

# **HITGRAPHIC**

## **User's Guide**

Document Version 3.6.00

### **for the**

## **Hierarchical Evaluation Tool**

### **HIT**

Version 3.6.000

### **and**

## **HITGRAPHIC**

Version 3.6.00

April 1996



# HITGRAPHIC User's Guide

Michael Sczittnick (Editor)

Martin Büttner

Elke Heck

Rainer Lange

Michael Strüwer

Monika Verwohlt

Christel Wysocki

Copyright © 1991, 1993-1996: Universität Dortmund, Informatik IV.  
ALL RIGHTS RESERVED.

## **Abstract:**

HITGRAPHIC provides a graphical user interface for the performance evaluation tool, HIT. In addition to the hierarchical model specification language HI-SLANG it allows for the graphical specification of HIT models by composing separate graphical presentations of model elements to a graphical representation of a complete model. The same holds for experiments. Subsequently, the graphical representation of the model is transformed automatically into its HI-SLANG representation.

HITGRAPHIC has been developed at the chair of Prof. Dr.-Ing. H. Beilner, Informatik IV, Universität Dortmund partly in cooperation with Nixdorf Computer AG and with the support of the BMFT (German Federal Ministry of Research and Technology).

This document is released for internal and external use. Corrections, comments, criticism and suggestions for improvement of this document are welcome.

## **Address:**

Universität Dortmund

Informatik IV

Prof. Dr.-Ing. H. Beilner

D-44221 Dortmund

Telefon: +49 231 755 2704

Telefax: +49 231 755 4730

E-Mail: [hit@ls4.informatik.uni-dortmund.de](mailto:hit@ls4.informatik.uni-dortmund.de)

Telex: 822465 unido d



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	What is HIT and What is HITGRAPHIC? . . . . .	1
1.2	Documents on HIT . . . . .	2
1.3	Conventions Used in this Manual . . . . .	2
1.4	Structure of the Manual . . . . .	3
<b>I</b>	<b>Reference Guide</b>	<b>4</b>
<b>2</b>	<b>First Steps</b>	<b>7</b>
<b>3</b>	<b>Elements of Usage</b>	<b>11</b>
3.1	HITGRAPHIC User Interface Conventions . . . . .	11
3.1.1	Mouse Handling . . . . .	11
3.1.2	Where to Find Menus? . . . . .	11
3.1.3	Dialog Boxes . . . . .	12
3.2	Window Types in HITGRAPHIC . . . . .	13
3.2.1	Environment Window . . . . .	14
3.2.2	Survey Window . . . . .	14
3.2.3	Component Type Graphic Window . . . . .	15
3.2.4	Control Procedure Window . . . . .	16
3.2.5	Editor Window . . . . .	17
3.2.6	Aggregation Description Window . . . . .	18
3.2.7	Evaluation Window . . . . .	18
3.2.8	Evaluation Object Window . . . . .	19
3.2.9	Start/Stop Window . . . . .	20
3.2.10	Hierarchy Survey Window . . . . .	20
3.2.11	Hierarchy Window . . . . .	21
3.2.12	Experiment Window . . . . .	22
<b>4</b>	<b>Functional Description</b>	<b>23</b>
4.1	The Environment Window . . . . .	24
4.1.1	Global Functions . . . . .	25
4.1.2	Functions on Kinds of Modelling Objects . . . . .	28
4.1.3	Functions on Modelling Objects . . . . .	31
4.2	The Survey Window . . . . .	46
4.2.1	Global Functions . . . . .	49
4.2.2	Popup Menu in the Survey Window . . . . .	49

4.2.3	Function on Line in the Type Structure . . . . .	50
4.3	The Component Type Graphic Window . . . . .	50
4.3.1	Global Functions on Model Types . . . . .	52
4.3.2	Global Functions on Component Types . . . . .	54
4.3.3	Popup Menus in the Graphic Region . . . . .	56
4.4	The Control Procedure Window . . . . .	70
4.4.1	Global Functions . . . . .	70
4.4.2	Functions on Control Procedures for Component Types . . . . .	71
4.4.3	Functions on Control Procedures for Components . . . . .	72
4.5	The Editor Windows . . . . .	76
4.5.1	HI-SLANG in Editor Windows . . . . .	78
4.5.2	Structure of the Editor Window . . . . .	83
4.5.3	Global Functions on the Head Label Area . . . . .	85
4.5.4	Global Functions on the Editor's Pane . . . . .	86
4.5.5	Text Selection . . . . .	86
4.5.6	Global Functions on the Editor Label Area . . . . .	87
4.5.7	Functions on the Editing Area . . . . .	90
4.6	The Aggregation Description Window . . . . .	92
4.6.1	Global Functions . . . . .	93
4.6.2	Specification of Population . . . . .	94
4.7	The Evaluation Window . . . . .	95
4.7.1	Global Functions . . . . .	96
4.7.2	Specification of the Evaluation . . . . .	97
4.8	The Evaluation Object Window . . . . .	102
4.8.1	Global Functions . . . . .	103
4.8.2	Specification of the Evaluation Object . . . . .	104
4.9	The Start/Stop Window . . . . .	108
4.9.1	Global Functions . . . . .	109
4.9.2	Specification of Start respectively Stop Conditions . . . . .	110
4.9.3	Specification of Values . . . . .	111
4.10	The Hierarchy Survey Window . . . . .	114
4.10.1	Global Functions . . . . .	114
4.10.2	Functions on Hierarchies . . . . .	115
4.11	The Hierarchy Window . . . . .	117
4.11.1	Global Functions . . . . .	119
4.11.2	Functions Referring to Load Path Selection . . . . .	119
4.11.3	Functions Referring to the Selection of Views . . . . .	120
4.11.4	Functions Referring to the Current Load Path . . . . .	120
4.12	The Experiment Window . . . . .	122
4.12.1	Global Functions . . . . .	123
4.12.2	Operations in the Method Area . . . . .	125
4.12.3	Functions on Evaluations . . . . .	126
4.12.4	Output Area . . . . .	127
<b>5</b>	<b>Teamwork with HITGRAPHIC</b>	<b>129</b>
5.1	Modes of Modelling Objects . . . . .	129
5.2	General Hints for Teamwork Support . . . . .	130
5.3	Documentation Support . . . . .	131

<b>6</b>	<b>Questions and Answers</b>	<b>133</b>
6.1	List of Questions and Problems . . . . .	133
6.2	Solutions . . . . .	134
<b>II</b>	<b>Tutorial</b>	<b>142</b>
<b>7</b>	<b>A Tutorial</b>	<b>145</b>
7.1	Introducing Remarks and Perspective . . . . .	145
7.2	How to Start a HITGRAPHIC Session . . . . .	145
7.3	Model Specification . . . . .	146
7.3.1	Model 1 . . . . .	146
7.3.2	Model 2 . . . . .	155
7.3.3	Model 2a – Aggregation . . . . .	160
7.3.4	Model 3 . . . . .	166
7.3.5	Model 4 . . . . .	170
7.4	Evaluation and Experiment Specification . . . . .	174
7.4.1	Evaluation 1 . . . . .	174
7.4.2	Experiment 1 . . . . .	180
7.4.3	Evaluation 2/2a . . . . .	183
7.4.4	Experiment 2/2a . . . . .	183
7.4.5	Evaluation 4 . . . . .	184
7.4.6	Experiment 4 . . . . .	200
7.5	Epilogue . . . . .	201
	<b>Index</b>	<b>203</b>





# Chapter 1

## Introduction

### 1.1 What is HIT and What is HITGRAPHIC?

This document introduces the usage of HITGRAPHIC, the high level user interface for the performance evaluation tool, HIT. In the following we sketch very briefly some features of the underlying HIT System, for details see the “HI-SLANG Reference Manual” or “HIT and HI-SLANG: An Introduction” (cf. Section 1.2).

HIT is a performance evaluation tool which allows the structured specification and the quantitative evaluation of computing system models. Since system design and development are nowadays usually based on a layered model with functional abstraction, HIT employs hierarchical modelling techniques allowing the separate specification and analysis of model components.

HIT provides for

- the specification of (models of) dynamic, discrete-event, stochastic systems by a particular model description language, HI-SLANG, for the description of model structures;
- the (performance) analysis of correspondingly specified models by a variety of techniques of the simulative, analytical-algebraic and analytical-numerical types.

Although originally developed with the objective of evaluating computing system performance, HIT also lends itself to the analysis of “similar” systems such as communication and office systems, transport and logistic systems and other systems of the specified dynamic, discrete-event, stochastic type. The HIT model world is tailored upon the prevailing view of computing system structures which partitions a system

- vertically, into a sequence of layers and levels, as communicating via function calls, and jointly representing a hierarchy of virtual machines;
- horizontally, into independent, mutually well-protected, information-hiding modules each one realizing some subset of functions to be provided at a particular level.

The corresponding HI-SLANG specification maintains as far as possible the conventional, high-level-language (HLL) approach, assumed to be well-known to and convenient for the envisaged user community of the tool. With the availability of the graphical interface, HITGRAPHIC, an important step towards wider applications of performance evaluation has been made.

Modelling and evaluation should be done by the designer, consultant, salesperson or engineer himself without the need to be an expert in simulation, statistics, queueing theory, numerical analysis and related techniques for quantitative system evaluation. Therefore, it was an important requirement for the HIT system to provide for a high level user interface.

As a consequence, in addition to the HIT system with its hierarchical model description language HI-SLANG, the interface HITGRAPHIC for the graphical specification of HIT models is offered. In HITGRAPHIC the construction of HIT models is done by composing graphical representations of model elements to a graphical representation of a complete model. The same holds for experiments. Subsequently the experiment can be transformed into its HI-SLANG representation, followed by an automatical start of the HIT system. Thus HITGRAPHIC liberates the user from writing potentially lengthy model specifications in the HI-SLANG modelling language.

Notice: When working with HITGRAPHIC it is required that the language version of HIT is already known by the user. So you should read the HIT Introduction before continuing with HITGRAPHIC, and use the HI-SLANG Reference Manual as a source of detailed information in case of practical problems beyond the graphical scope (cf. Section 1.2).

## 1.2 Documents on HIT

- Weißenberg, N. (ed.); HIT and HI-SLANG: An Introduction, Version 1.1.00; Universität Dortmund, Informatik IV, 1992
- Büttner, M. (ed.); HI-SLANG Reference Manual, Version 3.4.00; Universität Dortmund, Informatik IV, 1994

## 1.3 Conventions Used in this Manual

This document uses the following conventions:

- Notions, which appear for the first time, or essential terms are printed in **bold-face types**.
- Printing in *italics* is used to emphasize a topic or for menu items.
- “Double quotes” are used for items occurring in examples.
- The terms for the HI-SLANG Syntax refer to appendix A of the HI-SLANG Reference Manual.

## 1.4 Structure of the Manual

This manual is divided in two parts, the reference guide and a tutorial.

The novice user should start reading Chapter 1. Chapter 2 contains some technical prerequisites to be fulfilled before the work with HITGRAPHIC can start. Chapter 3 gives a brief overview on HITGRAPHIC. Then it would be best to switch to the tutorial (Chapter 7), which contains examples of the work with HITGRAPHIC. It is intended that the user can perform the described steps on the computer in parallel to reading the tutorial.

The main part of the reference guide is built up by Chapter 4 and Chapter 5. They provide detailed information about each function of HITGRAPHIC. Chapter 6 gives answers to specific questions, possibly arising during a HITGRAPHIC session.

**Part I**  
**Reference Guide**





# Chapter 2

## First Steps

For a user of HITGRAPHIC, there are some preparation steps necessary before the normal operation can start. To understand the instructions some basic knowledge of the UNIX system is required. Also the knowledge of an editor would be helpful. If you have problems understanding this chapter, ask your system administrator. He should be able to help you. It is recommended that you read this chapter completely and perform the described actions in a second run.

Users of previous versions of HITGRAPHIC should also read this chapter, because it contains some new aspects and hints for the database conversion.

First of all, you need to know the **installation directory** of HITGRAPHIC, for example `/usr/local/Hitgraphic.sun4`. In the following we will refer to this name using the abbreviation `"${hitgraphic}"`. The directory `"${hitgraphic}"` should contain the subdirectories `"bin"`, `"dbase"` and `"lib"`. The first interest is in the `"bin"` directory since it contains the programs `"hitgraphic"`, `"unset_busy"` and `"db_admin_prg"`, which can be called by the user. Therefore, the directory `"${hitgraphic}/bin"` should be added at the end of the search path for executable programs. This usually requires a change in your startup files (`".login"`, `".cshrc"`, `".profile"` or something else, depending on your system configuration).

The user of HITGRAPHIC is responsible for his (user) databases. He must create a **directory for his databases**, for example by performing `"mkdir .DB_DIR"` in the home directory. The HITGRAPHIC system needs to know the name of this directory. Therefore, an **environment variable** named **"DB\_DIR"** must be set to this name. In our example, you may add a line `"setenv DB_DIR $HOME/.DB_DIR"` to a startup file if using the C-shell. By setting the environment variable `"DB_DIR"` to the directory for databases of another user, you may get access to those databases when starting HITGRAPHIC.

Next, we will create a new database. For this, we change to the directory for databases, for example by `"cd $DB_DIR"` assuming that `"DB_DIR"` is already defined. New databases will be **created by copying existing databases**. Therefore, an **empty database** exists in the HITGRAPHIC installation directory. The copy process can be initiated by `"cp -r ${hitgraphic}/dbase/init.db <my_own>.db"`. The name of the database, `"my_own.db"` in our example, may not be longer than 79 characters.

Now, as you have created your first database, it is time to discuss **access rights** to the database. They can be controlled by giving the directory and all its containing subdirectories and files the desired UNIX access rights by using “chgrp” and “chmod”. You may also set the s-bit for groups for the subdirectory “long\_field” in the database directory. New files will only be created in this directory, and the s-bit will control the group attribute of new files to be identical to the one of the directory. The access rights of new files depend on the actual umask of the process that started HITGRAPHIC.

During the design of your database structure you should keep in mind that it is quite easy to copy modelling objects between environments within one database, but an automated copy from database to database is not supported.

After having started the X Window System it is now time to **call HITGRAPHIC** on the command line by “hitgraphic <database>” (in our example “hitgraphic my\_own.db”). The environment and the startup window appear and you can begin a HITGRAPHIC session by creating a new environment. You can also start HITGRAPHIC directly with an environment by using its name as second parameter (“hitgraphic <database> <environment>”).

In the following you will see an example session in a C-shell creating the directory for databases and a database with exclusive read and write access for the owner and the members of a group named “hitgr”. In this session the change of startup files is avoided, but this is not recommended for frequent use of HITGRAPHIC.

Please note: This is an example session. It probably will not work on your machine because groups or paths do not exist or you are using another kind of shell.

```
1% setenv PATH ${PATH}:/usr/local/Hitgraphic.sun4/bin
2% cd
3% mkdir .DB_DIR
4% setenv DB_DIR $HOME/.DB_DIR
5% cd $DB_DIR
6% cp -r /usr/local/Hitgraphic.sun4/dbase/init.db my_own.db
7% chmod 755 .
8% chgrp -R hitgr my_own.db
9% chmod 770 my_own.db
10% chmod 660 my_own.db/*
11% chmod 770 my_own.db/long_field
12% chmod g+s my_own.db/long_field
13% cd
14% csh -c "umask 007; hitgraphic my_own.db" &
```

If you have been using HITGRAPHIC Version 2.0.00 or later, user databases will automatically be converted with the first call of the new HITGRAPHIC version. Older databases are no longer supported. The database conversion is irreversible. If you need a high degree of safety it is recommended to copy the database (e.g., on a backup device) before starting the new version of HITGRAPHIC with it. Database conversions may last some minutes depending on the size of the database. Please, do not interrupt them.



After the conversion of databases prior to version 3.2.00 it may be necessary to add a `%END` at the end of a control part of an experiment because this is not part of the automatic conversion.

A `%END` must terminate the control part. Earlier versions of HITGRAPHIC have been inconsistent concerning this aspect.

From version 3.3.00 onwards it is possible to change the font used within the HITGRAPHIC editor from the default `fixed` by the resource `Hitgraphic*font`.

Usually this is all you need to know about databases and the call of HITGRAPHIC. But problems may occur in case of machine crashes or abnormal program (e.g., X server) terminations. The most common problem is that modelling objects cannot be opened anymore because they are “busy” although no one has opened them. First, make sure that nobody uses the database. Then, call “unset\_busy” with the database (“unset\_busy <database>”).

Some more operations can be performed by the program “db\_admin\_prg”. It is called in the same way (“db\_admin\_prg <database>”). Here again nobody should actually use the database. A menu appears on your terminal.

- “Check\_db” checks the internal hash keys of the database.
- “Repair\_db” partially checks the integrity of the database and resets all “busy” locks.
- “Simple\_file\_check” searches for unreferenced files.

In some cases you are asked to confirm that corrections should be performed. The program produces report files with suffix “\_trace” in the caller’s home directory.

If HITGRAPHIC internally detects an error, it may create a file called “DB\_trace” in the user’s home directory. Its content may help the HITGRAPHIC development team to localize the problem. Please do not hesitate to inform or ask the HITGRAPHIC development team.

If you have a problem using HITGRAPHIC you should consult Chapter 6, where you may find a solution.



# Chapter 3

## Elements of Usage

### 3.1 HITGRAPHIC User Interface Conventions

HITGRAPHIC is the graphical user interface of HIT. It is based on the X Window System. For all windows of HITGRAPHIC there is a common style in the user interface. These conventions are described here.

#### 3.1.1 Mouse Handling

If the *right* mouse button is held pressed down, popup menus show up (provided that the cursor has been located in a sensitive area). Choose a function from the menu and release the button.

The *left* mouse button is used for the selection of so-called buttons in dialog boxes. The boxes must be closed in some way before other actions are enabled. Additionally the left mouse button can be used for other functions, e.g., for selection or to mark the new position of an object during a move operation. Please, note that HITGRAPHIC uses X buffers.

Within the HITGRAPHIC editor the buttons are used in the conventional way (cf. Section 4.5).

#### 3.1.2 Where to Find Menus?

In general popup menus are offered on

- title bars;
- objects, represented by labels and graphical symbols (boxes, circles, ...);
- the background of a window.

In Chapter 4 all popup menus are listed in detail for each window.

### 3.1.3 Dialog Boxes

Many of the functions available from the popup menus need an additional interaction with the user. Dialog boxes can appear near the last cursor position or in the upper left corner of the corresponding (sub)window. Note that boxes are different from windows. Boxes cannot be iconified.

Boxes serve for the following purposes:

- Displaying error messages, e.g., in case a name already in use has been chosen.
- Requiring input from the user, e.g., rename.
- Requiring confirmation of critical functions, e.g., delete.

No further actions can be executed or proceed until you have confirmed the message by clicking *OK* with the left mouse button or entering Carriage Return while the cursor is inside the box. Selections with the left mouse button are possible, if they are requested in the box.

#### Message Boxes

Message boxes mainly serve for error information. There are three types of errors:

- Errors due to unfeasible user actions.
- Errors due to the underlying graphic system (X Window System), showing up with the message “X11-error: ...”. In this case you should contact your system administrator.
- Internal errors, showing up with the message “Error detected”. In this case it is advisable to check the information in the actual window and to start the program “db\_admin\_prg” (cf. Chapter 2) after leaving HITGRAPHIC.

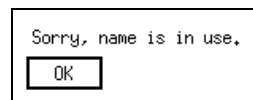


Figure 3.1: Message Box

#### Input Boxes

Input boxes require a text string. They are automatically placed at the cursor position so that the string can be typed on the keyboard without any mouse movements. Names can consist of letters, digits and underlines. Syntactical errors or HI-SLANG keywords are recognized immediately when selecting the *OK* button or pressing Carriage Return. Please, note that name checks are performed case insensitive. Input boxes may initially contain default strings.

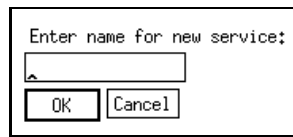


Figure 3.2: Input Box

### Confirm Boxes

To protect the user from an unintentional execution of “dangerous” functions, like delete, a confirmation is requested. It can be given by clicking the *OK* button, otherwise the *Cancel* button must be selected. Carriage Return can be used as an accelerator for selecting *OK* when the cursor is inside the box.

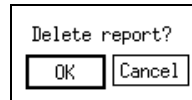


Figure 3.3: Confirm Box

### Other Boxes

There are some other special boxes, which will be introduced in later sections.

## 3.2 Window Types in HITGRAPHIC

The following types of windows are distinguished and shortly described in this section:

- environment window
- survey window
- component type graphic window
- control procedure window
- editor window
- aggregation description window
- evaluation window
- evaluation object window
- start/stop window
- hierarchy survey window
- hierarchy window
- experiment window

### 3.2.1 Environment Window

The environment window serves as lead-in to the modelling activities. It is the working environment for the modeller. The current set of modelling objects (model types, component types, aggregation descriptions, evaluations, experiments) is displayed, and the environment window provides the user with some basic functionality, i.e., operations to create, delete, copy, open ... modelling objects.

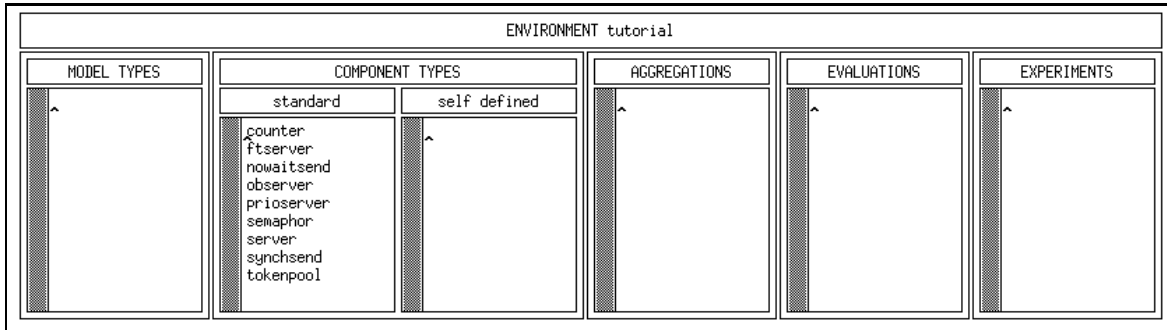


Figure 3.4: Environment Window

### 3.2.2 Survey Window

The survey window displays the structure of the model/component type under study with respect to its hierarchical (sub)structure. It serves as a link between the environment window and component type graphic windows by providing the user with clear structural information about the relations between model/component types or between corresponding instantiations, respectively.

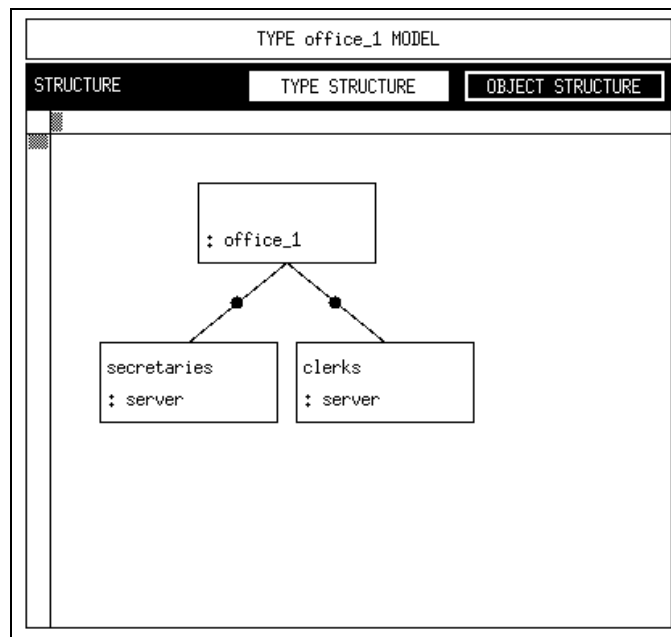


Figure 3.5: Survey Window

### 3.2.3 Component Type Graphic Window

The task of constructing a model/component type is performed in the component type graphic window. It provides the user with means for modelling services, used services, instantiating objects of given component types, referring used to provided services and the like. This work is simplified by comprehensive graphical support.

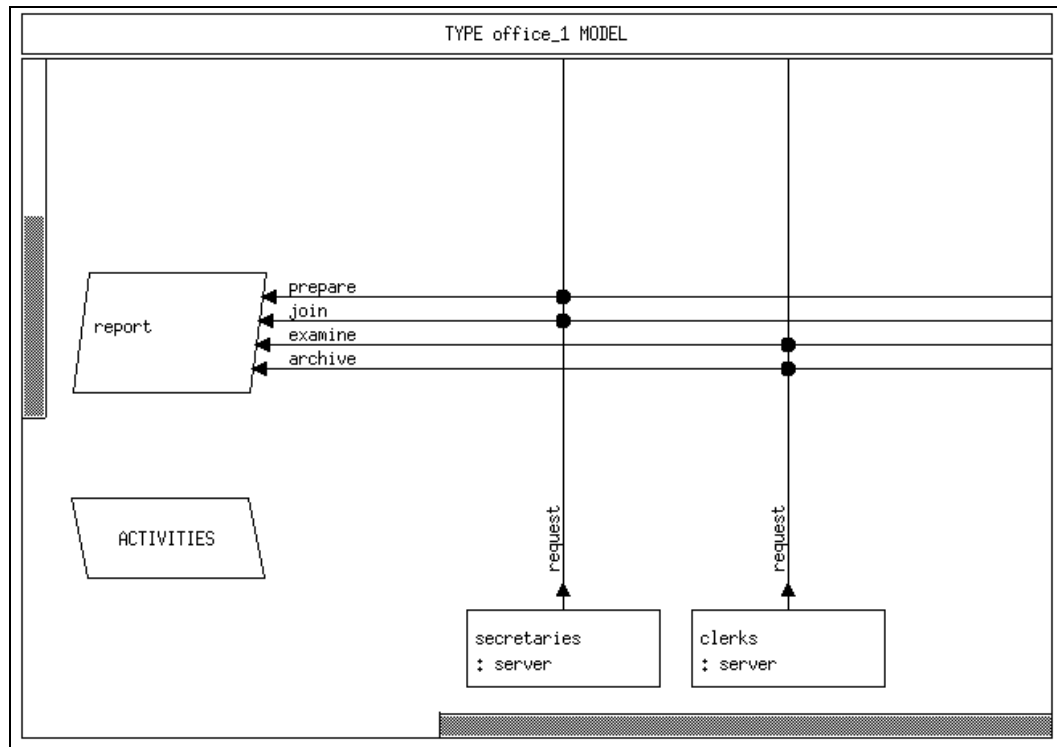


Figure 3.6: Component Type Graphic Window

### 3.2.4 Control Procedure Window

The goal of control procedure windows is (of course) to specify control procedures of component types respectively their instantiations. Modelling control procedures is performed by selecting predefined standard control procedures or, with a wider range of application, describing own procedures.

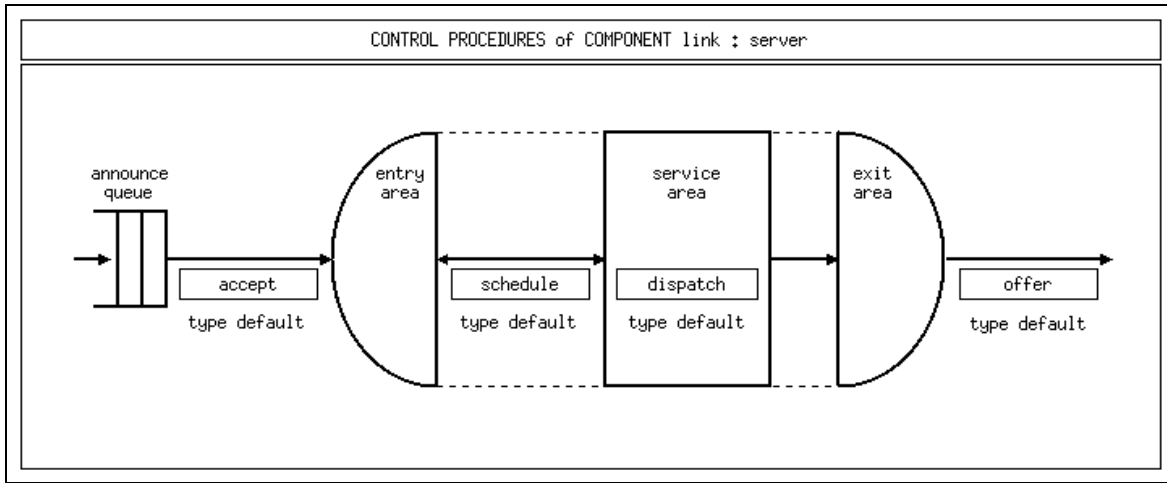


Figure 3.7: Control Procedure Window



### 3.2.5 Editor Window

Although HITGRAPHIC supports the user by providing a graphical interface, some editing remains to be done, in particular the behaviour pattern of services must be entered textually.

For editing purposes HITGRAPHIC provides a text editor. Depending on the given context it is presented as a simple editor or as a partitioned editor consisting of several subwindows.

Standard operations (search, search&replace, ...) are supported as well as a cut and paste mechanism which serves as support for window-to-window copies of text blocks. Additionally check functions are provided for a local check of the entered parts of code.



Figure 3.8: Editor Window

### 3.2.6 Aggregation Description Window

An aggregation description is always dependent on the underlying component type. To perform an aggregation run of HIT it is just necessary to specify the maximal population of each provided service. The aggregation description window provides the user with a comprehensive style of specifying these populations.

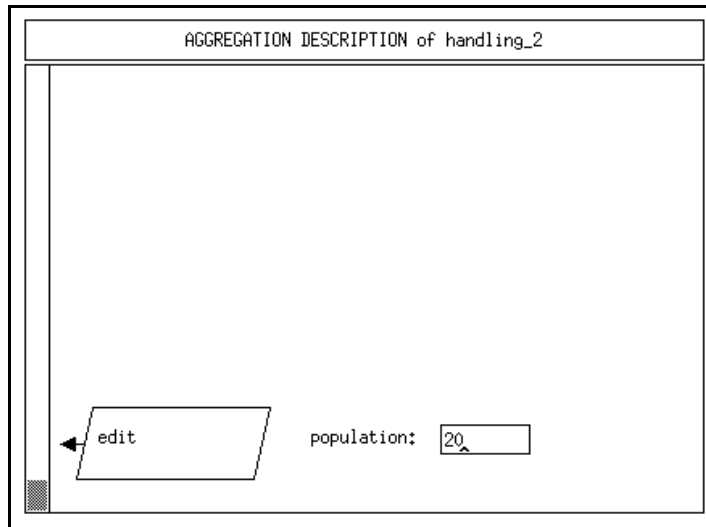


Figure 3.9: Aggregation Description Window

### 3.2.7 Evaluation Window

The evaluation window serves as a basis for the specification of various kinds of measurements as well as related fields. Starting with the well-known object structure of the survey window the evaluation window allows for the basic operations on evaluation objects, that represent a complete measurement specification. Especially the specification, **where** a measurement will take place, is done in this window.

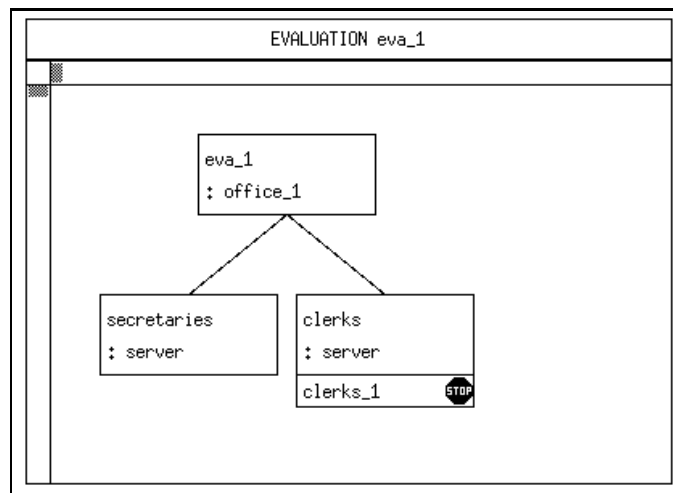


Figure 3.10: Evaluation Window

### 3.2.8 Evaluation Object Window

Whereas the evaluation window provides the user with basic operations on evaluation objects, evaluation object windows are concerned with details of measurements, i.e., they describe **what** should be measured and **how** it should be done. They allow for the specification of estimators, streams, hierarchies and even more detailed information.

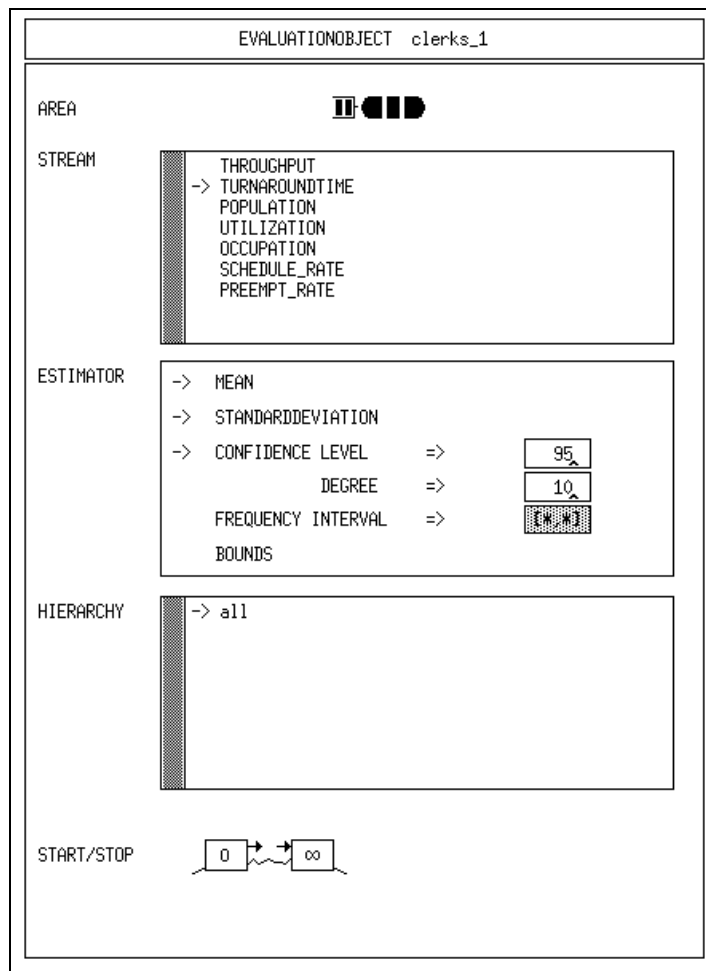


Figure 3.11: Evaluation Object Window

### 3.2.9 Start/Stop Window

At various steps during the specification of evaluations the necessity of start and stop controls arises. Start/stop windows are used to specify boolean expressions, which mirror the desired control.

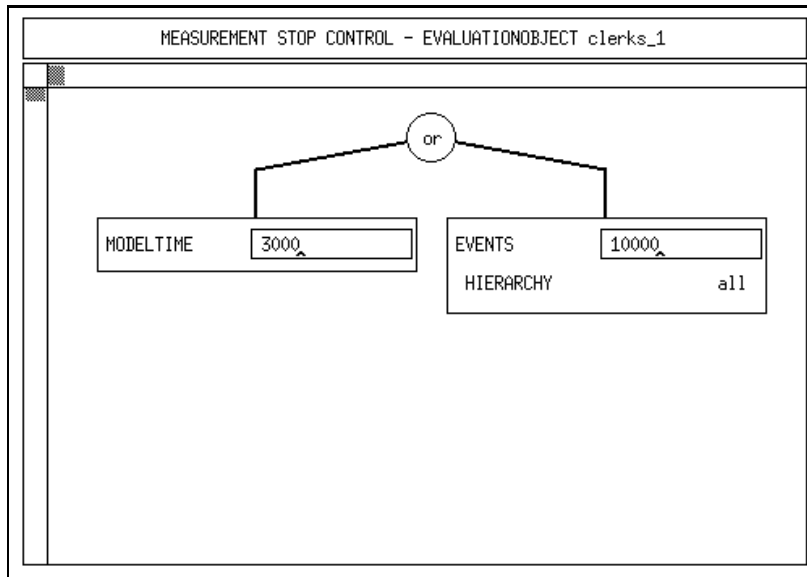


Figure 3.12: Start/Stop Window

### 3.2.10 Hierarchy Survey Window

The need to distinguish load by the point of generation and the path down the model hierarchy, which finally leads to the location of the measurement, is accomplished by the concept of hierarchies. Hierarchy survey windows provide the user with basic operations on hierarchies.

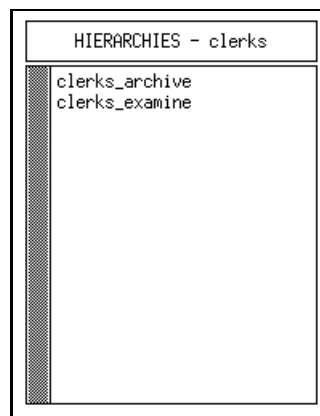


Figure 3.13: Hierarchy Survey Window

### 3.2.11 Hierarchy Window

Again, graphical elements of the object structure are partially used as a comprehensive, well-known and general presentation of structural relations within the model type. Load paths are specified by selecting a path to the location of interest. All paths within a hierarchy window are implicitly merged and form a load filtering hierarchy.

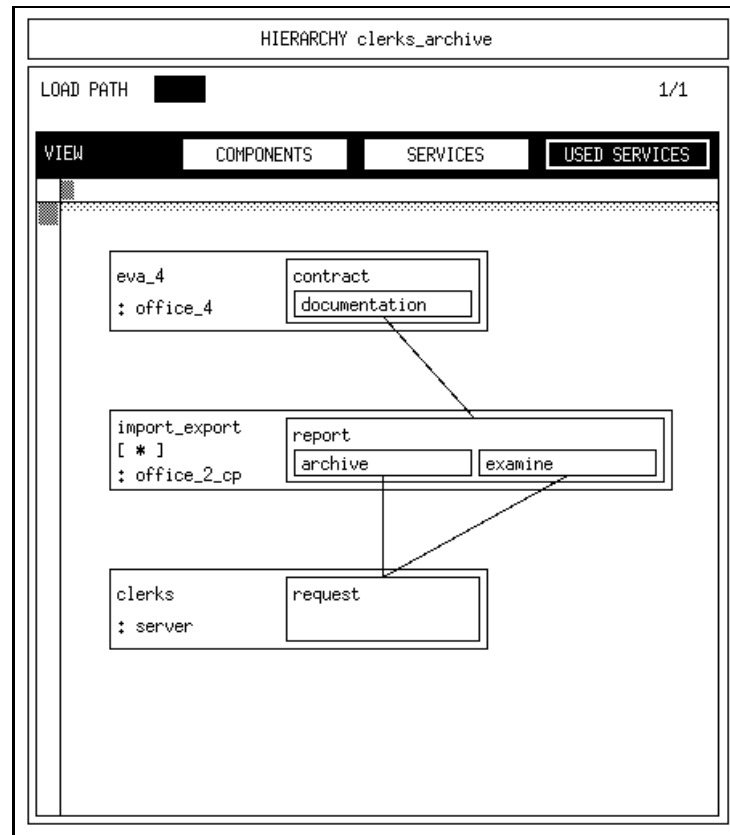


Figure 3.14: Hierarchy Window

### 3.2.12 Experiment Window

Experiment windows serve as a bridge from all recent specifications to complete HIT runs. They allow for designing experiments, which combine selected evaluations with solution techniques (since the evaluation descriptions are independent of the solver used) as well as with some general information constraining the experiment execution.

