Freja
An object-oriented CASE Tool
Tutorial and Reference Manual

Mjølner Informatics Report
MIA 93-31(3.0)
February 1997

Contents

1. Introduction

Freja is an object-oriented CASE tool supporting system development with the Unified Modeling Language [UML1.0] and with BETA as the implementation language. Together with Sif, the Mjølner Source Browser and Editor [MIA 91-11], Freja supports a smooth transition from design diagrams to implementation code and vice versa.

The CASE tool offers:

- **Diagram editing**
  Design diagrams can be created and modified.
- **Automatic code generation**
  Code skeletons are generated automatically from the design diagrams.
- **Reverse engineering**
  Design diagrams can be automatically created from the program code.
- **Simultaneous editing of design descriptions and program code**
  The design descriptions and the corresponding program code can be viewed and edited simultaneously, i.e. modifications in the design diagrams are reflected immediately in the program code and vice versa.
- **Structure editing**
  The editing technique in the diagram editor as well as the program code editor is structure editing, which means that syntax errors are prevented. The user is offered templates of the legal constructs of the language.
- **Integrated system development environment**
  The diagram editor Freja is integrated with the textual structure editor Sif which in turn is integrated with the rest of the Mjølner BETA System, e.g. the compiler and the debugger Valhalla. In this way the user is offered a system development environment that supports development of design diagrams, generation of program skeletons, filling out the code details, compiling and debugging. During the detailed implementation, testing and debugging process, the overall structure of the program may have changed, which makes the original design diagrams obsolete, but then new design diagrams can be generated automatically.

Freja is a Design/OA [Jensen 91] application written in BETA. Access to Design/OA is provided through a BETA library called DesignEnv [Knudsen 94], that interfaces to the Design/OA C library.

2. The Notation

As mentioned the graphical design notation used in Freja is UML.

UML is the result of the efforts of Jim Rumbaugh, Grady Booch and Ivar Jacobson[1] to unify their methods: OMT (Object Modeling Technique), Booch and OOSE (Object-Oriented Software Engineering).

The current version of Freja supports to types of diagrams: Class diagrams and object diagrams.

The class diagrams illustrate the static structure of the model with symbols for class, aggregation, association and specialization. In Freja design diagrams and code are tightly integrated. As a consequence class diagrams can also be seen as an illustration of the static structure of a BETA program. In this sense UML class symbols correspond to BETA patterns, UML aggregations to BETA whole-part and reference compositions and UML specializations to BETA specializations. A symbol that is not directly found as a language construct in BETA is an association.

Object diagrams focuse on objects and the dynamic aspects of a BETA program. Object diagrams illustrate static and dynamic objects, composition, dynamic object creation and operation calls. The object diagrams in Freja in general correspond to UML collaboration diagrams, although in the current version of Freja the graphical symbols do not exactly match the ones used in UML.

Action sequences (i.e. the Do-Parts of objects) and enter/exit parts are not described graphically.

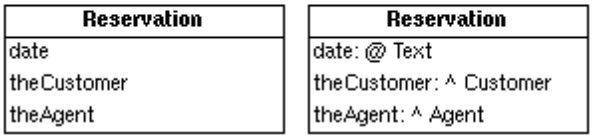
The most recent updates on the Unified Modeling Language are avialable via the worldwide web: http://www.rational.com.

2.1 Class Diagrams

2.1.1 Class

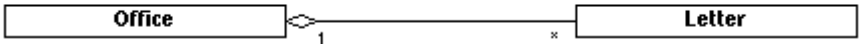
A class is shown as a box with a title (the name of the class) and an optional list of local attributes, operations and classes. The two class symbols shown below are actually two different views on the same class (`Reservation`). The left symbol showing only the names of the attributes and the right symbol showing full type information of the attributes. In the following examples one or the other view will be used depending on the illustrative purposes of the example.

[1] All employees of Rational Software Corporation



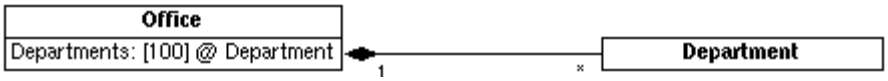
2.1.2 Aggregation

Aggregation is shown as a line connecting two class symbols. A diamond is attached to one end of the line. The diamond is attached to the class that is the aggregate.



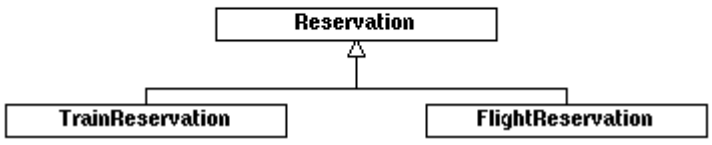
Multiplicities are shown at each end of the aggregation. Multiplicities may either by an integer value or an integer interval on the form *lower-bound...upper-bound*. In addition, the star character (*) may be used for upper-bound, denoting an unlimited upper bound. In the above example the multiplicities mean that an office may contain zero or more letters.

The diamond attached to the aggregate may either be hollow or filled. If it is hollow the implementation of the aggregation is said to be *by-reference* and if it is filled it is said to be *by-value*. The first case corresponds to reference composition in BETA, the second to whole-part composition.



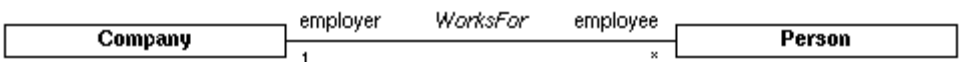
2.1.3 Specialization

Specialization is illustrated as a tree of class symbols. A line is drawn between the subclass and the superclass with an arrow at the end connected to the superclass.



2.1.4 Association

An association is shown as a line connecting two class symbols.



Multiplicities are shown on each end of the association. The syntax of the multiplicities on associations is exactly the same as the syntax of the multiplicities on aggregations. In the above example the multiplicities mean that a company may be associated to zero or more persons through the relation works-for and a person is related to exactly one company (i.e. in this example a person may not have more than one employer).

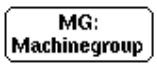
Apart from multiplicities role names may be attached to each end of the association. A role name indicates the role played by the class attached to the end of the line near the role. For instance in the above example the role of a company seen from the view of a person is an employer (and vice versa).

Finally the association may be given a name. In the above example the name of the association is `WorksFor`.

2.2 Object Diagrams

2.2.1 Static Object

Rounded box in solid linestyle. If the object is pattern-defined the name of the pattern is shown following the ":" after the name of the object. If the object is singularly defined only the name of the object is shown.



2.2.2 Dynamic Object

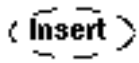
Rounded box in dashed linestyle. The name of the pattern that the object is an instance of is shown inside the box.

Notice:

A dynamic object is shown inside the object where the corresponding pattern is defined and not where the object is created ("the dynamic object is shown where it conceptually belongs").

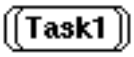
2.2.3 Operation

Ellipse in a dashed linestyle. The name of the operation is shown inside the ellipse. Detailed operations are for practical reasons shown as rounded boxes in a dashed linestyle. Detailed operations can be distinguished from detailed dynamic objects through the fact that the text "(PROC)" is appended to the name of the operation when it is detailed.



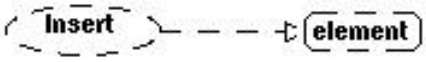
2.2.4 Active Object

Both static and dynamic objects can be active ("have an execution thread of their own", components in BETA terms). These are shown with two parallel lines inside the object - one in each side.



2.2.5 Dynamic Object Creation

Arrow in dashed linestyle. Going from the creating object to the created dynamic object.



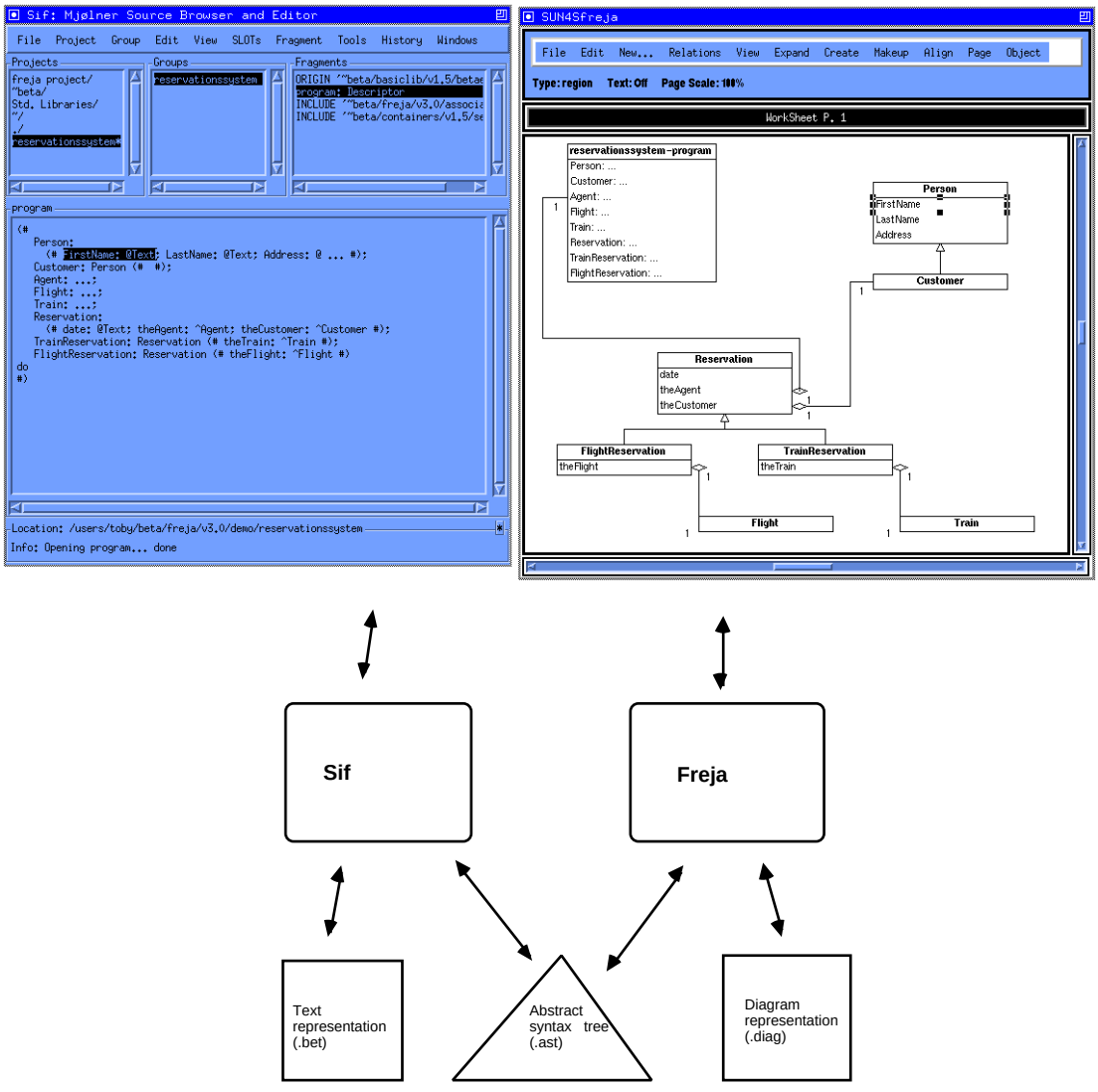
2.2.6 Operation Call

Arrow in solid linestyle. Going from the calling object to the called operation.

2.2.7 Composition

Currently only whole-part composition is displayed in the object diagrams. In the notation, part objects are simply shown nested inside their enclosing objects.

3. The Architecture



The figure illustrates how Freja and Sif work together. Freja and Sif are two applications that are tightly integrated in two ways.

Firstly Freja and Sif share one common representation of the program being developed. The structure editing technique gives a convenient representation, namely an abstract syntax tree (AST). The AST is presented (prettyprinted) textually in Sif and graphically in Freja.

Secondly, they communicate about changes in focus or modifications of the AST. Sif manipulates the AST through the textual representation and Freja manipulates the AST through the graphical representation. Whenever one of the editors modifies the AST, the other editor is notified and the other representation can be updated accordingly. Since the graphical UML syntax only reflects the overall structure of BETA programs, many modifications in the textual representation may not affect the graphical representation. Conversely, almost every modification of the design

diagram, except changing the layout of the diagrams, implies that the textual representation must be updated.

Both representations need not be visible at the same time. In the start of the development process the user might prefer only to see the diagrams, whereas the textual representation becomes more important later on. Since the diagrams and the program are prettyprints of the AST, they can always be reproduced. The textual prettyprinting is always reproducible. However if the user makes changes to the layout of the diagrams and want to keep the changes, the diagrams must be saved like the AST.

There are 3 important file types: .ast, .bet and .diag files.

The .ast and the .bet files are well-known to BETA programmers. The .ast file contains the AST and the .bet file is the textual version of the AST. Sif automatically produces the .bet file from the .ast file.

Layout of diagrams is only preserved through use of Freja

The .diag file contains a representation of the diagrams including layout information and references to the corresponding nodes in the AST. This means that if the user wants to keep the layout of the diagrams, the AST must only be modified through Freja (and/or Sif, when Sif has been.started through Freja).

4. Tutorial

4.1 How to Get Started

4.1.1 Class Diagrams

Freja is started from an xterm in one of the following ways:

```
freja
freja foo.bet
freja foo.ast
freja foo.diag
freja foo
```

Using the first option only a menu bar will be presented to the user.

The second and the third option opens a BETA fragment (`foo`) and the fourth opens a saved diagram. The last option checks the timestamps of the files on disk and opens whichever is newest.

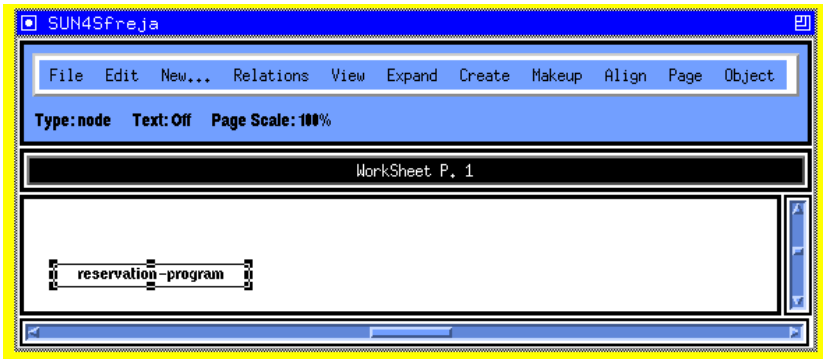
4.1.2 Object Diagrams

Object diagrams can be generated for any BETA program that has been checked. If the BETA program has not been checked or contains semantic errors object diagrams **cannot** be generated.

