two distinct roles in their involvements with other entity-sets or relationship-sets. For example, the involvement of entity-set PERSON in relationship-set EMPLOYED in figure B.1, is replaced in figure B.2 by the involvement of entity-set EMPLOYEE in EMPLOYED, so that only a PERSON in the role of EMPLOYEE is associated by relationship-set EMPLOYED.

### B.2.5 Aggregation

Aggregation is intended as a construct that can be applied over previously aggregated objects as many times as one wishes. For example, suppose that a relationship-set ASSIGNMENT associates entity-sets PROJECT and DEPARTMENT, as shown in figure B.2. We wish to relate PERSON (EMPLOYEE) and ASSIGNMENT. We could define a ternary relationship-set WORKS between entity-sets PROJECT, DEPARTMENT, and PERSON, but then relationships of this ternary relationship-set could associate PROJECT and DEPARTMENT entities that are not associated by any ASSIGNMENT relationship, contrary to our intention. The obvious and natural solution is to specify the relationship-set WORKS between relationship-set ASSIGNMENT and entity-set EMPLOYEE, as shown in figure B.2. Note that no extension is needed for the EER diagram in order to accommodate this new aggregation construct because of the use of directed edges.

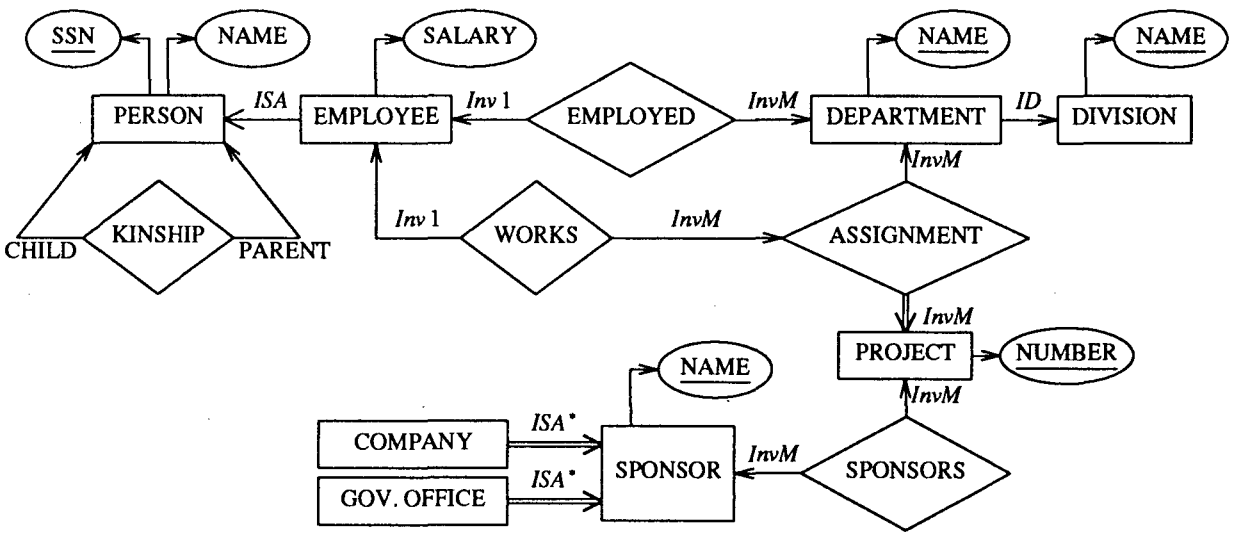



Figure B.2 An Extended Entity-Relationship Diagram Example.

LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
INFORMATION RESOURCES DEPARTMENT
BERKELEY, CALIFORNIA 94720