EXPERIMENT exp_4	
METHOD	<input type="checkbox"/> SIMULATIVE    CPUTIME => <input type="text" value="10000"/> MODELTIME => <input type="text" value="10000"/> TRACE ALL => on
	ANALYTICAL - algebraical: <input checked="" type="checkbox"/> DQ4 <input type="checkbox"/> LIN2    ACCURACY => <input type="text" value="1"/> - numerical: <input type="checkbox"/> MARKOV    ACCURACY => <input type="text" value="0.1"/> CPUTIME => <input type="text" value="5000"/>
EVALUATIONS	<input type="text" value="eva_4"/>
OUTPUT	DUMPFIL SYSOUT -> TABLE

Figure 3.15: Experiment Window

# Chapter 4

## Functional Description

Before the HITGRAPHIC windows are discussed in detail, some general aspects concerning the usage of HITGRAPHIC should be mentioned.

HITGRAPHIC is built on top of databases (a standard database and user databases). For this reason a user needs access to a user database.

A user database is structured into **environments**. Each environment consists of a collection of user defined **model types**, **component types**, **aggregation descriptions**, **evaluations** and **experiments** (henceforth called modelling objects) and the **standard component types** as defined by HIT (which are automatically added from the standard database). The creation and usage of environments is recommended as a structuring principle of HITGRAPHIC databases to distinguish different user communities or project activities and their subsets of modelling objects, respectively.

This chapter contains a detailed description of all functions in HITGRAPHIC. It is subdivided into sections (4.1 - 4.12) according to window types the user interface of the HITGRAPHIC system is composed of. The functions can be selected in popup menus which are displayed while pressing the right mouse button. The entries of the popup menus vary according to the object or area the cursor is pointing at. An entry is selected by positioning the cursor and releasing the right mouse button. Some operations have to be confirmed within a message box by the user, e.g., delete or rename. This is done either by clicking *OK* with the left mouse button or entering Carriage Return. Screendumps displayed in this chapter will help to attach objects and areas to their menus.

Some conventions concerning the entries in popup menus are specific to this chapter:

- Normal entries have the form “• *entry*”, “- *entry*” or “\* *entry*”.
- Pullright menus are indicated by a right-pointing small arrow. Entries, that provide pullright menus, have the form “• *entry* ->”.
- Switches between two entries are indicated by a slash: “• *entry1*/*entry2*”. In a popup menu that contains a switch, either *entry1* or *entry2* can be selected.

## 4.1 The Environment Window

The environment window forms the basis for modelling activities supported by HITGRAPHIC. It is present during a complete HITGRAPHIC session and displays the available model/component types, aggregation descriptions, evaluations and experiments. Additionally to the user defined component types it provides all standard component types available in HIT.

After starting a HITGRAPHIC session (cf. Chapter 2 or Section 7.2) the startup window and an empty environment window are displayed.

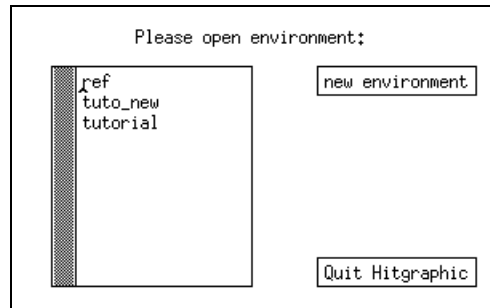


Figure 4.1: Startup Window

In a subwindow within the startup window all available environments of the specified database are listed. The popup menu on each list item provides the user with basic functions concerning the environment.

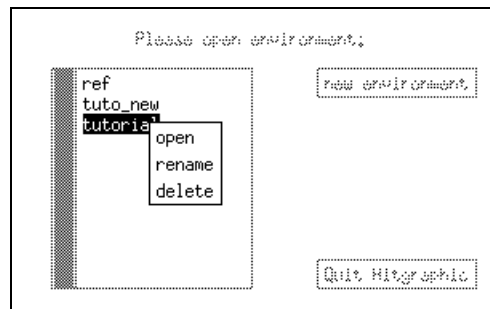


Figure 4.2: Menu on Environments

- *open*  
opens the selected environment.
- *rename*  
renames the selected environment.  
A dialog box is displayed requesting the new name. After entering the name and confirming it the name of the environment is updated.
- *delete*  
deletes the complete environment.  
This operation should be performed carefully because the objects are really



deleted within the database. Therefore a double beep can be heard and a confirmation of this operation is requested.

During the *delete* operation, which may take some time, the confirm box remains visible.

A new environment can be created by clicking with the left mouse button on the *new environment* button. A dialog box is displayed requesting the name of the new environment. After entering the new name and confirming it the environment window displays the new environment initially only consisting of the standard component types.

The *Quit Hitgraphic* button can be used to close the HITGRAPHIC session.

The environment window is composed of a title bar and five (respectively six) subwindows. The subwindows themselves contain a title bar and a list of subwindow-specific modelling objects (cf. Figure 4.3). The (sub)title bar as well as the list entries allow for some basic functions (new modelling object, copy, rename, ...) on modelling objects.

Besides these standard operations some features are provided. The most important one, the operation *transform & run*, generates the HI-SLANG source code of the selected experiment and starts the HIT system automatically.

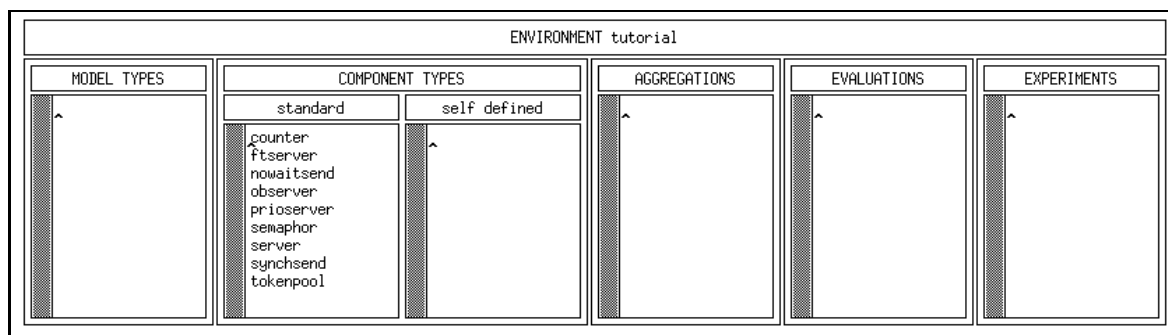


Figure 4.3: Environment Window

### 4.1.1 Global Functions

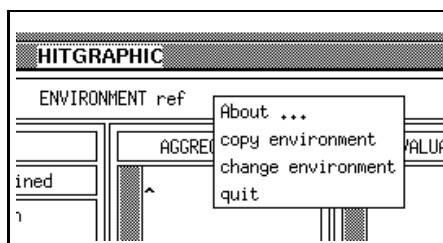


Figure 4.4: Menu on the Title of the Environment Window

- *About ...*

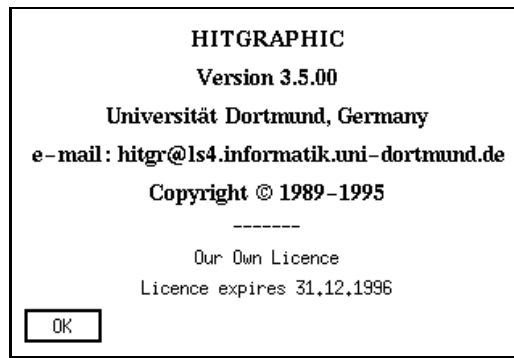


Figure 4.5: HITGRAPHIC Information

opens a window displaying information about the tool (version, address, license). After confirmation you can continue working with HITGRAPHIC.

- *copy environment*

opens a startup like window to copy the complete environment or a selected element.

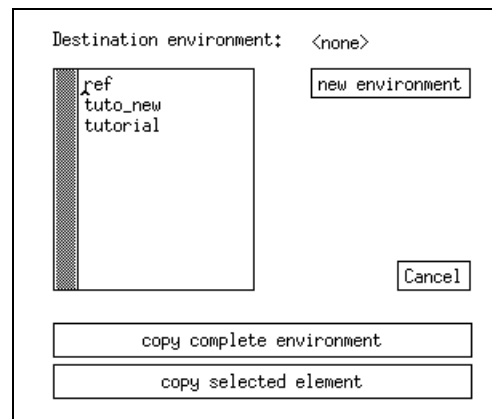


Figure 4.6: Subwindow to Copy to Other Environments

Firstly the destination environment has to be specified. This is done either by performing the *select* operation on a label within the environment list or by clicking on the *new environment* button if you want to create a new environment. In this case a dialog box is displayed requesting the new name. After entering the name and confirming it, the destination environment gets this name.

The two buttons at the bottom of the copy window allow to copy a *complete environment* or only a *selected element*. An element is selected with the left mouse button within the environment window.

During the *copy* operation, which may take some time, the corresponding button remains inverted.

If you have finished copying environments or elements the *Cancel* button is used to close the window.

The entry *copy environment* is not selectable, if other subwindows concerning a model type, a component type, an aggregation description, an evaluation or an experiment are still open.

- *change environment*

opens a startup like window to manipulate environments.

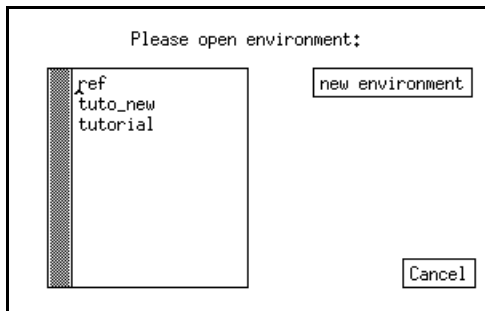


Figure 4.7: Subwindow to Change to Another Environment

There is a popup menu available on each environment within the displayed environment list.

- *open*

opens the selected environment and pops down the change window.

- *rename*

renames the selected environment.

A dialog box is displayed requesting the new name. After entering the name and confirming it the name of the environment is updated.

- *delete*

deletes the complete environment.

This operation should be performed carefully because the objects are really deleted within the database. Therefore a double beep can be heard and a confirmation of this operation is requested.

During the *delete* operation, which may take some time, the confirm box remains visible.

A new environment can be created by clicking with the left mouse button on the *new environment* button. A dialog box is displayed requesting the name of the new environment. After entering the new name and confirming it the environment window now displays the new environment only consisting of the standard component types.

The *Cancel* button can be used to close the window and to return to the environment window. If there exists no current environment, this button changes to *Quit Hitgraphic* and can be used to close the HITGRAPHIC session.

The entry *change environment* is not selectable, if other subwindows concerning a model type, a component type, an aggregation description, an evaluation or an experiment are still open.

- *quit*

terminates a HITGRAPHIC session.

This entry is not selectable, if other subwindows concerning a model type, a component type, an aggregation description, an evaluation or an experiment are still open.

## 4.1.2 Functions on Kinds of Modelling Objects

For any kind of modelling object a newly created object initially has the mode *free*. Next the functions are described in detail separated for each kind of modelling object.

### Model Types

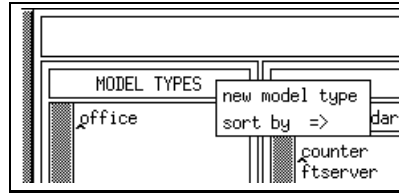


Figure 4.8: Menu on the Title of Model Types

- *new model type*

creates a new model type.

A dialog box is displayed, requesting the name of the new model type. After confirming the name by clicking *OK* with the left mouse button or entering Carriage Return, the new model type is inserted in the list of all model types according to the current sorting order.

An error occurs, if there already exists a model type, an evaluation or an experiment description with the entered name or if you entered the name of a standard component type.

- *sort by ->*

sorts the list of model types ...

- *names (asc)*

... alphabetically in ascending order.

- *names (desc)*

... alphabetically in descending order.

### Component Types

Component types are divided in standard component types (counter, ftserver, nowait-send, observer, prioserver, semaphor, server, synchsend and tokenpool) and self defined component types. Standard component types cannot be changed and only a restricted set of operations is possible.

- *new component type*

creates a new component type.

A dialog box is displayed, requesting the name of the new component type. After confirming the name the new component type is inserted in the list of the self defined component types according to the current sorting order.

An error occurs, if there already exists a component type with the entered name.

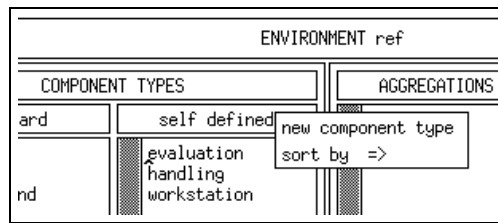


Figure 4.9: Menu on the Title of Component Types

- *sort by =>*  
sorts the list of component types ...
  - *names (asc)*  
... alphabetically in ascending order.
  - *names (desc)*  
... alphabetically in descending order.

### Aggregation Descriptions

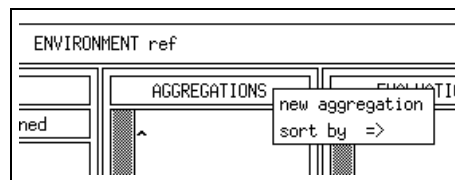


Figure 4.10: Menu on the Title of Aggregations

- *new aggregation*  
creates a new aggregation description.  
A dialog box is displayed, requesting you to select the component type to aggregate in the list of self defined component types with the left mouse button. After confirmation the new aggregation description automatically gets the name of the selected component type and is inserted in the list of aggregation descriptions according to the current sorting order. Note that standard component types cannot be aggregated.  
An error message is displayed, if there already exists an aggregation description for the selected component type or if the selection was invalid.
- *sort by =>*  
sorts the list of aggregation descriptions ...
  - *names (asc)*  
... alphabetically in ascending order.
  - *names (desc)*  
... alphabetically in descending order.

## Evaluations



Figure 4.11: Menu on the Title of Evaluations

- *new evaluation*

creates a new evaluation.

A dialog box is popped up, requesting you to select the model type to evaluate in the list of all model types with the left mouse button. After clicking *OK*, a second dialog box is displayed, asking for the name of the new evaluation. If you enter a correct name and confirm it with *OK* or Carriage Return, the new evaluation is inserted in the list of evaluations according to the current sorting order.

An error message is displayed, if the selection was invalid, or, if there already exists an evaluation or a model type with the entered name.

- *sort by =>*

sorts the list of evaluations ...

- *names (asc)*

... alphabetically in ascending order.

- *names (desc)*

... alphabetically in descending order.

## Experiments

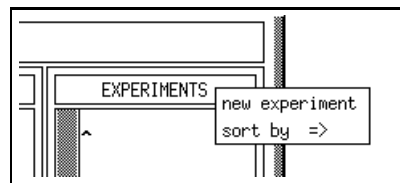


Figure 4.12: Menu on the Title of Experiments

- *new experiment*

creates a new experiment.

A dialog box is displayed, requesting a name for the new experiment. After entering a correct name and confirming it, the new experiment is inserted in the list of experiments according to the current sorting order.

An error message is displayed, if there already exists an experiment or a model type with the entered name.

- *sort by* ->
  - sorts the list of experiments ...
    - *names (asc)*
      - ... alphabetically in ascending order.
    - *names (desc)*
      - ... alphabetically in descending order.

### 4.1.3 Functions on Modelling Objects

#### Functions on a Model Type

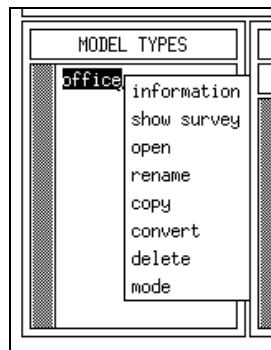


Figure 4.13: Menu on Model Types

- *information*
  - opens the HITGRAPHIC text editor with the information file of the selected model type.
  - This entry is not selectable, if the text editor with this information file is already open.
  - The information file is displayed in read-only mode, if the selected type is locked by another user or released or if file permissions are not sufficient for write mode.
- *show survey*
  - opens the survey window of the selected model type.
  - This entry is not selectable, if the survey window of this model type is already open.
- *open*
  - opens the component type graphic window that corresponds to the selected type.
  - This entry is not selectable, if the requested component type graphic window has already been opened.
  - A message is displayed, if another user has already opened this window. In this case the window is not opened. This behaviour prevents changes to get lost during concurrent editing.

If the selected type is locked by another user or released you can only have a look on the corresponding component type graphic, but not change it.

- *rename*

allows to rename the selected model type.

A dialog box is displayed, requesting the new name of the model type. After confirmation the model type is reinserted in the list of all model types according to the current sorting order.

In order to avoid misstatements of names, *rename* is not selectable, if a subwindow of the model type is open.

If the selected type is locked by another user or released, this operation cannot be performed. An appropriate message is displayed in this case.

An error occurs, if there already exists a model type, an evaluation or an experiment description with the entered name or if you entered the name of a standard component type.

- *copy*

copies the selected model type.

A dialog box is displayed, requesting the name of the copy of the selected model type. After confirming the name the new model type is inserted in the list of the model types according to the current sorting order. Initially, the mode of the new type is *free*.

*Copy* is not selectable, if a subwindow of this model type is still open.

An error occurs, if there already exists a model type, an evaluation or an experiment description with the entered name or if you entered the name of a standard component type.

- *convert*

converts the selected model type into a component type (without deleting the model type).

A dialog box is displayed, requesting the name for the component type. After confirmation the resulting component type is inserted in the list of the self defined component types according to the current sorting order.

Converting a model type into a component type causes the following changes concerning the converted type in the component type graphic window:

- A component type has no global declarations (a model type may have). They get lost in the new type generated by *convert*.
- Component types have control procedures (model types have not). After conversion they have the default set of control procedures.

*Convert* is not selectable, if a subwindow of this model type is still open.

An error occurs, if there already exists a component type with the entered name.

- *delete*

deletes the selected model type.

A dialog box is displayed, asking for confirmation to delete the model type. After clicking *OK* with the left mouse button, the model type is removed from



the list of all model types. Note that evaluations created from this model type are deleted, too. If you select *Cancel*, the model type is not deleted.

This entry is not selectable, if a subwindow of this model type is still open.

If the selected type is locked by another user or released, this operation cannot be performed. An appropriate message is displayed in this case.

- *mode*

permits you to change the mode of the selected model type, if the current mode is free or locked by you.

Section 5.1 describes in detail the functionality of this operation.

You cannot change the mode, if the model type is locked by another user or released or if someone is just working on that model type. In this case you can only select the *Cancel* button.

## Functions on a Component Type

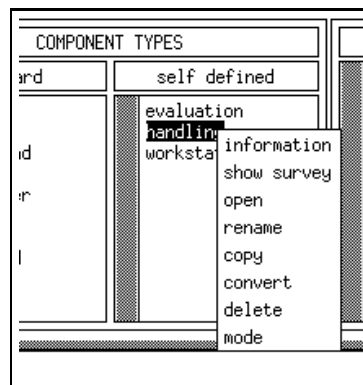


Figure 4.14: Menu on Component Types

On standard components only the entries *information* and *open* are available.

- *information*

opens the HITGRAPHIC text editor with the information file of the selected component type.

This entry is not selectable, if the text editor with this information file is already open.

The information file is displayed in read-only mode, if the selected type is locked by another user or released, if you select a standard component type or if file permissions are insufficient for write mode.

- *show survey*

opens the survey window of the selected self defined component type.

This entry is not selectable, if the survey window of this component type is already open.

- *open*

opens the component type graphic window that corresponds to the selected

type.

This entry is not selectable, if the requested component type graphic window has already been opened.

A message is displayed, if another user has already opened this window. In this case the window is not opened. This behaviour prevents changes to get lost during concurrent editing.

If the selected type is locked by another user or released or if a standard component type is selected, you can only have a look on the corresponding component type graphic, but not change it.

- *rename*

allows to rename the selected self defined component type.

A dialog box is displayed, requesting the new name of the component type. After confirmation the component type is reinserted in the list according to the current sorting order. Note that a corresponding aggregation description is renamed and reinserted in its list, too.

This entry is not selectable for standard component types or, in order to avoid misstatements of names, if a subwindow of the component type is open.

If the selected type is locked by another user or released or if a subwindow of the corresponding aggregation description is open, *rename* cannot be performed. An appropriate message is displayed in this case. The mode of the corresponding aggregation description, if it exists, is not considered.

An error occurs, if there already exists a component type with the entered name.

- *copy*

copies the selected self defined component type.

A dialog box is displayed, requesting the name of the copy of the selected component type. After confirmation the new component type is inserted in the list according to the current sorting order. Initially, the mode of the new type is *free*.

This entry is not selectable, if a subwindow of this component type is still open. An error occurs, if there already exists a component type with the entered name.

- *convert*

converts the selected self defined component type into a model type (without deleting the component type).

A dialog box is displayed, requesting the name for the model type. After confirmation the resulting model type is inserted in the list of the model types according to the current sorting order.

This operation causes the following changes concerning the converted type in the component type graphic window:

- Provided services and procedures in the component type are not provided in the converted (model) type.
- Shared components in the component type change to normal components in the converted type.
- The model type has no control procedures (the component type has).

- The model type can have global declarations (the component type does not have), after *convert* they are initially empty.

This entry is not selectable, if a subwindow of the component type is still open. An error occurs, if there already exists a model type, an evaluation or an experiment description with the entered name or if you entered the name of a standard component type.

- *delete*

deletes the selected self defined component type.

A dialog box is displayed, asking for confirmation to delete the component type. After clicking *OK* with the left mouse button, the component type is removed from the list. If there exists a corresponding aggregation description, it is also deleted. Note that all components that are incarnations of this type, original or aggregated, are deleted, too. Pressing *Cancel* aborts the *delete* operation.

This entry is not selectable, if a subwindow of this component type is still open. *Delete* cannot be performed, if the selected type is locked by another user or released or if the corresponding aggregation description is open. An appropriate message is displayed in this case. (The mode of the corresponding aggregation description is not considered when performing an “automatic” *delete*, it is deleted anyway.)

- *mode*

permits you to change the mode of the selected model type, if the current mode is free or locked by you.

Section 5.1 describes in detail the functionality of this operation.

You cannot change the mode, if the component type is locked by another user or released or if someone is just working on that component type. In this case you can only select the *Cancel* button.

## Functions on Aggregation Descriptions

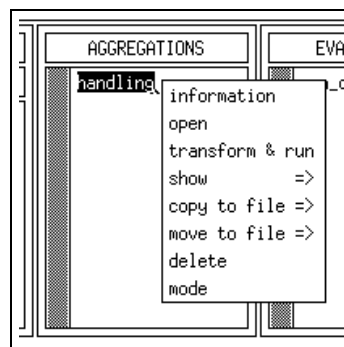


Figure 4.15: Menu on Aggregation Descriptions

- *information*

opens the HITGRAPHIC text editor with the information file of the selected

aggregation description.

This entry is not selectable, if the text editor with this information file is already open.

The information file is displayed in read-only mode, if the selected aggregation description is locked by another user or released or if file permissions are insufficient for write mode.

- *open*

opens the aggregation description window of the selected aggregation description.

This entry is not selectable, if this aggregation description window is already open.

A message is displayed, if another user has already opened this window. In this case the window is not opened. This behaviour prevents changes to get lost during concurrent editing.

If there are no provided services or there are provided procedures within the corresponding component type, the aggregation description window will appear, but only to inform you with a message box that aggregation is not possible in this case.

The aggregation description window is opened in read-only mode, if the selected aggregation description is locked by another user or released.

- *transform & run*

generates the HI-SLANG source code of the selected aggregation description and starts HIT.

If an error or a warning occurs during the transformation from the graphical description to HI-SLANG a window is popped up displaying the error message(s) (cf. Figure 4.19). It provides the following functions:

- *search*

causes a subwindow to appear allowing search operations on the error messages.

- *quit*

closes the transformation errors window.

If there are only warning messages initially no functions are available and the transformation errors window is automatically closed after performing the *run* within the run control popup. If no HIT run should take place, the transformation errors window can be closed with the *quit* function after the run control popup has been quitted.

After correct completion of transformation or in case of warnings the run control popup (cf. Figure 4.20) will be opened. In the run control popup the mode of an aggregation run (*dialog* or *batch*) and the main memory for HIT compilation and aggregation run can be specified.

On the run control popup, there are again some functions.

- *run*

starts the HI-SLANG compilation and, if no error occurred during the compilation, HIT will start the aggregation run (in the specified run mode). Generated results, listing and protocol files will be automatically added to the database (related to the corresponding aggregation description).

– *quit*

aborts the run specification and no HIT compilation and run will be done.

If the run mode BATCH was selected and the run was started the run control popup is automatically quitted and the compilation and experiment is done in the background.

If the run mode DIALOG was selected compilation messages and the experiment protocol are displayed in a protocol window.

The memory limitations for HIT compilation and experiment run must be positive integers.

The entry *transform & run* is not selectable, if the corresponding aggregation description window is open.

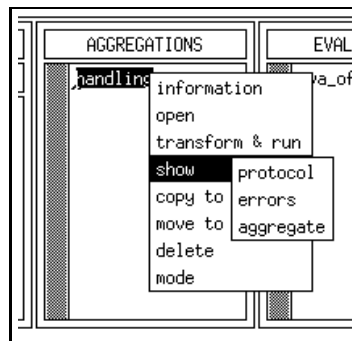


Figure 4.16: Show Function on Aggregation Descriptions

• *show* ->

opens a HITGRAPHIC text editor or subwindow in read-only mode.

It shows ...

– *protocol*

... the standard output (HI-SLANG compilation messages and aggregation protocol) and the HI-SLANG listing that were generated during the latest run of HIT.

– *errors*

... a list of errors and warnings of the latest HIT run and their relation to objects in HITGRAPHIC if possible (cf. Figure 4.22). In the popup menu of the title bar, the following functions are provided:

\* *search*

causes a subwindow to appear allowing search operations on the error messages.

\* *quit*

closes the subwindow.

– *aggregate*

... the results of the latest HIT run.

A single entry is not selectable, if the corresponding text editor is already open.

- *copy to file* ->

copies the corresponding file from the database to the file system.

A popup is displayed requesting the name of the destination file for the *copy* operation. The input box will contain the following default names dependent on the selected button.

<i>listing</i>	<aggregation name>.lis
<i>standard output</i>	<aggregation name>.pro
<i>aggregate</i>	<aggregation name>.agg

The directory where you have called HITGRAPHIC is used as current directory for the file name, but you can also give absolute file names.

- *move to file* ->

moves the corresponding file from the database to the file system. Afterwards the file is no longer available in the database.

A popup is displayed requesting the name of the destination file for the *move* operation. The input box will contain the following default names dependent on the selected button.

<i>listing</i>	<aggregation name>.lis
<i>standard output</i>	<aggregation name>.pro
<i>aggregate</i>	<aggregation name>.agg

The directory where you have called HITGRAPHIC is used as current directory for the file name, but you can also give absolute file names.

- *delete*

deletes the selected aggregation description and, if existing, the corresponding additional information, i.e., HI-SLANG listing, protocol file and the results of the aggregation.

A dialog box is displayed, asking for confirmation to delete the aggregation description. After clicking *OK* with the left mouse button, the aggregation description is removed from the list of all aggregation descriptions. Note that all components, that are incarnations of the corresponding aggregate, are changed to components of the corresponding (original) component type. Pressing *Cancel* aborts the *delete* operation.

This entry is not selectable, if the corresponding aggregation description window or the text editor with the information file of the selected aggregation description is still open.

If the selected aggregation description is locked by another user or released, this operation cannot be performed. An appropriate message is displayed in this case.

- *mode*

permits you to change the mode of the selected aggregation description, if the current mode is free or locked by you.

Section 5.1 describes in detail the functionality of this operation.

The mode cannot be changed, if the aggregation description is locked by another user or released or if someone is just working on the selected aggregation description. In this case you can only select the *Cancel* button.

## Functions on Evaluations

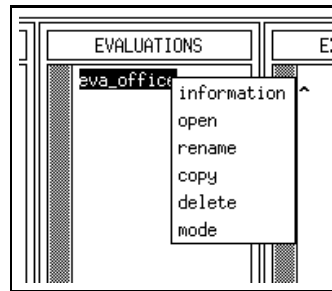


Figure 4.17: Menu on Evaluations

- *information*

opens the HITGRAPHIC text editor with the information file of the selected evaluation.

This entry is not selectable, if the text editor with this information file is already open.

The information file is displayed in read-only mode, if the selected evaluation is locked by another user or released or if file permissions are not sufficient for write mode.

- *open*

opens the corresponding evaluation window.

This entry is not selectable, if the window of the selected evaluation is already open.

A message is displayed, if another user has already opened this window. In this case the window is not opened. This behaviour prevents changes to get lost during concurrent editing.

If the selected evaluation is locked by another user or released, you can only have a look on the evaluation description, but not change it.

- *rename*

allows to rename the selected evaluation.

A dialog box is displayed, requesting the new name of the evaluation. After confirmation the evaluation is reinserted in the list of evaluations according to the current sorting order.

In order to avoid misstatements of names, *rename* is not selectable, if any

subwindows of the selected evaluation are open.

If the selected evaluation is locked by another user or released, *rename* cannot be performed. An appropriate message is displayed in this case.

An error occurs, if there already exists an evaluation or a model type with the entered name.

- *copy*

copies the selected evaluation, whereby the relation to the referred model type is maintained.

A dialog box is displayed, requesting the name of the copy of the selected evaluation. After confirmation the new evaluation is inserted in the list of the evaluations according to the current sorting order. Initially, the mode of the new evaluation is *free*.

This entry is not selectable, if subwindows of this evaluation description are still open.

An error occurs, if there already exists an evaluation or a model type with the entered name.

- *delete*

deletes the selected evaluation.

A dialog box is displayed, asking for confirmation to delete the evaluation. After clicking *OK* with the left mouse button, the evaluation is removed from the list of evaluations. Note that this evaluation is also removed from the evaluation lists in experiments. If a related experiment window is still open, the evaluation will no longer be displayed after the next *save* operation in that window. Pressing *Cancel* aborts the *delete* operation.

This entry is not selectable, if the evaluation window or the text editor with the information file of that evaluation is still open.

If the selected evaluation is locked by another user or released, this operation cannot be performed. An appropriate message is displayed in this case.

- *mode*

permits you to change the mode of the selected evaluation, if the current mode is free or locked by you.

Section 5.1 describes in detail the functionality of this operation.

You cannot change the mode, if the evaluation is locked by another user or released or if someone is just working on that evaluation. In this case you can only select the *Cancel* button.



## Functions on Experiments

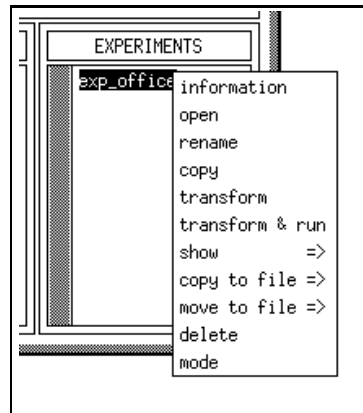


Figure 4.18: Menu on Experiments

- *information*

opens the HITGRAPHIC text editor with the information file of the selected experiment.

This entry is not selectable, if the text editor with this information file is already open.

The information file is displayed in read-only mode, if the selected experiment is locked by another user or released or if file permissions are not sufficient for write mode.

- *open*

opens the corresponding experiment window.

This entry is not selectable, if this experiment window is already open.

A message is displayed, if another user has already opened this window. In this case the window is not opened. This behaviour prevents changes to get lost during concurrent editing.

The experiment window is opened in read-only mode, if the selected experiment description is locked by another user or released.

- *rename*

allows to rename the selected experiment.

A dialog box is displayed, requesting the new name of the experiment. After confirmation the experiment is reinserted in the list of experiments according to the current sorting order.

In order to avoid misstatements of names, *rename* is not selectable, if any subwindows of the selected experiment are open.

If the selected experiment is locked by another user or released, *rename* cannot be performed. An appropriate message is displayed in this case.

An error occurs, if there already exists an experiment or a model type with the entered name.

- *copy*

copies the selected experiment.

A dialog box is displayed, requesting the name of the copy of the selected experiment. After confirmation the new experiment is inserted in the list of the experiments according to the current sorting order. Initially, the mode of the new experiment is *free*.

This entry is not selectable, if subwindows of this experiment description are still open.

An error occurs, if there exists already an experiment or a model type with the entered name.

- *transform*

generates the HI-SLANG source code of the selected experiment.

A popup is displayed requesting the name of the output file for the transformation. The default name is <experiment name>.hit in the directory where you have called HITGRAPHIC. After confirming the file name the transformation starts. During the execution of that operation the popup menu remains popped up.

If an error or a warning occurs during the transformation from the graphical description to HI-SLANG a transformation errors window is popped up displaying the error message(s).

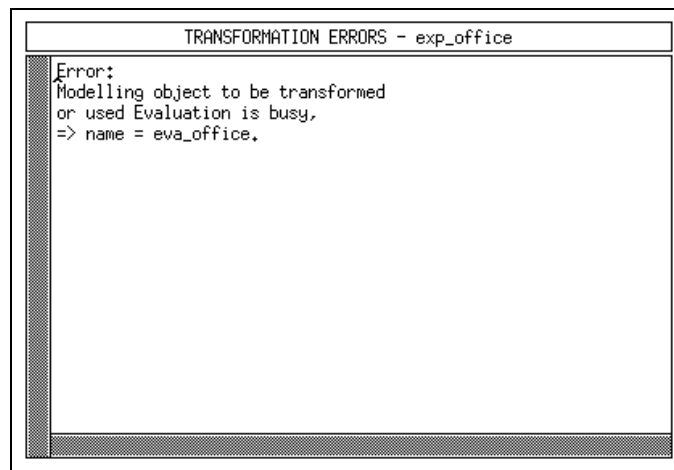


Figure 4.19: Transformation Errors Window

It provides the following functions:

- *search*

causes a subwindow to appear allowing search operations on the error messages.

- *quit*

closes the transformation errors window.

- *transform & run*

generates the HI-SLANG source code of the selected experiment and starts HIT.

If an error or a warning occurs during the transformation from the graphical

description to HI-SLANG a transformation errors window is popped up (cf. Figure 4.19) displaying the error message(s). It provides the following functions:

- *search*  
causes a subwindow to appear allowing search operations on the error messages.
- *quit*  
closes the transformation errors window.

If there are only warning messages initially no functions are available and the transformation errors window is automatically closed after performing the *run* within the run control popup. If no HIT run should take place, the transformation errors window can be closed with the *quit* function after the run control popup has been quitted.

After correct completion of transformation or in case of warnings the run control popup will be opened. In the run control popup the mode of an experiment run

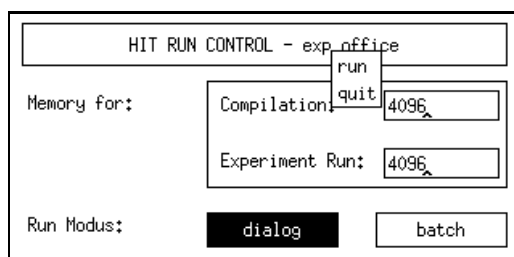


Figure 4.20: HIT Run Control Window

(*dialog* or *batch*) and the main memory for HIT compilation and experiment run can be specified.

On the run control popup, there are again some functions.

- *run*  
starts the HI-SLANG compilation and, if no error occurred during the compilation, HIT will start the experiment run (in the specified run mode). Generated results, trace, listing and protocol files will be automatically added to the database (related to the corresponding experiment).
- *quit*  
aborts the run specification and no HIT compilation and experiment run will be done.

If the run mode BATCH was selected and the experiment run was started the run control popup is automatically quitted and the compilation and experiment is done in the background.

If the run mode DIALOG was selected compilation messages and the experiment protocol are displayed in a protocol window.

The memory limitations for HIT compilation and experiment run must be positive integers.

The entry *transform & run* is not selectable, if the corresponding experiment window is open.

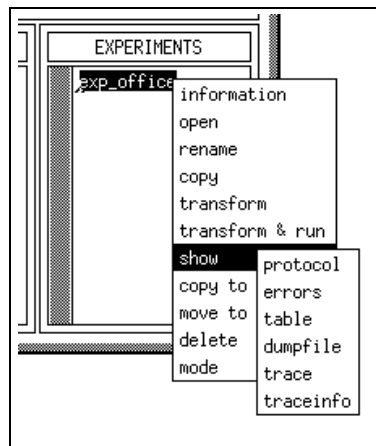


Figure 4.21: Show Function on Experiments

- *show* ->

opens a HITGRAPHIC text editor or subwindow in read-only mode.

It shows ...

- *protocol*

... the standard output (HI-SLANG compilation messages and experiment protocol) and the HI-SLANG listing that were generated during the latest run of HIT.

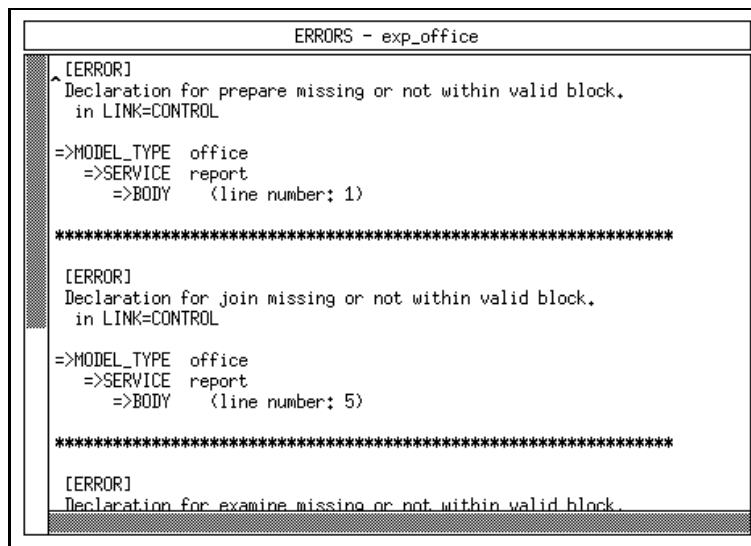


Figure 4.22: Errors of a HIT Run

- *errors*

... a list of errors and warnings of the latest HIT run and their relation to objects in HITGRAPHIC if possible. In the popup menu of the title bar, the following functions are provided:

- \* *search*

causes a subwindow to appear allowing search operations on the error messages.