4.2 Editing

4.2.1 Creating a New Diagram

To create a new diagram use the command New BETA program or **N**ew BETA library in the **File Menu**. Below the command **New BETA program** has been used:

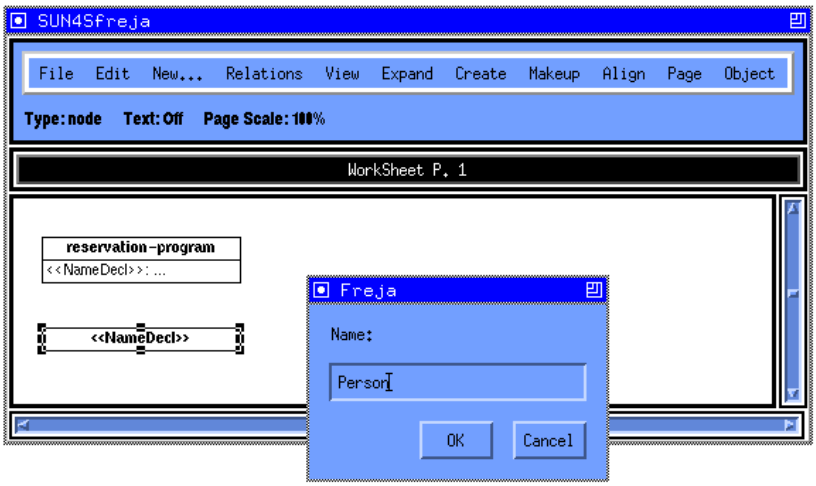


The title of the diagram corresponds to the name and the category (`program`) of the new BETA fragment:

```
ORIGIN '~beta/basiclib/v1.4/betaenv'
-- program: DescriptorForm --
(# do #)
```

4.2.2 Creating a New Class

When an attribute or a title of a diagram is selected it is possible to create a new class using the **Class** command in the **New...** menu. Doing this you get a dialog where the name of the new class can be typed in.

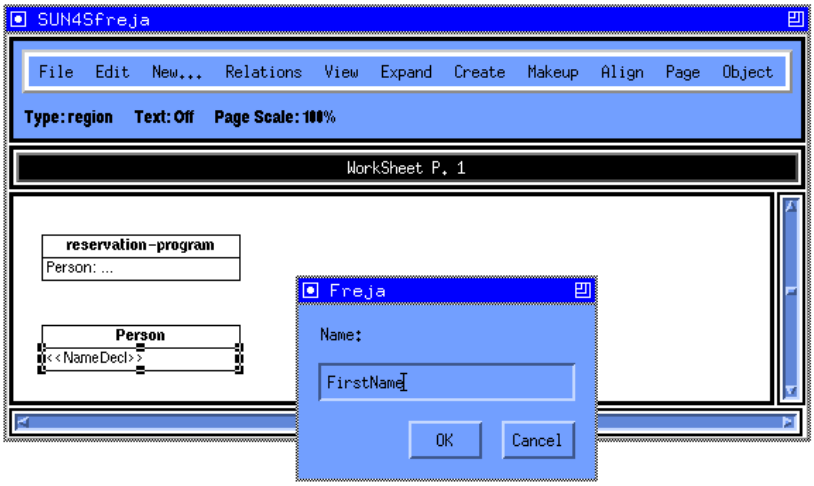


The BETA code corresponding to new class looks like this:

```
Person:(# #)
```

4.2.3 Adding Attributes and Operations

To add an attribute to a class, simply select the class and use the **Static Reference Attribute** or the **Dynamic Reference Attribute** command in the **New...** menu. As above a dialog is presented and you can now type in the name of the new attribute.



In case a static reference attribute is created the corresponding BETA code looks like this:

```
Person(# FirstName:@<<ObjectSpecification>> #)
```

And in case of a dynamic reference attribute like this:

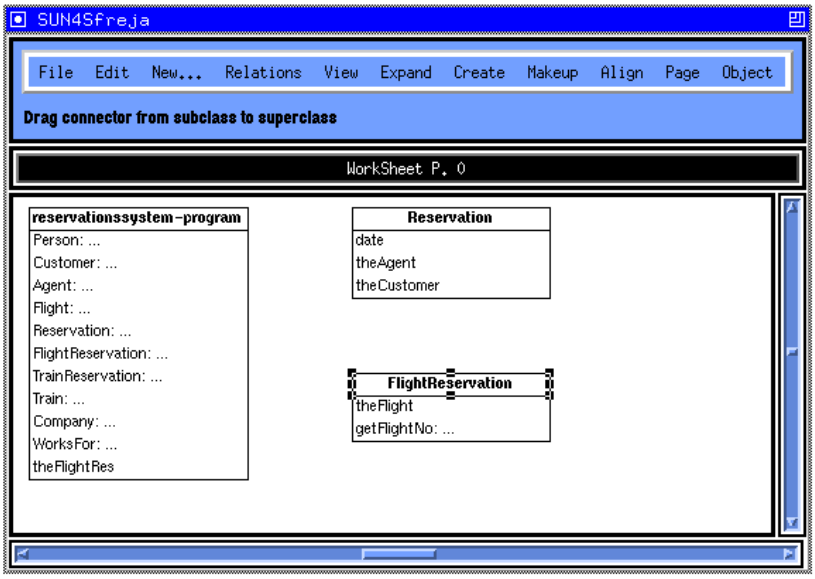
```
Person(# FirstName:^<<AttributeDenotation>> #)
```

Notice the `<<ObjectSpecification>>` and `<<AttributeDenotation>>` parts of the above code. These are placeholders or nonterminals. In the above two cases they ensure that, though the type of the attribute has not yet been specified, the code is still syntactically correct. Placeholders like the ones above can either be filled in using subeditors or Sif (see later).

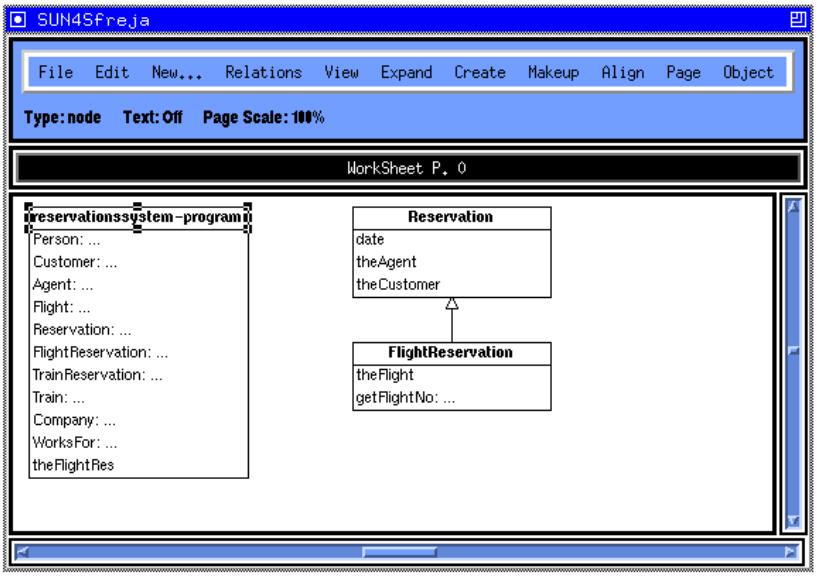
In the same manner as with attributes a new operation can be added to a class using the **Operation / Local Class** command in the **New...** menu.

4.2.4 Specifying Specialization

In the following situation two class declarations of the `reservation` program have been detailed. To specify that `FlightReservation` is a subclass of `Reservation` the **Specialization** command of the **Relations** menu is used. As the status message of the menubar states in the example below it is now possible to drag a specilization connector from the subclass to the superclass.



The result will look like this:



The code correspondingly changes from:

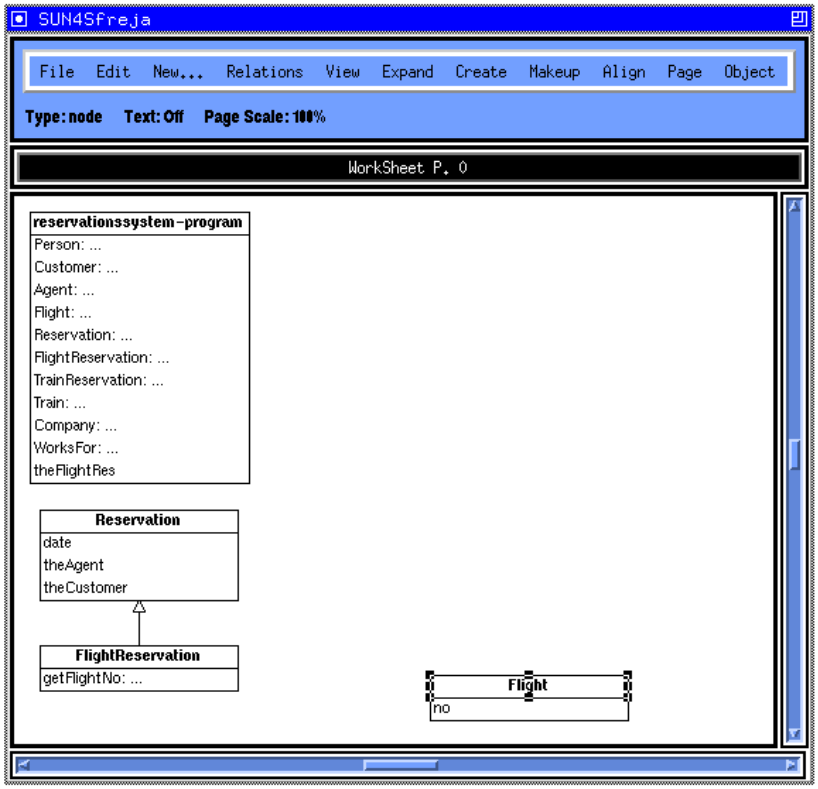
```
Reservation: (# ... #);
FlightReservation: (# ... #);
```

to:

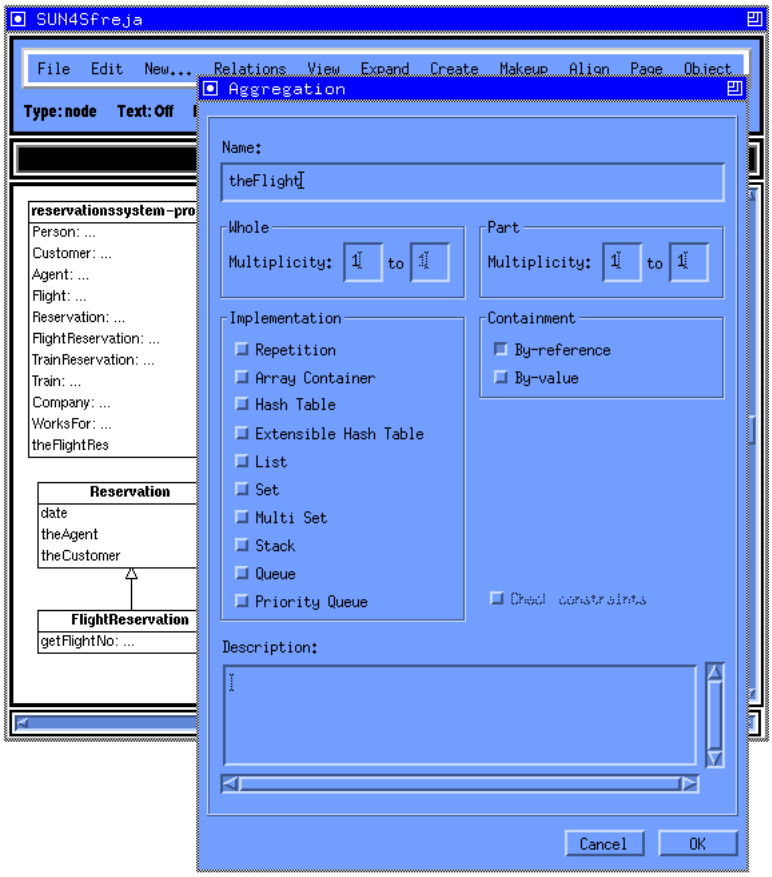
```
Reservation: (# ... #);
FlightReservation: Reservation(# ... #);
```

4.2.5 Specifying Aggregation

Below three class declarations of the `reservation` program have been detailed.



We wish to specifify a *by-reference* aggregation between `FlightReservation` and `Flight`. To do this select **Aggregation** in the **Relations** menu. This brings up the following dialog:

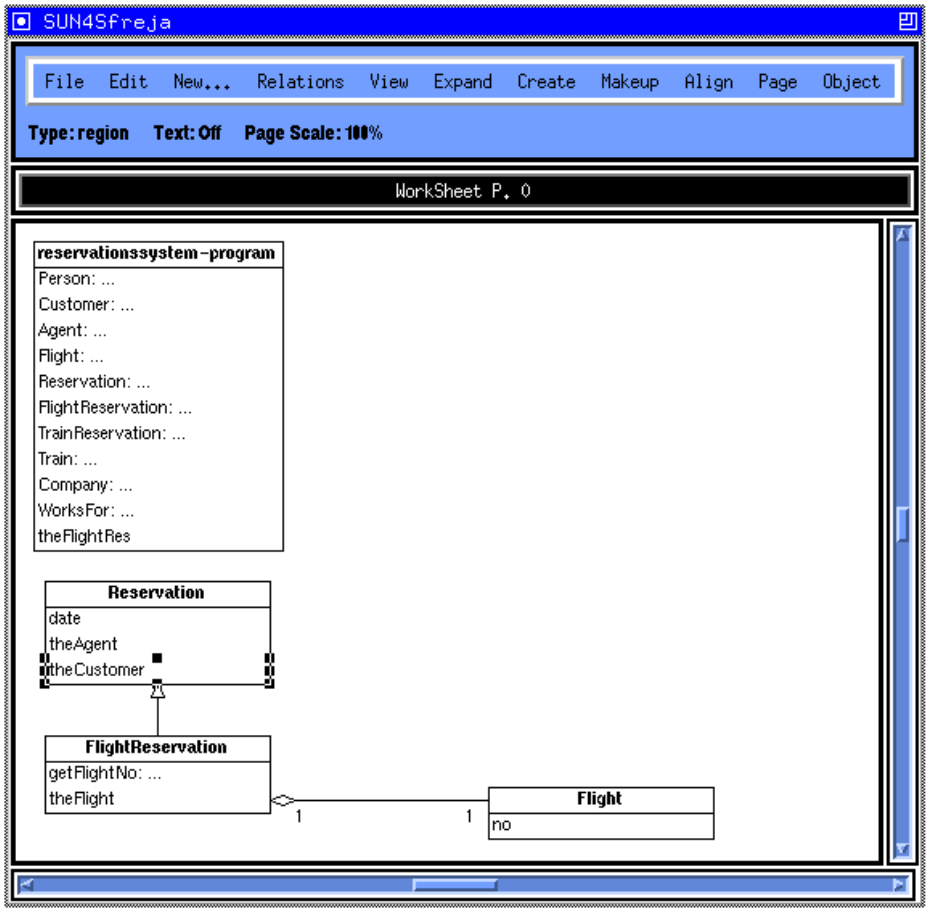


The aggregation dialog makes it possible to:

- give the aggregation a name,
- specifify the multiplicities of the whole and the part,
- specify the implementation of a *one-to-many*[1] aggregation; that is to specify if a repetition or one of the basic container classes should be used in the resulting code,
- specify if the aggregation is to be implemented *by-reference* or *by-value*,
- give a textual description of the aggregation.

In this example we only specify the name of the aggregation (defaults are such that we get a *one-to-one by-reference* aggregation).

After choosing **Ok** in the dialog drag a connector from the class designating the whole (`FlightReservation`) to the class designating the part (`Flight`).