- \* *quit*  
closes the subwindow.
- *table*  
... the table representation of the results if the output TABLE was specified in the experiment window.
- *dumpfile*  
... the dumpfile representation of the results if the output DUMPFILE was specified in the experiment window.
- *trace*  
... the trace if it was demanded for a trace in the experiment or evaluation window.
- *traceinfo*  
... information (size) about the tracefile if it was demanded for a trace in the experiment or evaluation window.

A single entry is not selectable if the corresponding text editor is still open. You will get a message if the requested text is not available.

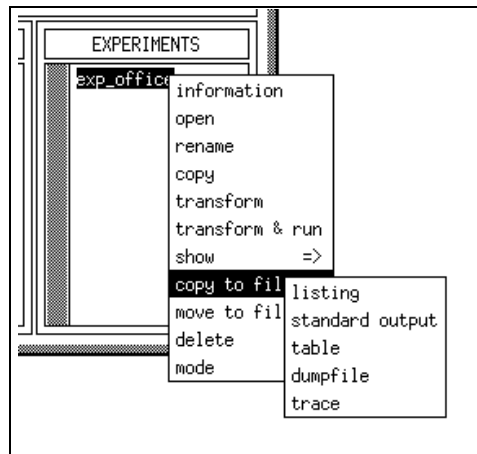


Figure 4.23: Copy Function on Experiments

- *copy to file* ->

copies the corresponding file from the database to the file system.

A popup is displayed requesting the name of the destination file for the *copy* operation. The input box will contain the following default names dependent on the selected button.

<i>listing</i>	<experiment name>.lis
<i>standard output</i>	<experiment name>.pro
<i>table</i>	<experiment name>.tab
<i>dumpfile</i>	<experiment name>.dmp
<i>trace</i>	<experiment name>.tra

The directory where you have called HITGRAPHIC is used as current directory for the file name, but you can also give absolute file names.

A message is displayed if the requested texts are not available.

- *move to file* ->

moves the corresponding file from the database to the file system. Afterwards the file is no longer available in the database.

A popup is displayed requesting the name of the destination file for the *move* operation. The input box will contain the following default names dependent on the selected button.

<i>listing</i>	<experiment name>.lis
<i>standard output</i>	<experiment name>.pro
<i>table</i>	<experiment name>.tab
<i>dumpfile</i>	<experiment name>.dmp
<i>trace</i>	<experiment name>.tra

The directory where you have called HITGRAPHIC is used as current directory for the file name, but you can also give absolute file names.

A message is displayed if the requested texts are not available.

It is recommended to use this function for traces if the size of the tracefile (see the item *traceinfo* in the *show* menu) is too large.

- *delete*

deletes the selected experiment.

A dialog box is displayed, asking for confirmation to delete the experiment. After clicking *OK* with the left mouse button, the experiment is removed from the list of all experiments. Pressing *Cancel* aborts the *delete* operation.

This entry is not selectable, if the corresponding experiment window or the text editor with the information file of the selected experiment is still open.

If the selected experiment is locked by another user or released, this operation cannot be performed. An appropriate message is displayed in this case.

- *mode*

permits you to change the mode of the selected experiment, if the current mode is free or locked by you.

Section 5.1 describes in detail the functionality of that operation.

The mode cannot be changed, if the experiment is locked by another user or released or if someone is just working on the selected experiment. In this case you can only select the *Cancel* button.

## 4.2 The Survey Window

The survey window is entered via a *show survey* being performed on a model/component type within the environment window.

It displays the hierarchical structure of the model/component type under study. The hierarchical structure in the given context is the result of the instantiations of component types within involved model/component types.

According to this understanding of hierarchical structure the survey window provides two different views. The first, which we refer to as the type structure, displays a type oriented view. The second view, the object structure, provides the same information based on instantiated objects.

The survey window is composed of three parts (cf. Figure 4.24). First a title bar provides the user with the name and the kind (model/component) of the type under study.

The buttons *TYPE STRUCTURE* and *OBJECT STRUCTURE* can be used to switch between the views, that are displayed in the scrollable region.

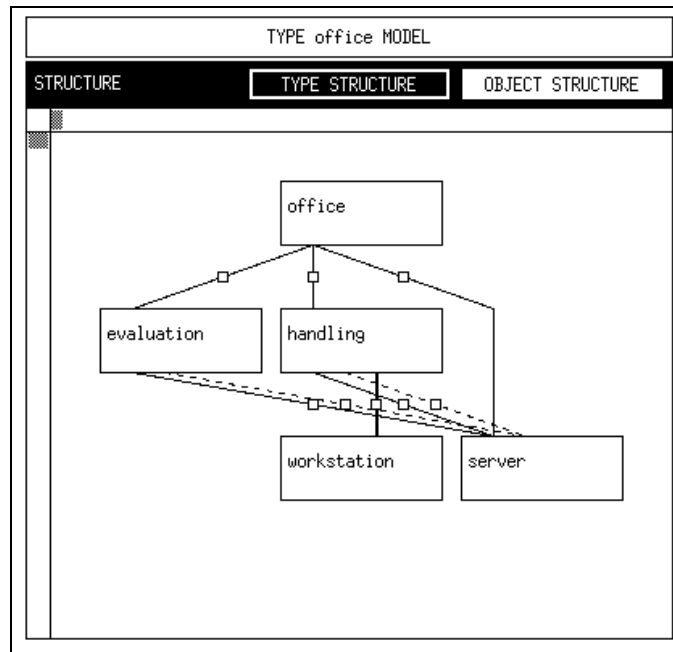


Figure 4.24: Survey Window – Type Structure

As mentioned above the *type structure* deals with model/component types and their hierarchical dependencies. Types are displayed in the form of **labelled boxes** which provide a *show type* operation to show the component type graphic window in read-only mode. The boxes are connected via lines that represent the hierarchical relation between types. Each connection represents at least one instantiation of a certain type due to the following constraints. An object can be instantiated single or as an array which is denoted by a **normal** respectively a **bold line**. Another criterion of instantiation results from the difference of a normal instantiation and a shared one where the latter refers to an object instantiated somewhere else. This difference is visualized by the use of **solid** respectively **dashed lines**. The combination of these criterions leads to four different kinds of lines. In Figure 4.24 three of them can be seen.

The names of the instantiated objects that are represented by one line can be popped up by selecting the small box on a line with the right mouse button. The list is

popped down, when the cursor leaves the box.

Note that within the type structure each type is displayed exactly once.

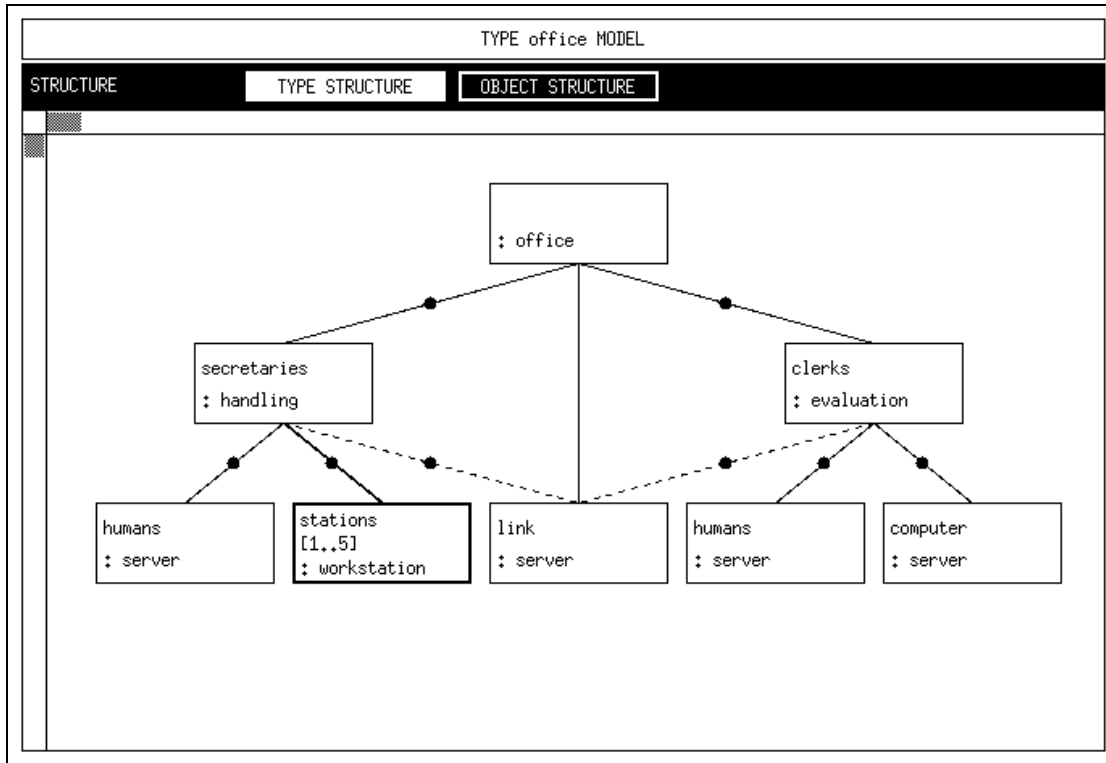


Figure 4.25: Survey Window – Object Structure

In contrast to the type structure the *object structure* deals with instantiated objects. Each instantiated object is represented by a solid labelled box. The semantics of **dashed/solid** and **normal/bold lines** are analogous to the type structure. A dot on the line indicates that there exists a binding of a used service in the upper component to a provided service of the lower component.

Note that an object may be accessed from different hierarchical levels, e.g., the object “link” of type “server” in Figure 4.25. The shared instantiation within the types “handling” and “evaluation” is within the scope of the normal instantiation in type “office”. For this reason the same object is referenced. If there was no normal instantiation of the component “link” within the scope of the model/component type under study each shared instantiation would reference to an own object (displayed as a dashed box).

The position of objects in the object structure can be influenced by the positions of the objects in the corresponding component type graphic windows. Changes have to be saved there, and the view in the survey window has to be actualized by a *refresh* operation.

By default the object structure is displayed because it contains a more significant view of the structure.



## 4.2.1 Global Functions

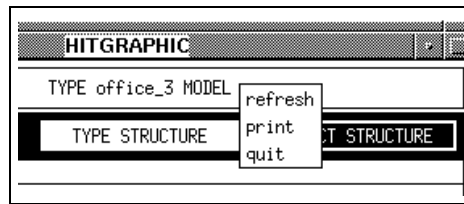


Figure 4.26: Menu on the Title of the Survey Window

- *refresh*  
updates the currently displayed view.  
This operation should be performed if modifications on underlying components have been made to mirror the actual status of the model respectively component type.
- *print*  
prints the object structure of the model or component.  
An input box requesting the settings for the PostScript file appears (cf. Section 5.3).  
This entry is only available if the object structure is selected.
- *quit*  
quits the survey window.  
This entry is not selectable, if a component type graphic window started from this survey window is still open.

## 4.2.2 Popup Menu in the Survey Window

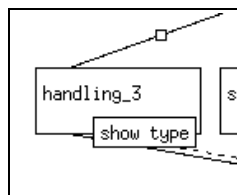


Figure 4.27: Menu on Types in the Survey Window

- *show type*  
shows the component type graphic window in read-only mode that corresponds to the selected type respectively object.  
This entry is not selectable, if the requested component type graphic window is already open.

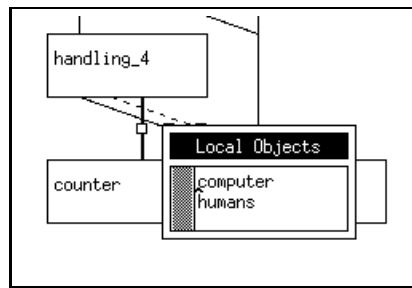


Figure 4.28: List of Local Objects

### 4.2.3 Function on Line in the Type Structure

Within the type structure of the survey window small rectangular boxes between the types provide the possibility to list all the local objects of the lower type used in the higher type. After performing a click with the right mouse button on the grip a scrollable list is displayed showing the local objects. The list disappears after a cursor movement out of the box.

## 4.3 The Component Type Graphic Window

The component type graphic window provides graphical support for the actual task of specifying a HIT model/component type. Performing an *open* operation on a model respectively component type within the environment window yields the component type graphic window showing the current “ingredients” of the chosen type.

To describe the whole extent of a component type graphic window, let us distinguish elements which are represented graphically from elements which can be referred through popup menus. The graphical representation of some elements is shown in Figure 4.29.

Components are placed on the bottom of the graphic region. A component is visualized by a box that is labelled with the name of the component and the component type. In Figure 4.29 you see that the component type “handling” includes the three components “humans”, “stations” and “link”. “Stations” is a component array which is emphasized by a shadowed box and the indication of the array bounds in the label. An important fact to mention is the spatial separation of **normal** and **shared** components. Normal components are placed left hand from the shared components which are symbolized through dashed boxes drawn on a grey background (like the component “link”).

Components may provide **services** and/or **procedures**, e.g., components of the type “server” provide the service “request”. Provided services are represented by a vertically drawn line starting at the upper side of a component box, provided procedures are drawn with a dashed line.

Services and procedures that are subordinated to the current model/component type are represented by parallelograms, dashed in case of a procedure, placed on the left side of the graphic region. A parallelogram is labelled with the name of the service/procedure it represents and, if the service/procedure is provided to an upper

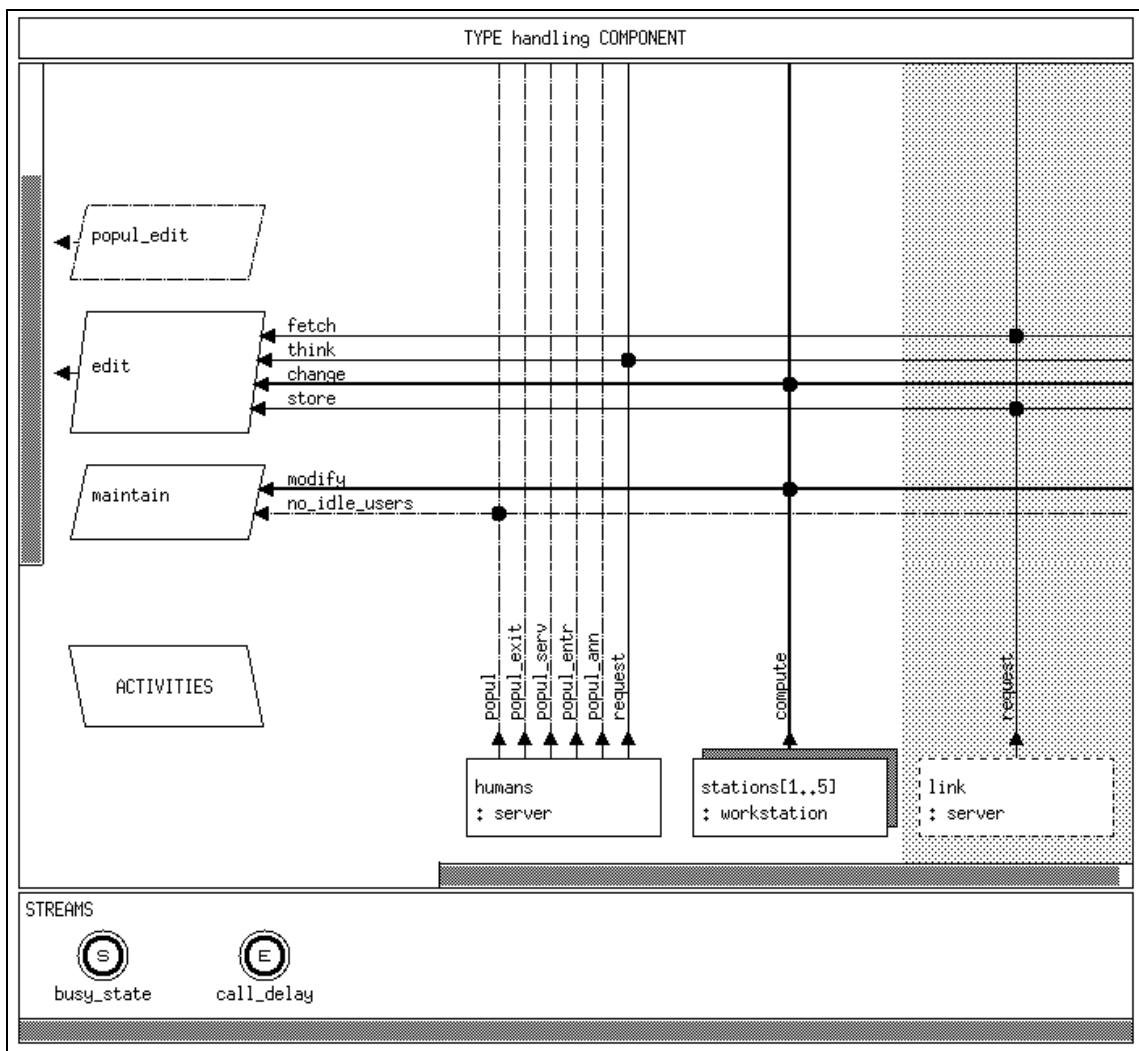


Figure 4.29: Component Type Graphic Window

level, a small left-pointing arrow is drawn at the left edge. Here, we have a spatial separation between services and procedures that are provided and internal services and procedures: the first are always placed on top of the latter.

**Used services/procedures** are represented by horizontal arrows pointing to the parallelogram of the using service/procedure. The service “edit” for example, which is provided to an upper layer, uses four services.

To sum up the following line styles are used:

- *solid* for services,
- *dashed* for procedures,
- *bold* for arrays.

To indicate that a provided service/procedure refers to a used service/procedure a **dot** is drawn at the point of intersection. The binding is also called a connection.

The parallelogram on the bottom represents the **activities** of the current model/component type. Opening it will yield a text editor where you can enter HI-SLANG code

for the creation of activities and for other initialisations.

**Streams** within a component type are displayed in a subwindow at the bottom of the component type graphic window as you can see in Figure 4.29. The subwindow is only displayed if a stream exists in the model/component type. A stream is labelled with its name and a capital letter that indicates the **stream type** (C for count, E for event, or S for state stream).

Let us now briefly discuss the non-graphical elements. If you work on a model type, the popup menu on the title bar provides entries to specify *formal parameters* and global and local *declarations*. These tasks are, as mentioned above, done in dedicated text editors occurring when the corresponding entry is selected.

If you work on a component type, besides the entries *formal parameters* and *locals* you will find an entry for modelling the *control procedures* of the type. This is done in a special window that will be discussed in detail in Section 4.4.2. The same is valid for control procedures of components. Other elements that can be specified via text editors are information files, actual parameters of components, bodies, formal parameters and local declarations of services/procedures as well as formal parameters of used services/procedures.

### 4.3.1 Global Functions on Model Types

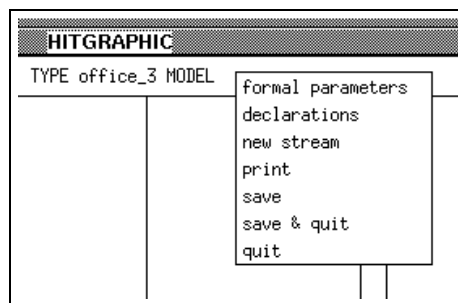


Figure 4.30: Menu on the Title of the Component Type Graphic Window

- *formal parameters*

opens the HITGRAPHIC text editor to specify the formal parameters of the model type.

The corresponding actual parameters have to be added within the experiment window, cf. Section 4.12.3.

This entry is not selectable, if the text editor with this file is already open. The file is displayed in read-only mode, if the model type is locked by another user or released or if file permissions are not sufficient for write mode.

- *declarations*

opens the HITGRAPHIC text editor consisting of two subwindows for global and local declarations, e.g., variables or record types.

This entry is not selectable, if the text editor with this file is already open. The files are displayed in read-only mode, if the model type is locked by another user or released or if file permissions are not sufficient for write mode.

- *new stream*

creates a new stream.

A dialog box is displayed, requesting the name for the new stream. After you have entered a correct name (i.e., a name that is not existing within the current type graphic window) and confirmed it the new stream is appended at the stream list as an event stream.

As standard streams are considered automatically, you must not create them. The error message “Sorry, name is in use.” is displayed, if there already exists a stream, a service or procedure, a used service or a used procedure, a component or a component type with the entered name. Error messages concerning name conflicts may also occur during HI-SLANG compilation.

This function is not selectable, if the current type is locked by another user or released.

- *print*

prints the type graphic of the model type.

An input box requesting the settings for the PostScript file appears (cf. Section 5.3).

- *save*

saves the current state of the model type.

This *save* does not include the contents of other windows, that provide a *save*, e.g., text editors or windows to enter control procedures. For the duration of this operation, the popup menu remains visible on the display. When the model type is saved, the type graphic is redrawn. At this point you see the actual state of the model type, i.e., how it is stored in the database. If you miss some objects, the corresponding types must have been deleted in the environment window meanwhile.

*Save* is not selectable, if the model type is locked by another user or released or if windows opened from the component type graphic window are not closed.

- *save & quit*

saves the current state of the model type and quits the component type graphic window.

If any problems occur during the save operation, the quit operation will not be performed, but the window will be actualized according to the current state of the database.

This entry is not selectable, if a window started from this component type graphic window is still open or if the model type is locked by another user or released.

- *quit*

quits the component type graphic window.

If any changes were made since the last *save* operation, you are asked for confirmation.

This entry is not selectable, if a window started from this component type graphic window is still open.

## 4.3.2 Global Functions on Component Types

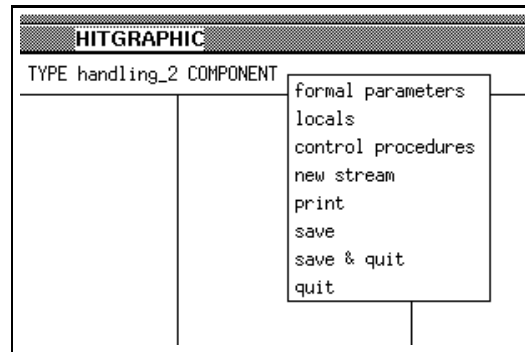


Figure 4.31: Menu on the Title of the Component Type Graphic Window

- *formal parameters*

opens the HITGRAPHIC text editor to specify the formal parameters of the component type.

This entry is not selectable, if the text editor with this file is already open.

The formal parameters are displayed in read-only mode, if the component type is a standard component type, if it is locked by another user or released, or if file permissions are not sufficient for write mode.

Among the standard component types the types “counter”, “semaphor”, “tokenpool”, “nowaitsend”, “ftserver”, and “observer” have formal parameters:

– counter:

```
min : ARRAY OF INTEGER;  
max : ARRAY OF INTEGER;  
init: ARRAY OF INTEGER
```

– semaphor:

```
sem_init: INTEGER DEFAULT 1
```

– tokenpool:

```
no_of_tokens: INTEGER
```

– nowaitsend:

```
no_of_buffers: INTEGER DEFAULT 1
```

– ftserver:

```
processors : INTEGER;  
degmax      : INTEGER DEFAULT 1;  
repair_units: INTEGER DEFAULT 1;  
failure_rate: REAL;  
repair_rate : REAL;  
dormancy    : REAL DEFAULT 1.0
```

– observer:

```
obs_interval: REAL;  
interactive : BOOLEAN DEFAULT FALSE
```

- *locals*

opens the HITGRAPHIC text editor to enter local declarations of the component type, e.g., local variables.

This entry is not selectable, if the text editor with this file is already open or if the type graphic shows a standard component type.

The file is displayed in read-only mode, if the component type is locked by another user or released or if file permissions are not sufficient for write mode.

- *control procedures*

opens the control procedure window for this component type.

This entry is not selectable, if the control procedure window for this component type is already open or if the type graphic shows a standard component type.

If the component type is locked by another user or released, you can only have a look on the control procedures, but not change them.

For further explanations cf. Section 4.4.2.

- *new stream*

creates a new stream.

A dialog box is displayed, requesting the name for the new stream. After you have entered a correct name (i.e., a name that is not existing within the current type graphic window) and confirmed it the new stream is appended at the stream list as an event stream.

As standard streams are considered automatically, you must not create them.

The error message “Sorry, name is in use.” is displayed, if there already exists a stream, a service or procedure, a used service or a used procedure, a component or a component type with the entered name. Error messages concerning name conflicts may also occur during HI-SLANG compilation.

This function is not selectable, if the current component type is locked by another user or released or if it is a standard component type.

- *print*

prints the type graphic of the component type.

An input box requesting the settings for the PostScript file appears (cf. Section 5.3).

- *save*

saves the current state of the component type.

This *save* does not include the contents of other windows, that provide a *save*, i.e., text editors or windows to enter control procedures. For the duration of this operation, the popup menu remains visible on the display. When the component type is saved, the type graphic is redrawn. At this point you see the actual state of the component type, i.e., how it is stored in the database. If you miss some objects, the corresponding types must have been deleted in the

environment window meanwhile.

*Save* is not selectable, if the component type is locked by another user or released, if windows opened from the component type graphic window are not closed, or if the type graphic shows a standard component type.

- *save & quit*

saves the current state of the component type and quits the component type graphic window.

If any problems occur during the save operation, the quit operation will not be performed, but the window will be actualized according to the current state of the database.

This entry is not selectable, if a window started from this component type graphic window is still open, if the component type is locked by another user or released, or if the type graphic shows a standard component type.

- *quit*

quits the component type graphic window.

If any changes were made since the last *save* operation, you are asked for confirmation.

This entry is not selectable, if a window started from this component type graphic window is still open.

### 4.3.3 Popup Menus in the Graphic Region

#### The Background Popup Menu

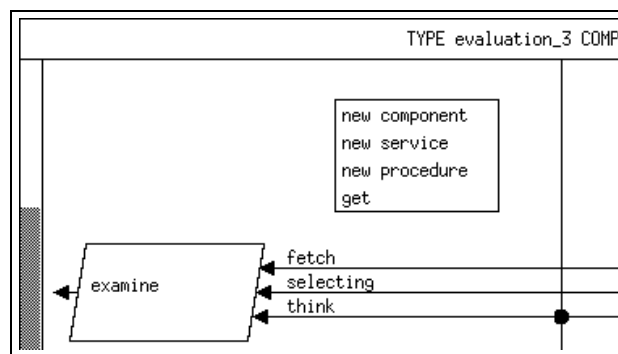


Figure 4.32: Menu on the Background of the Component Type Graphic Window

The entries of the popup menu are not selectable for standard component types.

- *new component*

is used to create an incarnation of an existing component type.

A dialog box is displayed, asking you to select a component type. At this point you have to turn to the environment window, select a component type with the left mouse button and then give your *OK* in the dialog box. If you really selected a component type – not, e.g., a model type or even nothing –, another



dialog box is displayed, requesting the name for the new component. Check the type name in the question – so you can be sure to incarnate the right object. After entering a correct name, i.e., not the name of another component or component type, service, procedure or stream in the current type, you will find the new component in your type graphic, placed at the right end of the normal components (as a normal component in the white area). If you did something wrong during this operation, error messages will tell you the problem.

*New component* is not selectable, if the current type is locked by another user or released.

- *new service*  
creates a new service.

- *new procedure*  
creates a new procedure.

In both cases a dialog box is displayed, requesting the name of the new service respectively procedure. After entering a correct name, i.e., not the name of another component or component type, service, procedure or stream in the current type, and confirming it you will find the new service respectively procedure placed at the upper end of the already existing internal services and procedures (as an internal service respectively procedure).

*New service* respectively *new procedure* is not selectable, if the current type is locked by another user or released.

- *get*  
inserts the component respectively service or procedure that was selected by the most recent *put* operation (cf. this section) into the current type graphic window.

A dialog box is displayed, requesting the name for the new component respectively the new service or procedure. After entering a correct name, i.e., not the name of another component or component type, service, procedure or stream in the current type, and confirming it you will find a copy of the object, on which the *put* operation was applied before.

This operation includes the copy of all files (e.g., parameter files) and, dealing with components, the control procedures of the original component. If the *put* operation was performed on a service respectively procedure, all used services and used procedures (including their files) are copied, too. This explains that you possibly may wait some seconds for the completion of this operation.

An error message is displayed, if no object from a previous *put* operation can be retrieved.

It is not necessary that the *put* and *get* operations are performed in the same type graphic window.

*Get* is not selectable, if the current type is locked by another user or released.

## Function on Activities

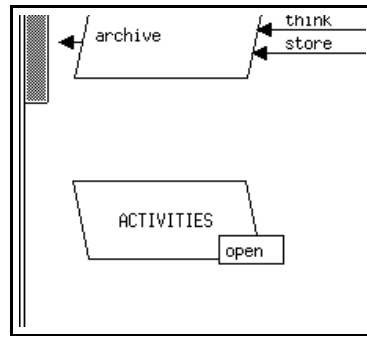


Figure 4.33: Menu on Activities in the Component Type Graphic Window

- *open*

opens the HITGRAPHIC text editor to enter HI-SLANG code for the creation of activities or for initialisations of the current component type respectively model type.

To ease the handling of specified parameters and declarations within the specification of activities they are also shown in read-only mode.

This entry is not selectable, if the text editor with the activities file is already open or the type graphic shows a standard component type. The file is displayed in read-only mode, if the component type is locked by another user or released or if file permissions are not sufficient for write mode.

## Functions on Components

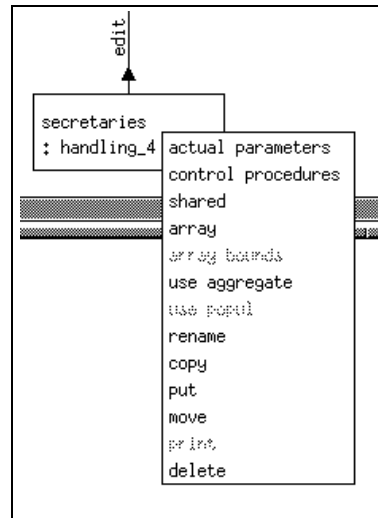


Figure 4.34: Menu on Components in the Component Type Graphic Window

- *actual parameters*

opens the HITGRAPHIC text editor with two subwindows to enter the actual

parameters of the component.

In the upper part of the editor the formal parameters of the type are displayed. This is useful, because the actual values must be entered in correct form and sequence in the lower part of the editor. Note that the formal parameters cannot be changed at this place.

If the formal parameters have been changed or deleted within the corresponding component type the actual parameters have to be updated in this window by the user himself.

The function is not selectable, if the text editor with the actual values of this component is already open. The actual parameters are displayed in read-only mode, if the current type is locked by another user or released or if file permissions are not sufficient for write mode.

- *control procedures*

opens the control procedure window for the component.

If this function is performed for the first time for this component, a dialog box is displayed asking for a *save* before opening the control procedure window.

This entry is not selectable, if the requested control procedure window is already open.

If the current type is locked by another user or released, you can only read the control procedures of the component, but cannot change them.

For explanations on control procedures concerning components cf. Section 4.4.3.

- *shared/normal*

transforms a normal component into a shared one respectively a shared component into a normal one.

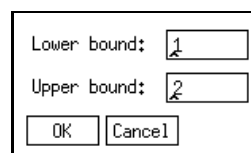
If you switch from *normal* to *shared*, the component is moved into the grey area and is placed there at the first position. If you switch from *shared* to *normal*, the component is moved into the white area and is placed there at the last position.

In model types, the switch is not selectable. It is also not selectable, if the current type is locked by another user or released.

- *array/single*

converts a single component into a component array respectively a component array into a single component.

If you switch from *single* to *array*, a dialog box is displayed, requesting integer values for the array bounds:



Lower bound:	<input type="text" value="1"/>
Upper bound:	<input type="text" value="2"/>
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

Figure 4.35: Array Bounds Window

If your *OK* is responded with a beep, maybe one of the values contains letters. Note that letters and the following characters are not ignored, if an entry begins

with a correct integer value.

An error message is displayed, if the lower bound value is greater than the upper bound value.

If the values are correct, they are appended at the component name in the type graphic. They are replaced by a “[\*]”, if component name and bounds do not fit in the component box.

After switching from *single* to *array* and performing a *save* operation all references to the single component in any evaluation are replaced by a reference to the first component with the index identical to the lower bound.

After switching from *array* to *single* and performing a *save* operation all references to any component or any range of components of the array in any evaluation are replaced by a reference to the single component.

This switch is not selectable, if the current type is locked by another user or released.

- *array bounds*

enables you to change the array bounds of the component array.

The values can be changed in the “array bounds” dialog box that is described for the *array/single* switch.

A change of bounds may cause components of the array to disappear, which are referenced in an evaluation. After a *save* operation all parts of any evaluation will be deleted, that contain such a reference to a no longer existing component in a component array. This can also cause a reduction of the range of the component array in an evaluation object.

This entry is not selectable, if you have selected a single component or if the current type is locked by another user or released.

- *use aggregate/use original*

performs a switch between aggregated and detailed representation for a component. Aggregated components are marked by a label below their box.

Please, note that all components of the same type exist in the same representation (within the component graphic window), that means either in aggregated or original representation. A representation switch for one component causes a representation switch for all objects of the same type within the component graphic window. If new components within the scope of this window are generated, their representation depends on existing components of the same type (within the component graphic window). If no other components exist the default “original” will be used.

Copying a component will not change its representation.

The switch to aggregate or original is only selectable if an aggregation description exists for the component type. The representation of a shared component cannot be changed, it depends on the information at the point of instantiation of the component.

Performing the switch from *original* to *aggregate* causes the control procedures to change to default values, which are “always” for the accept procedure and “all” for the offer procedure. Also evaluation objects and hierarchies within subcomponents are no longer available together with the corresponding specifications.

After a switch from the *original* to the *aggregated* representation and a *save* operation all components, that have been initiated because of the use of the original representation, will be deleted, because they no longer appear in the aggregated representation. This will also cause a deletion of all parts of any evaluation, that contain a reference to these now deleted components. The switch is not selectable, if the current type is locked by another user or released.

- *use popul/no popul*

performs a switch between normal and expanded view of the component. If you switch to *popul* additionally there are provided *popul* procedures which enable to determine the current number of processes in this component. Different procedures are available for the component areas and the complete component.

- *popul\_ann*
- *popul\_entry*
- *popul\_serv*
- *popul\_exit*
- *popul*

If the component type provides more than one service, for each of them another set of *popul* procedures exists.

This entry is only available for components of standard component types. It is not available, if the current type is locked by another user or released.

- *rename*

allows you to rename the component.

The new name can be entered in the dialog box that is popped up. After confirmation the component is redrawn with the new name in its box.

An error message is displayed, if there already exists a component or a component type, a service, a procedure or a stream with the entered name.

*Rename* is not selectable, if subwindows of this component are open or if the current type is locked by another user or released.

- *copy*

copies the component.

A dialog box is displayed, requesting the name for the copy of the component. After entering a correct name and confirming it the copy is inserted right hand from the selected component. This operation includes the copy of actual parameters and control procedures.

An error message is displayed, if there already exists a component or component type, a service, a procedure, or a stream with the entered name.

This function is not selectable, if the current type is locked by another user or released.

- *put*

stores the component in an internal buffer in order to prepare the next *get*

operation.

If the current component was not saved before, a dialog box is displayed asking for a *save* before performing the *put* operation.

- *move*

allows you to move the component to another position.

Move the cursor to the new position and press the left mouse button. The component will move from its old position to the new one.

A position is only accepted, if the cursor points between two components or left hand from the first component or right hand from the last component. Note that components can only be moved within their area, i.e., normal components within the white area and shared components within the grey area. Otherwise an error message is displayed.

The position of a component in the type graphic window affects the layout of object structures in survey windows containing objects of the type.

*Move* is not selectable, if the current type is locked by another user or released.

- *print*

prints the actual parameters and control procedures of the component.

(This is not yet implemented in the current HITGRAPHIC version.)

- *delete*

deletes the component.

A dialog box is displayed, asking for confirmation to delete the component. Clicking *OK* deletes the component.

*Delete* is not selectable, if subwindows of this component are open or if the current type is locked by another user or released.

## Function on Provided Services and Provided Procedures

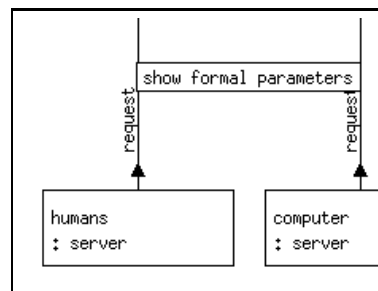


Figure 4.36: Menu on Provided Services

- *show formal parameters*

opens the HITGRAPHIC text editor with two subwindows showing the formal parameters of the provided service respectively procedure in the upper part and the result parameters in the lower part.

You can only have a look at the contents of the files with this operation; changing the contents is possible with the *formal parameters* function on this provided service in the type graphic of the underlying component type.

## Functions on Services and Procedures

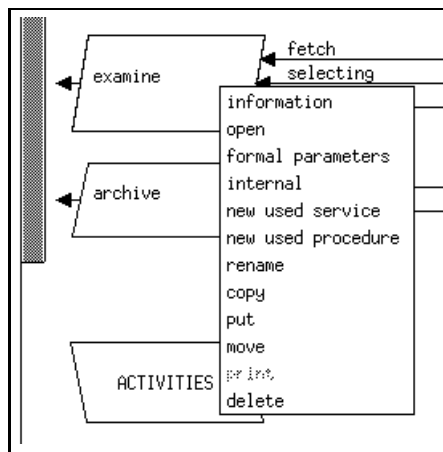


Figure 4.37: Menu on Services

Within standard component types only the entries *information* and *formal parameters* are selectable, and the displayed files are only readable.

- *information*

opens the HITGRAPHIC text editor with the information file of the service or procedure.

*Information* is not selectable, if the text editor with this information file is already open. The information file is displayed in read-only mode, if the current type is locked by another user or released or if file permissions are not sufficient for write mode.

- *open*

opens the HITGRAPHIC text editor to enter the interior of the service or procedure, i.e., the local declarations (variables or records) and the body.

The interface, i.e., formal parameters and results, are shown in read-only mode.

*Open* is not selectable, if the text editor with the body file is already open. The files are displayed in read-only mode, if the current type is locked by another user or released or if file permissions are not sufficient for write mode.

- *formal parameters*

opens the HITGRAPHIC text editor with two subwindows to enter or change the interface of the service respectively procedure, i.e., the formal parameters in the upper part and the result parameters in the lower part.

This entry is not selectable, if this text editor is already open. The files are displayed in read-only mode, if the current type is locked by another user or released or if file permissions are not sufficient for write mode.

- *provide/internal*

provides a service or procedure for external use respectively converts a provided service or procedure into an internal one not provided any longer.

If you select *provide* on an internal service or procedure, it is moved to the

lower end of the already existing provided services and procedures. A small left-pointing arrow indicates the fact, that the service or procedure is provided now.

If you select *internal* on a provided service or procedure, it is moved to the upper end of the already existing internal services or procedures, and the small arrow is removed.

This function is not selectable, if the current type is a model type or if the current type is locked by another user or released.

- *new used service*

creates a new used service.

*New used service* is not selectable for procedures.

More information can be found under the next item.

- *new used procedure*

creates a new used procedure.

In both cases (*new used service* and *new used procedure*) a dialog box is displayed, requesting the name of the new used service respectively procedure. After entering a correct name, i.e., a name that is not existing among the used services, the used procedures and streams, and confirming it you will find the new used service respectively procedure placed at the lower end of the already existing used services and procedures of the selected service or procedure.

The error message “Sorry, name is in use.” is displayed, if the service or procedure already has a used service or a used procedure with the same name or if there exists a stream with the same name.

This function is not selectable, if the current type is locked by another user or released.

- *rename*

allows you to rename the service respectively procedure.

Enter the new name in the dialog box that is popped up. After confirmation the service respectively procedure is redrawn with the new name.

An error message is displayed, if there already exists a service or procedure or a component or component type or a stream with the entered name.

*Rename* is not selectable, if subwindows of this service respectively procedure are open or if the current type is locked by another user or released.

- *copy*

copies the service respectively procedure.

A dialog box is displayed, requesting the name for the copy of the selected service respectively procedure. After entering a correct name and confirming it the copy is inserted directly below the selected service respectively procedure. This operation includes the copy of used services, used procedures and files.

An error message is displayed, if there already exists a service or procedure or a component or component type or a stream with the entered name.

*Copy* is not selectable, if the current type is locked by another user or released.



- *put*

stores the service respectively procedure in an internal buffer in order to prepare the next *get* operation.

If the current component type was not saved before, a dialog box is displayed asking for a *save* before performing the *put* operation.

- *move*

allows you to move the service respectively procedure to another position.

Move the cursor to the new (relative) position and press the left mouse button. The type graphic is redrawn, showing the service respectively procedure at its new position.

A position is only accepted, if the cursor points between two services or procedures or directly below or above the first respectively last service or procedure. Note that internal services and procedures can only be moved within the scope of internal services and procedures, provided services and procedures only within the scope of provided services and procedures. Otherwise an error message is displayed.

*Move* is not selectable, if the current type is locked by another user or released.

- *print*

prints the information file, formal parameters, locals and the body of the service respectively procedure.

(This is not yet implemented in the current HITGRAPHIC version.)

- *delete*

deletes the service respectively procedure.

A dialog box is displayed, asking for confirmation to delete the selected object. Clicking *OK* deletes the service respectively procedure and redraws the type graphic.

*Delete* is not selectable, if subwindows of this service respectively procedure are open or if the current type is locked by another user or released.

## Functions on Used Services and Used Procedures

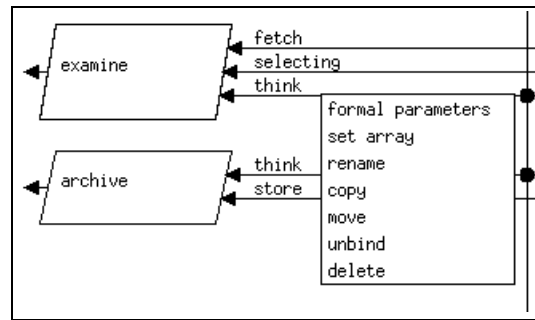


Figure 4.38: Menu on Used Services

- *formal parameters*

opens the HITGRAPHIC text editor with two subwindows to enter or change the formal parameters of the used service respectively procedure in the upper part and the result parameters in the lower part.

This entry is not selectable, if this text editor is already open. The files are displayed in read-only mode, if the current type is locked by another user or released or if file permissions are not sufficient for write mode.

- *set array/unset array*

transforms the used service or procedure into an array respectively transforms a used service or procedure array into a single used service or procedure.

Be sure, that you have connected component arrays with used service respectively procedure arrays and single components with single used services respectively procedures. Otherwise, you will get error messages during HI-SLANG compilation.

This entry is not selectable, if the current type is locked by another user or released.

- *rename*

allows you to rename the used service respectively used procedure.

Enter the new name in the dialog box that is popped up. After confirmation the used service respectively used procedure is redrawn with the new name.

The error message "Sorry, name is in use." is displayed, if there already exists a stream, a used service or a used procedure with the entered name at the current service or procedure.

*Rename* is not selectable, if the text editor with the formal parameters of this used service respectively used procedure is open or if the current type is locked by another user or released.

- *copy*

copies the used service respectively used procedure.

A dialog box is displayed, requesting the name for the copy of the selected used service respectively procedure. After entering a correct name and confirming it

the copy is inserted directly below the selected used service respectively used procedure. This operation includes the copy of the formal parameters.

The error message “Sorry, name is in use.” is displayed, if there already exists a stream, a used service or a used procedure with the entered name at the service or procedure you are working on.

*Copy* is not selectable, if the current type is locked by another user or released.

- *move*

allows you to move the used service respectively used procedure to another position within the other used services and used procedures of the current service respectively procedure.

Move the cursor to the new (relative) position and press the left mouse button. The service respectively procedure is redrawn, showing the used service respectively used procedure at its new position.

A position is only accepted, if the cursor points between two used services or procedures or below or above the first respectively last used service or procedure. Otherwise an error message is displayed.

*Move* is not selectable, if the current type is locked by another user or released.

- *unbind*

is used to delete connections.

*Unbind* is only selectable, if the used service respectively used procedure is already connected. If you select *unbind*, the connection is deleted.

This entry is not selectable, if the current type is locked by another user or released.

- *delete*

deletes the used service respectively used procedure.

A dialog box is displayed, asking for confirmation to delete the selected object. Clicking *OK* deletes the used service respectively used procedure and redraws the type graphic.

*Delete* is not selectable, if the text editor with the formal parameters of this used service respectively used procedure is open or if the current type is locked by another user or released.

## Functions on Streams

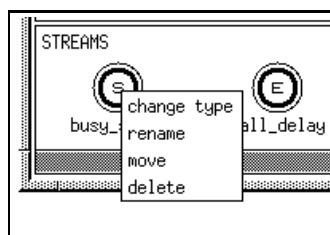


Figure 4.39: Menu on Streams

- *change type*

displays the stream dialog box to select the type of the stream.

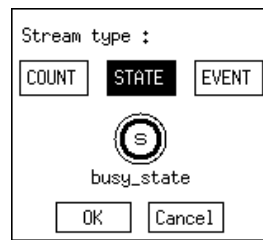


Figure 4.40: Stream Dialog Box

Select the stream type *COUNT*, *STATE* or *EVENT* with the left mouse button. The selected stream type is inverted. Click *OK* to confirm your selection.

The letter of the corresponding stream cycle in the type graphic, that indicates the selected stream type is updated. E stands for event stream, S for state stream and C for count stream.

*Change type* is not selectable, if the current type is locked by another user or released.

- *rename*

allows you to rename the stream.

Enter the new name in the dialog box that is popped up. After confirmation the stream is redrawn with the new name.

The error message “Sorry, name is in use.” is displayed, if there already exists a stream, a service or procedure, a used service or a used procedure, a component or a component type with the entered name within the current type.

*Rename* is not selectable, if the current type is locked by another user or released.

- *move*

allows you to move a stream to another position within the stream area.

Move the cursor to the new (relative) position and press the left mouse button. The stream area is redrawn, showing the stream at its new position.

A position is only accepted within the stream area. Otherwise an error message is displayed.

*Move* is not selectable, if the current type is locked by another user or released.

- *delete*

deletes the stream.

A dialog box is displayed, asking for confirmation to delete the stream. Clicking *OK* deletes the stream and redraws the type graphic.

*Delete* is not selectable, if the current type is locked by another user or released.

## Functions on Connections

A popup menu exists on the point of intersection of a line representing a provided service respectively procedure and a line representing a used service respectively used

procedure. The operations in the menu depend on the current state, i.e., whether a connection dot exists or not.

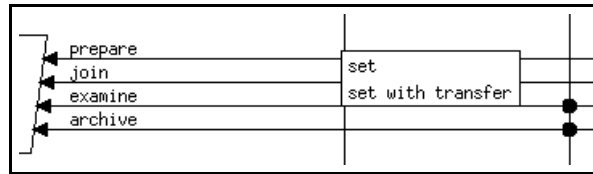


Figure 4.41: Menu on a Point of Intersection without a Dot

- *set*

is used to set a connection between used and provided service respectively procedure.

*Set* is not selectable, if the used service or procedure is already bound, if you try to connect a service and a procedure, a single line and an array line, or if the current type is locked by another user or released.

- *set with transfer*

operates like *set*, but additionally the formal parameters of the provided service respectively procedure are transferred to the used service respectively procedure. The transfer operates directly on the database without an explicit save operation. Therefore, a confirm box may appear.

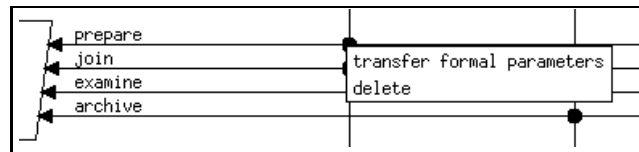


Figure 4.42: Menu on a Connection Dot

- *transfer formal parameters*

transfers the formal parameters of the provided service respectively procedure to the used service respectively procedure.

This function operates directly on the database without an explicit save operation. Therefore, a confirm box may appear.

This operation is not selectable, if the current type is locked by another user or released.

- *delete*

deletes the connection between used and provided service respectively procedure.

*Delete* is not selectable, if the current type is locked by another user or released.

## 4.4 The Control Procedure Window

The control procedure window supports the user in controlling the progress of a service call within a component. According to the general view of a component as an autonomous object, a component is (virtually) divided into four areas respectively queues. The transitions between the areas and the behaviour within the service area are controlled by control procedures. For each kind of control procedure (accept, schedule, dispatch, offer) certain standard disciplines are selectable in dependence on the component type under study. If necessary, actual parameters, i.e., speeds or population bounds, can be specified via an editor.

The control procedure window can be used in two situations. For a self defined component type the default control procedures can be overwritten by self defined control procedures. Their implementation is described in HI-SLANG within a text editor.

For a component the control procedure window is used to select the control procedures that should be used for this component.

Note that switching to another procedure causes a loss of the former specifications (i.e., parameters or self defined procedures).

### 4.4.1 Global Functions

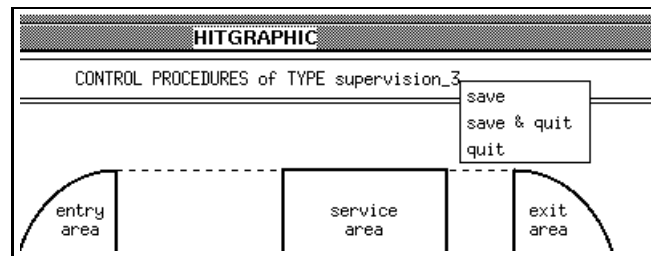


Figure 4.43: Menu on the Title of the Control Procedure Window

- *save*

saves the current choice of control procedures.

This save does not include the contents of HITGRAPHIC text editors opened from this window, i.e., self defined HI-SLANG procedures or parameter files must be saved before quitting the corresponding editor.

Note that the *save* operation causes the loss of former specifications (i.e., parameters or self defined procedures) if you have changed a procedure.

*Save* is not selectable, if subwindows are still open or if the component type is locked by another user or released.

- *save & quit*

saves the current choice of control procedures and quits the control procedure window.

This save does not include the contents of HITGRAPHIC text editors opened

from this window, i.e., self defined HI-SLANG procedures or parameter files must be saved before quitting the corresponding editor.

If any problems occur during the save operation, the quit operation will not be performed, but the window will be actualized according to the current state of the database.

*Save & quit* is not selectable, if subwindows are still open or if the component type is locked by another user or released.

- *quit*

quits the control procedure window.

If any changes were made since the last *save* operation, you are asked for confirmation.

*Quit* is not selectable, if text editors opened from this window are not closed.

## 4.4.2 Functions on Control Procedures for Component Types

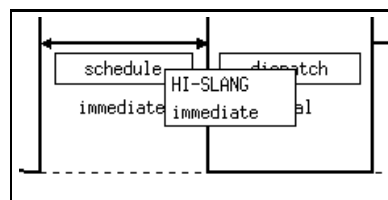


Figure 4.44: Menu on a Control Procedure

- *HI-SLANG*

switches to a self defined HI-SLANG control procedure as default for the type. Indicating the use of an own control procedure, the string “HI-SLANG” is displayed below the corresponding selection field.

To specify or change the control procedure within the HITGRAPHIC text editor you have to perform an *open* operation on the displayed *procedure* label. If you opened the editor for the first time a *save* is requested before the editor is displayed.

*Open* is not selectable, if the text editor is already open.

- *<type default>*

sets the default control procedure of the type to a standard default control procedure (in Figure 4.44: *immediate*).

The following table shows the standard default control procedures for the available component types. Self defined types have the same standard default control procedures as servers.

	accept	schedule	dispatch	offer
self defined type	always	immediate	equal	all
server	always	immediate	equal	all
prioserver	always	prionp	equal	all
ftserver	always	prionp	equal	all
counter	–	crandom	–	–
semaphor	–	–	–	–
tokenpool	–	–	–	–
synchsend	–	–	–	–
nowaitsend	–	–	–	–
observer	–	–	–	–

The name of the current default procedure is displayed below the corresponding selection field.

This entry is not selectable, if the component type is locked by another user or released.

## Specifying a Procedure

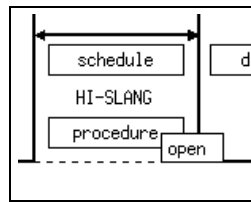


Figure 4.45: Menu on “procedure”

- *open*

opens the HITGRAPHIC text editor with three subwindows. In the subwindow on the right the control procedure is specified. The other subwindows display formal parameters and local declarations of the component type in read-only mode.

If a save operation for the control procedures is required before the *open* operation, you are informed about this by a dialog box and asked for confirmation. *Open* is not selectable, if this text editor is already open. The procedure is displayed in read-only mode, if the type is locked by another user or released or if file permissions are not sufficient for write mode.

### 4.4.3 Functions on Control Procedures for Components

The popup menus described in this section provide control procedures (accept, schedule, dispatch, offer), which can be attached to the component. To avoid repetitions, the common features of the menus are described now and therefore are omitted in the following sections:

- As the setting of control procedures for components depends on the corresponding type, only the valid entries are selectable in each menu.



- If the selected control procedure requires actual parameters, a parameter button appears. To enter or change the parameters within the editor you have to perform an *open* operation. If the editor is opened for the first time a *save* operation is requested before.
- *Type default* always stands for the procedure specified as default in the corresponding type (HI-SLANG or a standard default).
- All entries except *open* on the *parameters* label are not selectable, if the type is locked by another user or released.

The selected procedure is displayed below the selection field.

### Control Procedure “accept”

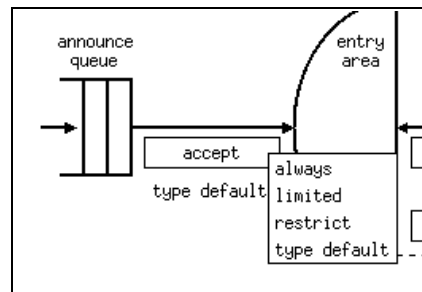


Figure 4.46: Menu on the Control Procedure “accept”

- *always*  
is valid for components of type “server”, “prio-server”, “ftserver”, and for components of a self defined or aggregated type.
- *limited*  
is valid for components of type “server”, “prio-server”, and for components of a self defined or aggregated type.
- *restrict*  
is valid for components of type “server”, “prio-server”, “ftserver”, and for components of an aggregated type.
- *type default*  
is valid for all components.

### Control Procedure “schedule”

- *fcfs*  
is valid for components of type “server” and for components of a self defined type.

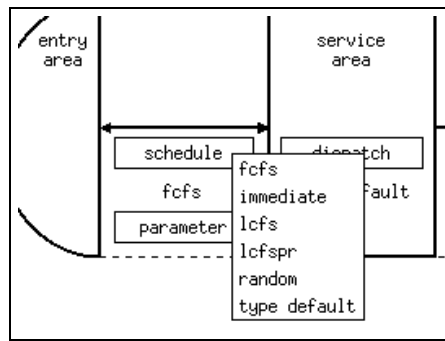


Figure 4.47: Menu on the Control Procedure “schedule”

- *immediate*  
is valid for components of type “server”, “prio-server”, and for components of a self defined type.
- *lcfs*  
is valid for components of type “server” and for components of a self defined type.
- *lcfspr*  
is valid for components of type “server” and for components of a self defined type.
- *random*  
is valid for components of type “server”, “prio-server”, “ftserver”, and for components of a self defined type.
- *crandom*  
is valid for components of type “counter”.
- *cprio*  
is valid for components of type “counter”.
- *priomp*  
is valid for components of type “prio-server” and “ftserver”.
- *prioprep*  
is valid for components of type “prio-server” and “ftserver”.
- *priopres*  
is valid for components of type “prio-server”.
- *type default*  
is valid for all components.

## Control Procedure “dispatch”

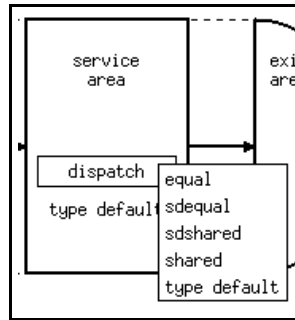


Figure 4.48: Menu on the Control Procedure “dispatch”

- *equal*  
is valid for components of type “server”, “prio-server”, “ftserver”, and for components of a self defined type.
- *sdequal*  
is valid for components of type “server”, “prio-server”, “ftserver”, and for components of a self defined type. f a self defined type.
- *sdshared*  
is valid for components of type “server”, “prio-server”, and for components of a self defined type.
- *shared*  
is valid for components of type “server”, “prio-server”, and for components of a self defined type.
- *type default*  
is valid for all components.

## Control Procedure “offer”

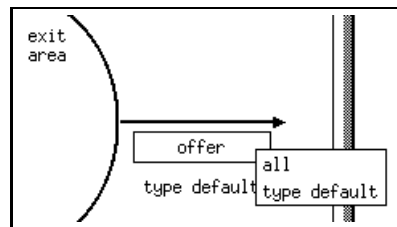


Figure 4.49: Menu on the Control Procedure “offer”

- *all*  
is valid for components of type “server”, “prio-server”, “ftserver”, and for components of a self defined or aggregated type.

- *type default*  
is valid for all components.

## Specifying Parameters

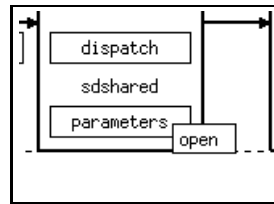


Figure 4.50: Menu on “parameters”

- *open*  
opens the HITGRAPHIC text editor with two subwindows to enter or change the actual parameters of the selected control procedure. In the upper part of the editor the formal parameters of the procedure are displayed in read-only mode.  
If a save operation for the control procedures is required before the *open* operation, you are informed about this by a dialog box and asked for confirmation. *Open* is not selectable, if this text editor is already open. The actual parameters are displayed in read-only mode, if the type is locked by another user or released or if file permissions are not sufficient for write mode.

## 4.5 The Editor Windows

As mentioned in the previous sections, some parts of HI-SLANG models are described in textual form. The HITGRAPHIC text editor may have several subwindows to show you different but related texts at the same time. Some subwindows display files in read-only mode, e.g., the formal parameters when editing the actual parameters.

These textual parts concern the following items:

- Global declarations of variables, constants, types for the model type (in HI-SLANG: those located outside the model type).
- Local declarations for model/component types, service types, procedures and experiments.
- Activities of model/component types, bodies of service types, procedures and experiments, i.e., statements to be executed at instantiation.
- Implementations of HI-SLANG control procedures.
- Formal parameters of model/component types, service types and procedures (and result parameters where appropriate).

- Actual parameters of components, predefined control procedures and evaluations.

Furthermore you can give some comments (information) about model/component types, service types, aggregation descriptions, evaluations and experiments. Files resulting from a HIT run (e.g., listing or table) are only readable.

The following list contains all menu items that cause an editor window for HISLANG to appear. If the editor consists of multiple subwindows, they are also listed. (Sub)windows that cannot be changed are marked by “(ro)” which stands for read-only mode.

- Environment window:
  - model type *information*
  - component type *information*
  - aggregation
    - \* *information*
    - \* *show protocol*: standard output (ro), listing (ro)
    - \* *show aggregate* (ro)
  - evaluation *information*
  - experiment
    - \* *information*
    - \* *show protocol*: standard output (ro), listing (ro)
    - \* *show table* (ro)
    - \* *show dumpfile* (ro)
    - \* *show trace* (ro)
- Type graphic window:
  - *formal parameters*
  - model type *declarations*: globals, locals
  - component type *locals*
  - service/procedure
    - \* *information*
    - \* *formal parameters*: parameters, results
    - \* *open*: parameters (ro), results (ro), locals, body
  - used service/procedure *formal parameters*: parameters, results
  - model type activities *open*: formal parameters (ro), globals (ro), locals (ro), activities
  - component type activities *open*: formal parameters (ro), locals (ro), activities
  - provided service/procedure at component *show formal parameters*: parameters (ro), results (ro)

- component *actual parameters*: formal parameters (ro), actual parameters
- Control procedure window:
  - procedure *open*: formal parameters of the component type (ro), locals of the component type (ro), HI-SLANG control procedure
  - parameters *open*: formal parameters (ro), actual parameters
- Evaluation object window:
  - frequency interval *open* (on shaded label)
- Experiment window:
  - *open*: control part, locals, body
  - *actual parameters* on evaluation: formal parameters of the model type (ro), actual parameters for the model

### 4.5.1 HI-SLANG in Editor Windows

In this section we give examples for texts you are expected to fill in. We suppose you are fairly familiar with the programming kernel of HI-SLANG, so we do not explain HI-SLANG syntax.

#### Information

You should use this possibility to comment your modelling objects wherever the *information* entry is available. The entered texts will be included in the generated code as comments (with a “% ” at the beginning of the line inserted by HITGRAPHIC), so you need not worry about any syntactical restrictions.

Example (Information of Standard Component Type “server”)

```
The component type "server" is used to model time consumption.
It provides the service "request" which has a real parameter
"amount". The parameter expresses the amount of work to be
done by the server.
```

```
Of course, the service time depends on the speed of the
server, described in the dispatch procedure selected for the
component. By default a component of type "server" models an
"infinite server", i.e., scheduling is "immediate" and
dispatching is "equal" with speed 1.
```

```
In HI-SLANG notation the component type "server" has the
following structure:
```

```

TYPE server COMPONENT;

    PROVIDE
        SERVICE request (amount : REAL);
    END PROVIDE;
    ...
END TYPE server;

```

## Formal Parameters

Here you can declare formal parameters for model and component types and for services and procedures. The formal parameters are everything between the parentheses “(” and “)”. There are no further restrictions.

{HI-SLANG syntax: non-terminal formal\_par without the parentheses [, ...]}

Example

```

no_of_users, access_limit : INTEGER; clock_rate : REAL

```

## Result Parameters

If you give result parameters in the result subwindow, the keyword “RESULT” will be generated followed by your text.

{HI-SLANG syntax: simple\_type [, ...]}

Example

```

POINTER FOR pid, BOOLEAN

```

## Actual Parameters

In giving actual parameters, you are as free as you know it from HI-SLANG: You can give all parameters in their order, you can rearrange them by “LET ...”, and you can omit some parameters if you have given a default in the formal parameters. The text you edit will be inserted between the generated parentheses.

{HI-SLANG syntax: non-terminal act\_par without the parentheses }

Example

```

1, LET speed := 32

```

## Globals

The editor where you can define the global declarations (in short form called globals) is displayed together with the editor for the locals if you select the declarations entry

within the menu on the title of the type graphic window. Globals are defined for model types only.

Here you can edit declarations that should be global for all component types of this model type, even for the model type itself. Typical applications are declarations of record types, which are allowed only here. Please, note the danger of name conflicts in the globals when there are multiple EVALUATE statements in an experiment.

Example

```
TYPE pid RECORD (proctype : TEXT; procid : INTEGER);
VARIABLE
  active      : BOOLEAN DEFAULT TRUE;
  pred, succ : POINTER FOR pid DEFAULT NONE;
END TYPE pid;

CONSTANT speed_of_medium : REAL DEFAULT 200E6; {meters per second}
```

## Locals

This editor serves for the local declarations of a modelling object, i.e., variables, procedures, etc. You can insert any declarations here, in the generated code this will be the text before the “BEGIN” of the modelling object.

Example

```
VARIABLE
  number : INTEGER;
  duration : REAL;

PROCEDURE hour RESULT INTEGER;
BEGIN
  RESULT time/3600; {here: time has second as unit}
END PROCEDURE hour;
```

## Body of a Service Respectively Procedure

The body editor for a service respectively procedure serves for the specification of statements to be executed when the service or procedure is called. It contains the specification of the behaviour pattern of the service respectively the specification of statements to be executed sequentially and without model time consumption in case of a procedure.



Example

```
BRANCH
  PROB 0.6: hier_modelling (negexp (1.0/(1*60*60.0)));
  PROB 0.3: loss_modelling (negexp (1.0/(19*60.0)));
  PROB 0.1: AVERAGE 3 TIMES LOOP
            detailed_simul (negexp (1.0/(18*60*60.0)));
            END LOOP;
END BRANCH;
IF draw (0.15) THEN
  phone_partner (negexp (1.0/(15*60.0)));
END IF;
LOOP
  write_section (negexp (1.0/(26*60.0)));
END LOOP until draw (0.95);
```

### Activities of a Model or Component Type

This editor contains the specification of CREATE statements for the services within the model respectively component type or other initialisations. The following is an example for the creation of a service named “batch”.

Example

```
CREATE 1 PROCESS batch EVERY negexp(1/100);
```

### HI-SLANG Control Procedure

When using self defined control procedures, you should be familiar with HI-SLANG syntax. In case of a *schedule* procedure the “CONTROL PROCEDURE schedule;” and “END PROCEDURE;” will be generated by HITGRAPHIC, so you only have to take care for the local declarations, the “BEGIN” and the statements (ending with “;”). The last-come-first-scheduled preemptive resume schedule procedure is given as an example.

Example

```
VARIABLE found : BOOLEAN DEFAULT FALSE;

BEGIN

    INSPECT ENTRY_AREA WHILE NOT found LOOP REVERSE
        SELECT;
        found := TRUE;
    END LOOP;

    IF found THEN
        INSPECT SERVICE_AREA LOOP
            SELECT;
        END LOOP;
    END IF;

```

## Frequency Interval

If the estimator `FREQUENCY INTERVAL` is selected within the evaluation object window, the intervals can be entered in this editor window. The estimator `FREQUENCY INTERVAL` can only be applied to event streams (e.g., `TURNAROUND-TIME`). For each interval, you will get the number of updates on the stream that fall into the interval. The lower bound is included, the upper bound excluded.

Example

```
20 .. 23.3, 23.3 .. 26.6, 26.6 .. 30
```

## Control Part of Experiment

In the control part you have the possibility to specify some `HI-SLANG` control statements used by the `HI-SLANG` compiler.

Do not forget the “`%END`” at the end of the specification.

Example

```
%PARM = UPDATES
%PARM = INDENT=| 1
%END
```

## Body of Experiment

The instruction part of an experiment is given in the body where you can specify the course of experiment execution.

Execution of evaluations are requested by the EVALUATE statement. It is an abbreviation of the EVALUATE statement in HI-SLANG. Here, it consists of the keyword EVALUATE followed by the name of an evaluation without any parameters and finished by a semicolon. The EVALUATE statement must be separated in a single line. Actual parameters are specified in the experiment window. For compatibility reasons an old form of the EVALUATE statement with the evaluation name in embedding “\$” symbols is also still supported.

Example

```
FOR no := 1 STEP 1 UNTIL 3
  LOOP
    EVALUATE bibsystem_eva;
  END LOOP;

no := 10;
EVALUATE bibsystem_eva;
```

## Other Aspects of the HI-SLANG Code

In the generated code, there is automatic indentation (by steps of 3), so you may disable indentation for the listing written by the compiler. Your edited texts have an initial indentation with respect to the surrounding code, but further indentation within your text will not be done, so you should do it when editing (for better readability).

When using HI-SLANG keywords, you should use upper-case letters to conform with the generated part of the code, but as HI-SLANG is case insensitive, it will not result in an error if you use your own upper-/lower-case rules.

Information texts are inserted in the generated code as comment lines.

The names of hierarchies, except the predefined hierarchy “all”, are extended in the generated code by the concatenation of an underline and the name of the evaluation object, where the hierarchy is used. The extended names are visible in the result output files.

### 4.5.2 Structure of the Editor Window

The HITGRAPHIC text editor provides the user with a variable number of editor areas which can be edited “in parallel”. In case that there are more editor areas than just one, they are placed side by side and/or one beneath the other. This leads to a two dimensional structure which consists of columns and rows of editor areas.

The window of the HITGRAPHIC text editor consists of the head label area and the editor’s pane. In the head label area a text string is displayed which contains information about the logical relationships of the files which can be edited in the editor’s pane. In the example depicted by Figure 4.51 the string “SERVICE – report”



Figure 4.51: An Example Structure of the HITGRAPHIC Text Editor

is displayed in the head label area. The editor's pane is located below the head label area and contains one editor area for each file to be edited or shown.

Each editor area consists of an editor label area and an editing area which is located below the editor label area. In the editor label area a text string is displayed containing information about the file which can be edited in the corresponding editing area. In the editor label areas of the example of Figure 4.51 the files "PARAMETERS", "RESULTS", "LOCALS" and "BODY" are displayed. The value behind the label indicates the line number of the current cursor position. The text in the editing areas can be editable (i.e., read/write) or read-only. If the editing mode of an editing area is read-only, any attempt to modify the text will cause a beep.

### 4.5.3 Global Functions on the Head Label Area

Depending on showing just one or multiple files the provided functions of the text editor vary, confirm Figures 4.52 and 4.53.

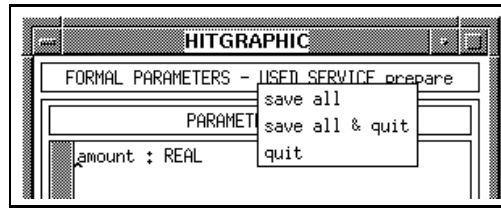


Figure 4.52: Menu on the Title of the Editor Window with Multiple Files

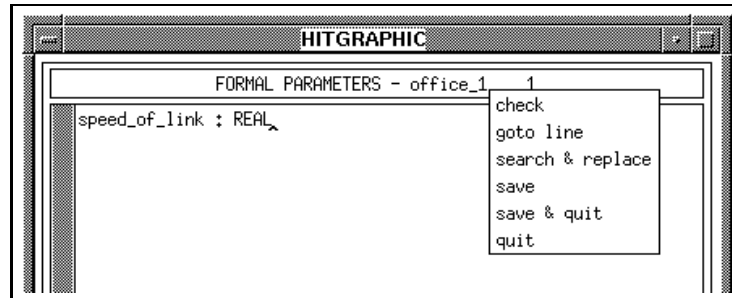


Figure 4.53: Menu on the Title of the Editor Window with a Single File

- *check*

performs a local check (of syntactical and semantical conditions) of the HI-SLANG part based on the currently displayed version, which may differ from the last saved version, and displays locally unknown identifiers or error messages in a check message window.

This function is only available in case of a single file editor window. If there is more than one editor area, it is provided from the menu on the editor label area. For a detailed explanation of the function please confirm Section 4.5.6. The check function is not provided in context with information files or result files from an experiment run.

- *goto line*

pops up a request box to enter the desired line number.

This function is only available in case of a single file editor window. If there is more than one editor area, it is provided from the menu on the editor label area. For a detailed explanation of the function please confirm Section 4.5.6.

- *search & replace*

causes a popup menu to appear which enables search and replace operations on the text.

This function is only available in case of a one dimensional editor window. If there is more than one editor area, it is provided from the menu on the editor label area. For a detailed explanation of the function please confirm Section 4.5.6.

- *save all* respectively *save*  
saves the contents of all editing areas which contain unsaved changes respectively the contents of the only area.  
If there are no unsaved changes, this function has no effect.
- *save all & quit* respectively *save & quit*  
saves the contents of all editing areas which contain unsaved changes respectively the contents of the only area, and then closes the editor window.
- *quit*  
exits the HITGRAPHIC text editor.  
If any changes were made since the last *save* operation, you are asked for confirmation.

#### 4.5.4 Global Functions on the Editor's Pane

A special feature of the HITGRAPHIC text editor are the grips which allow to resize the editor areas within the bounds of the editors pane in case there are more than one editor area in the editor's pane. If there is only one editor area in the editor's pane the whole HITGRAPHIC text editor window has to be resized in order to resize the editor area.

Grips are depicted as small rectangular regions on the border lines between columns and rows of editor areas. Grips between columns are called horizontal grips because they enable to move the border separating the columns horizontally. Correspondingly, the grips between rows are called vertical grips. In the example of Figure 4.51 there are one horizontal grip and two vertical grips.

When the cursor touches a horizontal grip, the cursor shape switches into a horizontal double arrow. In the same way, the cursor switches into a vertical double arrow over a vertical grip. Then the corresponding border line can be moved by pressing and holding down a mouse button, shifting the cursor to the desired new position and releasing the mouse button. Immediately after having released the button, the columns and rows of the editor's pane will resize.

#### 4.5.5 Text Selection

Some operations of the HITGRAPHIC text editor are designed to be applied on text blocks. These text blocks which we refer to as "selected text", are displayed invertedly during selection.

To select a text, three methods are proposed.

1. The simplest method of selecting text is to press the left mouse button at the start/end of the block, drag the cursor to the end/start of the block and then release the mouse button. This method only works on text actually displayed in an editor area.

2. Another method is to press and release the left mouse button at the start/end of the block, move the cursor to the end/start of the block and there press and release the right mouse button. This method allows to select text blocks of arbitrary length.
3. Some kind of canonical text blocks can be selected by multiple fast clicks with the left mouse button according to the following.
  - Two clicks  
select the word under the cursor. A word boundary is defined as a Space, Tab or Carriage Return.
  - Three clicks  
select the line under the cursor.
  - Four clicks  
select the paragraph under the cursor. A paragraph boundary is defined as two Carriage Returns in a row with only Spaces or Tabs between them.
  - Five clicks  
select the entire contents of the editing area.

A text selection which is currently displayed invertedly can be modified by clicking the right mouse button near either the end or the start of the selection to adjust. This end of the text selection may be moved while holding down the right mouse button until the button is released.

A main application for selected text blocks is to paste them into other editing areas. After having selected the desired text block, you move the cursor into the other editing area. There you click the left mouse button at the destination point. Then you click the middle mouse button. This causes the selected text to be inserted at the desired position.

The selected text is buffered until the next text selection is made, i.e., even if the text block is not longer displayed invertedly, the selection can be pasted to other locations.

#### 4.5.6 Global Functions on the Editor Label Area

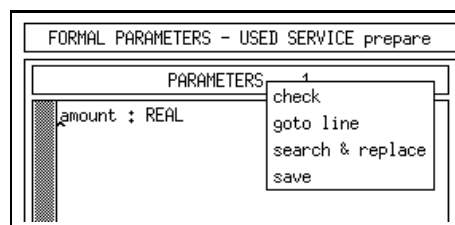


Figure 4.54: Menu on the Title of the Editor Area

- *check*

performs a local check (of syntactical and semantical conditions) of the HHSLANG part based on the currently displayed version, which may differ from the last saved version, and displays locally unknown identifiers or error messages in a check message window.

During the check operation the menu remains popped up. If everything is ok the menu will pop down. Otherwise a check message window is displayed showing errors and/or warnings and, in case of syntactical correctness, the locally unknown identifiers which can be compared with the declarations made in corresponding editors.

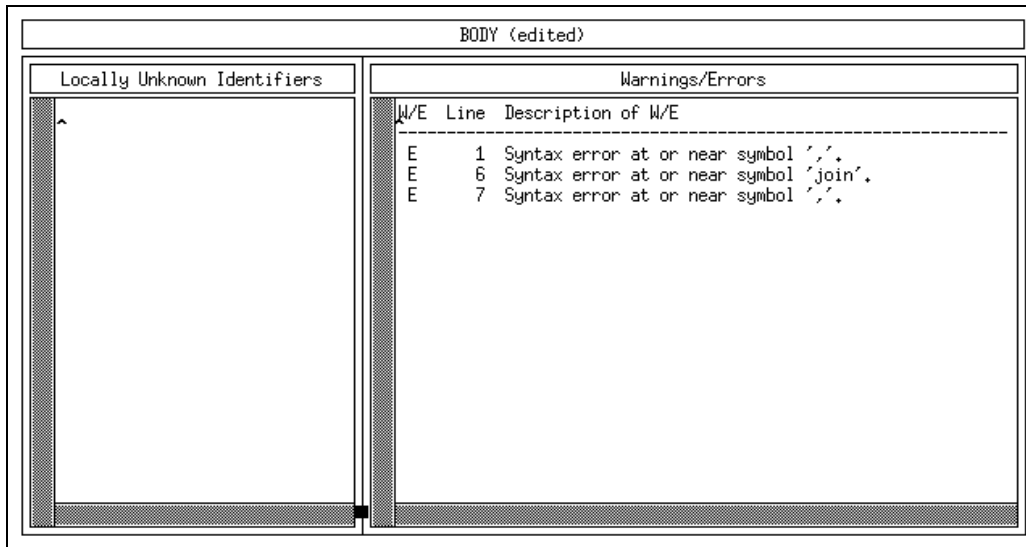


Figure 4.55: The Check Message Window

The displayed line numbers indicate the occurrence of locally unknown identifiers or errors. Selecting a line number (e.g., by a double click) automatically places the cursor within the corresponding editing area at the beginning of this line. The placement is only done if the selection has changed.