The BETA code correspondingly changes from:

```
Flight: (# ... #);
FlightReservation: Reservation(# getFlightNo: ... #)
```

to:

---

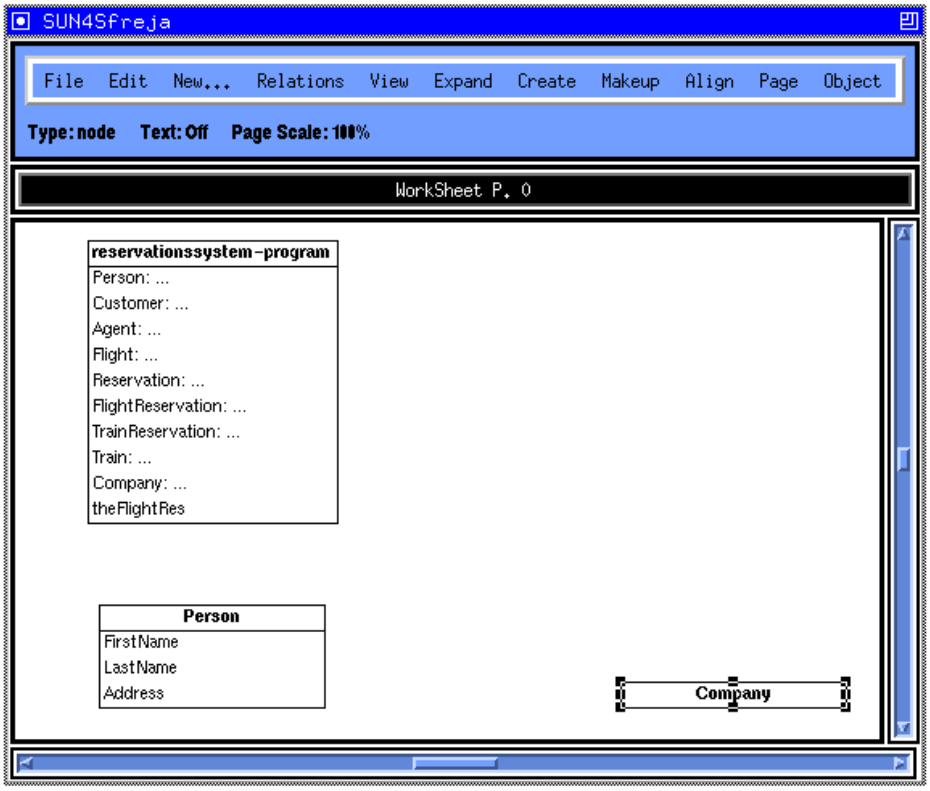
[1] An aggregation is said to be *one-to-many* if the multiplicity of the part is one and the multiplicity of the whole is more than one.

```
Flight: (# ... #);
FlightReservation: Reservation(# getFlightNo: ...; theFlight:^Flight #)
```

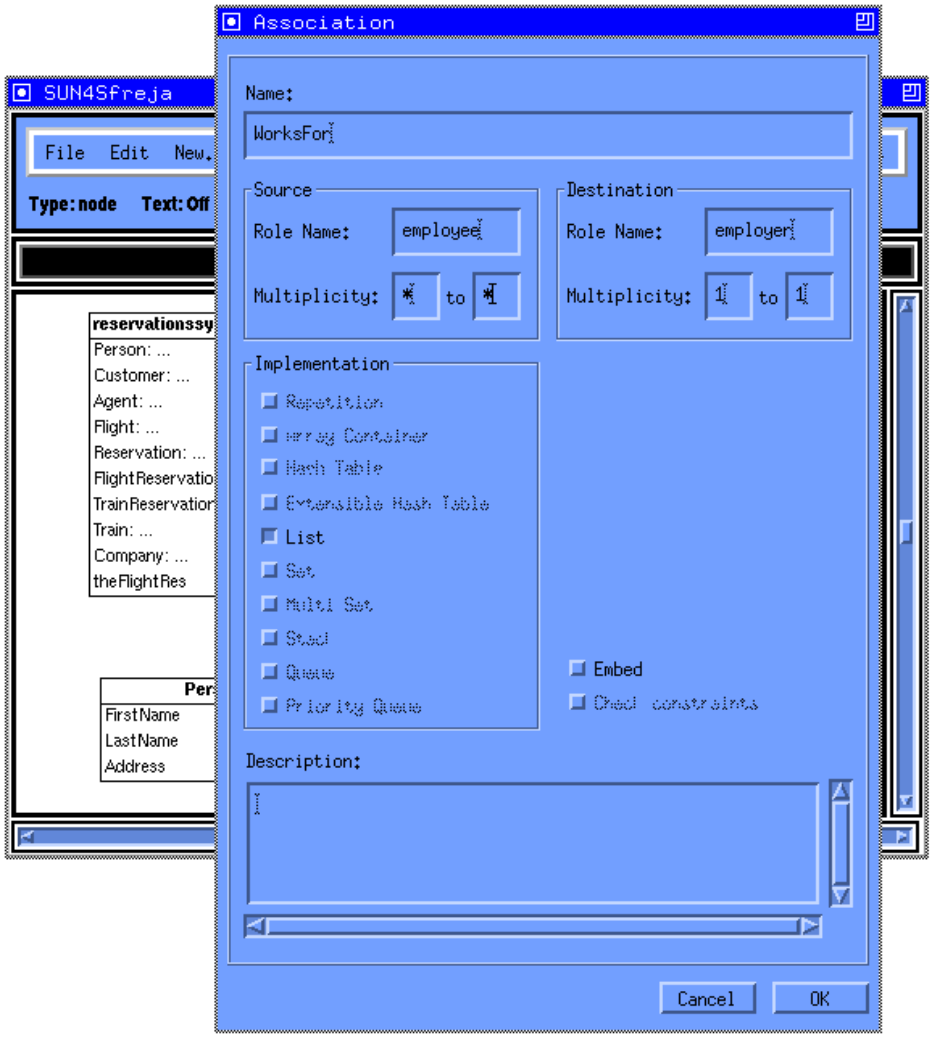
As can be seen the name of the aggregation is reused in the code and the fact that we created a *by-reference* aggregation is reflected by the dynamic reference implementation of the new declaration (had we instead chosen *by-value* a static reference would have been declared).

4.2.6 Specifying Association

Below two class declarations of the `reservation` program have been detailed.



We wish to specify a one-to-many association between `Company` and `Person`. To do this select **Association** in the **Relations** menu. This brings up the following dialog:

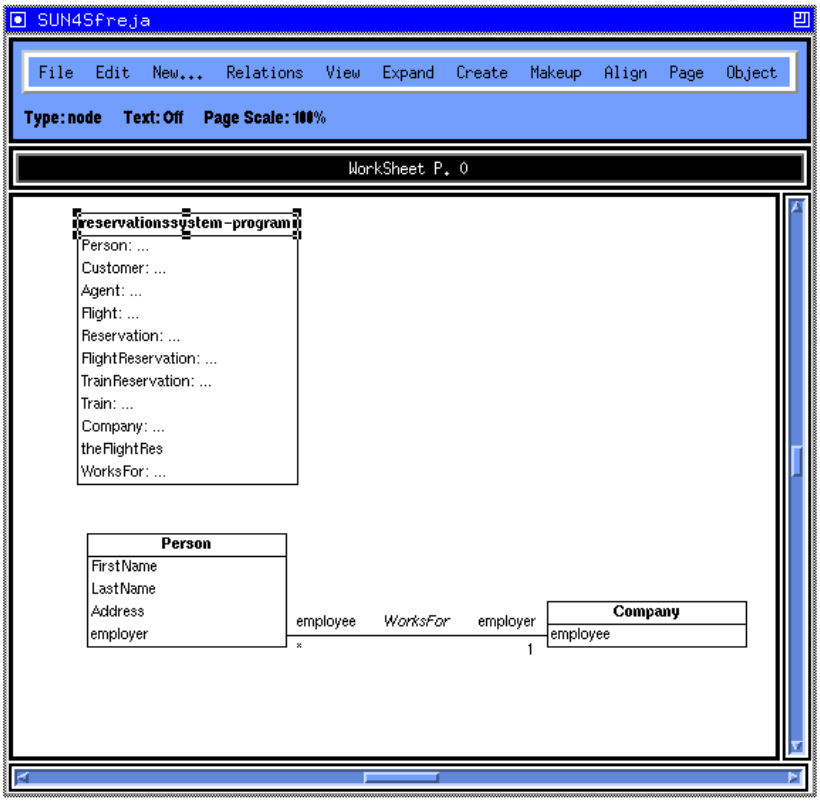


The Association dialog makes it possible to:

- give the association a name,
- specify role names for both sides of the association,
- specify multiplicities for both sides of the association,
- specify the implemention of a *one-to-many* or *many-to-many* association; that is to specify if a repetion or one of the basic container classes should be used in the resulting association code,
- specify if the association should be *embeded* or not; that is to specify if the generated code is to result in a separate association class or if it is to be embedded in the source and destination classes (see below),
- give a textual description of the association.

In this example we specify the name of the association, `WorksFor`, the name of the source role, `employee`, the name of the destination role, `employer`, and the multiplicities. Specifying that the multiplicity of the `employee` end of the association is more than one (in this case "`*`") automatically sets the implementation to the default, `List`.

After choosing **`Ok`** in the dialog drag a connector from the class designating the source (`Person`) to the class designating the destination (`Company`).



The resulting code looks like this:

```
Person: (# ...; employer:^WorksFor #);
Company: (# employee:^WorksFor #);
WorksFor: OneToManyAssociation
  (# oneType:: Company; manyElmType:: Person #)
```

Where the things added to code are the `employer` and `employee` declarations and the `WorksFor` class.

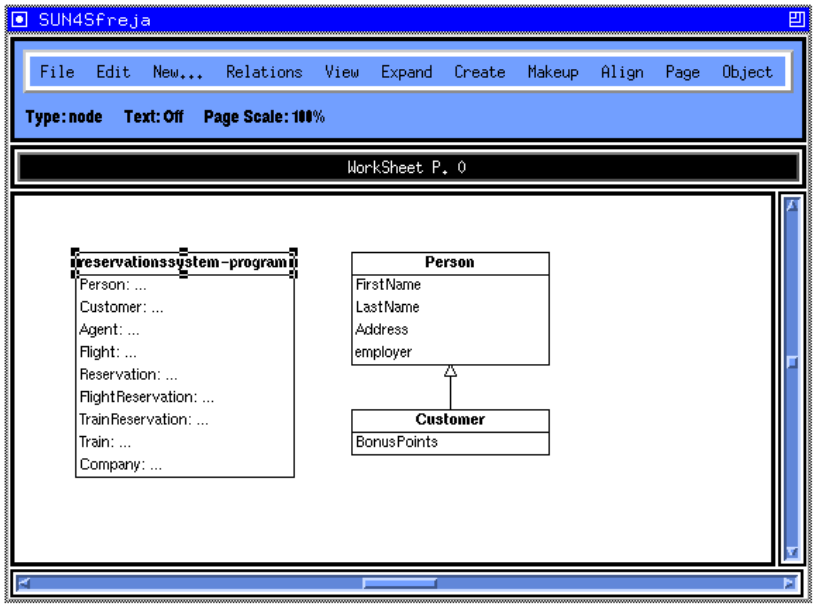
Had the embed option been chosen the code would have looked like this:

```
Person: (# ...; employer:@AssociationOne(# element:: Company #) #);
Company: (# employee:@AssociationMany(# element:: Person #) #)
```

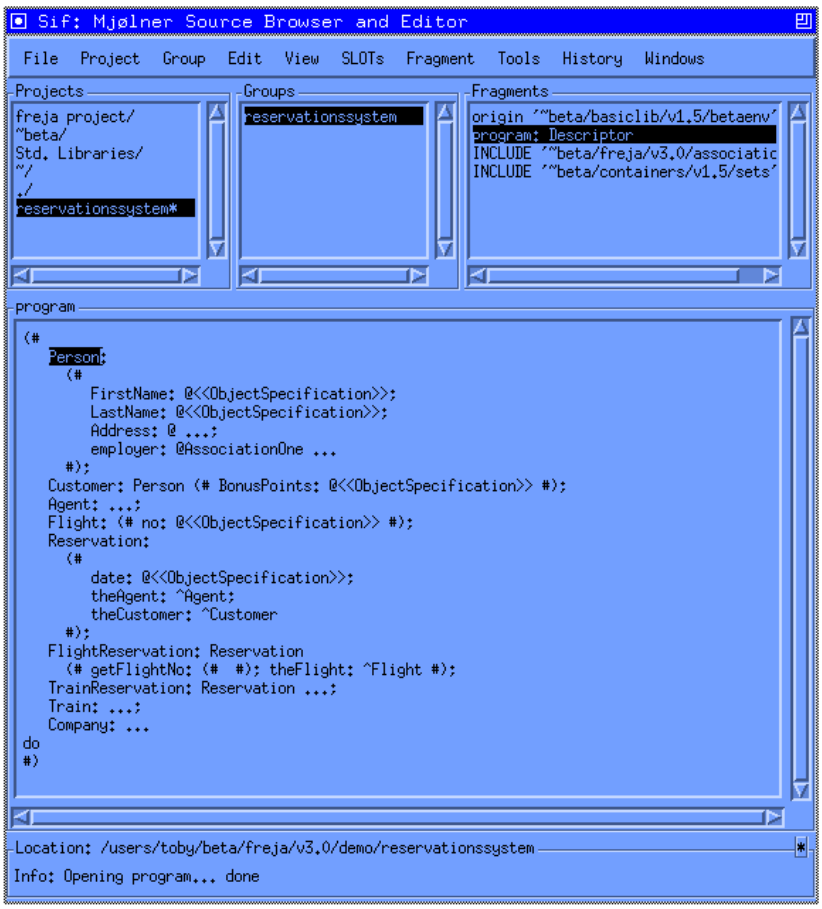
Notice that in this case no separate association class is generated.

4.2.7 Completing the Code in Sif

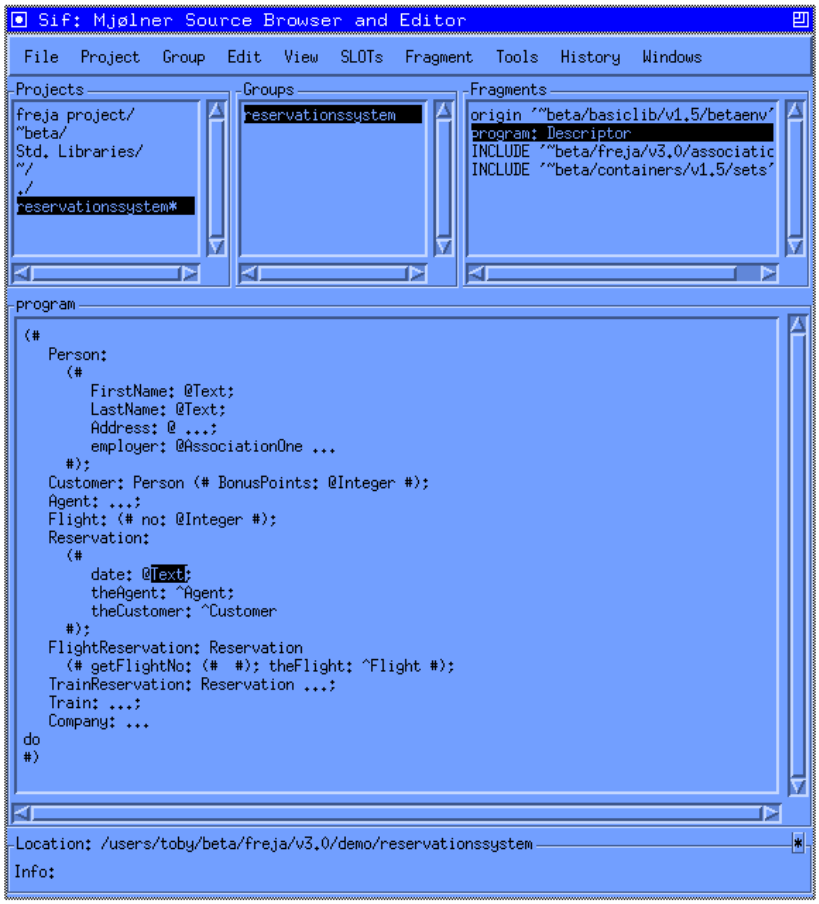
Consider the following design diagram which could be considered as complete at the design level:



To fill out the code details, Sif, the textual representation editor must be activated, if not already visible (use **Show Sif** in the **File** menu to make Sif visible):



The textual representation of the design is, however, not complete. It still contains unexpanded placeholders. For example the type of the `FirstName` attribute of the `Person` class needs to be specified. To do this simply select the placeholder `<<ObjectSpecification>>` and start typing the name of the type, e.g. `Text`. Finish this textediting by choosing `Parse Text` form Sif's `Edit` menu (or rather use the keyboard shortcut `ctrl-t`). In this manner all the placeholders can be filled in:



In this case all placeholders disappear, but this is not always the case. Optional placeholders may exist in the program. For example the `getFlightNo` operation of the FlightReservation class may look like this:

```
getFlightNo:
  (# <<AttributeDeclOpt>>
     <<EnterPartOpt>>
     do theFlight.no->putint
     <<ExitPartOpt>>
   #)
```

To get rid of these optionals select the part of the code containing them and choose **Remove Optionals** from Sif's or from Freja's **Edit** menu. In the above example the effect will be:

```
getFlightNo:
  (#
     do theFlight.no->putint
   #)
```

After filling out more code in Sif, e.g. the action sequences of the objects, the compiler is activated in the **Tools** menu of Sif (or alternatively, if one only whiches to check the program for semantic errors, the checker can be activated using the **Check** entry of Freja's **Edit** menu).

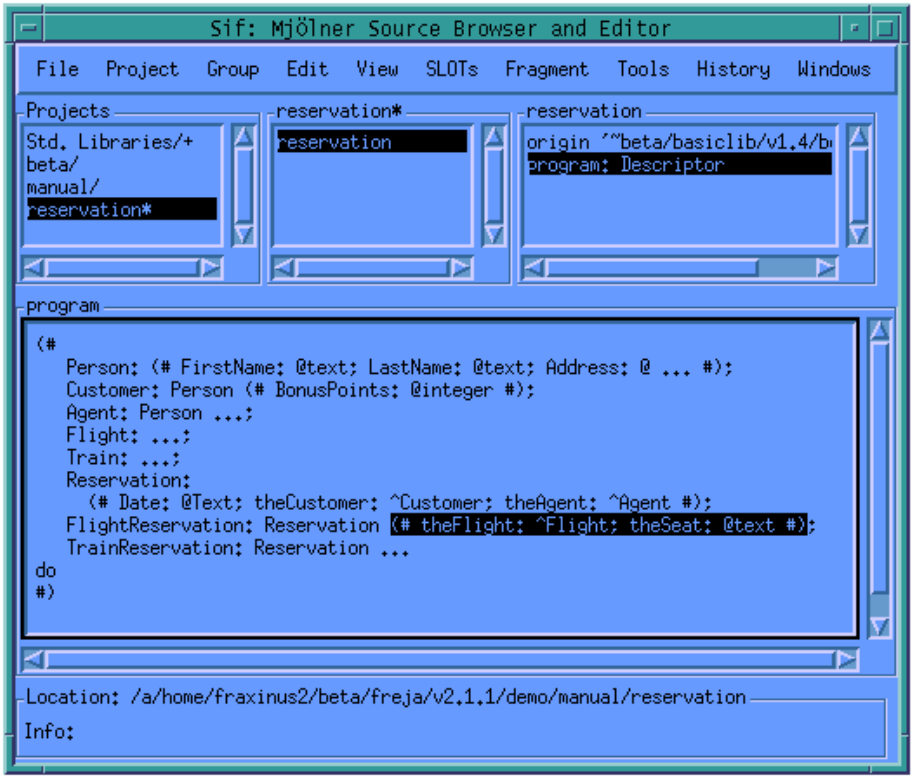
If the compiler reports a semantic error, the corresponding node is also selected in Freja.

When the program is semantically correct and it can be executed and tested. In this phase Freja will typically not be activated. When the program has been tested and considered fulfilling the requirements, the design diagrams may have become obsolete, and need to be updated, i.e. reverse engineering is necessary. This is done by activating Freja on the BETA program.

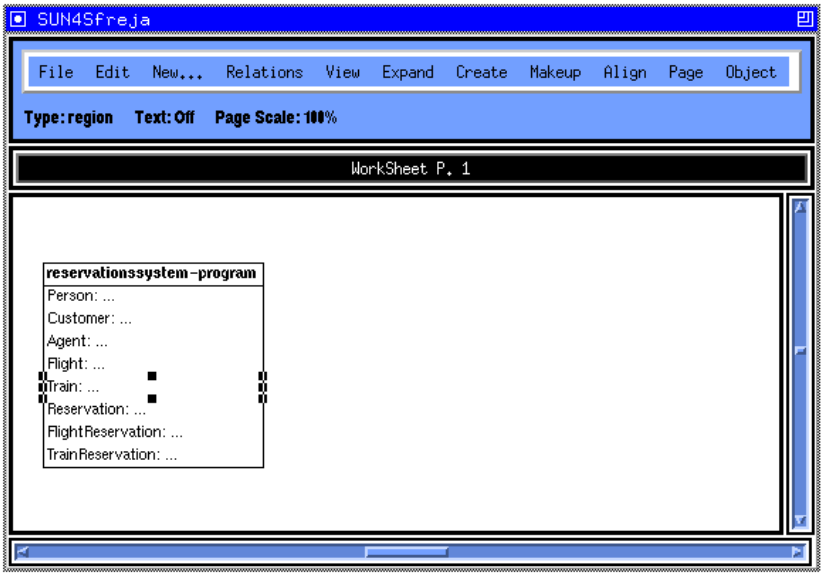
4.3 Reverse Engineering

4.3.1 Class Diagrams

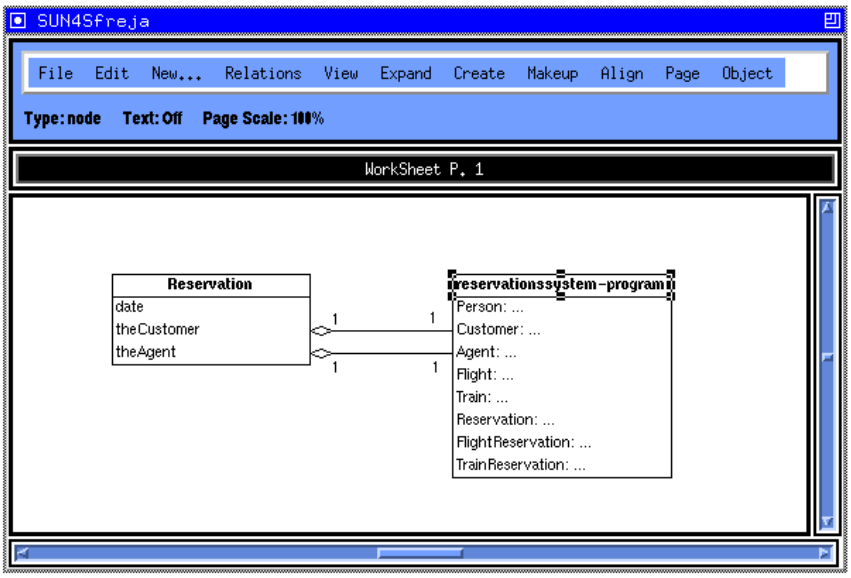
To extract the design diagrams from the BETA code, the program is opened in Freja. Consider the following program shown in Sif:



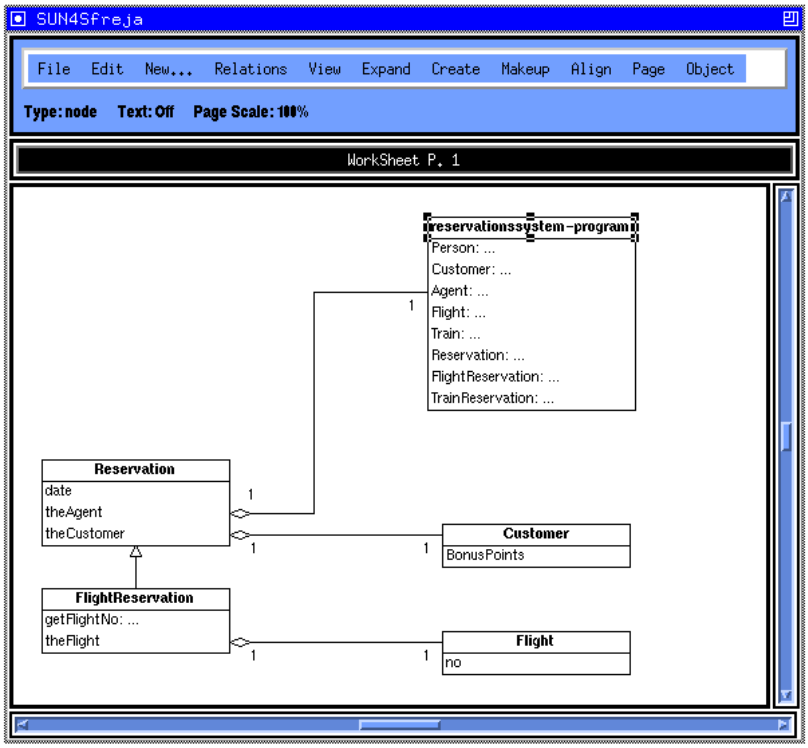
When opening this program in Freja the following diagram initially appears:



This is an abstract design view of the program. More detailed design diagrams can be generated by detailing selected parts of the diagrams. If an attribute contains three dots (...) it can be detailed. Below the class `Reservation` is detailed by double-clicking on the `Reservation` attribute:

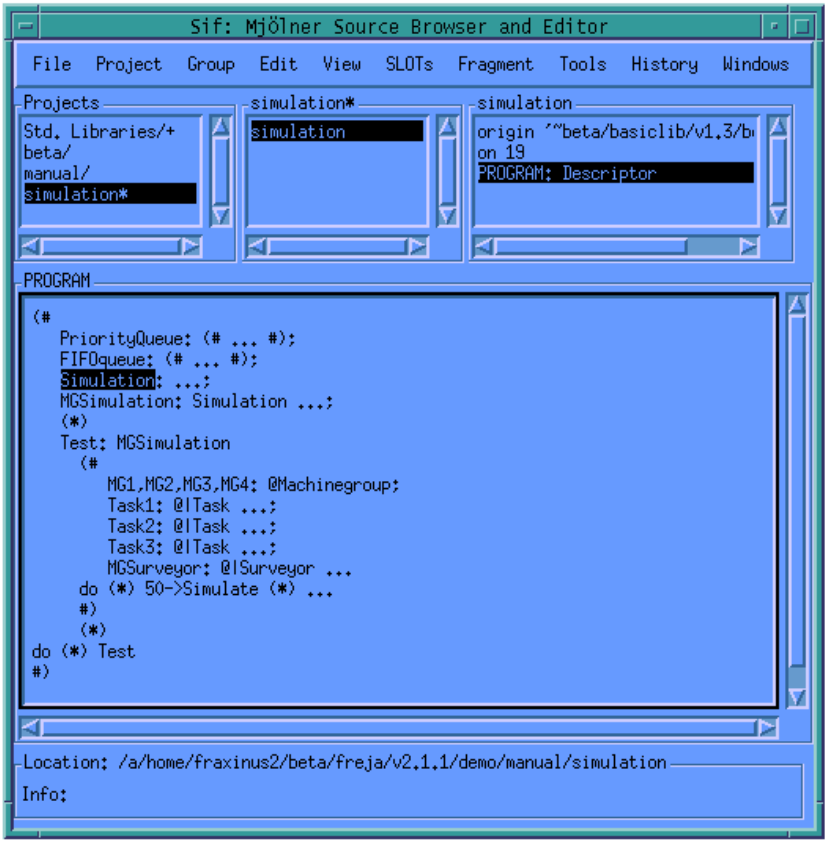


If a detailed class is related to other visible classes, possible relations like specialization or dynamic reference is automatically shown (if the program has been checked for semantic errors):



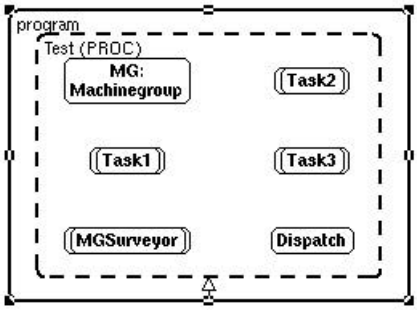
4.3.2 Object Diagrams

Object diagrams are generated in the same dynamic way as when generating the class diagrams. The reason for not detailing all the diagrams automatically is that, a fully detailed diagram often will evolve into something to big and to difficult to overview. Consider the following BETA program shown in Sif.

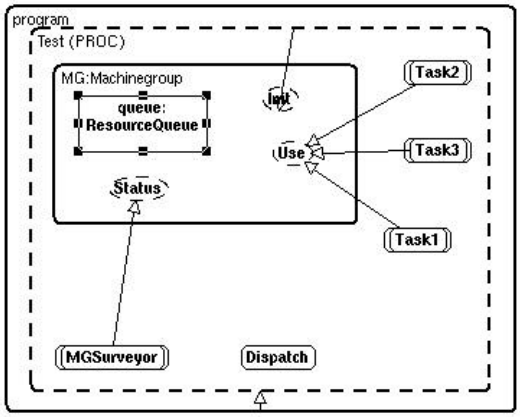


To get an object diagram for this program the **Show Object Diagram** command is used in the Freja's **Object** menu and a new window, a so-called object page is opened, containing:

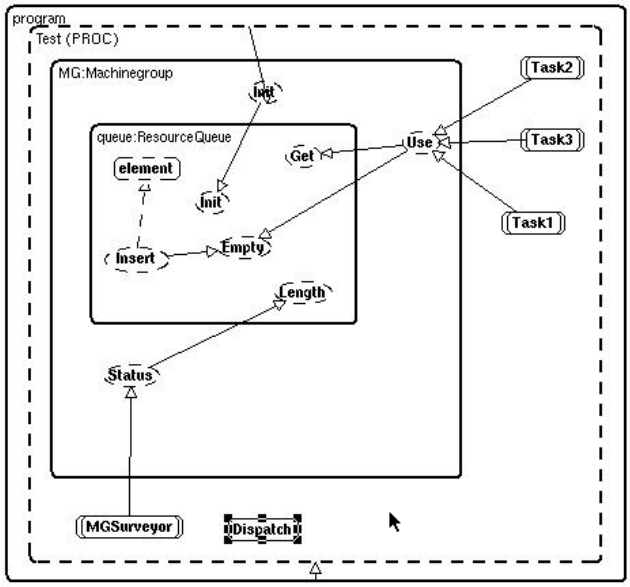
Which tells that the do part of the object program calls an operation `test`. Double-clicking on test shows the local objects of the operation:



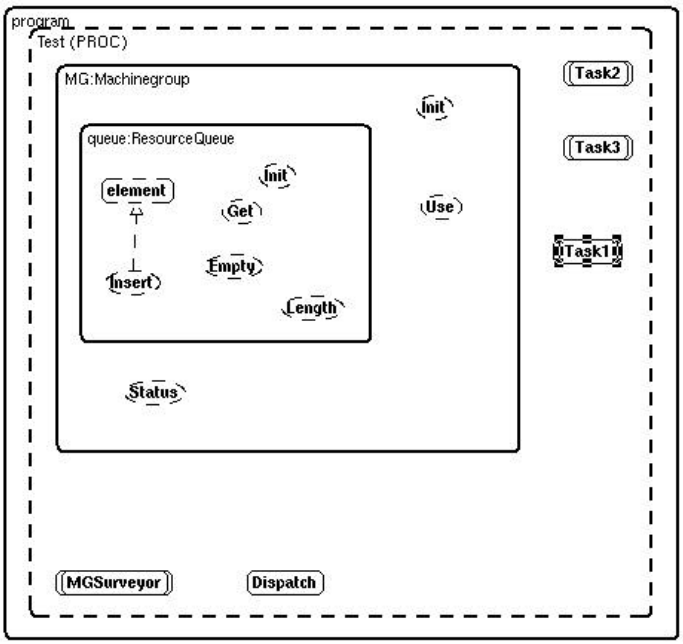
It contains 4 active objects and 2 static objects. Detailing `MG` gives:



Now we can, among other things, see that `MGSurveyor` calls the local operation pattern `Status` of `MG`. Detailing `queue` gives an even more detailed picture:



We can see that `Insert` creates the dynamic object `element`, and that `MGSurveyor` calls Status which in turn calls the Length operation of queue. If the number of arrows get to many, they can be filtered away. Below the operation call arrows are hidden:



5. Reference Manual

5.1 How to Get Started

5.1.1 Class diagrams

Freja is started from an xterm in one of the following ways:

```
freja
freja foo.bet
freja foo.ast
freja foo.diag
freja foo
```

Using the first option only a menu bar will be presented to the user. How to continue from here: see the **File Menu** section.