The check message window provides a *quit* menu, but it is automatically closed if a new *check* operation is started within the same editing area or if the editor is closed itself.

The *check* function is not provided in context of windows showing files resulting from a HIT run.

- *goto line*

pops up a request box to enter the desired line number.

The cursor is placed at the beginning of the line with the entered number. The editing area is scrolled so that the actual line is placed in the middle of the visible area.

- *search* respectively *search & replace*

pops up a popup menu which enables search and replace operations on the text in the corresponding editing area.

If an editing area is in write mode this function is labelled *search & replace* while



it is labelled *search* in case of a read-only area, where no replace operations are permitted.

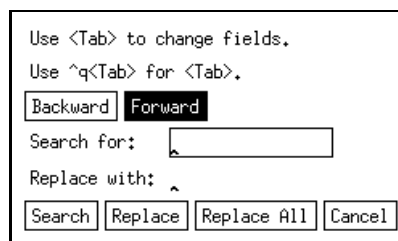


Figure 4.56: The Search & Replace Box

At the top of the search popup there are two buttons labelled *Backward* and *Forward*, which indicate the direction in which the search will be performed. The *Forward* button will be highlighted by default, but the desired direction can be changed by clicking on the appropriate button.

Below these buttons there are two text areas, one labelled *Search for* and the other labelled *Replace with*. If the text of the editing area is displayed in read-only mode, the replace field is insensitive and no replacements are allowed.

To the right of each of these labels there is a text field which allows to enter a string to search for and a string to replace with. Only one of these text fields will have a border around it. This field is the active field. Any key presses that occur in the search popup are directed to the active field. There are a few key sequences with special meaning to the active field.

- Carriage Return  
executes the action and removes the popup from the screen.
- Tab  
executes the search operation, if the search field is active, and moves to the next field.
- Shift Carriage Return  
executes the search operation, if the search field is active, and moves to the next field.
- Control-q Tab  
inserts a Tab into the active field.
- Control-c  
removes the popup from the screen.

At the bottom of the search popup there is a row of buttons. The use of these buttons performs the following actions.

- *Search*  
searches for the string in the search field. The popup remains on the screen.
- *Replace*  
replaces the currently highlighted string with the string in the replace field,

and moves to the next occurrence of the search string in the editing area. The popup remains on the screen.

– *Replace All*

replaces all occurrences of the search string with the replace string from the current position of the text cursor to the end (or beginning) of the file. The popup remains on the screen.

– *Cancel*

removes the search popup from the screen.

By clicking the middle mouse button, selected text blocks can be pasted into the search and replace fields. Within the fields only the last line of the block is visible.

● *save*

saves the contents of the corresponding editing area.

If the text of the corresponding editing area is displayed in read-only mode, this function is insensitive. If there are no unsaved changes the *save* function has no effect.

When the first editing operation in the editing area of an editor area is performed, the string (*edited*) is appended to the string displayed in the corresponding editor label area. A *save* operation, either from the editor label popup or from the head label popup then deletes this appendix from the displayed string.

## 4.5.7 Functions on the Editing Area

Most of the editing actions described in this section refer to the position of the text cursor within an editing area. This position will be called the insertion point.

Depending on the number of text columns displayed in an editing area not all lines of a text may be shown completely. If the text of a file in one line is wider than the editing area, a small rectangle is painted in the right margin of the editing area to indicate that the line is too long. The tail of the line can be made visible by adjusting the corresponding horizontal scrollbar.

### Scrollbars

Every editing area permanently has two scrollbars: a vertical scrollbar at the left hand side and a horizontal scrollbar at the bottom.

The background of a scrollbar is called the scroll region. It represents the length of the area that users can scroll. The dark area is the slider representing a window through which a user looks at the displayed data. So the slider marks the location of a user's viewport in relation to the total scrollable area. The slider also indicates the relation between the visible part of the file and its invisible part. The larger the slider, the larger is the visible part of the file in comparison to the invisible part.

The user's viewport can be modified by changing the position of the slider. A click with the left mouse button on the scroll region moves the viewpoint downwards in case of a vertical scrollbar and to the right in case of a horizontal one. A click with the right mouse button moves the viewport upwards respectively to the left. The range of the movement depends on the cursor position within the scroll region.

Pressing and holding down the middle mouse button on the scrollbar enables to slide to an arbitrary position.

## File Insertion

Typing the key sequence Meta-i in an editing area activates the file insert popup which appears under the cursor. Any text typed while the cursor is in this popup is directed to the text field used for the filename. When the desired filename has been entered, a click on the *Insert File* button or a Carriage Return will cause the file to be inserted at the position of the insertion point.

If an error occurs while opening the file, an error message will be displayed, prompting to enter the filename again. The file insertion operation may be aborted by clicking on the *Cancel* button. If Meta-i is typed at an editing area in read-only mode, the bell will beep, as no file insertion is allowed.

By clicking the middle mouse button, selected text blocks can be pasted into the text field.

## Editing Actions

The following keystroke combinations are defined to cause editing actions.

Ctrl-a	sets the insertion point to the beginning of the line
Ctrl-b	sets the insertion point one character backward
Ctrl-d	deletes the next character
Ctrl-e	sets the insertion point to the end of the line
Ctrl-f	sets the insertion point one character forward
Ctrl-g	resets the keystroke multiplication factor to one after a Ctrl-u (i.e., cancels Ctrl-u)
Ctrl-h	deletes the previous character
Ctrl-j	inserts a newline and indents the new line to the line above at the next keystroke
Ctrl-k	kills all characters to the end of the line
Ctrl-l	redraws the display
Ctrl-m	inserts a newline; insertion point is the beginning of the new line
Ctrl-n	moves the insertion point one line down
Ctrl-o	inserts a newline; insertion point is the end of the old line
Ctrl-p	moves the insertion point one line up
Ctrl-r	activates the search/replace popup for backward search
Ctrl-s	activates the search/replace popup for forward search

Ctrl-t	exchanges the positions of characters left and right of the insertion point and sets the insertion point one character forward
Ctrl-u	the next keystroke will be executed four times
Ctrl-v	moves the insertion point one page down (a page is defined by the number of lines currently visible in the editing area)
Ctrl-w	kills currently selected (i.e., invertedly displayed) text
Ctrl-y	undo last kill command (e.g., Ctrl-k or Ctrl-w)
Ctrl-z	scrolls the visible part of the file one line down
Meta-b	sets the insertion point one word backward
Meta-d	deletes the word right from the insertion point
Meta-D	kills the word right from the insertion point
Meta-f	sets the insertion point one word forward
Meta-h	deletes the word left from the insertion point
Meta-H	kills the word left from the insertion point
Meta-i	activates the file insertion menu
Meta-k	kills the rest of the paragraph
Meta-q	formats the current paragraph in a way that its lines first are joined together and then split into new lines which fit into the current width of the editing area
Meta-v	moves the insertion point one page up (a page is defined by the number of lines currently visible in the editing area)
Meta-y	inserts the currently selected text at the insertion point
Meta-z	scrolls the visible part of the file one line down
Meta-<	sets the insertion point to the beginning of the file
Meta->	sets the insertion point to the end of the file
Meta-[	sets the insertion point to the beginning of the current paragraph
Meta-]	sets the insertion point to the end of the current paragraph
Meta-Delete	deletes the word left from the insertion point
Meta-Shift Delete	kills the word left from the insertion point
Meta-Backspace	deletes the word left from the insertion point
Meta-Shift Backspace	kills the word left from the insertion point

## 4.6 The Aggregation Description Window

The aggregation description window is used, when an automated aggregation of a component type is desired.

The aggregation of component types is performed to reduce the complexity of models and their analysis. This goal is reached by generating a substitute representation of the original component type.

The technique is only applicable on component types, that fulfill certain properties, e.g., they must have neither parameters nor parameterized provided services. There should be at least one provided service and provided procedures are not allowed.

The result of the aggregation, henceforth called the **aggregate**, is a simple load-dependent server, that

- provides the same interface (provided services) and
- behaves similarly to the original component type in terms of quantitative aspects.

To generate a load-dependent server, it is necessary to specify upper bounds for the population of each provided service. The aggregation description window allows the user to enter these bounds.

### 4.6.1 Global Functions

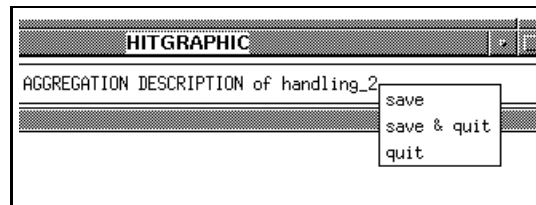


Figure 4.57: Menu on the Title of the Aggregation Description Window

- *save*  
saves the current population values of the aggregation description.  
If the values are correct, i.e., a non-negative integer value was entered for each population, the *save* operation is executed (and the corresponding database is modified). For the duration of this operation the popup menu remains visible on the display. The contents of the aggregation description window is redrawn, when the operation is finished. At this point the actual state of the aggregation description will be displayed. If the number or sequence of the provided services is different now, the corresponding component type must have been changed meanwhile.  
*Save* is not selectable, if the aggregation description is locked by another user or released.  
An error message is displayed, if the population values are not correct.
- *save & quit*  
saves the current population values of the aggregation description in the same way as the *save* operation does, and quits the aggregation description window after a successful save.  
*Save & quit* is not selectable, if the aggregation description is locked by another user or released.
- *quit*  
quits the aggregation description window.  
If any changes were made since the last *save* operation, you are asked for confirmation.

## 4.6.2 Specification of Population

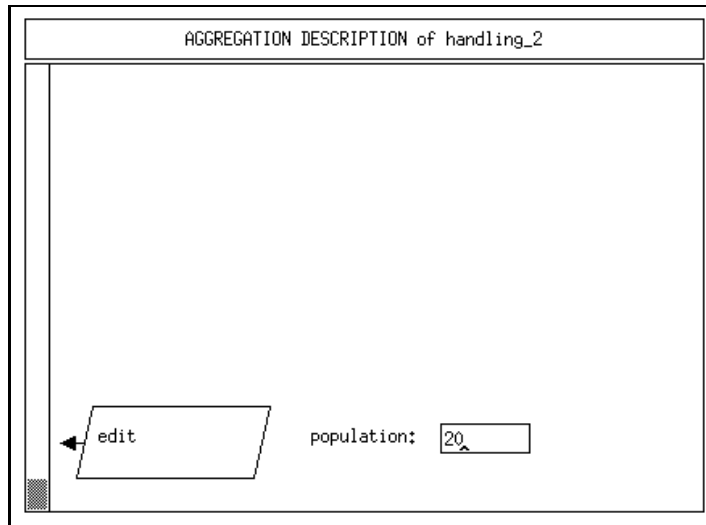


Figure 4.58: Aggregation Description Window

As shown by Figure 4.58, the graphic area of the aggregation description window displays the provided services of the component type as done within the component type graphic window.

For each provided service, an associated text box on the right hand allows for the specification of the population bound. The specified values are syntactically checked, when the *save* or *save & quit* operation on the title bar is performed.

## 4.7 The Evaluation Window

The evaluation window serves as the basis for evaluation specification. Its graphics consist of the object structure and some additional information, which will be explained later in this section.

All specifications done in this window are somehow independent of the analysis technique, which is applied via HIT. Some specifications, which are only suitable for simulative analysis, will be ignored and not be generated (during the transformation step) for analytical experiments.

Since evaluation specification is only sensible for completed model type specifications, an error occurs if the model type includes enclosed components without main declarations. In this case, an error message is displayed and the evaluation window is opened in read-only mode.

The example in Figure 4.59 is used to discuss the evaluation window.

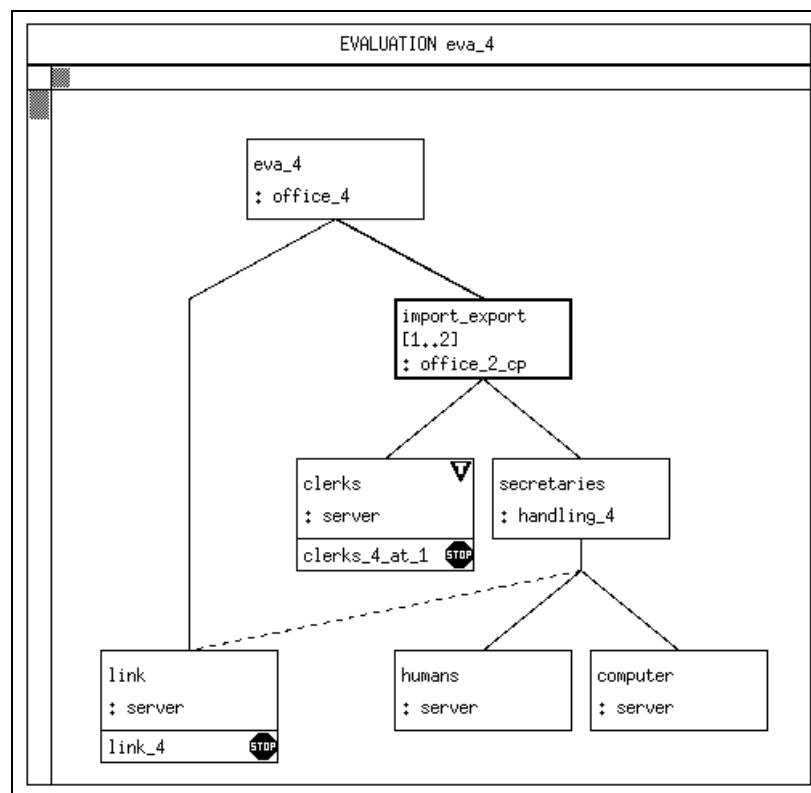


Figure 4.59: Evaluation Window

The first point to be addressed is the concept of **evaluation objects**. An evaluation object determines a location within a model, where measurements should be performed. The evaluation window allows for their creation and further specification. Evaluation objects are displayed as rectangular boxes underneath the associated component.

Figure 4.59 displays two evaluation objects: “clerks\_4\_at\_1” at the component “clerks” (type “server”) and “link\_4” at the component “link” (type “server”).

A problem arises during the creation of evaluation objects, if a component array is

located on the path from the model down to the associated component, following the instantiating declarations (see “clerks\_4\_at\_1”). This path is called component declaration path. To distinguish the subcomponents of a component array in a component declaration path the specification of an identifying array index is required. If the associated component is a component array itself, you can select one component by a single index or a set of components by an index range. For each component of the set the same kind of measurement, as described in the evaluation object, is performed separately.

Additionally, the evaluation window allows for creation of a simulation stop control at an evaluation object, which determines when a simulative analysis should stop. These stop controls are displayed by usual stop symbols, which are borrowed from the corresponding traffic sign. Multiple simulation stop conditions are connected by a “logical or” operation to build the overall simulation stop condition of the evaluation.

Finally, the evaluation window allows to specify that certain components should be traced during the simulation. This is indicated by a triangle (see component “clerks” of type “server”).

### 4.7.1 Global Functions

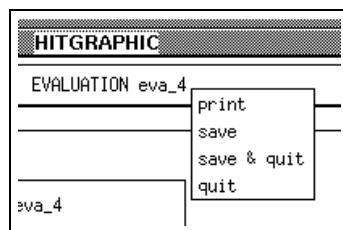


Figure 4.60: Menu on the Title of the Evaluation Window

- *print*

prints the evaluation window.

An input box requesting the settings for the PostScript file appears (cf. Section 5.3).

- *save*

saves the current evaluation.

This *save* does not include the contents of other windows, which provide a save, i.e., evaluation object, hierarchy survey, hierarchy and start/stop control windows. For the duration of this operation the popup menu remains visible on the display. The evaluation window is redrawn, when the operation is finished. At this point you see the actual state of the evaluation, i.e., how it is stored in the database. If the object structure looks different now, component types or the corresponding model type must have been changed in the meantime.

*Save* is not selectable, if the evaluation is locked by another user or released. In order to avoid inconsistencies, *save* is also not selectable, if subwindows of this evaluation window are open.



- *save & quit*

saves the current evaluation in the same way as *save* does. If the saving is successful, the evaluation window is closed. Otherwise the actual information of the database will be retrieved and displayed.

*Save & quit* is not selectable, if the evaluation is locked by another user or released or if subwindows of this evaluation window are open.

- *quit*

quits the evaluation window.

If any changes were made since the last *save* operation, you are asked for confirmation.

*Quit* is not selectable, if a window started from this evaluation window is still open.

## 4.7.2 Specification of the Evaluation

### Functions on the Model respectively on Components

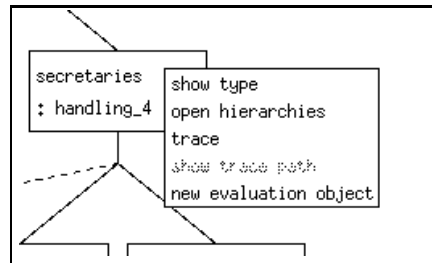


Figure 4.61: Menu on Models and Components

- *show type*

shows the corresponding model respectively component type.

*Show type* opens the type graphic window corresponding to the type of the selected model or component in read-only mode.

*Show type* is not selectable, if the type graphic was opened before from this entry and is still open.

- *open hierarchies*

opens the window with the list of hierarchies belonging to the selected model or component.

*Open hierarchies* is not selectable, if the window with the list of hierarchies is already open.

A modification of the hierarchy list is not possible if the evaluation is locked by another user or released.

- *trace/no trace*

sets the trace option respectively resets the trace option for the selected component.

If the trace option is set, a trace file will be generated during a simulative HIT run. *Trace* will be ignored for analytical evaluations.

A trace flag marks the selected object, if you switch to *trace*. Otherwise the trace flag is removed.

If the *trace* option is set, along the component declaration path component arrays will be searched. For every component array an index will be asked for to set the trace for the right components. If the associated component is a component array, you can set the *trace* option for one component by one index or for multiple components of the array by an index range.

The switch is not selectable, if the evaluation is locked by another user or released.

- *show trace path*

opens a popup displaying the component declaration path (possibly with array indices) to the corresponding object (cf. *trace/no trace*).

This entry is only selectable, if the trace option is set.

- *new evaluation object*

creates a new evaluation object.

A dialog box is displayed, requesting the name for the new evaluation object. After giving a valid name and clicking *OK* with the left mouse button, component arrays will be searched along the component declaration path from the object to the model. For every component array an index will be asked for. If the associated component itself is a component array, you can attach the evaluation object to one or multiple components of the array by specifying an index or an index range.

The new evaluation object is appended underneath the already existing evaluation objects of the selected component or model.

An error message is displayed, if the name of the evaluation object already exists as an evaluation object name within the current evaluation.

*New evaluation object* is not selectable, if the evaluation is locked by another user or released.

## Index or Range Specification

Enter index of component import\_export [1..2] : office\_2\_cp:

Index  Lower bound

Upper bound

Figure 4.62: Box to Select an Index or an Index Range

Associated components of component arrays are selected by a dedicated input box. A single component is specified by the index in the index field, a set of components by a range of indices given in a lower and upper bound field. At a time, only the index or the range variant is active. You can change the variant with the *Switch to*

*range* respectively the *Switch to index* button. The *OK* and *Cancel* button operate in a similar way as for standard input boxes.

## Functions on Evaluation Objects

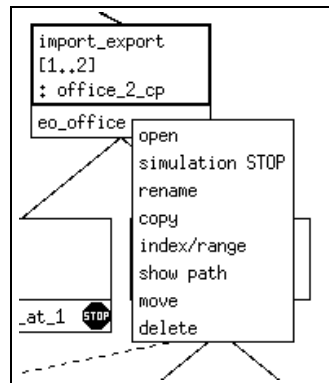


Figure 4.63: Menu on Evaluation Objects

- *open*

opens the corresponding evaluation object window.

If a new evaluation object should be opened, a dialog box is displayed asking for a *save* before performing the *open* operation.

*Open* is not selectable, if the current evaluation object window is already open. No modifications are possible in the evaluation object window if the evaluation is locked by another user or released.

- *simulation STOP*

offers the possibility to specify simulation stop conditions for the experiment run which concern the selected evaluation object.

To indicate the fact that the evaluation object is now provided with simulation stop conditions, it is marked with a *STOP* symbol. These stop conditions are only significant in simulative experiments. If an analytical method is selected, they will be ignored.

This entry is not selectable, if the selected evaluation object is already provided with a simulation stop control or if the current evaluation is locked by another user or released.

- *rename*

allows you to rename the selected evaluation object.

A dialog box is displayed, requesting the new name of the evaluation object. After entering a correct name and confirming it the old name is replaced by the new name in the evaluation window.

An error message is displayed, if the new name already exists as an evaluation object name within the current evaluation.

*Rename* is not selectable, if the corresponding evaluation object window is open or if the current evaluation is locked by another user or released.

- *copy*

copies the selected evaluation object.

A dialog box is displayed, requesting the name for the copy. After entering a correct name and confirming it the new evaluation object is created and appended underneath the already existing evaluation objects. Note that if you copy an array element, the copy has the same array index as the selected evaluation object. As a consequence, you cannot copy one array element into another array element. If some start and/or stop control statements are specified for the selected evaluation object, they are copied, too.

An error message is displayed, if the entered name already exists as an evaluation object name within the current evaluation.

*Copy* is not selectable, if the corresponding evaluation object window is open or if the current evaluation is locked by another user or released.

- *index/range*

opens the dedicated input box for index or range specification. This item is only selectable, if the evaluation object is attached to a component array. The association of the evaluation object to components of the array can be changed.

*Index/range* is not selectable, if the current evaluation is locked by another user or released.

- *show path*

opens a popup displaying the component declaration path (possibly with array indices) to the corresponding object.

- *move*

allows to change the position of an evaluation object in the list of evaluation objects at a component. After selecting the *move* operation, the new position is determined by a mouse click with the left button on the box, below that the evaluation object should be moved. A message box may inform you about illegal positions.

*Move* is not selectable, if the corresponding evaluation object window is open or if the current evaluation is locked by another user or released.

- *delete*

deletes the selected evaluation object.

A dialog box is displayed, asking for the confirmation to delete the selected evaluation object. After clicking *OK* with the left mouse button, the evaluation object including its start and stop conditions is deleted. Clicking *Cancel* aborts the *delete* operation.

*Delete* is not selectable, if the corresponding evaluation object window is open or if the current evaluation is locked by another user or released.

## Functions on a STOP Symbol

- *open*

opens the corresponding stop window to specify simulation stop conditions.

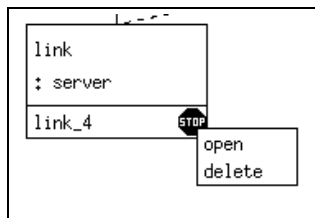


Figure 4.64: Menu on the STOP Symbol

If the stop window is opened for the first time, a dialog box is displayed asking for a *save* before performing the *open* operation.

*Open* is not selectable, if this stop window is already open.

If the corresponding evaluation is locked by another user or released, no modifications are possible in the stop window.

- *delete*

deletes the simulation stop conditions of the corresponding evaluation object.

A dialog box is displayed, asking for confirmation to delete the simulation stop conditions. After clicking *OK* with the left mouse button, the STOP symbol is removed and the simulation stop conditions are deleted. Clicking *Cancel* aborts the *delete* operation.

*Delete* is not selectable, if the stop window is open or if the current evaluation is locked by another user or released.

## 4.8 The Evaluation Object Window

The evaluation object window serves for the specification of measurements that are to be performed at the evaluation object.

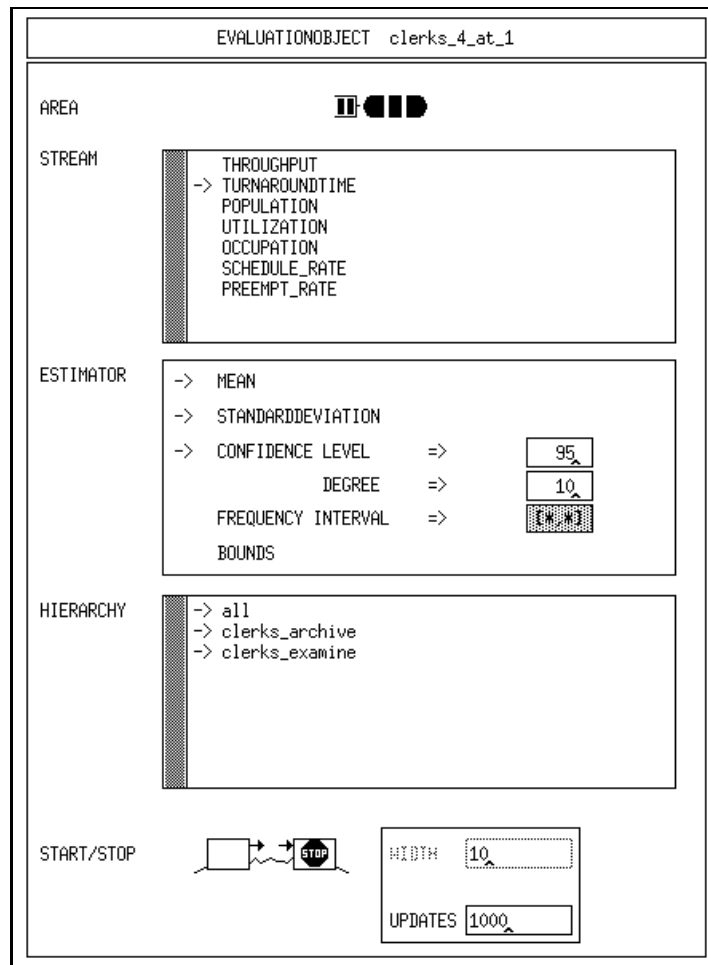


Figure 4.65: Evaluation Object Window

It is divided into five specification areas, which will be discussed now with the help of Figure 4.65:

- The upper symbol represents the three AREAs of a component and its ANNOUNCE QUEUE. It is possible to select one or all areas by clicking with the left button on the appropriate field of the symbol. By default, the complete component is selected.
- The next subwindow, which is entitled STREAM, allows the specification of one or more performance indices that are to be measured. It contains the so-called standard streams in the first part of the list, as well as the user defined streams that were specified in the component type graphic window of the corresponding component type. Streams are selected or deselected by clicks with the left mouse button. Note that it is impossible to deselect a stream that is the last selected one. In Figure 4.65 the stream TURNAROUNDTIME is selected.

- The following subwindow, named ESTIMATOR, allows to specify the estimator, which should be applied for measurement purposes. The estimators STANDARDDEVIATION, CONFIDENCE LEVEL and FREQUENCY are only evaluable during simulation and are ignored otherwise.
- The next subwindow allows for the selection of HIERARCHYs. It contains the standard hierarchy “all” as well as all appropriate user defined hierarchies, that were defined and specified in the hierarchy survey and hierarchy window for the corresponding component.
- Finally, the subwindow START/STOP displays two rectangular symbols with initial values “0” and “∞”, respectively. They represent the start/stop controls that determine the start and stop time of the current measurement during simulative analysis. An empty box expresses that a measurement control condition is specified. A stop symbol in the right box indicates a simulation stop condition based on the measurements. This condition is directly described here in a box with a popup menu and two value fields for the WIDTH, that the confidence interval should fall below, and the number of UPDATES, that should be reached. The simulation stops, if all simulation stop conditions of all measurements are fulfilled.

### 4.8.1 Global Functions

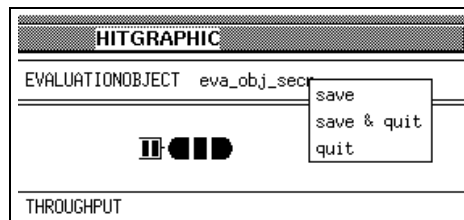


Figure 4.66: Menu on the Title of the Evaluation Object Window

- *save*

saves the current evaluation object.

For the duration of this operation the popup menu remains visible on the display. The window is refreshed, when the operation is finished. At this point you see the actual state of the evaluation object as it is stored in the database. If names or the number of streams or hierarchies are different now, the corresponding component type respectively the corresponding hierarchy list must have been changed in the meantime.

*Save* is not selectable, if the corresponding evaluation is locked by another user or released.

If values are not correct, an error message is displayed and the *save* operation is aborted.

- *save & quit*

performs a save in the same way as the *save* operation does, but also quits the

evaluation object window after a successful save.

*Save & quit* is not selectable, if the corresponding evaluation is locked by another user or released or if a subwindow of this evaluation object window is not closed.

- *quit*

quits the evaluation object window.

If any changes were made since the last *save* operation, you are asked for confirmation.

*Quit* is not selectable, if a window opened from this evaluation object window is not closed.

## 4.8.2 Specification of the Evaluation Object

### Selection of the Area

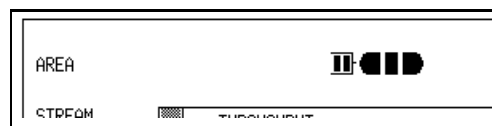


Figure 4.67: The Selection of the Area

The evaluation of a model or component can be restricted to one area: announce queue, entry area, service area or exit area. Selecting another single area is performed by clicking on it with the left mouse button. To select all areas when a single area is highlighted, click on the already highlighted area with the left mouse button.

By default the evaluation is not restricted to a single area.

If any of the streams UTILIZATION, SCHEDULE RATE and PREEMPT RATE or a self defined stream is selected, restriction to one area is not possible.

A selection is not possible as well, if the corresponding evaluation is locked by another user or released.

### Selection of Streams

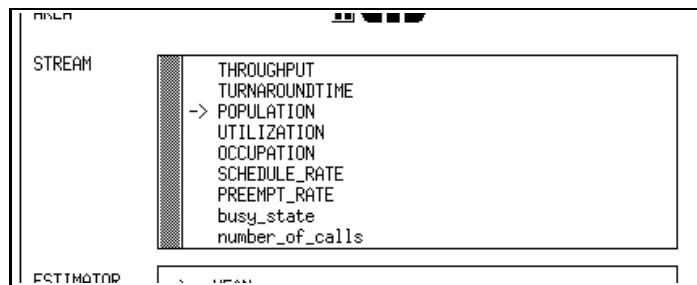


Figure 4.68: The Selection of Streams

The stream list contains all streams which are available for the current evaluation object. If the whole model respectively component is to be evaluated, these are the



standard streams THROUGHPUT, TURNAROUNDTIME, POPULATION, UTILIZATION (only in case of a server or prioserver), OCCUPATION, SCHEDULE RATE and PREEMPT RATE, and all self defined streams (i.e., the streams that were specified in the component type graphic window, that corresponds to the type of the model respectively component the evaluation object is related to). The standard streams UTILIZATION, SCHEDULE RATE and PREEMPT RATE as well as all self defined streams are not selectable, if the evaluation is restricted to one area.

POPULATION is selected as the default.

One or more streams can be selected. A selection can be reset with the left mouse button. Selected streams are indicated by a right-pointing, small arrow. To insure that at least one stream is selected, a reset of the last stream is prevented.

Note that during the *transform* operation selected streams are ignored, if they cannot be evaluated with the method chosen for the experiment.

Changing the stream selection is not possible, if the corresponding evaluation is locked by another user or released.

## Selection of Estimators

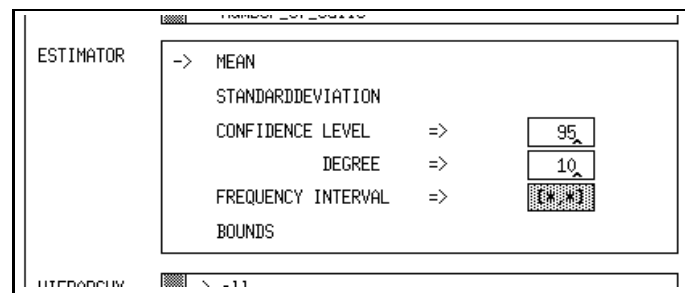


Figure 4.69: The Selection of Estimators

The estimator table provides the five entries MEAN, STANDARDDEVIATION, CONFIDENCE LEVEL, FREQUENCY INTERVAL and BOUNDS. The default selection is estimator MEAN.

Selecting CONFIDENCE LEVEL entails an automatically selection of MEAN and STANDARDDEVIATION, because they must be computed, too. MEAN is also selected, if STANDARDDEVIATION or BOUNDS are requested.

Estimators can be selected respectively a selection can be reset with the left mouse button. Selected estimators are indicated by a right-pointing, small arrow. To insure that at least one estimator is selected, a reset of the last and only estimator causes the selection of the default estimator MEAN.

The estimator CONFIDENCE LEVEL requires two integer values for level and degree. The level value must be in the interval  $[90,99]$ , the degree value in the interval  $[1,20]$ . Default values are 95 and 10, respectively.

The estimator FREQUENCY INTERVAL requires intervals. They can be edited in an editor window, which is displayed when performing an *open* operation on the “[\*,\*]”-symbol with the right mouse button. There is no initial default interval.

Changing the estimator selection is not possible, if the corresponding evaluation is locked by another user or released.

## Selection of Hierarchies

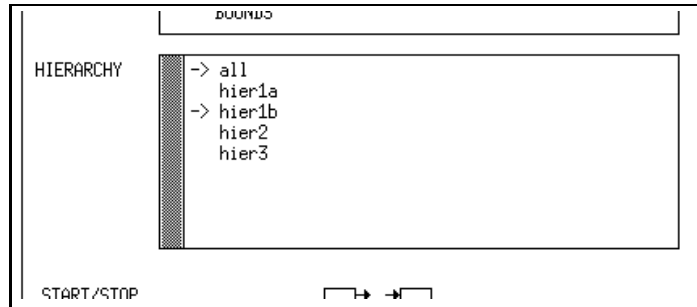


Figure 4.70: The Selection of Hierarchies

The list contains on one hand the standard hierarchy “all”, which is the default of this list, and, on the other hand, all hierarchies, that were generated for the model respectively component the current evaluation object is related to.

Hierarchies can be selected respectively a selection can be reset with the left mouse button. Selected hierarchies are indicated by a right-pointing, small arrow. To insure that at least one hierarchy is selected, a reset of the last hierarchy is prevented.

Changing the hierarchy selection is not possible, if the corresponding evaluation is locked by another user or released.

## Functions Concerning Start/Stop Controls

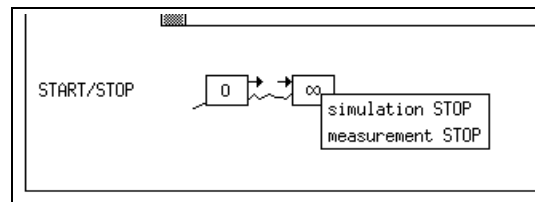


Figure 4.71: Menu for Stop Control Selection

- *simulation STOP*

A simulation stop control condition is added to the measurements of the evaluation object.

- *measurement STOP*

A measurement stop control condition is added to the measurements of the evaluation object. This function implicitly generates a default measurement stop control (CONFIDENCE LEVEL 95, DEGREE 10, WIDTH 10, POPULATION, HIERARCHY all).

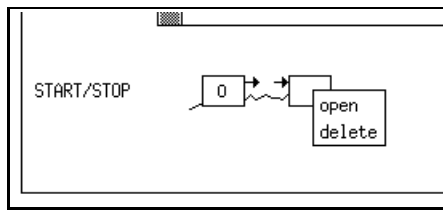


Figure 4.72: Menu on Measurement Start/Stop Control

- *open*

opens the corresponding start respectively stop window.

In case of a measurement start control this function implicitly generates a default start control (MODELTIME 0), if it is performed for the first time. An empty box always indicates an existing measurement start or stop condition.

If there was no measurement start respectively stop control, i.e., the symbol “0” respectively “∞” is visible, a dialog box is displayed asking for a *save* before performing the *open* operation.

*Open* is not selectable, if this window is already open.

- *delete*

deletes the start respectively stop controls.

A dialog box is displayed, asking for confirmation to delete the start respectively stop conditions. After clicking *OK* with the left mouse button, the conditions are deleted and, if the start control symbol was selected, a “0” is written into the symbol, or, if the stop control symbol was selected, the symbol contains a “∞” now.

*Delete* is not selectable, if no start respectively stop control exists, i.e., if the symbol contains a “0” respectively a “∞”, if the start respectively stop window is open or, if the corresponding evaluation is locked by another user or released.

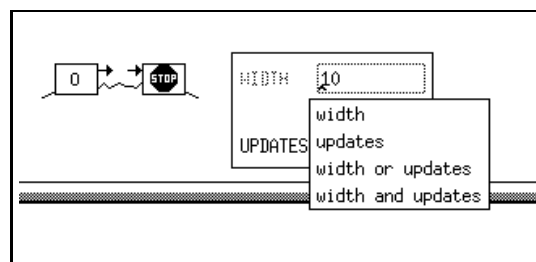


Figure 4.73: Menu on Simulation Stop Control

- *width*

Only the width condition for the confidence interval is used for simulation stop control, whereas the updates condition is ignored.

Selection of *width* is not correct if CONFIDENCE LEVEL is not selected as estimator.

- *updates*

Only the updates condition is used for simulation stop control, whereas the width condition is ignored.

- *width or updates*

The simulation stop control will be composed of the width and updates condition connected by a “logical or”. The operator is displayed in the box.

Selection of *width or updates* is not correct if CONFIDENCE LEVEL is not selected as estimator.

- *width and updates*

The simulation stop control will be composed of the width and updates condition connected by a “logical and”. The operator is displayed in the box.

Selection of *width and updates* is not correct if CONFIDENCE LEVEL is not selected as estimator.

## 4.9 The Start/Stop Window

The start/stop window is used to specify simulation stop conditions (called from the STOP symbol within the evaluation window) as well as measurement start and stop controls (called from the evaluation object window). They only make sense in combination with simulation.

The specification principle of this window is as follows:

Single conditions are composed via conjunction/disjunction to form one boolean expression. The conjunctions are listed in columns that are composed in a disjunctive fashion.

The resulting expression serves as control, because the measurement or simulation starts or stops as the corresponding boolean expression becomes true. This principle of control specification holds for the start as well as for the stop control. Different boolean expressions are available, depending on the application of the control condition. If in the current window start conditions are specified, only the pullright entries *MODELTIME* and *EVENTS* are selectable. If simulation stop conditions are specified, only the entries *EVENTS* and *CONFIDENCE LEVEL* are selectable. For measurement stop conditions all entries are available.

In the following we describe the criteria without consideration of these restrictions in order to simplify the explanation.

In Figure 4.74 an example of a measurement stop control is given. It graphically presents the boolean expression

(CONFIDENCE LEVEL ... *and* MODELTIME 5000) *or* (EVENTS 10000).

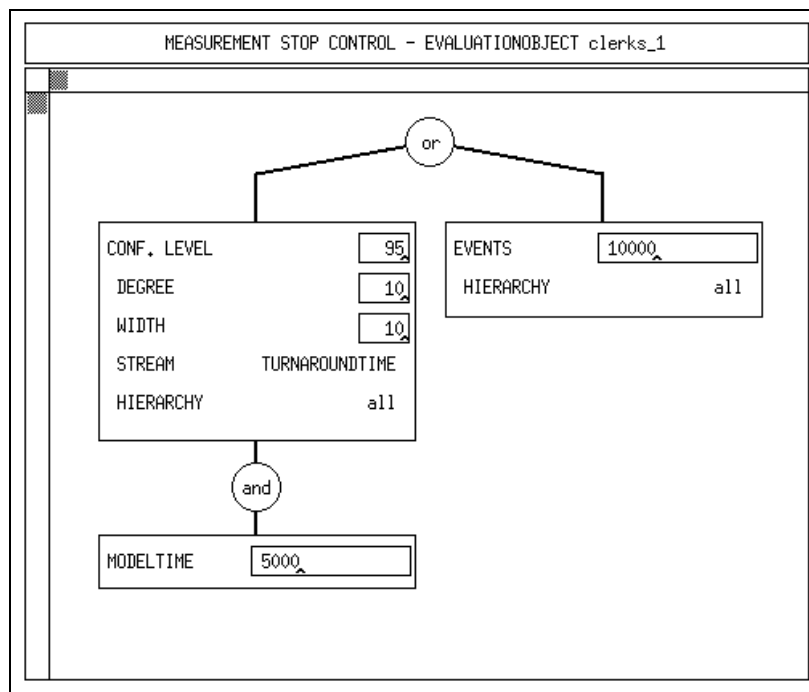


Figure 4.74: Start/Stop Window

### 4.9.1 Global Functions

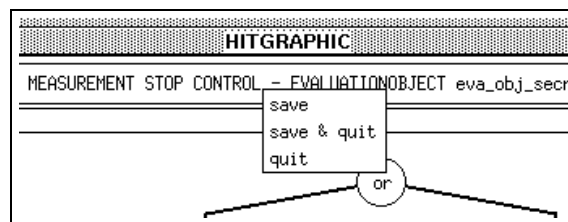


Figure 4.75: Menu on the Title of the Start/Stop Window

- *save*

saves the current start respectively stop conditions.

If the entered values of the conditions are correct, the *save* operation is executed. For the duration of this operation the popup menu remains visible on the display. The start/stop window is redrawn, after the operation has finished.

An error message is displayed, if any values are not correct.

*Save* is not selectable, if the corresponding evaluation is locked by another user or released.

- *save & quit*

saves the current start respectively stop conditions in the same way as *save* does and, if no errors have occurred, quits the start/stop window.

*Save & quit* is not selectable, if the corresponding evaluation is locked by another user or released.

- *quit*

quits the start/stop window.

If any changes were made since the last *save* operation, you are asked for confirmation.

## 4.9.2 Specification of Start respectively Stop Conditions

### Generating OR Conditions

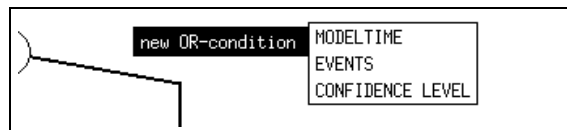


Figure 4.76: Generating OR Conditions

- *new OR-condition* ->

connects the existing “or conditions” with a new “or condition”, that has the criterion ...

– *MODELTIME*

... reached modeltime (default: 0)

– *EVENTS*

... number of events (default: 0, hierarchy default: all)

– *CONFIDENCE LEVEL*

... confidence interval width (level default: 95, degree default: 10, width default: 10, measure default: POPULATION, hierarchy default: all)

The new start respectively stop condition is appended at the right end of the existing conditions. Parts of the window are redrawn.

This entry is not selectable, if the corresponding evaluation is locked by another user or released.

### Functions on Conditions

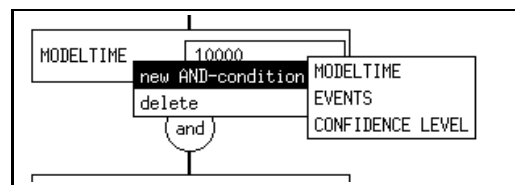


Figure 4.77: Generating AND Conditions

- *new AND-condition* ->

appends a new “and condition” to the current column. The new “and condition” has the criterion ...

- *MODELTIME*

- ... reached modeltime (default: 0)

- *EVENTS*

- ... number of events (default: 0, hierarchy default: all)

- *CONFIDENCE LEVEL*

- ... confidence interval width (level default: 95, degree default: 10, width default: 10, measure default: POPULATION, hierarchy default: all)

This entry is not selectable, if the corresponding evaluation is locked by another user or released.

- *delete*

deletes the selected condition.

A dialog box is displayed, asking for confirmation to delete the condition. After clicking *OK* with the left mouse button, the condition is deleted. If this was the last condition in a column, the whole column is deleted and the window is redrawn. Clicking *Cancel* aborts the delete operation.

*Delete* is not selectable, if the corresponding evaluation is locked by another user or released or if there is only one condition at all.

### 4.9.3 Specification of Values

#### Criterion MODELTIME

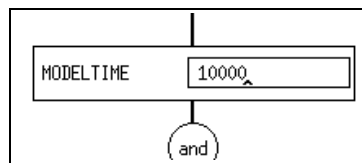
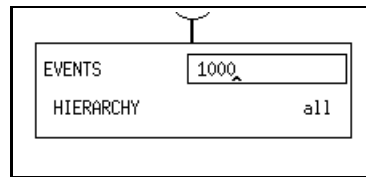


Figure 4.78: Changing the Modeltime

Valid values for modeltime are non-negative values (default: 0), which can also be entered in exponential form. The values are checked, when the *save* operation is performed.

The value cannot be changed, if the corresponding evaluation is locked by another user or released.

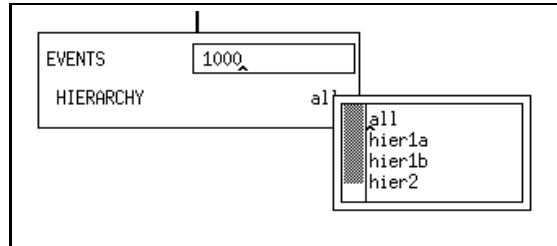
## Criterion EVENTS



EVENTS	1000
HIERARCHY	all

Figure 4.79: Changing the Number of Events

Valid values for events are non-negative integer values (default: 0). The values are checked, when the *save* operation is performed.



EVENTS	1000
HIERARCHY	all

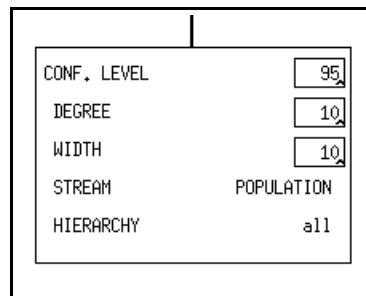
- all
- hier1a
- hier1b
- hier2

Figure 4.80: Changing the Hierarchy

Clicking the right mouse button on the hierarchy field displays a popup menu, showing a list of the hierarchies which belong to the corresponding model respectively component. A hierarchy can be selected with the left mouse button. The name of the selected hierarchy is then displayed in the hierarchy field.

The number of events cannot be changed and hierarchies cannot be selected, if the corresponding evaluation is locked by another user or released.

## Criterion CONFIDENCE LEVEL



CONF. LEVEL	95
DEGREE	10
WIDTH	10
STREAM	POPULATION
HIERARCHY	all

Figure 4.81: Changing the Confidence Level

The criterion confidence level requires three integer values, a stream and the name of a hierarchy. The value for the level must be in the interval  $[90,99]$  (default: 95), the degree value in the interval  $[1,20]$  (default: 10) and the value for the relative width of the confidence interval in the interval  $[1,50]$  (default: 10) given in percent. In the simulation the difference between any of the bounds of the confidence interval and the estimated mean value should become lower than the given percent value that relates



to the estimated mean. The entered values are checked, when the *save* operation is performed.

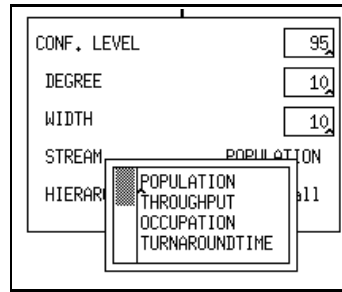


Figure 4.82: Changing the Stream for the Confidence Level

Clicking the right mouse button on the stream field displays a popup menu, showing a list of the streams which belong to the type of the corresponding model respectively component. A stream can be selected with the left mouse button. The name of the selected stream is then displayed in the stream field.

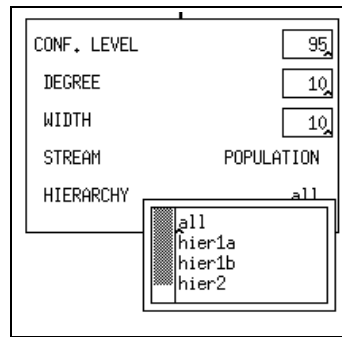


Figure 4.83: Changing the Hierarchy for the Confidence Level

Clicking the right mouse button on the hierarchy field displays a popup menu, showing a list of the hierarchies which belong to the corresponding model respectively component. A hierarchy can be selected with the left mouse button. The name of the selected hierarchy is then displayed in the hierarchy field.

Values cannot be changed and streams and hierarchies cannot be selected, if the corresponding evaluation is locked by another user or released.

## 4.10 The Hierarchy Survey Window

The hierarchy survey window is called from the evaluation window and is associated with a component. It provides the user with basic functions to specify self defined hierarchies that end in the associated component.

All user defined hierarchies are displayed in a list and offer operations via a popup menu.

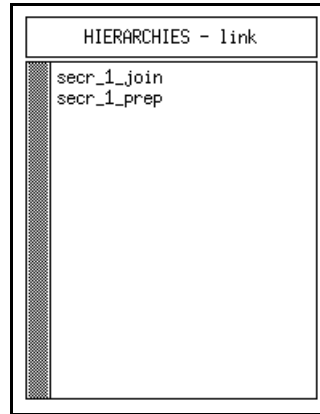


Figure 4.84: Hierarchy Survey Window

In Figure 4.84 a hierarchy survey window is displayed showing the two hierarchies “secr\_1\_join” and “secr\_1\_prep”.

### 4.10.1 Global Functions



Figure 4.85: Menu on the Title of the Hierarchy Survey Window

- *new hierarchy*

creates a new hierarchy.

A dialog box is displayed, requesting a name for the new hierarchy. After entering a correct name and confirming it the new hierarchy is inserted in the alphabetically sorted list of hierarchies.

An error message is displayed, if the entered name already exists as a hierarchy in the current window.

*New hierarchy* is not selectable, if the corresponding evaluation is locked by another user or released.

- *save*

saves the hierarchy survey window.

*Save* is not selectable, if a hierarchy window started from this window is still open or if the corresponding evaluation is locked by another user or released.

- *save & quit*

saves and quits the hierarchy survey window, if the save operation has been successful.

*Save & quit* is not selectable, if a hierarchy window started from this window is still open or if the corresponding evaluation is locked by another user or released.

- *quit*

quits the hierarchy survey window.

If any changes were made since the last *save* operation, you are asked for confirmation.

*Quit* is not selectable, if a hierarchy window started from this window is still open.

## 4.10.2 Functions on Hierarchies

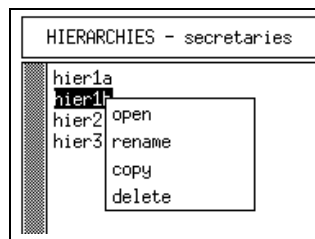


Figure 4.86: Menu on Hierarchies

- *open*

opens the corresponding hierarchy window.

If the current hierarchy is just created, a dialog box is displayed asking for a *save* before performing the *open* operation.

This entry is not selectable, if this hierarchy window is already open.

No modifications of the hierarchy are possible if the corresponding evaluation is locked by another user or released.

- *rename*

allows you to rename the selected hierarchy.

A dialog box is displayed, asking for the new name of the hierarchy. After entering a correct name and confirming it the hierarchy is renamed and reinserted in the list. If this hierarchy is used in evaluation object windows or start/stop windows of the current environment, it is renamed there, too.

An error message is displayed, if the entered name already exists.

*Rename* is not selectable, if the hierarchy window of the selected hierarchy is open or if the corresponding evaluation is locked by another user or released.

- *copy*

copies the selected hierarchy, i.e., copies all load paths of this hierarchy.

A dialog box is displayed, requesting the name for the copy. After entering a correct name and confirming it the new hierarchy is created and inserted in the list.

An error message is displayed, if the entered name already exists.

*Copy* is not selectable, if the corresponding evaluation is locked by another user or released or if the hierarchy window of the selected hierarchy is open.

- *delete*

deletes the selected hierarchy, i.e., deletes all load paths of this hierarchy.

A dialog box is displayed, asking for confirmation to delete the selected hierarchy. After clicking *OK* with the left mouse button, all load paths of this hierarchy are deleted and the hierarchy is removed from the list of hierarchies.

If this hierarchy is used in evaluation object windows or start/stop windows of the current environment, it is replaced there by the hierarchy “all”, if it was the only hierarchy selected there. Clicking *Cancel* aborts the *delete* operation.

*Delete* is not selectable, if the hierarchy window of the selected hierarchy is open or if the corresponding evaluation is locked by another user or released.

## 4.11 The Hierarchy Window

The hierarchy window is designed for the modelling of hierarchies via the concept of load paths. A load path specifies a special load filtering for measurement within a model, that ends in an associated component. The set of load paths defined in a hierarchy window is composed internally to form the entire hierarchy. The composed hierarchy may be used by its name within the start/stop or the evaluation object window.

The specification of a load path may be performed hierarchically on three levels. The first step is done on the component view, where only components are displayed. An example is given in Figure 4.87, which shows a unique path on the component level leading from “eva\_4” of type “office\_4” to the component “link” of type “server”.

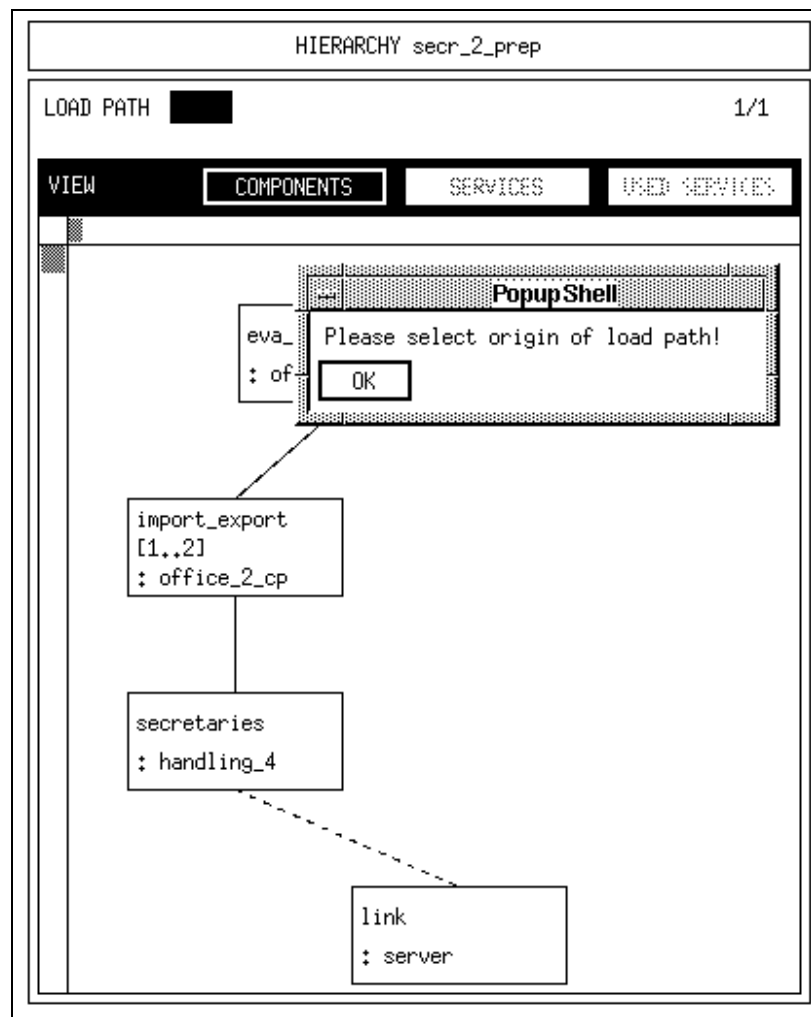


Figure 4.87: Hierarchy Window: Component View

When a load path specification on the component level describes a distinct component path after selection of appropriate components or paths, the additional service and used service views are accessible, which allow for a more detailed specification. In Figure 4.88 the previously shown path is displayed in the used services view showing all existing services and used services on the corresponding path.

In the hierarchy window each load path is saved separately. So you will not find a *save* operation for the complete set of load paths in the menu on the title. You will find the *save* operation for a load path on the background menu.

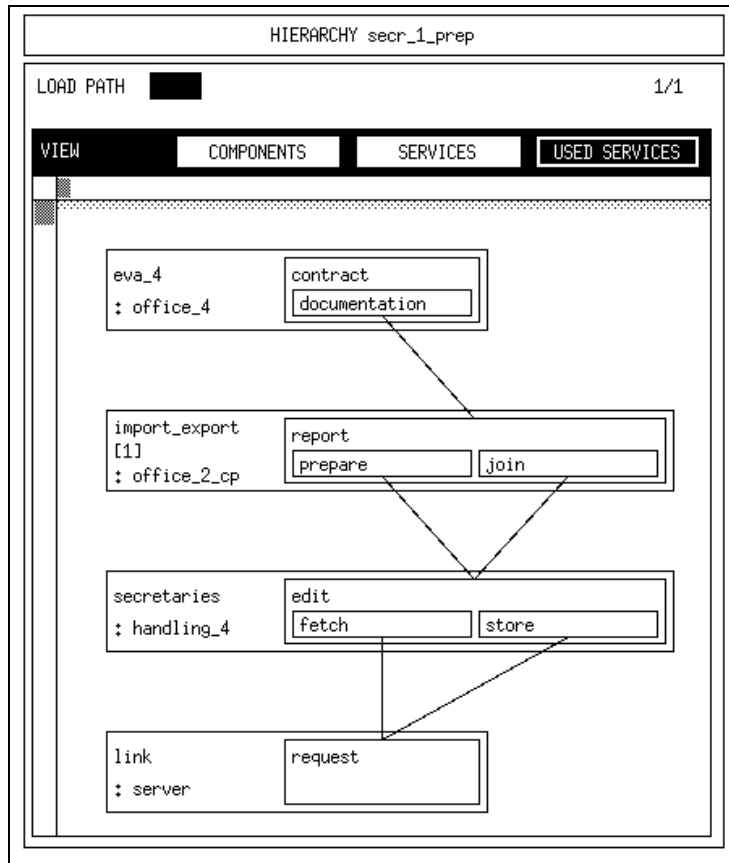


Figure 4.88: Hierarchy Window: Used Service View

The specification principles of this window are restricted by some phenomena due to certain model structures, for instance component arrays or shared components. A detailed discussion of the specification procedure as well as for the handling of the non-trivial model structures is presented in the tutorial (cf. Section 7.4.5).

### 4.11.1 Global Functions

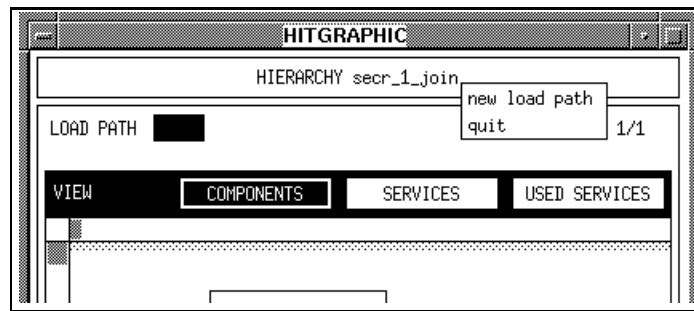


Figure 4.89: Menu on the Title of the Hierarchy Window

- *new load path*

appends a new load path at the end of the list of load paths.

If this is the only load path in the current window you get a look on the reduced object structure which ends in the component corresponding to this hierarchy. If a new load path is displayed (e.g., after a load path switch) a dialog box is displayed requesting the originator of the load path.

After selecting an object (performing the *select* operation with the right mouse button) the components without interest are cut out and the area between model and the originator turns grey.

Furthermore the paths (respectively part of these paths) between originator and basic component without any bound services (and consequently without any possibility to call) will be eliminated. Now the load path can be specified by sequentially activating the *select* operation. If no admissible path can be specified the load path will be deleted and an error message will be displayed. If you want to switch to another (for example a new) load path and there are unsaved changes within the current load path, the switch must be explicitly confirmed. Clicking *OK* with the left mouse button aborts the modification of the current load path; after clicking *Cancel* no load path switch is done.

*New load path* is selectable at any time, except the corresponding evaluation is locked by another user or released.

- *quit*

quits the hierarchy window.

If any changes have been made in the current load path since the last *save* operation respectively the very first displaying, you are asked for confirmation.

### 4.11.2 Functions Referring to Load Path Selection

Load paths can be accessed directly selecting an area of the load path survey with the left mouse button. Consequently the “black box” is moving to this selected area and in the load path subwindow the corresponding load path will be displayed. Further switches are possible at any time.

If any changes were made since the last *save* in the current load path the message “Switch without save?” is displayed. Clicking *OK* aborts the modification of the current load path; after clicking *Cancel* no switch is done.

### 4.11.3 Functions Referring to the Selection of Views

The various views *COMPONENTS*, *SERVICES* and *USED SERVICES* enable the specification of a load path with different degree of detail.

Note the following points:

- Initially the *COMPONENTS* view is chosen.
- Switching to another view is only possible if a single component path is definitely specified and saved in the *COMPONENTS* view.
- Before switching the view from *COMPONENTS* to another view it is checked whether the current load path is empty, i.e., whether there is no connection between provided and used services along the component path. If the load path is empty (in this sense) the switch is aborted and an error message is displayed.

### 4.11.4 Functions Referring to the Current Load Path

#### Functions on the Background

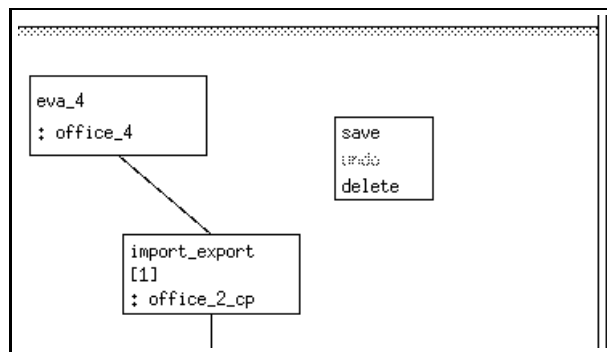


Figure 4.90: Background Menu on the Load Path

- *save*

saves the current specification of a load path.

For the duration of this operation the mouse cursor is displayed as a clock symbol. The window is refreshed when the operation is finished. In case of an empty load path an error message is displayed.

If the *save* is done in the view *COMPONENTS* an automatic switch to the view *SERVICES* is performed because at this time the component path is definitely specified and no more modifications can be done in this view (*COMPONENTS*). *Save* is not selectable if the load path is not specified definitely or if the corresponding evaluation is locked by another user or released.



- *undo*

undoes the last update (i.e., the last selection).

*Undo* can be called repeatedly until the last saved state is reached.

*Undo* is not selectable for a new window or immediately after a *save* operation. Furthermore this function is not selectable if the corresponding evaluation is locked by another user or released.

- *delete*

deletes the current load path specification.

A dialog box is displayed asking for confirmation to delete the current load path.

After the deletion the cyclically next load path specification is displayed. The load path survey is updated appropriately. If the last load path has been deleted, an empty hierarchy window is displayed. The load path survey is empty, too.

*Delete* is not selectable if the corresponding evaluation is locked by another user or released.

### Functions on Components and Edges (View: COMPONENTS)

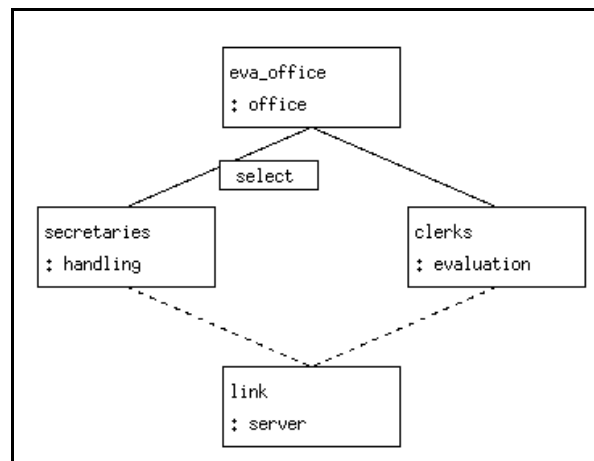


Figure 4.91: Menu on an Edge

- *select*

selects a component respectively an edge.

The load path will be restricted appropriately due to the selection.

The selection may cause that indices of arrays on the component path have to be specified, because these indices cannot be determined at the points of usage of the hierarchy. Dialog boxes may be displayed asking for these indices.

This function is not selectable if the corresponding evaluation is locked by another user or released.

## Functions on Services (View: SERVICES)

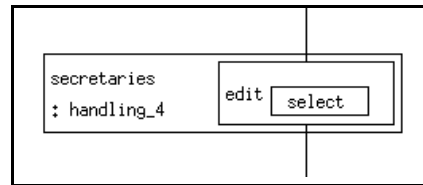


Figure 4.92: Menu on a Service

- *select*

selects a service.

The load path will be restricted appropriately due to the selection.

This function is not selectable if the corresponding evaluation is locked by another user or released.

## Functions on Used Services (View: USED SERVICES)

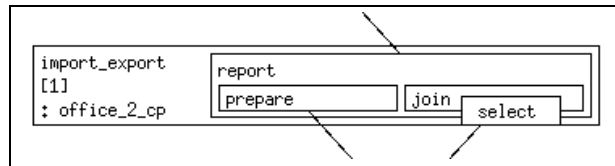


Figure 4.93: Menu on a Used Service

- *select*

selects a used service.

The load path will be restricted appropriately due to the selection.

This function is not selectable if the corresponding evaluation is locked by another user or released.

## 4.12 The Experiment Window

The experiment window serves as the basis for the complete experiment specification. It allows the user to specify the evaluations and their actual parameters, the solution method to be applied as well as the body of the experiment, which defines the course of experiment execution.

An example of an experiment window is given in Figure 4.94. It shows that the window is subdivided into three areas.

The first part entitled METHOD allows for the selection of the analysis method to be applied for the current experiment. For each method, some additional specifications can be made. The sample window displays the selected method SIMULATIVE and some stop conditions for each evaluation of the experiment run.

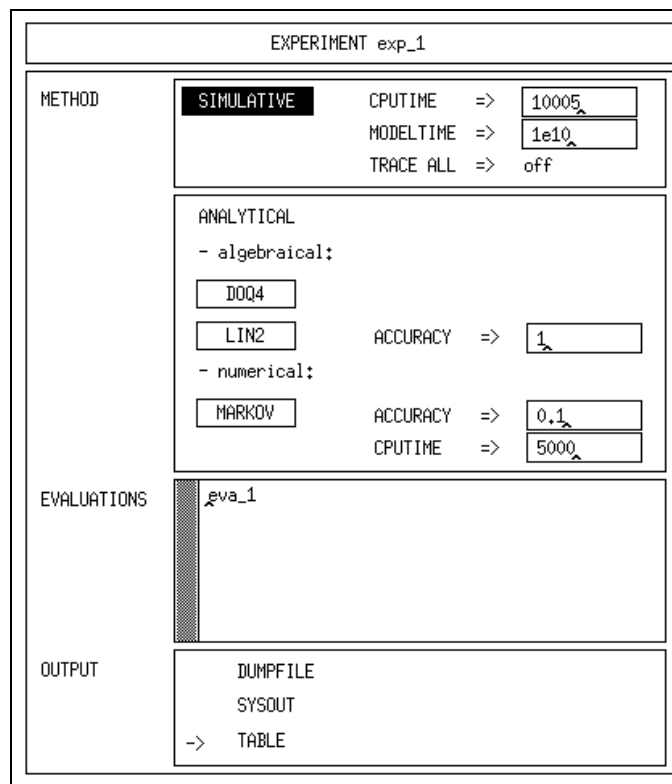


Figure 4.94: Experiment Window

The next subwindow is entitled EVALUATIONS. It displays an alphabetically sorted list of evaluations, that may be run and analysed within this experiment. To enter an evaluation, the *insert evaluation* operation of the title bar menu is activated, which causes a message box to be displayed. After the selection of the desired evaluation within the environment window the evaluation is inserted into the list of the subwindow EVALUATIONS. Note that only evaluations being displayed in this subwindow may be analysed within the current experiment. In Figure 4.94 the evaluation “eva\_1” is imported.

The last subwindow, OUTPUT, allows for the selection of the output representation of the analysis’ results. The default selection is TABLE, which is the most common form of output representation.

Two additional operations should be mentioned. The *open* operation on the title bar opens a text editor, which allows for the specification of the experiment body. Note that evaluations, which are referred to, are written in the form “EVALUATE my\_evaluation;” in a separate line.

Finally, the popup menu on evaluations provides the operation *actual parameters*. It opens a text editor, which allows to specify actual parameters for the evaluation respectively for the corresponding model type.

### 4.12.1 Global Functions

- *open*

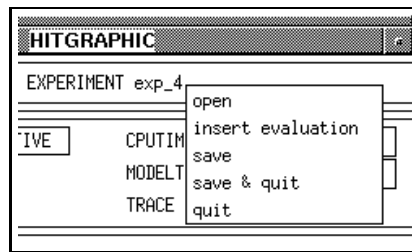


Figure 4.95: Menu on the Title of the Experiment Window

opens the HITGRAPHIC text editor with three subwindows to specify the following parts of an experiment:

– *CONTROL*

This editor subwindow serves for the specification of the HI-SLANG control part.

Do not forget the “%END” at the end of the control part.

– *LOCALS*

This editor serves for the specification of the declaration part.

– *BODY*

This editor serves for the specification of the instruction part.

Evaluations used in the instruction part must be referred to in a special statement (e.g., `EVALUATE eva_1;`).

If an evaluation is used in the instruction part which is not listed in the EVALUATIONS area an appropriate message will be displayed during the transformation and the transformation will be aborted.

*Open* is not selectable if the text editor is already open.

The files are only readable if the experiment is locked by another user or released or if file permissions are not sufficient for write mode.

- *insert evaluation*

inserts an evaluation into the list of evaluations in this experiment in alphabetical order.

A dialog box is displayed asking for the selection of an evaluation in the environment window using the left button. After clicking *OK* the evaluation is inserted into the list. If the experiment body is not currently open and can be edited, another box is displayed asking to append an `EVALUATE` statement for the inserted evaluation at the end of the experiment body. After clicking *OK* the experiment body is automatically updated.

An error message will be displayed if anything but an evaluation has been selected (e.g., a component type) respectively if the evaluation already exists in the list.

*Insert evaluation* is not selectable if the experiment is locked by another user or released.

- *save*

saves the actual state of the experiment window (i.e., the experiment specification).

An error message will be displayed if any values are incorrect.

*Save* is not selectable if the experiment is locked by another user or released or if a subwindow of the experiment window is still open.

- *save & quit*

saves the actual state of the experiment window as the *save* operation does. If saving has been successful, the experiment window is closed.

*Save & quit* is not selectable if the experiment is locked by another user or released or if a subwindow of the experiment window is still open.

- *quit*

quits the experiment window.

If any changes were made since the last *save* operation, you are asked for confirmation.

*Quit* is not selectable if a subwindow (i.e., a text editor) is still open.

## 4.12.2 Operations in the Method Area

In the method area the user may choose between the methods SIMULATIVE, DOQ4, LIN2 or MARKOV by clicking with the left mouse button. The default method is SIMULATIVE.

Changing the method is not possible if the experiment is locked by another user or released.

Additionally for the method SIMULATIVE it can be specified whether a trace should be generated for the current experiment or not. If TRACE ALL is clicked with the right button a popup menu is displayed showing the value *on* respectively *off*. The default value is *off*.

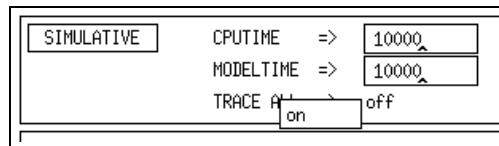


Figure 4.96: Menu on the Trace Switch in the Experiment Window

- *TRACE ALL on/TRACE ALL off*

causes a trace generation for this experiment respectively resets the option.

Switching is not possible if the experiment is locked by another user or released.

For the method SIMULATIVE the additional specification of the maximal runtime (CPU time) or modeltime for an evaluation is possible. Default values are 10000. The value 0 means an ignoring of this stop criterion. But you cannot set both values to 0. The values can also be given in exponential form (i.e., two values separated by an 'E' or 'e').

The complete simulative stop control of an evaluation within the experiment will be composed by a “logical or” of up to four conditions:

1. the `MODELTIME` condition of the experiment window, if enabled,
2. the `CPUTIME` condition of the experiment window, if enabled,
3. a “logical or” of all simulation stop conditions attached to evaluation objects in the evaluation window (cf. Sections 4.7 and 4.9), if any are specified,
4. a “logical and” of all simulation stop conditions in evaluation objects described in evaluation object windows (cf. Section 4.8.2), if any are specified.

Additional to the methods `LIN2` and `MARKOV` accuracy values (`ACCURACY`) for the results can be specified. The `LIN2` accuracy value refers to performance bounds and must be an integer value between 0 and 4, the `MARKOV` accuracy must be specified by a real value between 0 and 1. The values can be inserted directly into the corresponding text fields. Default accuracy values are 1 (`LIN2`) and 0.1 (`MARKOV`). A (syntactical and semantical) check will be done during a *save* operation.

Changing the values is not possible if the experiment is locked by another user or released.

### 4.12.3 Functions on Evaluations

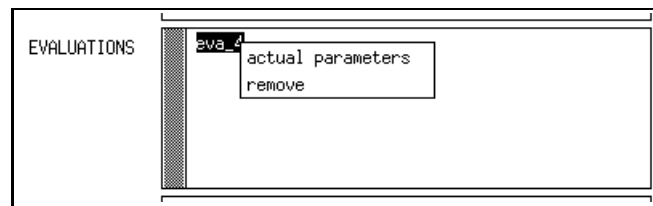


Figure 4.97: Menu on Evaluations in the Experiment Window

- *actual parameters*

opens the `HITGRAPHIC` text editor with two subwindows to display the formal model parameters in read-only mode and to specify the actual parameters of the model for this experiment.

If the formal parameters have been changed or deleted within the corresponding model type, the actual parameters have to be updated in this window by the user himself. Do not forget to change the experiment body and the locals, too. The operation is not selectable if the text editor with the same file is already open.

The actual parameters are only readable if the experiment is locked by another user or released or if file permissions are not sufficient for write mode.

- *remove*

removes the selected evaluation from the list.

It is not checked whether this evaluation is still used in the (experiment) instruction part. Possibly an interruption is forced during transformation (cf. above: *open* operation).

*Remove* is not selectable if the current experiment is locked by another user or released.

#### 4.12.4 Output Area

In the output area the result presentation can be selected respectively the selection can be reset with the left mouse button. The selected parts are indicated by a right-pointing, small arrow.

To insure that at least one result form is specified a reset of the last selection will force the selection of TABLE, which is the default output form.

If an interactive observer is used within the model actually being solved, one output direction should be SYSOUT, which pipes the table output to the protocol window of the HIT run.

Changing the result form is not possible if the experiment is locked by another user or released.





# Chapter 5

## Teamwork with HITGRAPHIC

### 5.1 Modes of Modelling Objects

In general many questions may arise in the context of teamwork, like

- How can modelling objects being in progress permanently be protected against unauthorized or ignorant changes even after the end of a HITGRAPHIC session?
- How can completely modelled objects be released for general use?

Therefore, cooperating modellers (modelling by teamwork) wish to get as much help as possible. If every user accepts the rules for using the system the problem can be solved in a really simple and effective manner. At every time every modelling object is in a mode which has to be set explicitly by the user. Each mode defines a scope for the operations permitted.

The operation *mode* on modelling objects within the environment window is used to set a mode. After you have done this the following window will appear.

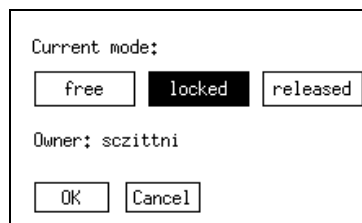


Figure 5.1: Mode Box

The modes can be classified and described as follows.

- *free*

A locked object can become unlocked by the *free* operation, which is executable only by the owner.

*Free* is the default mode of a modelling object. Modifications including changes of the mode are possible for all authorized users.

- *locked*

By the lock operation the user gets the exclusive permission to change a modelling object. In this mode the modelling object is coupled with the user, who now becomes “owner” of the modelling object. His identification is generally known by all users (it is the user name in UNIX). The duration of the *lock* does not depend on a session or process so that the *lock* operation is valid even after the end of a HITGRAPHIC session. Read and write permissions for the owner remain unchanged, but all other users can use this modelling object only for reading purpose.

By the lock operation all data belonging to the modelling object are locked and cannot be changed by anyone else than the owner. It might be desirable to lock all other modelling objects being concerned in it as well, e.g., for component types: the used component types and their interfaces. But as a consequence only bottom up design would be possible. Therefore the lock operation has only the effects described above. It is up to the user to get a more extended lock operation, i.e., he has to lock or release the additional modelling objects himself.

- *released*

After finishing the specification of a modelling object the modeller changes its mode into the *released* mode. But be careful because a released modelling object cannot be changed anymore, not even by the modeller himself. He becomes the “owner” of the object and his user identification will be linked to it.

The system allows all users to use or copy the modelling objects. The users can be sure that there will be no modification of this objects, because no changes are allowed. The only possible operation beside the *copy* operation is the *delete* function that can only be performed by the owner. After releasing a modelling object changes are only possible on copies of that released modelling object.

The domain of the release is the same as that of a lock.

Note that the *released* mode can never be set back to *free* or *locked*, not even by the owner of the modelling object.

## 5.2 General Hints for Teamwork Support

In Section 5.1 the term “modes of modelling objects” has been discussed. The objective of this section is to give some guidelines how to work with HITGRAPHIC in general, respectively how to use the presented features for teamwork support.