The second and the third option opens a BETA fragment (`foo`) and the fourth opens a saved diagram. All three of these situations are described in further detail under **Open...** in the **File** menu section. The last option checks the timestamps of the files on disk and opens whichever is newest.

5.1.2 Object Diagrams

Object diagrams can be generated for any BETA program that has been checked. If the BETA program has not been checked or contains semantic errors object diagrams **cannot** be generated.

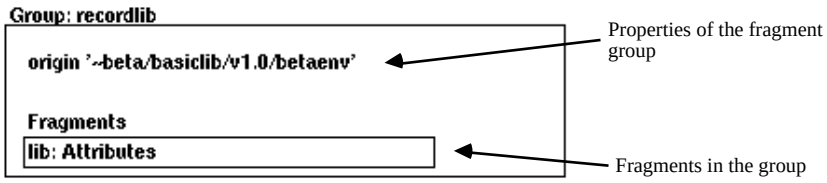
To activate the object diagrams you first have to activate the class diagram part of Freja. Having done this one or more work sheets containing class diagrams will be open. To generate an object diagram you select the title-node of a class diagram shown on a work sheet and choose Show Object Diagram from the **Object** menu. The result of this will be a new page ("**Object Page**") with a box whose title corresponds to the chosen class diagram. The only exception here is that you cannot generate an object diagram for a class diagram that corresponds to a BETA library. In that case you will have to detail one of the entries of this class diagram and then go forth as described above.

5.2 Work Sheets

The page title of a work sheet is on the form "**WorkSheet p. n**", where n is the page number. A work sheet is a page containing class diagrams. The class diagrams on a single work sheet is a reflection of the classes (patterns) contained in one or more BETA fragments.

5.3 The Group Page

If one or more work sheets with class diagrams are open choosing **Show Group Window** from the Pages menu a page called "**Group Window**" will be opened. This page will contain one or more *group diagrams*.



Group Diagram

The number of group diagrams in the Group Window corresponds to the number of currently open BETA fragment groups.

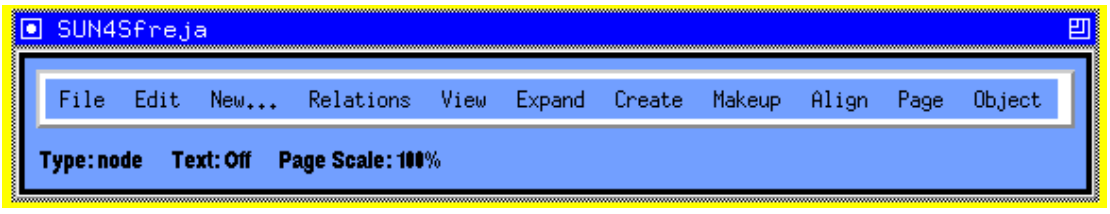
The Group Diagram displays the properties and the fragments defined in the group. The properties and the fragments can be detailed, i.e. shown in detail, by selecting a node and then select **Detail** in the **View** menu. The detailing can also be performed by double-clicking on the nodes. By e.g. double-clicking on the `lib: Attributes` node, the declarations in the fragment will be presented as a class diagram in on a work sheet (if it is already detailed, the detailed version will become current focus). Doubleclicking `Origin`, `Include` or `Body` fields in a group diagram will open that group and bring up a group diagram for it.

5.4 Object Pages

The page title of an object page is on the form "**Object Page p. n**", where n is the page number. An object page is a page containing an object diagram. The object diagram on an object page is a reflection of the dynamic aspects of a BETA program. Object diagrams illustrate static and dynamic objects, composition, dynamic object creation and operation calls.

The box that is initially shown on the object page is an entity whose contents corresponds to the objects contained in the object descriptor of the pattern that was chosen to be subject to object diagram generation (in the case where the chosen class diagram represents a BETA program-descriptor the object diagram will simply represent the objects contained in the program).

5.5 The Menu Bar



The majority of the commands in the menus operate on the active page. A page is made active either by clicking in the contents of the page or by clicking in the field containing the title of the page. This field is shown in reverse video when the page is active.

Many of the commands of the menus also operate on the current selection of the active page. The current selection is called current focus.

5.5.1 File Menu

New BETA program

Selecting this command will make a dialog popup. Entering a file name without extension and clicking **ok** will result in the creation of a minimal BETA program, which is given the entered file name. The immediate result of this will be a diagram showing up on the active work sheet.

The title of the diagram is the name of the new fragment group (the file name: `foo`) followed by a dash (-) and the name (`program`) of the new BETA fragment form:

```
ORIGIN '~beta/basiclib/v1.4/betaenv'
-- program: DescriptorForm --
(# do #)
```

Notice:

- Diagrams displaying the descriptor of a program, like the one mentioned above, are called descriptor diagrams.

New BETA library

Makes a dialog popup. Entering a file name without extension and clicking **ok** will result in the creation of a minimal BETA library, which is given the entered file name. The result of this will be a diagram, like the one shown above, showing up on the active work sheet. The title of the diagram is the name of the new BETA fragment group (the file name: `fee`) followed by a dash (-) and the name of the new BETA fragment form (`lib`). The single attribute shown below the title corresponds to the single class attribute of the library:

```
ORIGIN '~beta/basiclib/v1.4/betaenv'
-- lib: Attributes --
<<NameDecl>>: <<PrefixOpt>>
(# <<AttributeDeclOpt>>
   <<EnterPartOpt>>
   <<DoPartOpt>>
   <<ExitPartOpt>>
#)
```

Notice:

- Diagrams displaying the attributes of a library, like the one mentioned above, are called attributes diagrams.

A general note:

The common name for descriptor and attributes diagrams is *f*ragment diagrams.

New Graphics Page...

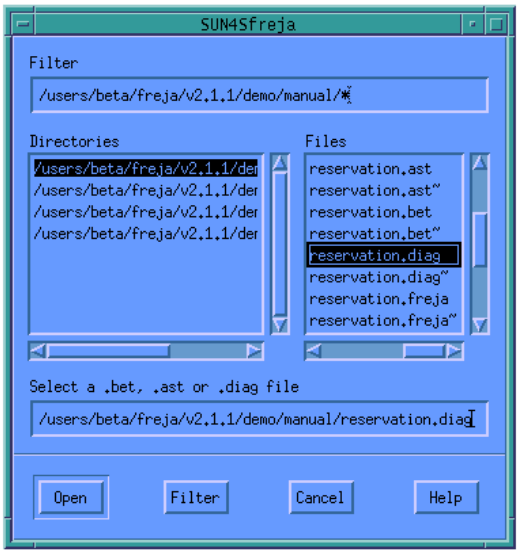
If other pages are already open invoking this command will create a new page entitled "**Graphics Page p. n**". If no other pages are open at the time where the

command is invoked two pages will be created - a graphics page and a group page.

Graphics pages are as the name implies ment for graphics only. That is all objects appearing on these pages are interpreted as having no other semantics than the purely graphical one. Basically the commands that apply on these pages are the purely graphical ones that can be found in the **Create**, **Makeup** and **Align** menues. Also **Cut**, **Copy**, **Paste** and **Clear** from the **Edit** menu applies to single objects, regions and groups of objects on these pages.

As a special case it should be mentioned that editing pattern and object diagrams as graphics only (i.e. editing the diagrams arbitrarily without this having any consequences on the underlying code) can be done simply by copying such a diagram from a work sheet or an object page and afterwards pasting it onto a graphics page (for an alternative way of editing these diagrams as pure graphics see **Load Graphics...**).

Open...



Selecting **Open...** makes the open dialog popup. From here a .bet, .ast file or .diag file can be selected and opened.

Selecting a .bet or .ast file will, depending on the category of the selected fragment, result in either a descriptor or attributes diagram on the active worksheet. This class diagram reflects the attributes of the opened fragment. One exception from this should be noted: When a file containing more than one fragmentform is opened the first thing to show up will be the group page with a diagram displaying the fragmentforms of the selected file (see section: 'The Group Page', below). Now, by doubleclicking one of these fragmentforms its attributes can be shown on the active worksheet as described above.

Selecting a .diag file will open pages containing class diagrams corresponding to the situation in Freja when a save was last performed (see **Save** and **Save As...** below). This also has the effect that all the BETA fragments related to the class diagrams shown on these pages are being opened. That is, if the class diagrams on the pages of the opened .diag file are edited and saved hereafter, the related fragments will be edited and saved correspondingly.

Notice:

- A BETA fragment is said to be *related* to a class diagram if the class diagram reflects attributes declared in this fragment.
- If the fragments related to a saved .diag file are being edited and saved outside the scope of Freja the .diag file becomes inconsistent with these and therefore the .diag file can no longer be opened. However the only data lost this way will be the layout of the diagrams on the pages, as new

diagrams, reflecting the changes, can be generated through opening the .bet or .ast files anew (here-through performing reverse engineering).

- Only one .diag file can be open at the time, but the different class diagrams on the work sheets can be related to several different BETA fragments.

Close

Closes all open pages in Freja and the fragments related to the class diagrams shown on these pages.

If any editing of the diagrams has been done since the last save the user is asked if he wishes to save these changes. Answering yes to this question results in a save to the currently open .diag file.

Save

Performs a save of all shown diagrams on all open pages to the currently open .diag file.

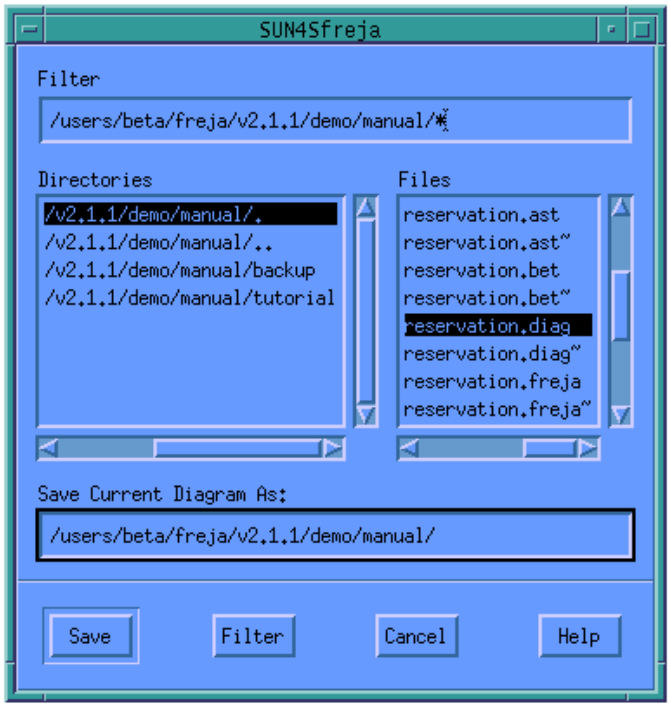
This means that the layout of the diagrams on the pages are saved, and that the fragments related to the diagrams are saved to their .bet and .ast files. The .diag file can then later be opened for further browsing and editing and eventually be saved again.

Notice:

- The default name for the diagram is the name of the BETA fragment that was opened or created first during the session. To change this name use **Save As** (see below).

Save As...

Using this command invokes the dialog:



Entering a name in the text field of the dialog with extension .diag and choosing Save will result in a save of all shown diagrams on all open pages to this file (as described above).

Revert

Reverts all edits done during the current Freja session. I.e. the diagram will appear as it did when it was initially opened.

Page Setup

Use **Page Setup** before printing. To get a satisfactory print the recommended settings are:

- **Paper**: **A4 letter**.
- **Output form: Scale to fit**.
- **Omit page borders**.

Print...

Initiating the print command will bring up a dialog asking the user to indicate which pages are to be printed. If all pages are to be printed simply select the **All** field of the dialog.

Having selected the **ok** button of the dialog the following string will appear in the xterm from which Freja was originally started: “Enter name of PostScript file (without .ps): “. Simply do so and both a PostScript and an Encapsulated PostScript version of the print will be available in the directory where Freja was started.

To this purpose the Freja script sets the environment variable "DesignPrintCommand" to "frejaprinter" if it does not already have a value. Alternatively the "DesignPrintCommand" can be used to specify that Freja is to print directly on a given printer. This can e.g be done like this:

```
>setenv DesignPrintCommand "lpr -Pr312"
```

Where `lpr -Pr312` is the command for printing on a printer called `r312`.

Preferences...

Brings up a dialog for manipulating general settings of the program.

Load Text...

Is used for loading text from a file on disc into a new box node in the diagram.

Save Text...

Saves the text contained in the node that is current focus onto a specified file on disc.

Load Graphics...

Makes a dialog popup from which you can choose a file on disc that contains pages that has been saved as graphics form within Freja at an earlier time. Theese pages can now be edited as graphics pages and saved again as pure graphics (see e.g. **New Graphics Page...**).

Also if in an earlier session pattern and/or object diagrams where created and saved togther with the code, these diagrams can at a later time be loaded into Freja as graphics only (i.e. without the code). This is done by inoking this menu item and selecting the .diag file (the graphics) that was saved earlier together with the .ast file (the code).

Notice:

- Having loaded a .diag file into Freja as graphics only it is generally not recommended to save this graphics under the *same* name again - i.e. as a .diag file. This will cause the .diag file to become inconsistent with the code saved in the .ast file. That is reverse engineering has to be performed before code and diagram can be edited simultanously again.

Save Page As Graphics...

Using this command you can, using the dialog that pops up, save the current page on disc as graphics only.

About

Will bring up a dialog displaying general information (version no. etc.) about the running Freja.

Help

Brings up a version of this document.