- In general working with HITGRAPHIC should be done in a somewhat sequential or reasonable manner. Performing parallel changes in several component type graphic windows should be performed carefully.
- If the *save* operation is performed in the component type graphic window, the changes will be documented within the database immediately. This fact should

lead to a somewhat careful style of modelling. The same holds for some more operations, especially for the *delete* operation within the environment window. These operations may cause a lot of elements to disappear from the database.

- Because displayed data may become obsolete in comparison to the state of the database, you should consider an update from time to time, for example by a *refresh*, *save* or *quit* and subsequent reopen operation.

Some more hints should be observed concerning the teamwork with HITGRAPHIC:

- The specification of interfaces should be done carefully and in early stages of the modelling task.
- If modelling objects are still under development, not tested or in any other “unreliable” state, they should be locked.
- The access and change of modelling objects in the free mode should be avoided when working in a team.
- The information file of modelling objects should really be used for documentation purposes in a reasonable manner. At least interfaces, respectively their changes, should be documented.

These hints should be acceptable for any user. As a result, a basis for common use and teamwork is formed.

### 5.3 Documentation Support

HITGRAPHIC supports documentation by providing a *print* function for several windows. A PostScript file is generated which can be included in papers or used for demonstration. The *print* function is available within the object structure of the survey window, the component type graphic window and the evaluation window.

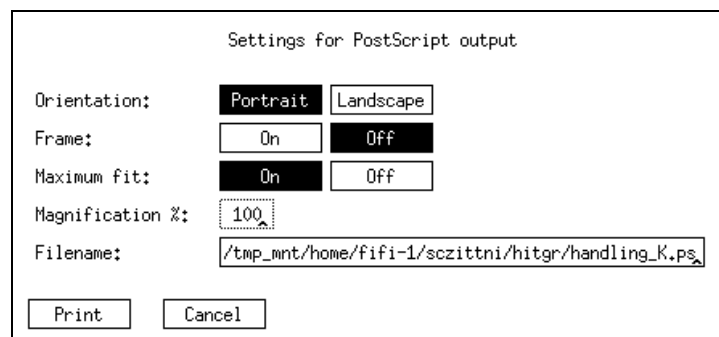


Figure 5.2: Print Box

To manipulate the settings for the PostScript output a subwindow is displayed. The related functions are explained in the following.

- Orientation

The button will toggle the output orientation between *portrait* and *landscape*. The default orientation is *portrait*.

- Frame

A rectangular frame consisting of the border of the output can be requested. The button will toggle between *on* and *off*. The default is *off*.

- Maximum fit

The button will toggle between *on* and *off*. If the default *on* is selected the generated figure is displayed with the maximum size fitted on an A4 paper. If *off* is selected the size depends on the specified magnification value.

- Magnification %

The magnification value can only be changed if no maximum fit is requested. The default value 100% results in a size of the figure approximately to the displayed window size.

It is guaranteed that the use of the same magnification value produces the same picture size even if you are working on different displays.

- Filename

A filename for the output is generated depending on the corresponding name of the element to be printed. It is editable by the user. Filenames can be absolute or relative to the current working directory of the HITGRAPHIC call.

# Chapter 6

## Questions and Answers

A collection of frequently appearing problems during the use of HITGRAPHIC is built up in this chapter. The first section contains a list of problems and questions, the second section offers solutions. This partitioning allows to quickly browse through the problems.

### 6.1 List of Questions and Problems

- 1) The *quit* operation is deactivated, but I want to **quit** HITGRAPHIC.
- 2) I cannot perform *open* because the modelling object is **busy**.
- 3) Which operations affect the **database** and when?
- 4) For which operations do I have to expect a rather long **execution time**?
- 5) I have inserted a new **service** in the component type but I cannot find it at an instantiated object of that type.
- 6) How does the HI-SLANG code of a **self defined control procedure** look like?
- 7) What is the difference between measurement and simulation **stop control**?
- 8) How can I change a **stop control** in a stop control window, e.g. from confidence level to events?
- 9) How does a **CONTROL part** of an experiment look like?
- 10) Can I set the **seed** for random drawing?
- 11) How can I describe **experiment series**?
- 12) Why does the **transformation to HI-SLANG** fail?
- 13) What can I do if I get **compile errors** from the HI-SLANG compiler?
- 14) Can I add **debugging code** during the development phase of a component type?

- 15) The HI-SLANG **listings** are getting too large!
- 16) Why does my database require so much **disk space**?
- 17) Results are **undefined**!
- 18) Can I get **intermediate results** of a simulation?

## 6.2 Solutions

- 1) The *quit* operation is deactivated, but I want to quit HITGRAPHIC.

The *quit* button is only activated, if all subwindows of the window are closed. A subwindow appears as the result of an operation in the window. For example, an experiment window is a subwindow of an environment window.

Due to a program error or a process kill a subwindow may not terminate correctly. Then there may be another option to quit the window, dependent on the X environment and window manager you use. Some window managers have an operation to terminate in the menu of the window. They cause a confirm box to appear, that warns you about a possible damage of your database. If any window of HITGRAPHIC is terminated without the usual *quit* operation, the program “unset\_busy” should be executed, cf. Chapter 2.

- 2) I cannot perform *open* because the modelling object is busy.

Under normal circumstances modelling objects (model types, component types, aggregation descriptions, evaluations, experiments) are busy if they are opened for editing or used in an experiment, that is actually running.

If you can make sure that this is not the case and the modelling object is still busy, you can use the program “unset\_busy” to reset the busy locks, cf. Chapter 2.

- 3) Which operations affect the database and when?

Within the environment window all operations affect the database directly. The same holds for the deletion of load paths within the hierarchy window. Otherwise the database is only modified by *save* respectively *save & quit* operations. Some operations require the actual data to be saved. In that case you are asked to confirm the *save* operation. If you select *Cancel* both the save and the activated operation will not be performed.

A new retrieve in the database is performed with the *save* operation. An exception of this rule is the case that illegal input values have been detected, because you should have the chance to correct without performing all changes again. Changes in the database made in other windows can only be seen after a new retrieve. The survey window has a *refresh* operation, because there is nothing to be saved.

- 4) For which operations do I have to expect a rather long execution time?

All operations that directly access the database take a bit more time. Copying between environments lasts long because a lot of data is affected. *Save* operations may take a longer time, especially if they require internal delete operations. They are complex, because the integrity of the database must be ensured.

The transformation to HI-SLANG may last longer, because all information referenced in the experiment must be retrieved from the database to compose the HI-SLANG source.

- 5) I have inserted a new service in the component type but I cannot find it at an instantiated object of that type.

Firstly, you have to check that the service is provided. This is indicated by a small arrow at the left of the service. Activate the *provide* operation in the menu of the service, if the arrow is missing. Then make sure that the changes of the component type are stored in the database by performing a *save* in the component type graphic window containing the description of the service. Then you still do not see the new provided service at the object of the type, because the changes in the database are not automatically updated in the windows. You have to perform a *save* in the component type graphic window containing the object. Alternatively you can *quit* and *open* that window again. Now you should find the provided service at the object.

- 6) How does the HI-SLANG code of a self defined control procedure look like?

The HI-SLANG code of self defined control procedures, that is entered in the text editor window, is an exception concerning BEGIN-END pairs. It is divided in two parts, the local declarations and the statement part. The parts are separated by a **BEGIN**. The corresponding **END** is not part of the user's input, it will be generated automatically.

For a detailed description of HI-SLANG control procedures please refer to the corresponding chapter in the HI-SLANG Reference Manual. Here we repeat the description of the last-come-first-scheduled preemptive resume schedule procedure, but only the part that is entered in the text editor of HITGRAPHIC.

```

VARIABLE found : BOOLEAN DEFAULT FALSE;

BEGIN

    INSPECT ENTRY_AREA WHILE NOT found LOOP REVERSE
        SELECT;
        found := TRUE;
    END LOOP;

    IF found THEN
        INSPECT SERVICE_AREA LOOP
            SELECT;
        END LOOP;
    END IF;

```

7) What is the difference between measurement and simulation stop control?

Stop controls can be used for measurements and for complete evaluations.

Measurement stop controls are stop conditions that are entered in a start/stop window and opened from within an evaluation object window. They influence the gathering of data for streams activated in the evaluation object window. The gathering stops if the stop condition becomes true. This only works when using the simulative solver. If the stop condition has become true, the estimated measures will no longer change, although the simulation may continue.

As indicated by the name, simulation stop conditions are only of interest to the simulative solver. They control the stopping of the simulation run for the actual evaluation of the experiment. There exist two kinds of simulation stop conditions.

The first kind is related to the determination of measured results. Simulation stop conditions of the first kind are therefore specified in the evaluation object window, only the criteria WIDTH and UPDATES are available for them. Note that the measurement for an evaluation object containing such a simulation stop condition will stop with the end of the simulative evaluation, not necessarily when the condition becomes true. The condition influences the decision whether to stop the simulation, see below.

Simulation stop conditions of the second kind are created and opened at evaluation objects in the evaluation window. They are determined independently from the estimation of results.

The complete simulative stop control of an evaluation within an experiment is composed by a “logical or” of up to four conditions:

- (a) the MODELTIME condition of the experiment window, if not disabled by the value 0,
- (b) the CPUTIME condition of the experiment window, if not disabled by the value 0,
- (c) a “logical and” of all simulation stop conditions of the first kind, if any are specified,



- (d) a “logical or” of all simulation stop conditions of the second kind, if any are specified.

For stationary analysis it should be sufficient to apply measurement start conditions to reduce the influence of the initial state of the simulation, simulation stop conditions of the first kind to request a certain accuracy of the results, and a CPUTIME condition, that can limit the simulation in some way, so that it fits in your personal time table. Other stop conditions are less useful for stationary analysis.

- 8) How can I change a stop control in a stop control windows, e.g. from confidence level to events?

There is no direct way to change the kind of a start or stop condition. The only way to achieve this is to create a new condition and delete the old one. Please, note that there must at least be one condition in a start/stop window, so you cannot delete the last condition.

- 9) How does a CONTROL part of an experiment look like?

The CONTROL part is described in a chapter of the HI-SLANG reference manual. You should avoid %BIND directives, especially for predefined link names. A quite useful CONTROL part for a simulative experiment is the following:

```
%COMPILER
%PARAM = XREF,INDENT=|1

%ANALYZER
%PARAM = UPDATES,MINMAX

%END
```

The %END at the end is necessary, although earlier versions of HITGRAPHIC behave differently than described. For details of each option, please refer to the HI-SLANG Reference Manual.

- 10) Can I set the seed for random drawing?

The seed is an implicit parameter of each model type. It can be set with the actual parameters at an evaluation in the experiment window. As an example we regard a model type with one formal parameter:

```
arrival_rate : REAL
```

The actual parameters given for the corresponding evaluation including the setting of the seed may look like this:

```
2.5, LET seed := 23
```

In the context of experiment series, the use of the procedures “set\_seed” and “last\_seed” may be of interest. Have a look at the next solution.

11) How can I describe experiment series?

It is possible to describe experiment series within one experiment description. To clear up the main idea we regard an example. A model type has the following formal parameter:

```
arrival_rate : REAL
```

Simulative evaluations should be performed for a parameter range from 2.0 to 5.0 with step width 0.5 and for the values 10.0 and 50.0. For each parameter value two evaluations should be performed with different sequences of the random number generator assuming that its period is large enough.

The first step is to introduce a local variable in the experiment for each formal parameter of the model that should be modified, in our case:

```
VARIABLE act_arrival_rate : REAL;
```

This variable is used as actual parameter at the evaluation in the experiment, that corresponds to the model type. In our example the evaluation is called "eva\_mod". For the seed we use the predefined procedure "last\_seed". This guarantees that the random number sequence will not be reset, but continued to avoid overlapping as far as possible. The actual parameters at the evaluation are:

```
act_arrival_rate, LET seed := last_seed
```

The setting of the actual parameter via the local variable and the activation of the evaluation are described in the experiment body:

```
set_seed (13);

FOR act_arrival_rate := 2.0 STEP 0.5 UNTIL 5.01 LOOP
  EVALUATE eva_mod;
  EVALUATE eva_mod;
END LOOP;

FOR act_arrival_rate := 10.0, 50.0 LOOP
  EVALUATE eva_mod;
  EVALUATE eva_mod;
END LOOP;
```

The extension of this concept for multiple parameters or multiple evaluations should be obvious.

12) Why does the transformation to HI-SLANG fail?

There may be several reasons for a failure of a transformation. Firstly, nothing that is referenced by the experiment/aggregation description may be opened for editing. The referenced evaluations should contain evaluation objects. A

frequent error is to forget or misspell an evaluation name in the experiment body. Note that after a rename of an evaluation the name is not automatically changed in experiment bodies.

You get more information about your problem in the transformation errors window, that is popped up automatically (cf. Figure 4.19) displaying message(s) in case of transformation errors.

13) What can I do if I get compile errors from the HI-SLANG compiler?

Compile errors usually indicate errors in the HI-SLANG code you have entered in a text editor.

First of all, a lot of syntactical errors can be avoided early by applying the *check* operation in text editors for HI-SLANG.

If you still get errors from the compiler, you can apply the *errors* operation in the submenu of *show* on the experiment or aggregation in the environment window. Based on the information in the generated listing it tries to relate the errors of the HI-SLANG compiler to the corresponding elements of the specification in HITGRAPHIC (cf. Figure 4.22), where they usually can be corrected. You should note that a single error may cause multiple error messages.

If the automated reverse mapping is not successful, please try to apply the following scheme.

To localize the problem you open the *protocol* of the experiment. In one subwindow you find the list of errors, in the other a listing of the generated HI-SLANG code. You should position the listing to the line of the first remaining error. The context of the line, especially some automatically generated comments and page titles should help you identifying the part to be corrected. Then you have to open the corresponding HITGRAPHIC windows and correct there. After this you may continue with the next error.

If you have an old control part without `%END` at the end, this could be the problem (cf. question 9). This will not be corrected by the database format conversion. It will produce compilation errors of module FAN. A manual insertion of `%END` at the end of the control part cures the problem.

14) Can I add debugging code during the development phase of a component type?

First of all, if you use the simulative method, the trace options can be very useful. You can turn on the trace globally in the experiment or for selected components in the evaluation window. In the control part the format of the trace can be selected, e.g., by a line containing `%PARM = TRACEFORMAT=3`.

The trace output can be deactivated respectively activated by the predefined procedures `trace_off` and `trace_on`. But be careful, you are programming concurrent processes. If you turn on the trace output at the start of a service and turn it off at the end of the service, you cannot be sure that all events from start to the end of a service execution are traced, because in the meantime another process may have turned off the trace output.

Additionally, the tracefile is accessible from within HI-SLANG by the predefined procedure `tracefile`. You can use `writeln` statements to add additional

output to the tracefile, for example the value of variables or the state of the component.

But before you start to add debugging code you should think about a strategy. A simple solution is to add the code and delete it if the component is successfully tested. A more sophisticated strategy is to use conditional compilation. For each component type you define a compiler switch whose name is composed by the prefix `DEBUG_` and the component type name. The debugging code is only compiled if the switch is set:

```
%IF DEBUG_component_type_name THEN
    trace_on;
    WRITELN FILE tracefile, ... ;
%FI
```

The switch can be set in the control part of the experiment after(!) the `%END` by a `%SET` directive. The advantage of this strategy is that the component can remain unchanged after the tests and the debugging code is still available if problems occur later.

#### 15) The HI-SLANG listings are getting too large!

This may be caused by the multiple use of component types within different component types. These component types are generated and listed for each component type that contains an object of that type. There is an opportunity to control the generation of the listing by compiler directives.

As an example we regard the large body of a service called “compute”, that should only be listed once. By convention we use a compiler switch named “service\_compute\_listed”. This name should be unique within the complete model. Adding the following compiler directives at the begin and the end of the body will result in the desired listing output:

```
%IF NOT service_compute_listed THEN
%SET service_compute_listed
%ELSE
%NOSOURCE
%FI

    < large body of service "compute", HI-SLANG >

%SOURCE
```

#### 16) Why does my database require so much disk space?

The most disk space consuming parts of the database are output files of the HIT system, especially traces and listings. A hint to reduce listing size is given in the previous solution. If you must be economical with disk space, delete experiments as soon as possible, use the *move to file* operation for output files you do not need any longer.

17) Results are undefined!

There may be several reasons for an undefined estimator. Have a look on the solver information automatically appended at the listing if not explicitly disabled via `%PARM = NOSOLVERINFO` in the CONTROL part. After the “evaluation trace” you find “information about estimators”.

For example no confidence interval is given, if the confidence interval is wider than the mean value itself. Or the estimator mean is undefined because the measure interval has length zero.

For more information concerning the solver information please confirm to the HI-SLANG Reference Manual.

In case of simulation a first step to solve your problem is to get information about the number of updates for the corresponding stream. You should use the control part of the experiment to demand for the number of updates:

```
%PARM = UPDATES
%END
```

Of course, the number of updates depends on the model behaviour, the stream definition, and the load filtering hierarchy, but usually the measurement start and stop condition at the evaluation object and all simulation stop conditions controlling the experiment run should be considered.

Check whether the measurement start condition has become true too late or the measurement stop condition has become true too early. If the measurement stop condition has not become true you may increase the experiment run length by changing the simulation stop conditions at or in the evaluation objects or in the experiment window. Again have a look on the solver information for the reason of the end of the experiment.

Note that all the conditions, measurement start, measurement stop, and simulation stop, are evaluated independently. For example, if you have set the measurement start condition to a high modeltime or event value the updates for the estimation of the result are not regarded before the start condition becomes true. For a measurement stop condition, e.g., the width of a confidence interval, all the data is taken into account. So it may be possible that the stop condition becomes true before there is enough data for the estimation.

Please, note that in a stationary analysis there are usually no reasons to use measurement stop conditions.

18) Can I get intermediate results of a simulation?

The standard component type “observer” is designed for this purpose. Please, read its information.

If you want to use an “observer” in interactive mode, you should additionally select *SYSOUT* for output in the experiment window. Your xterm should be configured with a scrollbar and a sufficiently large scrollbuffer (via resources of X).

# Part II

## Tutorial







# Chapter 7

## A Tutorial

### 7.1 Introducing Remarks and Perspective

A single example will be used throughout the tutorial to clarify the most important features of a HITGRAPHIC session. For reasons of readability and clearness only main concepts and functions are depicted in order to provide for a certain “feeling” for the work with HITGRAPHIC. The Chapters 2 and 3 of the reference part should be read before starting the tutorial. For information about text editing you can consult Section 4.5.

The tutorial is structured into two main parts. The first one deals with model specification, presenting refinement steps of a sample model. The second one is about evaluation and experiment specification based on the previously specified models.

The example goes as follows: Some company considers the introduction of computer support for its office work. Considerations start at supporting certain decisions and their preparations. Basically, one such decision activity is planned, per day per department. The company is, in a restructuring phase, moving from five towards ten departments; eventually twenty departments are to be catered for. The activities under consideration each consist of preparing three initial documents and subsequently joining these documents into some master report, which is later examined and eventually finished off.

### 7.2 How to Start a HITGRAPHIC Session

The first steps to create your own HITGRAPHIC environment are described in Chapter 2. If you have not yet created a database, please go to that chapter and return, if you have one.

The syntax of the HITGRAPHIC call is given by

```
hitgraphic <your database>
```

HITGRAPHIC prompts with an empty environment window and the startup window (cf. Figure 7.1).

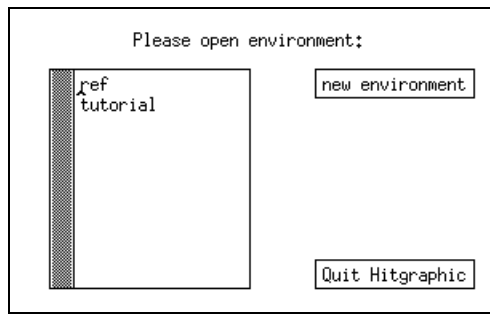


Figure 7.1: Startup Window of a HITGRAPHIC Session

In a subwindow the list of available environments of the specified database is displayed. You are required to select your working environment. The selection follows the general principles of operations as introduced in Chapter 3.

If you start with a new database, the list of available environments is obviously empty.

## 7.3 Examples of Model Specification

### 7.3.1 Model 1

After having started HITGRAPHIC (cf. Section 7.2) the name of the working environment is requested. Since a new environment should be created for the examples, the button *new environment* is selected with the left mouse button. As a consequence of this selection a dialog box is popped up allowing to enter the necessary name of the new environment (cf. Figure 7.2). By clicking the *OK* button, the displayed text string is confirmed and (if syntactically correct) accepted. The environment window is redrawn (cf. Figure 7.3) and the modelling may begin.

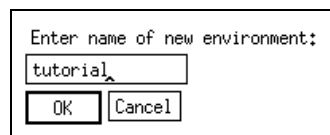


Figure 7.2: A Dialog Window

If there already exist environments and if you know the name of your current working environment, you may abbreviate the startup procedure by calling the program with two parameters:

```
hitgraphic <your database> <your environment>
```

In this case the environment window is displayed immediately without the startup window.

The new model type “office\_1” is created by performing the *new model type* operation on the label MODEL TYPES. After entering the mentioned name and clicking the *OK* button in the dialog window, the name of the new model type is displayed in the list of model types.

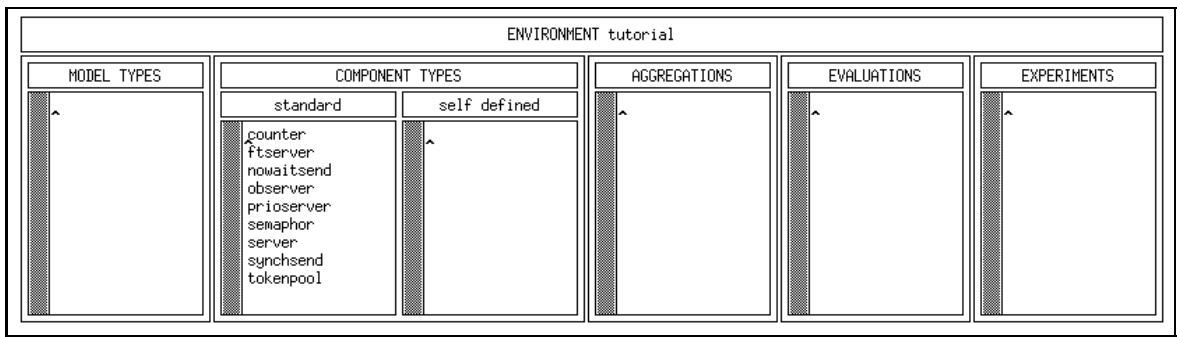


Figure 7.3: The Environment Window

To protect the new model type against unauthorized changes by other users, the next step is to perform the *mode* operation on the model type. A dialog box is displayed showing the current mode (*free*) and the current owner (no owner). The dialog box is given in Figure 7.4. To protect the model type the button *locked* is selected and confirmed by clicking the *OK* button. As a result, your login name is set for the mode *locked* as depicted in Figure 7.5, indicating that other users may *open* this model type in a read mode only.

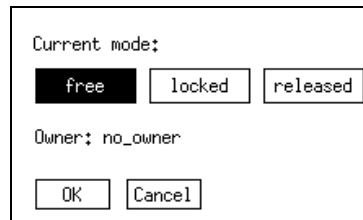


Figure 7.4: The Free Mode

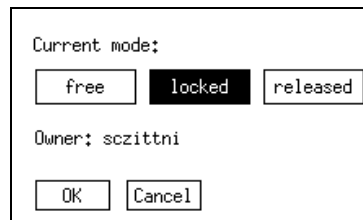


Figure 7.5: The Locked Mode

Up to now the preparatory steps are finished and the modelling of the model type “office\_1” may start. Performing the *show survey* operation on the model type pops up the survey window. It displays a single box labelled “office\_1” for the object structure as well as for the type structure.

To modify the model type “office\_1” the *open* operation on the model type in the environment window has to be performed. Then the component type graphic window appears (which is opened for model types as well as for component types). The “empty” model type is displayed in form of an empty graphical area with the activities symbol being located at its lower left corner.

The present total scenario is given in Figure 7.6.

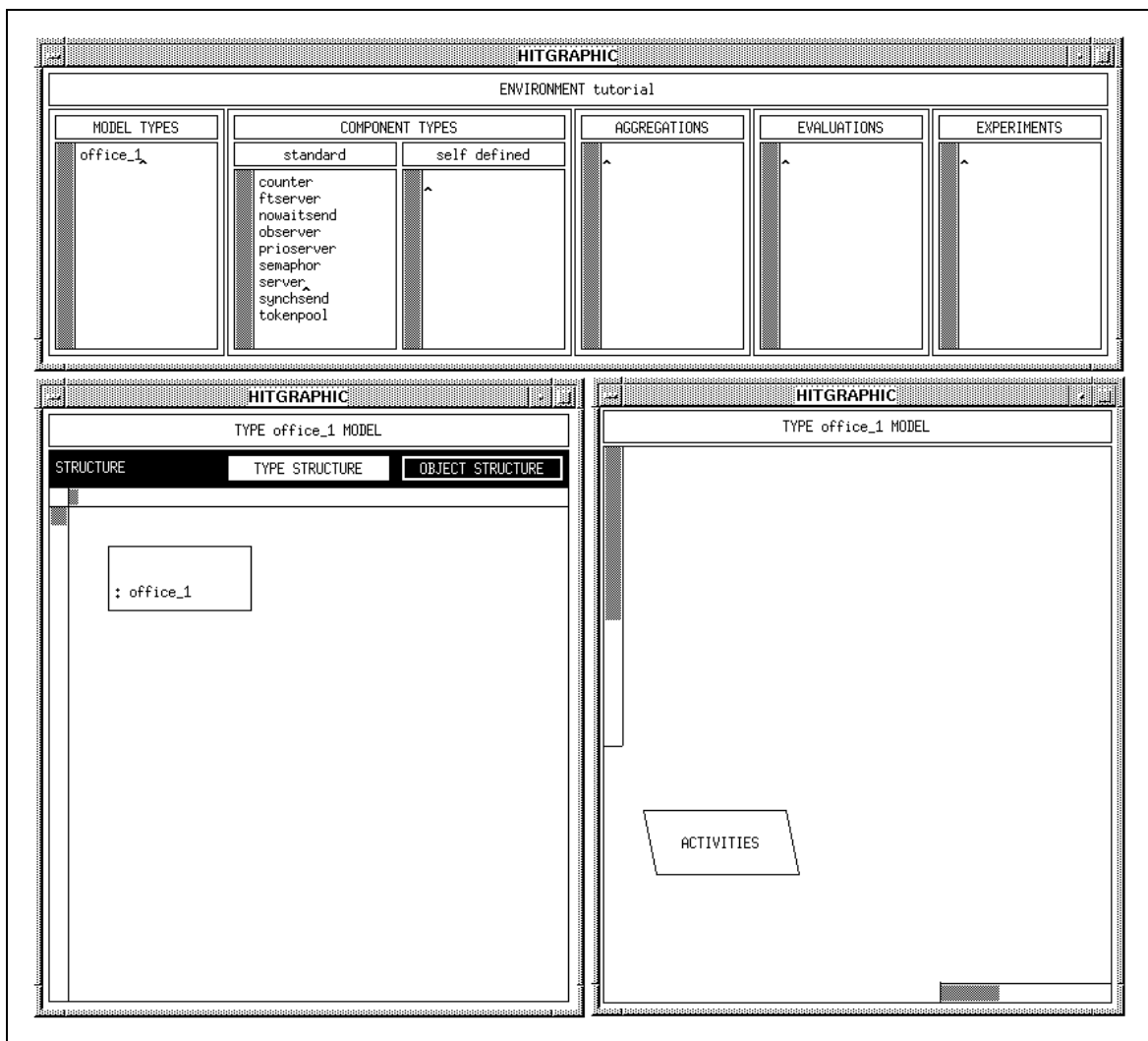


Figure 7.6: A HITGRAPHIC Scenario

Keeping the rough example description in mind, the first step to be performed in the component type graphic window is the creation of a service “report”, since the load of the office is given in terms of reports. Selecting *new service* from the background menu and giving the service the name “report” results in the first symbol to be displayed, which is the service “report” (cf. Figure 7.7).

The load is known at this point and additionally it is assumed that the services representing the load will not be parameterized. For these arguments it is reasonable now to think about how to generate the load. Following these thoughts immediately leads to the *opened* activities of the model type.

The text editor corresponding to the activities of model type “office\_1” is depicted in Figure 7.8 showing the statement which is responsible for the load to be CREATED. It is assumed that there are always 20 reports to be worked on. Having entered the text it is good practise to activate the *check* operation on the ACTIVITIES bar. This operation gives hints about errors or problems of the current HI-SLANG code. In our case it is a message about the locally unknown identifier “report”. Since this cannot be avoided, editing can be finished and you initiate a *save all & quit* operation on

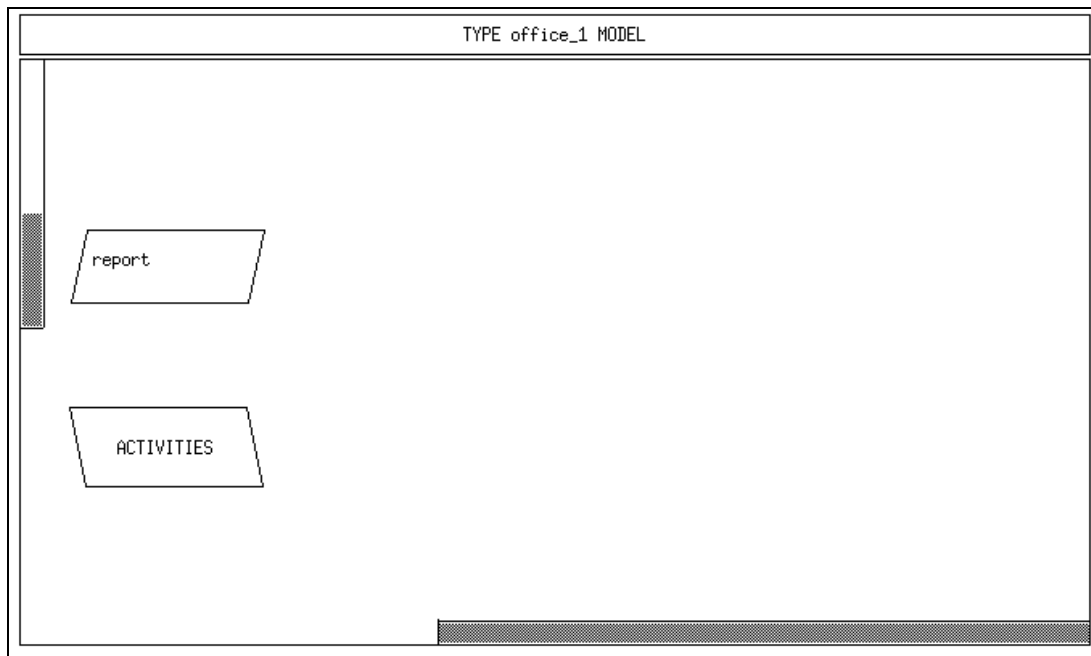


Figure 7.7: The Service “report” is Created

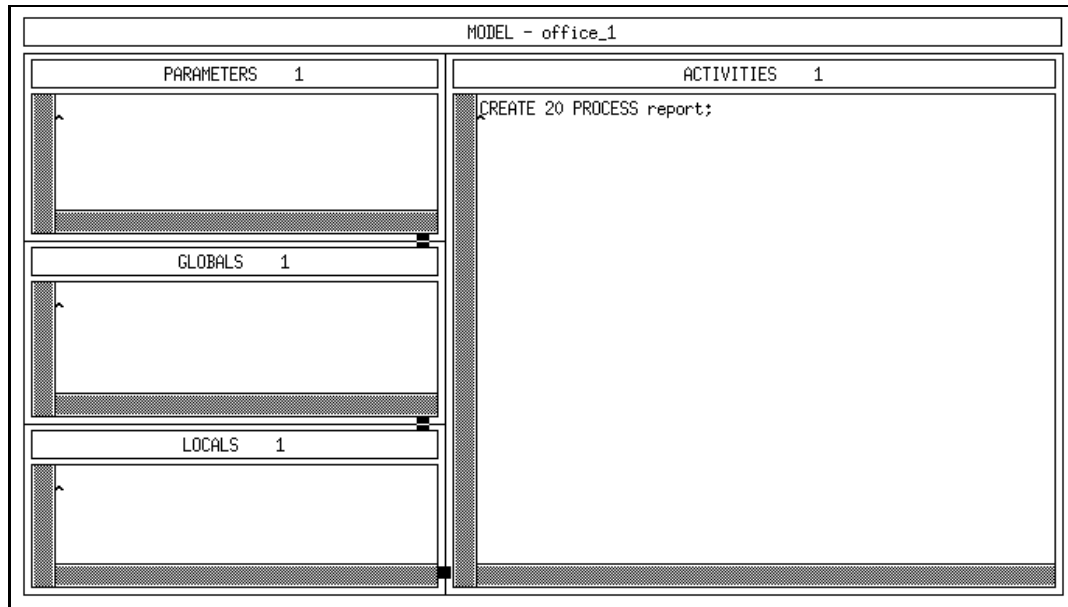


Figure 7.8: The Activities of “office\_1”

the title bar of the editor window. The text editor will then disappear.

So far it is known, that a certain load is created. But it is still unknown what the load will do, in terms of HIT, the load pattern of, i.e., using services (being provided somewhere by somebody else) needs to be specified. From the rough description of the sample model (cf. Section 7.1) four different subactivities are obvious, which will be named “prepare”, “join”, “examine” and “archive”. Following the structuring principles the used services will be declared first, then their usage within the service “report” will be given.

Performing the *new used service* operation on the service “report” yields a new used

service, which is named “prepare” in the usual manner (cf. Figure 7.9).

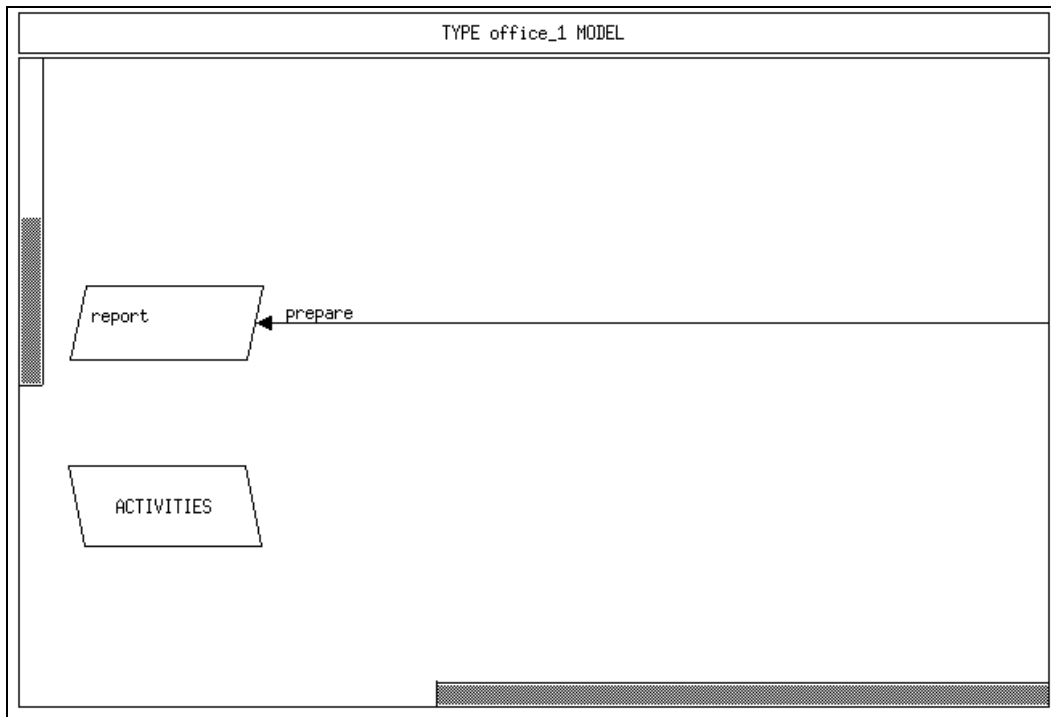


Figure 7.9: The Used Service “prepare” is Created

Before this service is effectively used a thought should be addressed to the way of usage, especially the parameters of “prepare”. Following the rough description of the office example (and remembering that at first a simple model is to be developed) the used service “prepare” as well as the (still missing) “join”, “examine” and “archive” models a simple time consumption.

Figure 7.10 depicts the corresponding formal parameter editor. It is opened via the *formal parameters* operation on a used service and allows the specification of parameters and results.

The advantage of completely specifying the used service “prepare” becomes obvious when the still missing used services are to be generated. Since they have identical formal parameters (in the model type “office\_1”) a simple *copy* on “prepare” is sufficient not only to generate each of them, but to copy the formal parameters automatically as well. Repetition of the *copy* operation and applying the *move* operation lead to the component type graphic window as shown in Figure 7.11. After activating a *move* on a used service the new position is selected with the left mouse button.