Show Sif / Hide Sif

This menu item initially contains the text **Show Sif** and by invoking it the Mjølner BETA Source Browser and Editor, Sif, will appear on the screen. In the case where a class diagram is currently being edited in Freja, Sif will appear with focus on the corresponding piece of code. The code can now be edited in Sif as well as through Freja and the changes will constantly be reflected in the other tool.

After invoking **Show Sif** the menu item will contain the text **Hide Sif** which can now be invoked to make Sif disappear from screen again.

Quit

Performs a **Close** (see above) and quits the application.

5.5.2 Edit Menu

Undo

Work sheets:

Choosing **Undo** from the **Edit Menu** will undo the latest editing operation on a work sheet in Freja. If this operation itself was **Undo** the state before the **Undo** will be restored.

The operations which can be undone are **Cut, Copy, Paste, Insert Before, Insert After, Paste Before, Paste After, Expand**.

Cut

In the general case **Cut** will remove current focus and put it on the graphical clipboard as a graphics only object. Also groups and regions of objects can be cut, but not groups that contain both graphics only objects and parts of class diagrams. See **Paste** for further explanation of when the graphics only clipboard is used and when code is also being pasted.

Work sheets:

Initiating **Cut** removes the contents of current focus on the active work sheet. The deleted part of the class diagram is moved onto a clipboard and can later be pasted into any other class diagram. Only titles and attributes can be cut from a class diagram (not any of the relations). Performing a **Cut** with a title of a class diagram as current focus results in the removal of the entire class diagram and the removal of the attribute of which this diagram is a detailed version. Notice that this type of cut will also cut the corresponding code.

Object Pages:

Only cutting of graphics only objects, regions and groups is currently implemented on object pages.

Copy

In the general case **Copy** will put current focus on the graphical clipboard as a graphics only object. Also groups and regions of objects can be cut. See **Paste** for further explanation of when the graphics only clipboard is used and when code is also being pasted.

Work sheets:

Copies the contents of current focus on the active work sheet onto the clipboard and can later be pasted into any other class diagram. Only titles and attributes can be copied from a class diagram (not any of the relations).

Paste Before

Work sheets:

When current focus is an attribute of a class diagram this command performs an **Insert Before** followed by a **Paste**.

Paste

Work sheets:

If current focus is part of a class diagram - i.e. a title or an attribute - and if the clipboard contains part of a class diagram, then **Paste** can only be performed when the contents of current focus and the class diagram part on the clipboard are of exchangeable syntactic categories. If this is the case and a **Paste** is performed the effect is that the contents of current focus is replaced by the class diagram part on the clipboard and the code is updated correspondingly.

In this context when cutting or copying with a title of a class diagram as current focus the syntactic category is the same as the syntactic category of the attribute of which the class diagram is a detailed version.

If the clipboard contains a graphics only object this object is simply pasted as such onto the work sheet.

Object pages:

On object pages only paste of pure graphics can be performed.

Graphics Pages:

If the clipboard contains a graphics only object this is pasted as such onto the graphics page.

If the clipboard contains part of a class diagram the purely graphical part of this will be pasted onto the graphics page.

Paste After

When current focus is an attribute of a class diagram this command performs an **Insert After** followed by a **Paste**.

Clear

Removes the contents of current focus on the active work sheet (nothing is moved to the clipboard).

Edit Name

If current focus is an attribute of a class diagram or the title of a diagram detailed from such an attribute, this command will invoke a dialog in which the name of the attribute can be entered. The entered name must be a lexem (i.e. some legal name in BETA). Therefore before accepting it, the name is parsed and if unlegal the dialog will ask the to change it.

Open Subeditor

If current focus is an attribute of a class diagram or the title of a diagram detailed from such an attribute, this command will invoke a socalled *subeditor* on the code that corresponds to current focus. The functionality of a subeditor is mainly identical to the functionality of the codeviewer/-editor of the browser window of Sif. Subeditors are therefore typically used when textual structure editing on some part of the code is called for.

Insert Before

If the contents of current focus is an attribute of a class diagram this command will insert a new unexpanded attribute before current focus.

Insert After

If the contents of current focus is an attribute of a class diagram this command will insert a new unexpanded attribute after current focus.

Remove Optionals

If current focus is part of a class diagram this command removes all nonterminals representing optional productions from the contents of current focus.

Show Optionals

Inserts all nonterminals representing optional productions in current focus, if it is part of a class diagram.

Check

Invokes the semantic checker on the code related to the class diagram which is current focus.

If any semantic errors are detected in the code it is reported in a “Semantic Error” dialog. Selecting an error in this dialog will make the attribute in Freja, referring to the code containing the error, become current focus.

If no errors are detected during checking all drawn relations, that have somehow become invalid using e.g text editing, are removed and the ones, that were not displayed before the check, are now drawn.

5.5.3 New... Menu

In general the **New..** menu contains entries that are composed of two or more ordinary commands and are provided to make some of the more frequently used series of commands easier to perform.

Class

This command basically creates a new class pattern and makes a corresponding class diagram show up on the active work sheet. The user is prompted for a class name.

The class is declared in what is called the *current fragment diagram*.

If the current focus is a title or an attribute of a class diagram P the current fragment diagram is the fragment diagram that is related to the same fragment as the diagram P.

Local Operation / Class

This command creates a new pattern. It inserts the new pattern attribute in the diagram where current focus is set. That is, the class diagram which includes the title or attribute which is current focus. This diagram is called the current diagram. The user is prompted for a name for the new pattern.

Static Reference Attribute

Inserts a new static item in the current diagram. Prompts the user for an attribute name.

Dynamic Reference Attribute

Inserts a new dynamic item in the current diagram. Prompts the user for an attribute name.

Virtual Operation/Class

Inserts a new virtual pattern definition. Prompts the user for a pattern name.

Binding of Virtual Operation/Class

Inserts a binding of a virtual pattern. Prompts the user for a pattern name.

Final Binding of Operation/Class

Inserts a final binding of a virtual pattern. Prompts the user for a pattern name.

5.5.4 Expand Menu

The content of the **Expand** menu is dependent on the current focus.

If current focus is anything else than an attribute of a class diagram it will be empty.

If current focus is an attribute of a class diagram the menu will contain an entry for each language construct that can replace a possible nonterminal in the attribute.

If the attribute that is current focus does not contain a nonterminal the **Expand** menu will be empty.

Having selected an attribute that does contain a nonterminal choosing one of the entries in the **expand** menu will result in the nonterminal being replaced with the chosen construct (see below). If the nonterminal in current focus is an optional or a list element the menu will also contain an **Empty** entry. Choosing this entry will remove the nonterminal.

In some cases the attribute that is current focus will apparently contain more than one nonterminal. For example:

```
<<NameDecl>>: <<ReferenceSpecification>>.
```

However in these cases the situation will always be that only one of the nonterminals can be replaced by a number of language constructs and all other present nonterminals will be so called *lexem nonterminals* that can not be further expanded. Therefore the Expand menu will at any time contain the language constructs which are possible replacements for the (single) replaceable nonterminal in the attribute. In the example above `<<NameDecl>>` is a lexem nonterminal and the replaceable nonterminal determining the contents of the menu is `<<ReferenceSpecification>>`. To replace a lexem nonterminal like `<<NameDecl>>` use the **Edit Name** command of the **Edit** menu. Other lexem nonterminals can e.g. be replaced using a subeditor (see **Open Subeditor**) on the attribute containing the lexem.

5.5.5 Relations Menu

The commands of the **Relations** menu all relate one node to another. For this reason they are all basically used in the same manner:

1. Choose the wanted relation,
2. possibly a dialog appears, then fill in the wanted settings and click **Ok**
3. click down the mouse on the source of the relation and
4. drag a connector to the destination of the relation.

Notice:

- The settings of some of the relations (aggregations and associations) can be altered by doubleclicking the relation. This will bring up the appropriate dialog with settings according to the doubleclicked relation.
- The source or destination of a relation can be altered after its creation by selecting the source/destination end of the relation connector and dragging it to the new endpoint.(on connectors in general: see **Connector** in the **Create Menu** section)

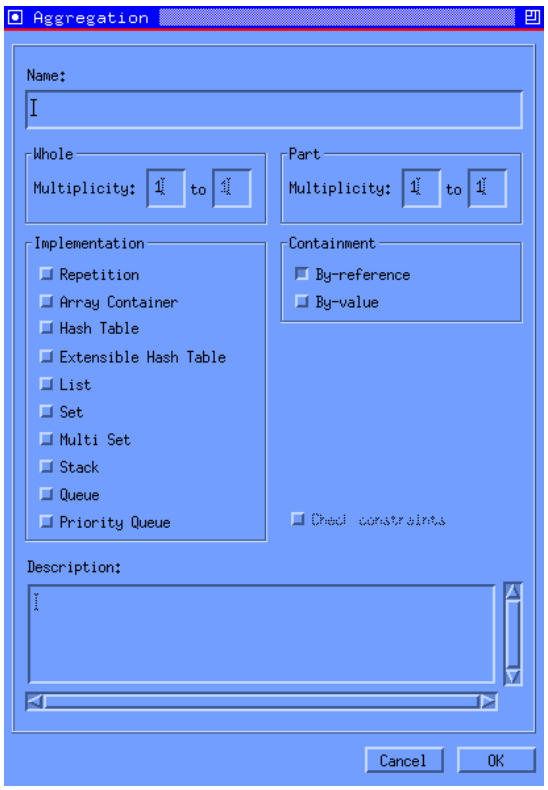
General Relationship

Any elements of a class diagram can be connected using through this relation. It has no formal semantics and no code results from creating it. It is intended as a

purely informal symbol, used in situations where the user has not yet decided which type of concrete relation (typically aggregation or association) should exist between two classes.

Aggregation

When executing this command a dialog appears:



The aggregation dialog makes it possible to:

- give the aggregation a name,
- specifify the multiplicities of the whole and the part,
- specify the implementation of a *one-to-many*[1] aggregation; that is to specify if a repetition or one of the basic container classes should be used in the resulting code,
- specify if the aggregation is to be implemented *by-reference* or *by-value*,
- give a textual description of the aggregation.

Note that default values are set, so that the user only has to specify the settings he actually wants to change.

When **Ok** has been clicked in the dialog an arrow cursor, ⇒ , indicates that the user now can specify the aggregation by dragging a connector from the whole to the part.

When dragging this connector the destination (the part) must either be a class declaration, a virtual class declaration, a binding of a virtual class declaration or a final binding of a virtual class declaration. The destination may either be detailed or not.

If the source (the whole) of this operation is the title of a class symbol the result will be that a new attribute containing the corresponding code is inserted in the

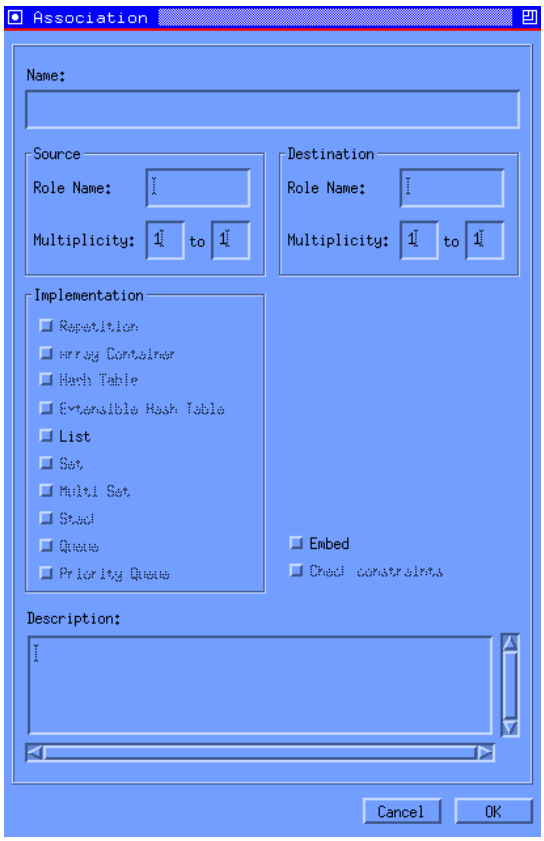
---

[1] An aggregation is said to be *one-to-many* if the multiplicity of the part is one and the multiplicity of the whole is more than one.

class. If the source is an attribute of a class and this attribute corresponds to a dynamic or static reference in the code the operation will result in this attribute being altered so that it now corresponds to the generated aggregation code.

Association

When executing this command a dialog appears:



The Association dialog makes it possible to:

- give the association a name,
- specify role names for both sides of the association,
- specify multiplicities for both sides of the association,
- specify the implemention of a *one-to-many* or *many-to-many* association; that is to specify if a repetion or one of the basic container classes should be used in the resulting association code,
- specify if the association should be *embeded* or not; that is to specify if the generated code is to result in a separate association class or if it is to be embedded in the source and destination classes,
- give a textual description of the association

Note that default values are set, so that the user only has to specify the settings he actually wants to change.

When **Ok** has been clicked in the dialog an arrow cursor, ⟶ , indicates that the user now can specify the association by dragging a connector from the source class to the destination class.

Creating this relation the source and destination must be detailed versions of either class declarations, virtual class declarations, binding of virtual class declarations or final binding of virtual class declarations.

Specialization

Creating this relation the source and destination must be detailed versions of either class declarations, virtual class declarations, binding of virtual class declarations or final binding of virtual class declarations.

The result of creating a **Specialization** relation will be that the descriptor corresponding to the source diagram gets the destination class inserted as its prefix.

Binding of Virtual

For this relation the source must either be an attribute containing an unexpanded attribute declaration or an attribute containing a pattern, virtual or binding declaration. The destination must be an attribute containing a binding or virtual declaration.

Having created a relation like this the source will have become a binding of the virtual of the destination.

Notice:

- A possible descriptor of the source will not be lost through this operation.

Final Binding of Virtual

Behaves exactly like the **Binding of Virtual** relation except that the source in this case becomes a final binding of the virtual of the destination.

Pattern Variable

For the **Variable Pattern** relation the source must be an attribute containing either an unexpanded attribute declaration or some kind of simple declaration. The destination must either be the title of a diagram that is a detailed version of an attribute containing a pattern declaration or it must be an attribute containing a pattern declaration.

The result will be that the source becomes a declaration of a pattern variable with the destination pattern as qualification.

5.5.6 View Menu

Abstract

Work sheets:

If current focus is a title of a class diagram **Abstract** will remove this class diagram and all diagrams detailed from it.

If current focus is an attribute that has been detailed the command will remove the detailed version of the attribute and all diagrams detailed from that.

Object Pages:

Abstracting an object (going from the detailed to the abstracted form of an object) is done by selecting a detailed object and choosing **Abstract** or by doubleclicking the detailed object (double-click inside the detailed object without hitting any of the contained objects).

Abstract Recursively

Work sheets:

If current focus is a title of a class diagram **Abstract Recursively** will remove all diagrams detailed from it (but *not* the diagram to which the title belongs).

If current focus is an attribute that has been detailed the command will remove the detailed version of the attribute and all diagrams detailed from that.

Object Pages:

Abstract does not apply to detailed objects that themselves contain detailed objects. To abstract such objects you will have to choose **Abstract Recursively** instead of **Abstract** (doubleclicking the object to abstract recursively is not possible).

Overview

Work sheets:

If current focus is a title or an attribute of a class diagram this command will remove all class diagrams not on the "detail path" to the diagram which is current focus.

Object Pages:

Having detailed a lot of objects in the object diagram you might want to focus on one specifically interesting object and so to speak “undo” all the other detailing that has been done. This is the purpose of the **Overview** command. To use it you select an object (of any kind, abstracted or detailed) and choose **Overview**. As a consequence of this all detailed objects in the surrounding of the selected object will be abstracted (except of course the objects containing the selected object itself).

Detail

Work sheets:

If current focus is an attribute of a class diagram and this attribute contains an object descriptor, like e.g. a pattern declaration does, **Detail** will display the attribute on detailed form; that is, as a class diagram displaying the attributes of this descriptor.

If current focus is a title of a class diagram **Detail** will perform a **Detail** on all attributes that can be detailed in the class diagram.

Notice:

- Doubleclicking a detailable attribute of a class diagram will perform a **Detail** on the attribute. If the attribute is already detailed doubleclicking it will make the title of the detailed version the new current focus.

Object Pages:

To display the contents of the first *abstracted object* select it and then choose **Detail** from the menu or you simply double-click the abstracted object. This will result in having the abstracted object displayed on *detailed* form; that is the objects that are directly contained in the detailed object (in the BETA program) will be displayed inside the box of the detailed object. These objects can now be detailed exactly the same way as the first one etc. - getting a diagram that illustrates the static and dynamic object structure and the creation and call relations between the objects of the initially chosen pattern.

Notice:

- When an object has been detailed the program automatically does a reprettyprint of the surrounding objects. The intention of this reprettyprint is to preserve the ordering of the objects on screen. In normal cases this works as intentioned, however, some abstracted objects may grow very large when being detailed with the effect that some of the surrounding objects are moved far apart. A help in preventing this to happen can be being aware of the fact that detailed objects only expand to the right and downwards (never to the left or upwards), so only objects residing in these locations may become subject to relocation

Detail Recursively

Work sheets:

If current focus is an attribute of a class diagram and this attribute contains an object descriptor, like e.g. a pattern declaration does, **Detail Recursively** will display the attribute on detailed form; that is, as a class diagram displaying the attributes of this descriptor. Then it will perform this command recursively on the detailable attributes of the diagram that has now been detailed.

If current focus is a title of a class diagram **Detail Recursively** will perform a **Detail Recvrsively** on all attributes that can be detailed in the class diagram.

Object Pages:

If you select an abstracted object and choose the entry **Detail Recursively** the operation **Detail** will recursively be performed on the contained objects of the abstracted object.

Detail on Different Page

Work sheets:

Not yet implemented.

Object Pages:

It is also possible to show the details of an object on another page than the one containing the abstracted form of the object. To do this select the abstracted object you wish to detail and choose **Detail on Different Page**. As a result of this the detailed form of the chosen abstracted object will be displayed on a new page. The abstracted object will still exist in its original surroundings but now displayed with a bold linestyle to indicate that the corresponding detailed object can be found on another page. In fact, doubleclicking an abstracted object like this will activate the page containing the detailed form of it.

Search

Work sheets:

By means of the **Search** command it is possible to search for a substring of a lexem, i.e. a name definition, a name application or a string.

Notice that this command does not search text auxillary to the class diagrams, e.g text in user created labels.

Replace...

Work sheets:

Extends the search dialog with a replace text field so that the found text can be replaced with a text specified in this text field.

Layout Sub-patterns

Work sheets:

If current focus is the title of a class diagram and if there are currently displayed sub-patterns of this pattern, **Layout Sub-patterns** will rearrange the positions of these sub-class diagrams in a manner that makes them appear "nicely" in a tree structure below the selected diagram.

Edged Specialization Connectors

Work sheets:

If current focus is the title of a class diagram and if there are currently displayed sub-patterns of this pattern, **Edged Specialization Connectors** will make specialization connectors only follow horizontal and vertical lines.

Text Attributes

Makes a dialog popup through which the text attributes of current focus can be changed.

Fit To Text

If current focus is a graphical object auxillary to the class diagrams, the command will make the object fit to the text contained in the object. This command does not work for attributes and titles of class diagrams.

Layout Preferences...

Makes a dialog popup where preferences regarding the layout of the class diagrams can be set.

The following explains in further detail the consequences of enabling or disabling the different checkboxes of the dialog.

Show**:**

- Attributes

  If enabled the attributes of the class diagrams are visible. If disabled the attributes of the class diagrams are invisible (only the titles of the diagrams will be visible).
- Aggregations

  If enabled the aggregations visible. If disabled the aggregations are invisible.
- Specializations

  If enabled the specilization connectors are visible. If disabled the specialization connectors are invisible.
- Associations

  If enabled the association connectors (and diamonds) are visible. If disabled the association connectors (and diamonds) are invisible.

Show attributes with:

The four following entries concerns the appearance of the information inside the atributes of the class diagrams - that is, the prettyprint of the attributes. Only one of the following can be checked in at the time.

- Name And Type

  In this mode both the name and the type of the declaration inside an attribute is shown, as for example in:

  ```
  anInstance: @aPattern
  ```

- Name

  This is the default prettyprint mode for attributes. This prettyprint mode only displays the name (the `NameDecl` part) of the declaration, making the above example appear as:

  ```
  anInstance
  ```

- Type

  This prettyprint mode only displays the type of the declaration, making the above example appear as:

  ```
  : @aPattern
  ```

- Name And Kind

  The **Name and Kind** prettyprint mode displays the name and the kind (e.g. static item symbolised by a "@") of the declaration and makes the example appear as:

  ```
  anInstance: @
  ```

5.5.7 Create Menu

The **Create** menu commands draw non-formal graphic objects (nodes and connectors).

Connector

Draws connectors between two nodes.

The cursor, a light arrow ⇾, indicates that the system is in connector creation mode.

Drawing Single-Segment Connectors

Click inside one node (the source node) and drag the connector tool inside another node (the destination node). Freja always draws connectors from border to border, along a line from the center of the source node to the center of the destination node.

Delete a connector by double-clicking outside a node during a drag. Connector length and placement are determined by the nodes they connect.

Drawing Multi-Segment Connectors

To create a segmented connector, follow the above process, with one difference: release the mouse at the location of the first bend point, outside the destination node. Subsequent clicks outside the destination node create other bend points. Click in the destination node to complete the connector.

Pressing the mouse on a bend point and pressing the ALT or META key snaps the two segments of the bend point into a right angle.

Delete a bend point by placing the pointer tool on the bend point and pressing the mouse button along with SPACEBAR.

Add a bend point by placing the pointer tool on a midpoint selection handle and dragging it to a location.

Effect of Moving Source/Destination Nodes

Freja automatically redraws connectors when their source and destination nodes are moved or resized.

Endpoint Handle Use

Reattach a connector by selecting the endpoint handle and dragging it to another object. After the connector has been created, use the endpoint regions to change the position of the source or destination endpoint.

Terminating Connector Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Box

Creates nodes with a box shape

The cursor, a box  , indicates that the system is in box creation mode.

Changing Size and Position

Mouse clicks create default size boxes. Holding down the mouse and dragging the cursor also creates a box, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the box.

Creating Multiple Boxes

While in box creation mode, each mouse click creates a new box.

Terminating Box Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Rounded Box

Creates nodes with a rounded box shape.

The cursor, a rounded box  , indicates that the system is in rounded box creation mode.

Changing Size and Position

Mouse clicks create default size rounded boxes. Holding down the mouse and dragging the cursor also creates a rounded box, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the rounded box.

Creating Multiple Rounded Boxes

While in rounded box creation mode, each mouse click creates a new rounded box.

Terminating Rounded Box Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Ellipse

Creates nodes with an ellipse shape.

The cursor, an ellipse  , indicates that the system is in ellipse creation mode.

Changing Size and Position

Mouse clicks create default size ellipses. Holding down the mouse and dragging the cursor also creates an ellipse, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the ellipse .

Creating Multiple Ellipses

While in ellipse creation mode, each mouse click creates a new ellipse.

Terminating Ellipse Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Polygon

Creates an node, shaped as a polygon.

The cursor, a polygon  , indicates that the system is in polygon creation mode.

Changing Size and Position

Mouse clicks create default size polygons. Holding down the mouse and dragging the cursor also creates a polygon, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the polygon.

Creating Multiple Polygons

While in polygon creation mode, each mouse click creates a new polygon.

Drawing Polygons

The command sets the first vertex or bend point. Subsequent vertices are created by clicking the mouse. To complete the polygon, double-click on the last vertex. A polygon must have at least three vertices or it will be removed when double-clicked.

Pressing the mouse on a bend point and pressing the ALT or META key snaps the two segments of the bend point into a right angle.

Delete a bend point by placing the pointer tool on the bend point and pressing the mouse button along with SPACEBAR.

Add a bend point by placing the pointer tool on a midpoint selection handle and dragging it to a location.

Terminating Polygon Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Regular Polygon

Creates nodes with a regular polygon shape.

The cursor, a regular polygon  , indicates that the system is in regular polygon creation mode.

Changing Size and Position

Mouse clicks create default size regular polygons. Holding down the mouse and dragging the cursor also creates a regular polygon, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the regular polygon.

Creating Multiple Rounded Boxes

While in regular polygon creation mode, each mouse click creates a new regular polygon.

Terminating Regular Polygon Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Wedge

Creates nodes with a wedge shape.

The cursor,  , indicates that the system is in wedge creation mode.

Changing Size and Position

Mouse clicks create default size wedges. Holding down the mouse and dragging the cursor also creates a wedge, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the wedge.

Creating Multiple Wedges

While in wedge creation mode, each mouse click creates a new wedge.

Terminating Wedge Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Line

Creates multi-point lines classified as nodes.

The cursor, a cross +, indicates that the system is in line creation mode.

Terminating Line Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Label

Creates nodes shaped as labels.

The cursor, a label  , indicates that the system is in label creation mode.

Label creates a special borderless node to contain text. Text is always turned on when the label tool is active. A click of the label tool places an insertion point on the page.

Dimensions

A label's dimensions are determined by the text typed into it. Freja does not word-wrap text typed in a label. Press the RETURN key to break lines.

Restrictions

A label's dimensions may not be changed by dragging or by any other graphic operation. For example, a label that is a group member does not change with the group, and a label that is a region does not change with its parent.

Labels cannot have fill added nor can the layering logic be altered.

Moving Labels

Move a label around the diagram with the graphic tool. The label's boundary is indicated by selection handles.

Terminating Label Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Make Region

Turns nodes into regions or selects a new parent for regions. The command subordinates the current node to another object on the page.

Creating a Region

1. The status bar prompts: Select a Node, Region, or Connector to contain the new Region.
2. Place the pointer within the region's parent.
3. When the object's borders flash, click the mouse and the action is complete.
4. Selection handles reappear on the boundary of the new region and the status bar displays: Type: Region.

Effect on Parents

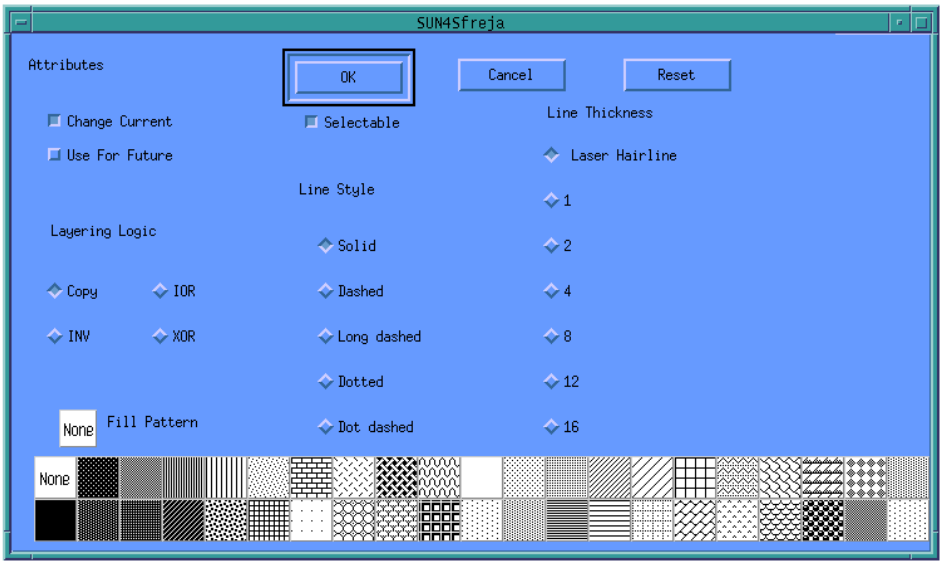
Some operations performed on a parent are also be performed on its regions (with the exception of a label), including moving and resizing.

Operations performed on a region do not affect its parent.

Unmake Region

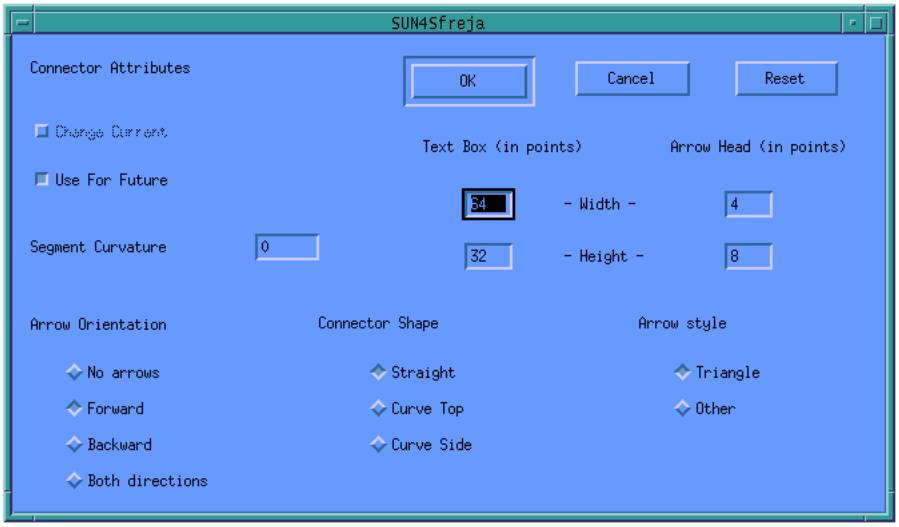
Turns regions into auxiliary nodes.

Set Attributtes...



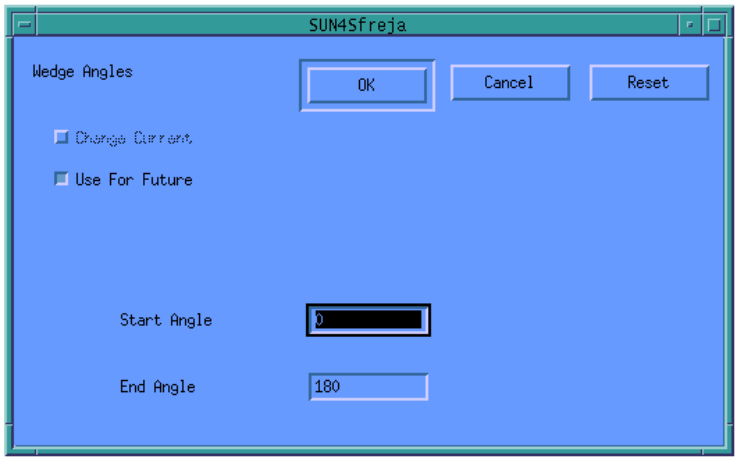
This command brings up a dialog through which general attributes of the graphical objects can be set

Set Conn Attributes,,,



This command brings up a dialog through which attributes of connectors can be set

Set Wedge Attributes...



This command brings up a dialog through which attributes of wedges can be set

Set Polygon Attributes...

This command brings up a dialog through which attributes of Polygons can be set

5.5.8 Makeup Menu

The **Makeup** menu commands manipulate and alter graphic objects.

Select

Selects objects by temporarily hiding other objects.

When objects are layered, closely spaced, or contained within other objects, they may be difficult to select with the graphic tool.

Selecting Visible Objects

Moving the pointer tool across an object causes its borders flash. Clicking on a flashing object makes it become the current object.

Selecting Hidden Objects

While the obscuring object is flashing, press SPACEBAR to make it invisible. Continue to hide objects until the desired object is visible. Select the object by clicking the mouse. All the hidden objects are restored.

Clicking anywhere in the active window restores all the temporarily hidden objects.

Adjust

Resizes the current node without moving its center.

Resizing in Two Dimensions

The hand pointer appears at the lower right corner of the current object. Drag the hand to resize the object in either dimension. Click the mouse to complete the operation.

Resizing in One Dimension

To resize one dimension while retaining the object's other dimension and its center, hold down the CTRL key.

Drag horizontally, with the vertical dimension locked.

Drag vertically and the horizontal dimension is locked.

Click the mouse to complete the operation.

Resizing Regions and Labels

When a label or a key region is selected, the command has no effect unless the object is part of a group. In such a case, the command does not affect the object itself but only its descendants, which get a new size and position.

When a selected group of regions contains ancestors/descendants of each other, some regions may change more than once.

Change Shape

Changes the size and shape of objects.

Change Shape changes the shape and size of the current node or group to match that of any other node on the page.

Shape Changing

The status bar prompts the selection of a node or region as a model for the shape and size change.

Placing the pointer within the selected object causes its borders flash.

Clicking on the flashing object completes the change.

Region Changing

Note that when a selected group of regions contains ancestors/descendants of each other, some regions may change more than once.

Drag

Repositions objects.

The drag tool  indicates that the system is in drag mode.

Use this tool to move the current object or group without placing the pointer inside a selected object. This is especially useful for moving small objects and objects inside other objects.

Dragging Text

If text is turned on, text can be moved as a block within its object. To begin a drag text, the drag tool must be inside the graphic object containing the text.

Terminating Drag Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Displace

Repositions objects by moving the current node or group the same distance (horizontally and vertically) as the last drag or alignment of an object or group.

Note: The command cannot be applied to groups of regions.

Move To...

Moves the current node or group to another page in the diagram.

Moving Nodes

The hierarchy page window appears and the status bar prompts: Click on page to receive object.

Move Node returns to the current page after the target page has been selected.

Effect on Regions and Connectors

Regions and connectors are moved, along with the corresponding nodes, in the usual way:

• A region is moved if the corresponding node is moved.

• A connector is moved if both the source and destination nodes are moved. If only one of these nodes is moved, the connector is deleted.

Hide Substructure

Hides descendants of selected objects , making them non-visible and non-selectable.

Nodes and Connectors

Hides all the descendants of the selected objects.

Regions

Hides the selected regions, together with all their descendants.

Show Substructure

Shows descendants of selected objects, making the descendants visible and selectable.

Next Object

Selects the object one layer above the current object.

This process is repeated with each selection of **Next Object** until the top layer of the stack of objects is reached. **Previous Object** (**Makeup** menu) reverses the process.

Previous Object

Selects the previous object (or text pointer).

Previous Object selects the object one layer below the current object. This process is repeated with each selection of **Previous Object** until the bottom layer of the stack is reached. **Next Object** (**Makeup** menu) reverses the process.

Bring Forward

Changes the object order of objects on a page.

Reordering Object Layering

1. Select the object to bring forward.
2. Choose **Bring Forward**.
3. Place the pointer on the object to put one layer below the selected object. The object chosen to underlie the selected object flashes.
4. Click to bring the selected object forward .

To travel through the layering order sequentially, use **Next Object** (**Makeup** menu) and **Previous Object** (**Makeup** menu).

Restrictions

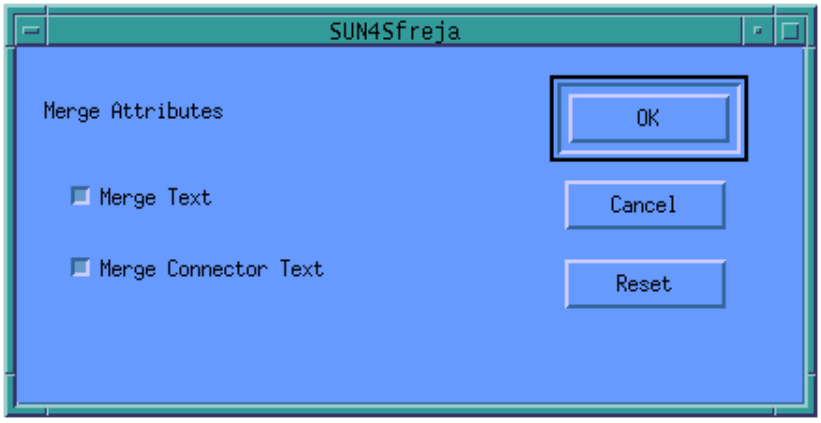
The command is not yet implemented for groups of regions.

Merge

Merges nodes into a single node by combining a node or group into a single target node.

Specify options for the merge with **Set Merge Attributes**.

Set Merge Attributes



Options for **Merge** can be set using this dialog.

Edit Polygon

Makes the vertices of the current polygon selectable.

The vertices of the polygon can then be edited in exactly the same way as bend points of connectors.

5.5.9 Align Menu

The **Align** menu commands reposition the currently selected node or group by aligning it to one or two reference objects.

Aligning is a two-step process: first select the command and then the reference object(s). A click of the mouse on the reference object accomplishes the alignment.

Horizontal

Performs a horizontal alignment.

Horizontal aligns the center of the current object along a horizontal axis drawn through the center of the reference object.

Vertical

Performs a vertical alignment.

Vertical aligns the center of the current object along a vertical axis drawn through the center of the reference object.

Left to Left

Performs a left-to-left alignment.

Left to Right

Performs a left-to-right alignment.

Right to Left

Performs a right-to-left alignment.

Right to Right

Performs a right-to-right alignment.

Top to Top

Performs a top-to-top alignment.

Top to Bottom

Performs a top-to-bottom alignment.

Bottom to Top

Performs a bottom-to-top alignment.

Bottom to Bottom
Performs a bottom-to-bottom alignment.

Center
Performs a center alignment.

Center places the center of the current object over the center of the reference object.

Between
Places the center of the current object at the midpoint of an imaginary line drawn between the centers of the two reference objects.

Projection
Places the current object at one end of an imaginary line, whose first endpoint is the center of the first reference object, and whose midpoint is the center of the second reference point.

5.5.10 Page Menu

Refresh
Refreshes the current page.

Refresh redraws the diagram on the current page, restoring the on-screen resolution of objects and text distorted by operations such as **Blowup** and **Reduce**.

Reduce
Reduces the page scale for the selected set of pages.

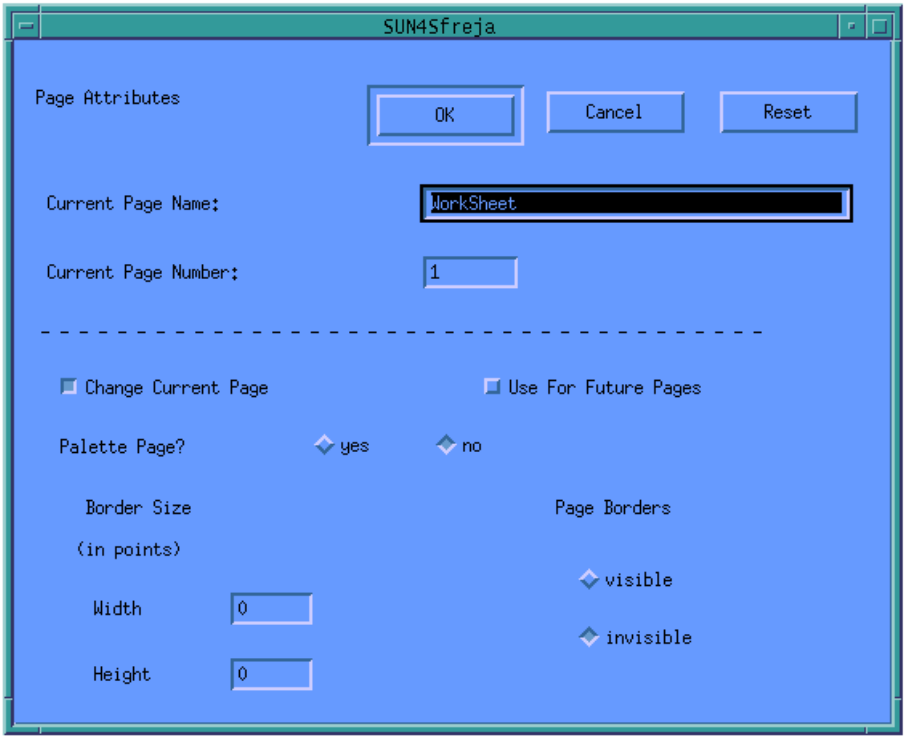
Each time **Reduce** is used, the active page is reduced to one half its present size to the minimum of 25 percent of normal. **Reduce** reverses the effect of **Blowup**, and like **Blowup**, may affect screen resolution.

Blowup
Enlarges the page scale for a selected set of pages.

Each time **Blowup** is used, it enlarges the active page to twice its present size up to the maximum of 400 percent of normal. The page scale is indicated in the status bar.

Some degrading of the screen image may occur. The actual diagram and the printed diagram are not degraded by **Blowup** or **Reduce**. Use the **Refresh** (**Page** menu) command to restore on-screen resolution after using **Blowup**.

Attributes...



The attributes of the current page can be set using this dialog.

Group Window

Makes the page entitled "**Group Window**" the current page.

If current object on the **Work Sheet** is a title or an attribute of a class diagram, the current object that is selected in the **Group Window** is the fragment that this title or attribute belongs to.

Work sheet

Makes the page entitled "**Work Sheet**" the current page.

Hide page

Hides the current page (i.e. makes it invisible).

Show all pages

If a number of pages has been hidden this command makes them all reappear on screen.

5.5.11 The Object Menu

Show Object Diagram

This menu entry is used when initially generating an object diagram. For further details see the Getting Started section.

The Search Modes

Having chosen **Show Object Diagram**, as explained above, the text "**Scanning AST...**" will appear in the status field of the menu bar (the field that initially contains the text "**Type: node**"). This means that an analysis is being performed on the target program to detect dynamic object generations and operation calls, so that these can be displayed in the resulting object diagram. Those BETA-fragments that are in some way used by the target program (included files, body files etc.) also have to be analysed to give a complete picture of the dynamics of

the program. Some programs use a lot of different fragments that themselves use a number of fragments etc. Analysing all these fragments can sometimes be a time-consuming task. Furthermore it may not be necessary to do this to get an object diagram that illustrates the essentials of the program. For this reason you may choose one of three different search modes before choosing **Show Object Diagram**. The three search modes are:

- Local Search: The default search mode. Analysis is only carried out on fragments located in the same directory as the target program or in subdirectories hereof.
- Global Search: All fragments used by the target program are analysed.
- Global Search with Confirmation: Like **Global Search** except that for every included fragment you are asked to confirm if the fragment should be analysed.

Notice:

- Excluding a fragment from analysis also means excluding all fragments used by this fragment.

During analysis the fragment that is currently analysed is printed out in the xterm from which the program was started.

Filters

Some programs may have a very complex object creation and/or operation call structure. The illustration of this in the object diagram may result in a very large number of arrows. This again may hinder the user in getting a clear picture of the dynamic structures of the program. To help out in these situations two filters are provided.

Toggle Dynamic Creations

To remove all dynamic creation arrows from an object diagram you choose **Toggle Dynamic Creations**. To have them displayed again at sometime afterwards you choose **Toggle Dynamic Creations** once more. Whether or not a check mark is present before this entry in the menu visualises whether or not dynamic creations are currently shown in the object diagram. Default is that dynamic creations are displayed in the diagram.

Toggle procedure calls

Same as for **Toggle Dynamic Creations**.

Notice:

- The filters on dynamic creations and operation calls work on page level. That is, if you choose **Detail on Different Page** for some object then both dynamic creations and operation calls will as a default be shown in the diagram on the new page . Afterwards checking out one or both of these will only have an effect on the page currently in focus.

Showing and Hiding Simple Objects

Instances of the basic patterns `Integer`, `Char` and `Boolean` are in the context of object diagrams considered to be *simple objects*. Simple objects are as a default not displayed in the object diagram. It is however possible to have them displayed as follows: Select the detailed object for which you wish to see the simple objects that are part of it. Then choose **Show Simple Objects**. The result of this is that all simple objects that are part of the chosen detailed object are displayed at the bottom of this. The simple objects of a detailed object can in a similar fashion be hidden again by selecting the detailed object and choosing **Hide Simple Objects**.

Compact and blowup

The objects in the diagram can freely be rearranged (moved) by the user except that objects that are part of another object cannot be moved outside the frame of the surrounding object. While rearranging the objects in some detailed object one might wish for more space to do so; that is to enlarge the rectangle that makes up the detailed object. To do this you select the detailed object and choose Blowup. The effect is that the detailed object is enlarged by 10% and the surrounding objects are reprettyprinted accordingly (of course this operation may be repeated until a satisfactory size is obtained).

If you on the contrary wish to have an object displayed on an as compact form as possible you select a detailed object and choose **Compact**. Thereby the rectangle of the chosen detailed object and those detailed objects containing it will be made as small as possible.

Notice:

- You should not resize objects using the standard “mouse dragging” method as the effect of this will be that all contained objects will also be resized and no reprettyprint will be performed.
- If you move a detailed object all objects that are part of it

are also moved. This effect can sometimes be a help in detecting which objects are part of a detailed object.

6. Bibliography

[Jensen 91] K. Jensen, S. Christensen, P. Huber, M. Holla: Design/OA: A Reference Manual, Meta Software Corporation, 125 Cambridge Park Drive, MA 02140, USA, 1991.

[Knudsen 94] J. L. Knudsen, M. Löfgren, O. L. Madsen, B. Magnusson (eds.): *Object-Oriented Environments – The Mjølner Approach*, Prentice Hall, 1994, ISBN 0-13-009291-6.

[Madsen 93] O. L. Madsen, B. Møller-Pedersen, K. Nygaard: *Object-Oriented Programming in the BETA Programming Language*, Addison-Wesley, 1993, ISBN 0-201-62430-3

[MIA 90-1] Mjølner Informatics: *The Mjølner BETA System: – Overview,* Mjølner Informatics Report MIA 90-1.

[MIA 90-2] Mjølner Informatics: *The Mjølner BETA System: BETA Compiler Reference Manual* Mjølner Informatics Report MIA 90-2.

[MIA 90-11] Mjølner Informatics: *Sif – A Hyper Structure Editor, Tutorial and Reference Manual* Mjølner Informatics Report MIA 90-11.

[MIA 92-12] Mjølner Informatics: *The Mjølner BETA System – The BETA Source-level Debugger – Users's Guide*, Mjølner Informatics Report MIA 92-12

[UML1.0] Rational Software Corporation: *Version 1.0 of the Unified Modeling language*, Rational Software Corporation, 2800 San Thomas Expressway, Santa Clara, CA 95051-0951. The most recent updates on the Unified Modeling Language are avialable via the worldwide web: http://www.rational.com.

I. Index