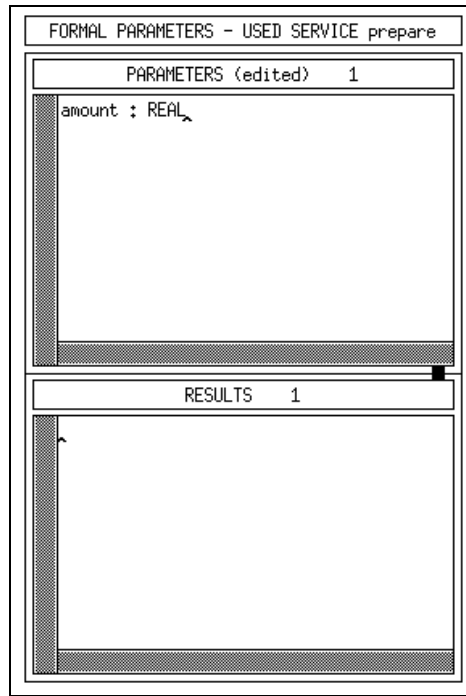


Figure 7.10: Formal Parameters of “prepare”

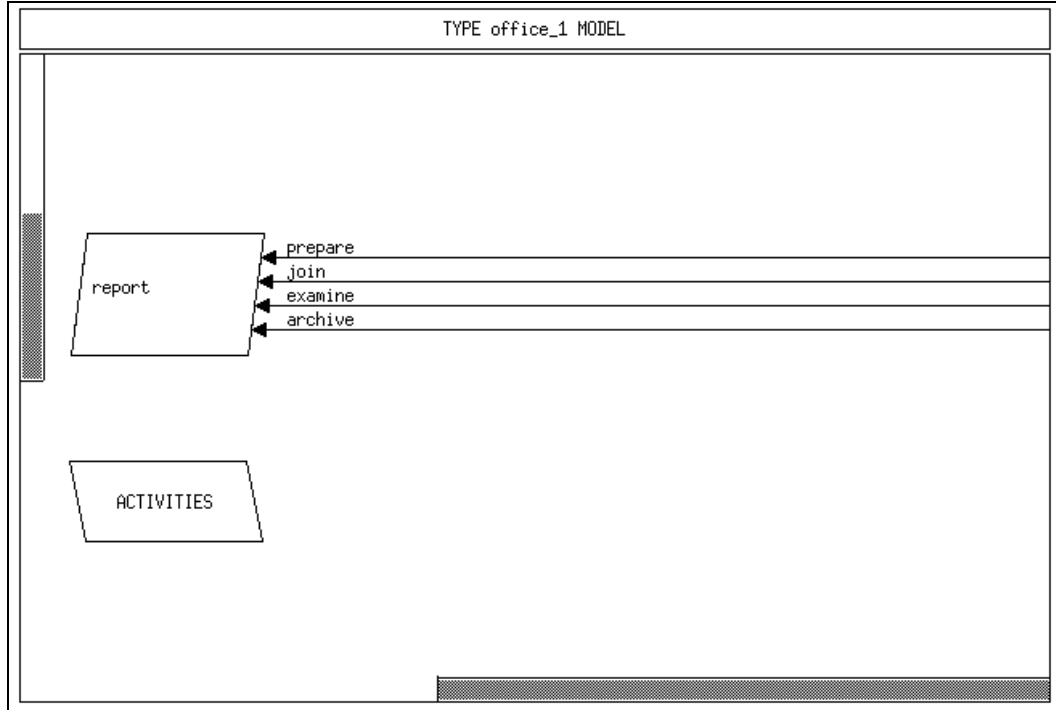


Figure 7.11: The Completed Service “report”

The final step of the load description is to specify what the service “report” will do, in other words editing the body of the service “report”. The statements shown in Figure 7.12 stem from the assumption that a report consists of three subdocuments. The subdocuments are prepared separately one after another, joined, examined and archived, where each of these steps takes a certain amount of time.



Figure 7.12: The Body of Service “report”

At this point the specification of the load is finished. Two parts of the specification of a model/component type are still missing, the specification of the machine and the reference between load and machine.

Starting with the description of the machine, the question arises which components of which component type are to be instantiated. The answer is simple, two service stations of type “server” (named “secretaries” and “clerks”) are available in “office\_1”. The standard component type “server” provides a single service “request”, which models a time consumption.

The instantiation of the “secretaries” proceeds as follows: the operation *new component* of the background menu of the component type graphic window is applied first. A confirm box pops up requesting a component type to be selected. The cursor is moved to the required component type (here: “server”) in the environment window, where the left mouse button is clicked. To confirm this selection the *OK* button inside the dialog box is pressed. After having named the new component (here: “secretaries”) it is drawn in the component type graphic window including its provided services. They are graphically represented as towards-the-top-pointing arrows. Repeating this procedure for the component “clerks” leads to the scenario depicted in Figure 7.13.

In the model type “office\_1” the components of type “server” are instantiated as infinite servers without any competition for resources, which is the default instantiation



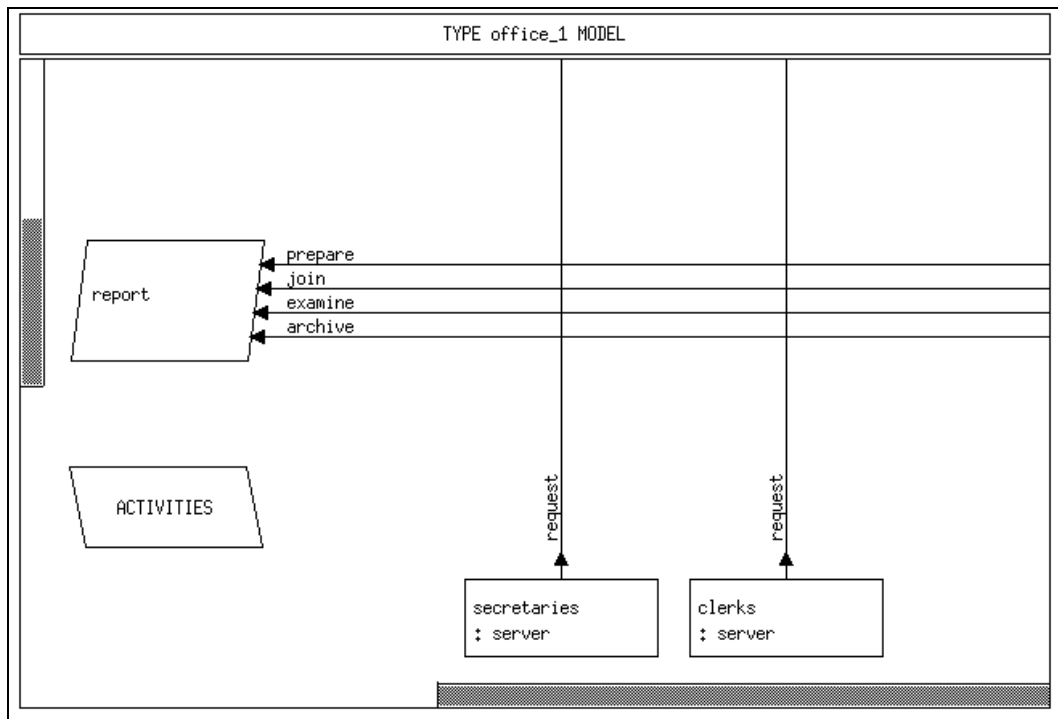


Figure 7.13: The Machinery is Inserted

for components of type “server”. For this reason no specification of control procedures is necessary.

The final points to be made concern the reference of load and machine. By placing a dot at the crossing of a used and a provided service these are bound, the used one being associated with the provided one. The way to bind services is to click the right mouse button at the corresponding crosspoint and perform the *set* operation.

In the example “prepare” and “join” are to be carried out by secretaries, so they are bound to the provided service “request” of component “secretaries”. The used services “examine” and “archive” are bound to the “request” of component “clerks”. The resulting graphical representation of the model type “office\_1” is given in Figure 7.14.

Note that each used service may at most be bound to one provided service, whereas a provided service may be bound to multiple used services.

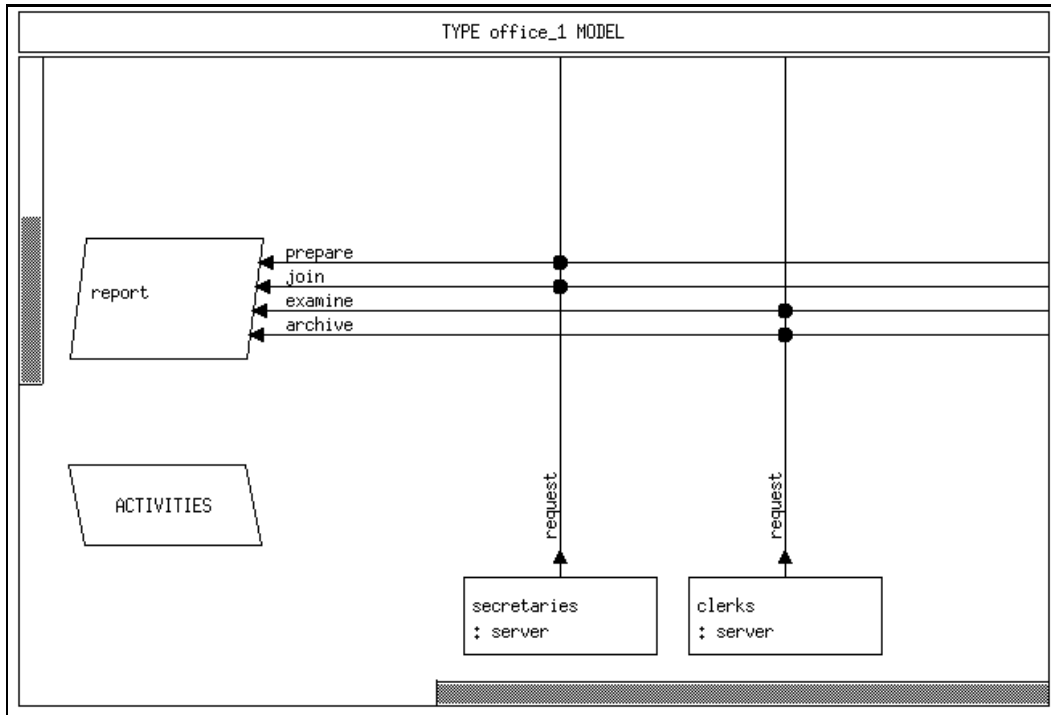


Figure 7.14: The Model Type "office\_1"

The specification of the model type "office\_1" is finished at this point. A final application of the *save* operation in the title bar popup menu followed by a *quit* finishes the work within the component type graphic window and closes it.

To get an updated view in the survey window, an activation of *refresh* in the title bar is necessary. The new type and object structures are given in Figures 7.15 and 7.16, respectively.

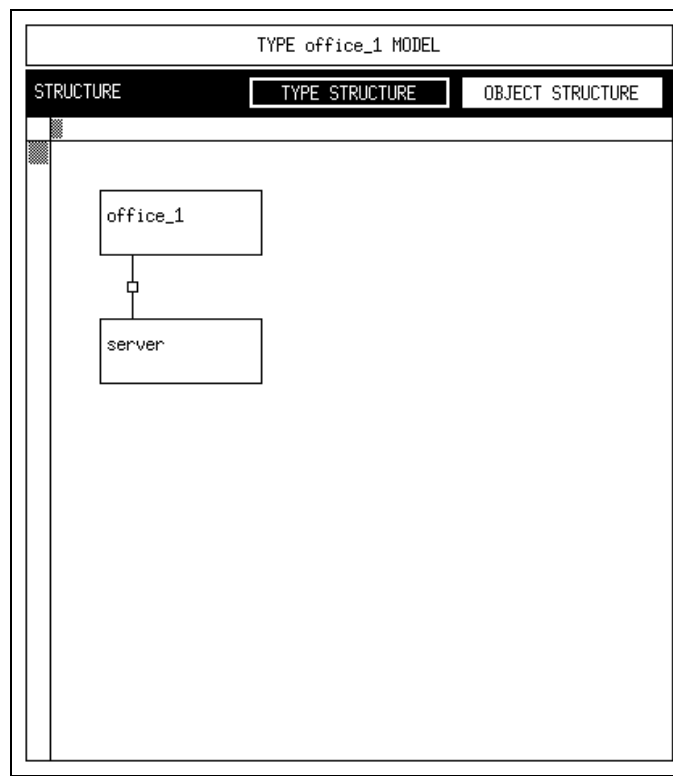


Figure 7.15: Type Structure of Model Type "office\_1"

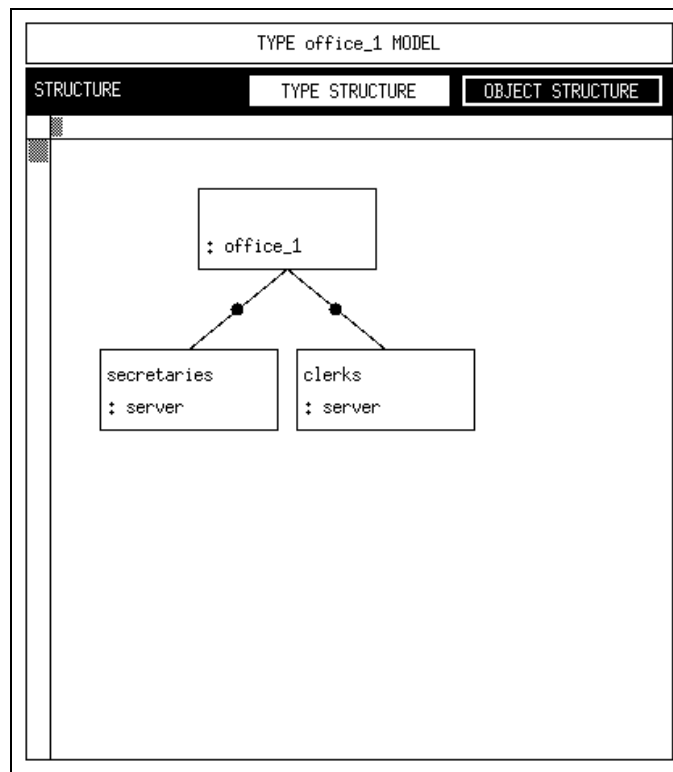


Figure 7.16: Object Structure of Model Type "office\_1"

### 7.3.2 Model 2

The first step towards a more detailed model is to refine the machinery doing the work of the secretaries. For this purpose a new component type "handling\_2" (the

number refers to the example) is created. As far as the necessary user actions were already described in the preceding section they will not be repeated.

Create and open the new component type “handling\_2” in the column of self defined component types. In the first model it was the secretaries’ job to “prepare” and “join” the subdocuments. A new service named “edit” is to be created for this purpose. A component of type “handling\_2” is providing the service “edit” to be used from outside. For this, we select the menu item *provide* in the service menu. For reasons which will be given later, this service is left parameterless. Instead of just modelling a simple time consumption, the service “edit” is refined. It uses the four services “fetch”, “think”, “change” and “store” as shown in Figure 7.18 and Figure 7.19.

Now these four used services model a time consumption (parameter amount: REAL), too, but the time is spent on a lower level. Some pages of a document are to be modified one after another. So each page is fetched via a communication link first. After thinking some time, some minor or major changes are made, until the document is stored (cf. Figure 7.19). From this description the necessary machinery can be concluded in a simple manner (cf. Figure 7.18 and Figure 7.20).

Now thoughts should be addressed towards the procedures controlling the components. Up to now all components were instantiated as infinite servers to model the fact that enough secretaries respectively clerks are available, one for each document. This assumption does not hold for the communication link, since it is not owned by a single secretary/clerk but used by all of them. So the component “link” should be instantiated with a processor sharing discipline. For this reason the *control procedures* operation is applied on the component “link”. After confirming the *save* operation the control procedure window pops up allowing the desired specification. To specify the processor sharing discipline the dispatch strategy is set to “shared” via the corresponding popup operation on the label *dispatch*. The resulting control procedure window is given in Figure 7.17.

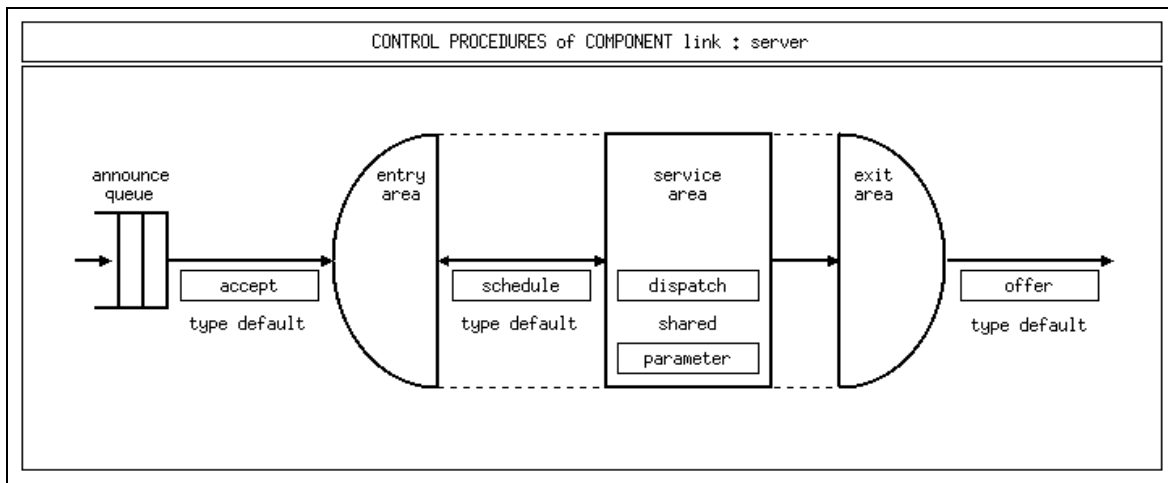


Figure 7.17: Control Procedures of Component “link”

The description of the completion of the component type “handling\_2” is omitted here, the final result is given in Figures 7.18, 7.19, 7.20.

The refinement of the secretaries’ job results in a new component type, which is to be embedded in the office model. As a first step the model type “office.1” is

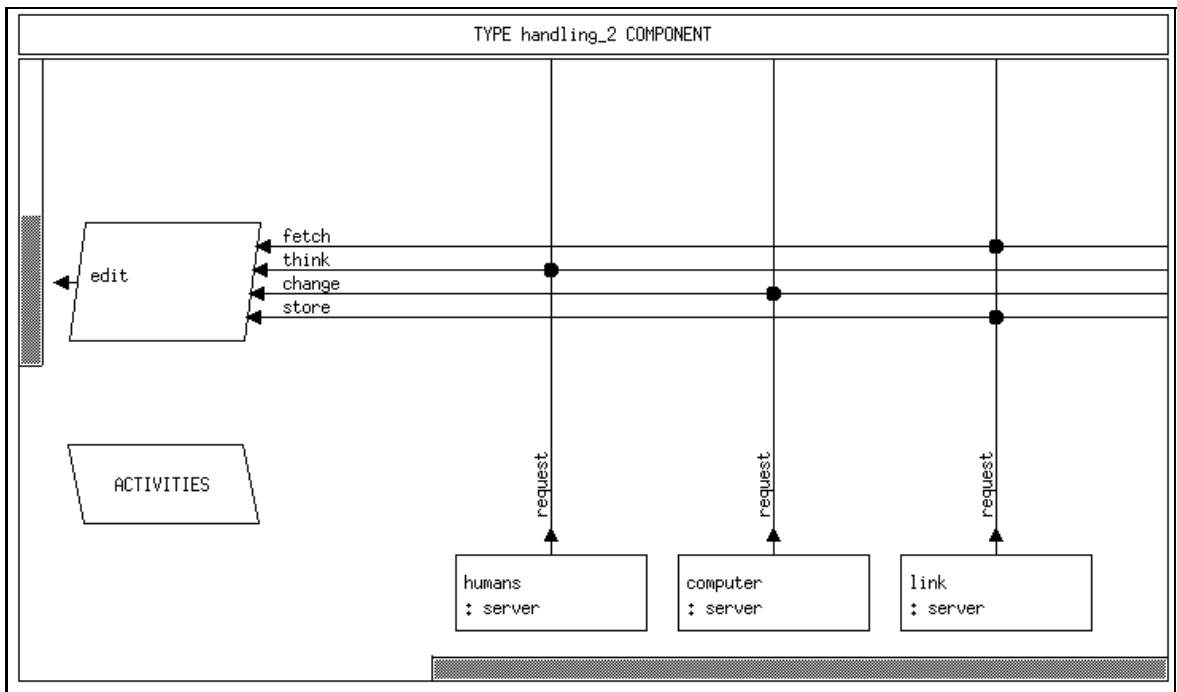


Figure 7.18: The Component Type “handling\_2”

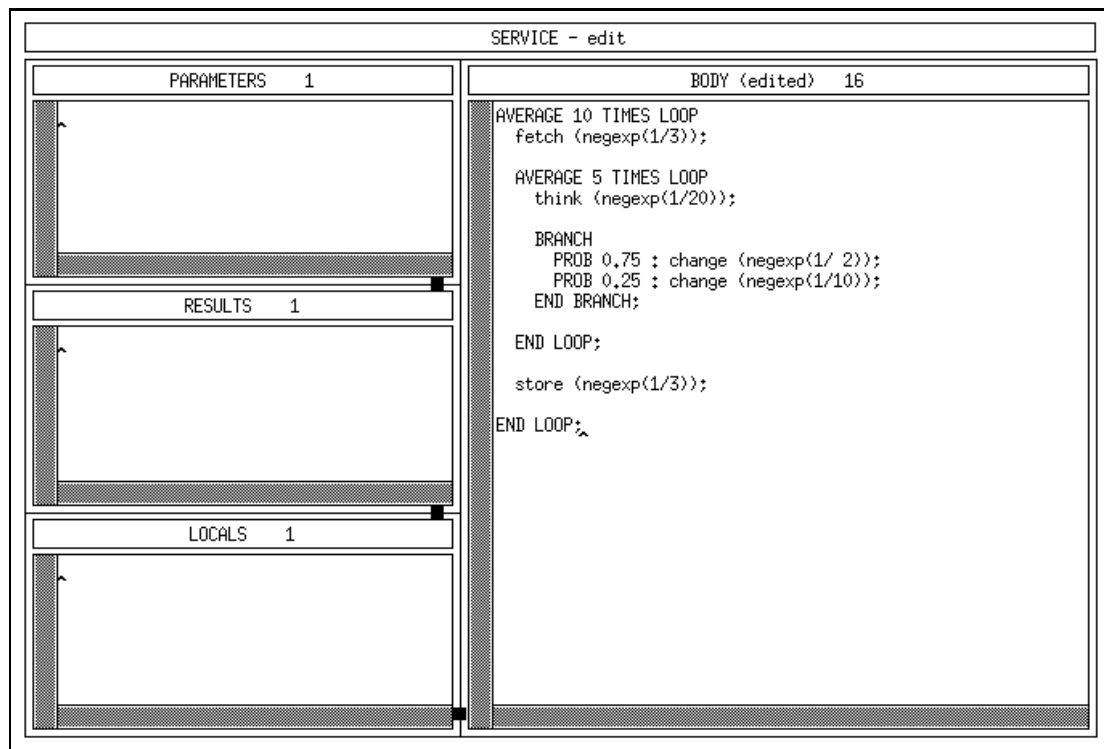


Figure 7.19: The Body of Service “edit”

copied within the environment window, the new model type is named “office\_2”. The following changes are to be performed on “office\_2” in order to refine the secretaries’ job.

Starting with the components, the existing component “secretaries” of type “server” is deleted by performing and confirming the *delete* operation on it. Instead, a new

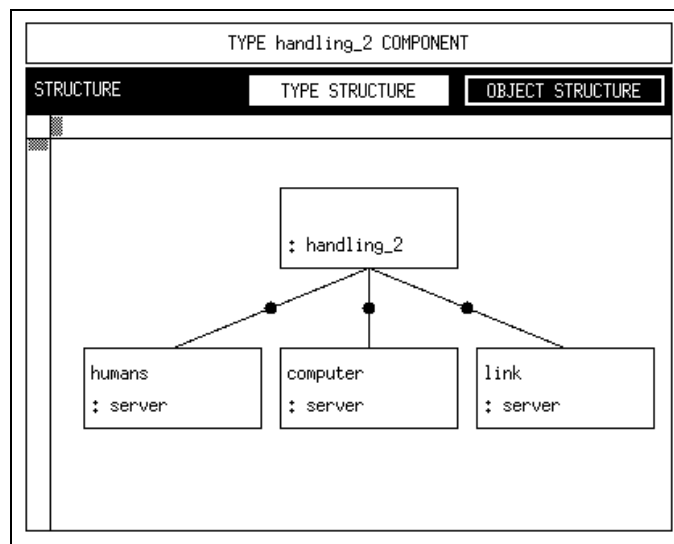


Figure 7.20: Object Structure of “handling\_2”

component “secretaries” of type “handling\_2” is instantiated. The used services “prepare” and “join” are referred to the provided “edit” service of the new component “secretaries”. Using *set with transfer* instead of *set* will transfer the parameters of the provided service to the parameters of the used service. Because “edit” has no formal parameters the formal parameters of the used services are deleted after a confirmation.

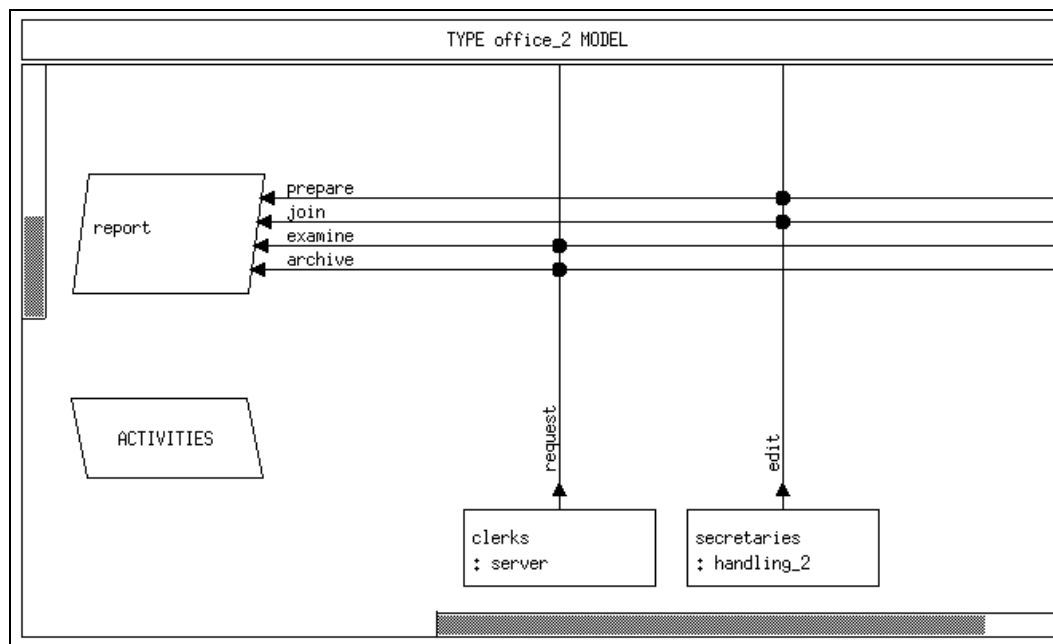


Figure 7.21: The Model Type “office\_2”

Stopping at this point would cause an error when compiling the generated code, since the load pattern within the body of service “report” must be changed, too. The actual parameters at the calls of the used services “prepare” and “join” have to be deleted, e.g., by selection and a Ctrl-w keystroke. After this is done, the specification of the model type “office\_2” is finished. The corresponding component type graphic window

as well as the resulting type and object structures are depicted in Figures 7.21, 7.22 and 7.23.

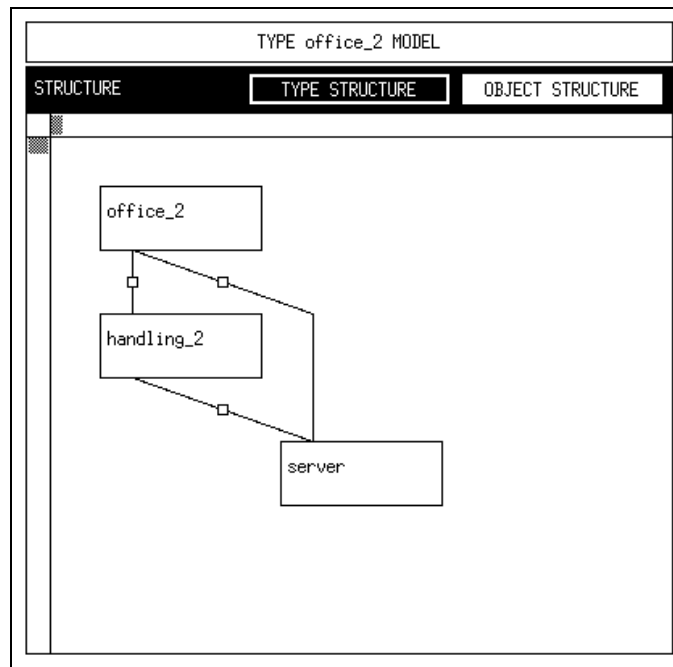


Figure 7.22: Type Structure of "office\_2"

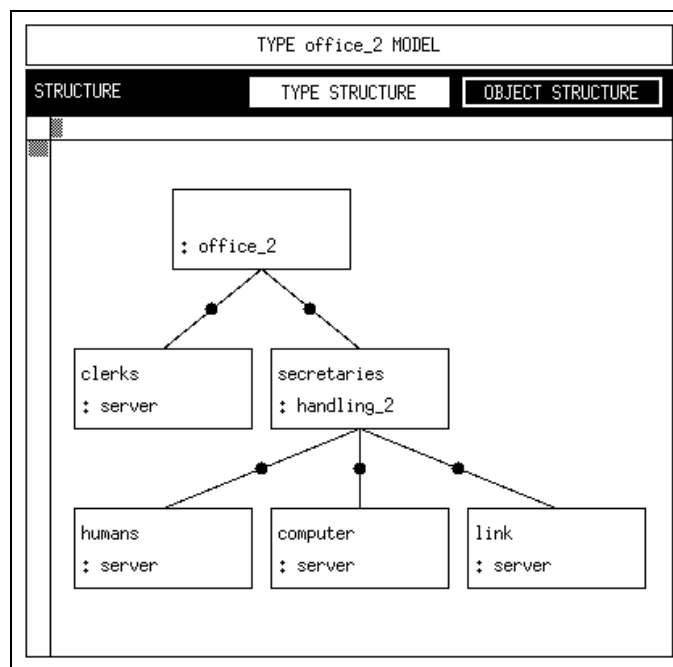


Figure 7.23: Object Structure of "office\_2"

### 7.3.3 Model 2a – Aggregation

The main subject of the last section was the refinement of the component “secretaries” of type server, which led to the component type “handling\_2”. Remember, that the service “edit” was refined from a simple time consumption to the use of four used services within nested loops (cf. Figure 7.19). This refinement follows the top-down development style, which is also widespread within the modelling discipline.

From the viewpoint of analysis techniques, this refinement is of no advantage. Analytical techniques have to face more stations and classes, which makes numerical solutions worse especially. Simulation has to deal with used service calls within nested loops instead of a single time consuming call. Hence, the runtime is expected to grow.

For these reasons, it is sometimes desirable to reduce the complexity of analysis by aggregating some submodels. If certain requirements hold, an aggregation of component types is accurate. In other words a flow-equivalent representation may be constructed, that exactly provides the same results as the original component type (concerning time consuming service calls/delay).

Considering the component type “handling\_2”, the requirements from the theory of aggregation are fulfilled. The automated aggregation of “handling\_2” is the topic of this section. The necessary steps for specification and execution as well as integration into the model type will be discussed next.

The first step is to create an aggregation description within the environment window. Selecting the entry *new aggregation* from the corresponding title bar displays a confirm box with the text “Please select component type”. After selecting “handling\_2” from the component type list and clicking the *OK* button, the new aggregation is inserted in the aggregation list. Aggregations are always named as the original component type, since they are just substitute representations. This principle of naming is performed automatically by HITGRAPHIC, even when the entry in the component type list is renamed. From now on, the component type entries are referred as original component types, whereas the aggregations are referred as aggregates or aggregated component types (after performing the aggregation).

Performing the *open* operation on the aggregation “handling\_2” opens the aggregation description window. This window displays the provided services of the original component type, as they are displayed in the component type graphic window. It allows for the specification of the maximum population of each service, for which the aggregation is to be performed.

In our example, 20 processes are generated. For this reason, it is sufficient (for the given framework) to specify a maximum population of 20 (cf. Figure 7.24).

At this point the aggregation specification is completed.



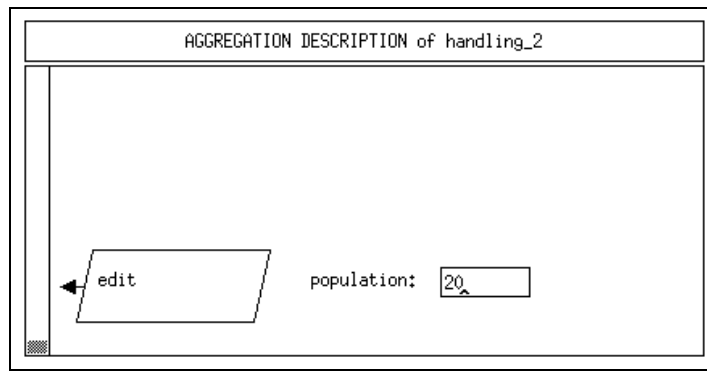


Figure 7.24: The Aggregation Description Window of “handling\_2”

The next step is to perform the aggregation with HIT. This is done by selecting the entry *transform & run* from the popup menu of the aggregation “handling\_2” within the environment window. When the transformation (source code generation for HIT) is finished, the HIT run control window appears. Here you start the aggregation with the *run* operation on the title bar popup menu. A terminal window is displayed to show the progress of the HIT call. After the execution of these steps, the terminal window disappears and the aggregation is well done. Selecting the *show ->* entries of the popup menu allows the user to inspect some information concerning the aggregation execution:

- protocol*: the protocol of the aggregation execution, containing all messages of the HIT system in the upper part and the source listing of the HI-SLANG code used for the aggregation in the lower part (cf. Figure 7.25);
- aggregate*: the source code of the aggregate, which is the substitute representation as a load-dependent server, given in HI-SLANG code (cf. Figure 7.26).

```

AGGREGATION PROTOCOL - handling_2

STANDARD OUTPUT 1

> PASS 3      :   Okay.                Cpu Time used :    0,050 Seconds.
> ACG        :   Okay.                Cpu Time used :    0,250 Seconds.
> T O T A L  :   Okay.                Cpu Time used :    1,750 Seconds.

                                Compile Rate : 88,439 Lines/Sec.
Lund Software Simula Runtime System Revision 4.13 Pool size 4096 K bytes.
HIT Our Own Licence Uni Dortmund, Inf.IV ends 95-12-31
HIT Version 3,4,167 HIT-Analyzer DOQ4 of 95-07-14 10:35

Please enter name of Analyzer CONTROL file:
/tmp/hitgr_0/agg.hit

> FAN        :   Okay.                Cpu Time used :    0,030 Seconds.

ANALYTICAL DOQ4: Exact evaluation of separable queueing model using CONVOLUTION
TECHNIQUES.

> DOQ4       :   Okay.                Cpu Time used :    0,160 Seconds.
> T O T A L  :   Okay.                Cpu Time used :    0,450 Seconds.

LISTING 1

12 18a: {DECLARATION OF COMPONENT TYPE handling_2}
13 19a:
14 20a: TYPE handling_2 COMPONENT;
15 21a:
16 22a: PROVIDE {of component/model type handling_2}
17 23a: SERVICE edit ;
18 24a: END PROVIDE; {of component type handling_2}
19 25a:
20 26a:
21 27a: {DECLARATION OF SERVICE edit}
22 28a: {WITHIN COMPONENT/MODEL TYPE handling_2}
23 29a:
24 30a: TYPE edit SERVICE;
25 31a:
26 32a: USE {of service/procedure edit}
27 33a: SERVICE store (
28 34a: time : REAL
29 35a: )
30 36a: ;
31 37a: SERVICE change (
32 38a: time : REAL
33 39a: )
34 40a: ;

```

Figure 7.25: Protocol and Listing of the Aggregation “handling\_2”

```

AGGREGATE - handling_2  1
^B
% Experiment Name      aggregation
% Component Type      handling_2
% Used Method          ANALYTICAL DQQ4
% Date of Compile     1995-07-17      Time of Compile   09:59
% Start Date of Run   1995-07-17      Start Time of Run 10:00

TYPE handling_2 COMPONENT;
PROVIDE SERVICE
edit;
END PROVIDE;

TYPE edit SERVICE;
END TYPE edit;

END TYPE handling_2;

% SPEEDS
NUMBER OF PROVIDES      1
edit                    POPULATION      20
7.936508E-004 7.918552E-004 7.899064E-004 7.877860E-004 7.854734E-004 7.829
7.700614E-004 7.659573E-004 7.614048E-004 7.563460E-004 7.507168E-004 7.444
7.114843E-004 7.009234E-004

% Stop Date of Run     1995-07-17      Stop Time of Run  10:00
% Used CPU Time        0.16000

```

Figure 7.26: The Source Code of the Aggregate “handling\_2”

The final step is the integration of the aggregated component type into the existing model type. To guarantee the availability of both model versions, the model type “office\_2” is copied to the model type “office\_2a”. The changes that are necessary to use the aggregate instead of the original component type are quite easy. They are performed within the component type graphic window of the model type “office\_2a” (cf. Figure 7.27), which is opened in the environment window.

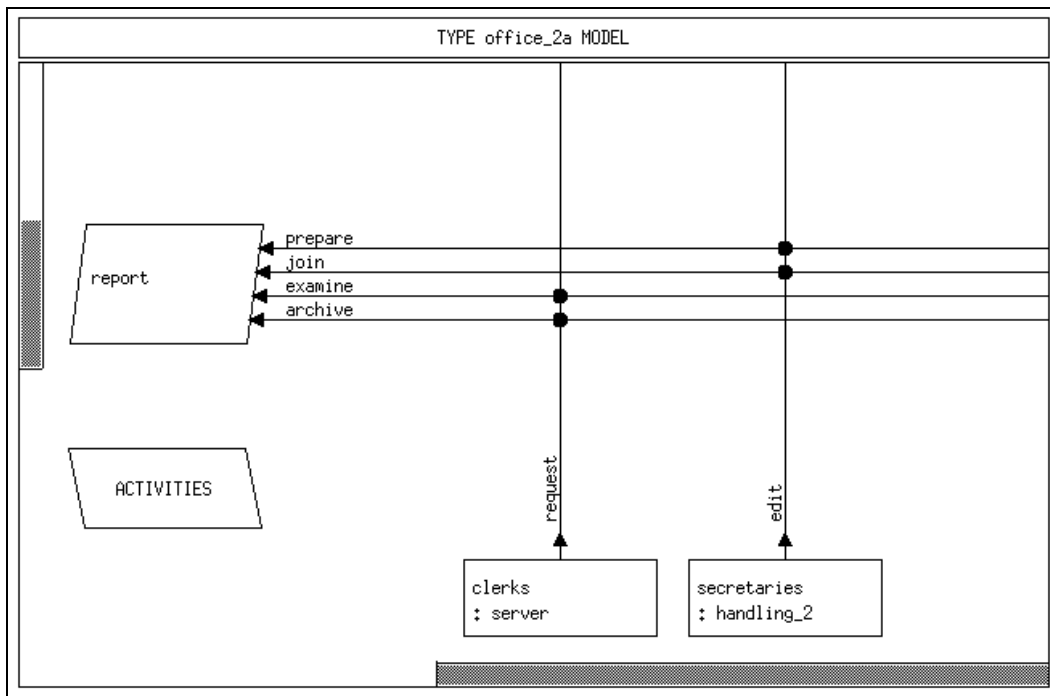


Figure 7.27: The Copied Component Type “office\_2a”

All that is needed is to perform the *use aggregate* operation on the component “secretaries” of type “handling\_2”. To represent the use of the aggregate graphically in the component type graphic window, the string “AGGREGATED” is displayed underneath the corresponding component (cf. Figure 7.28). Switching to the aggregated type causes the control procedures to be reset to default values (accept := always, offer := all).

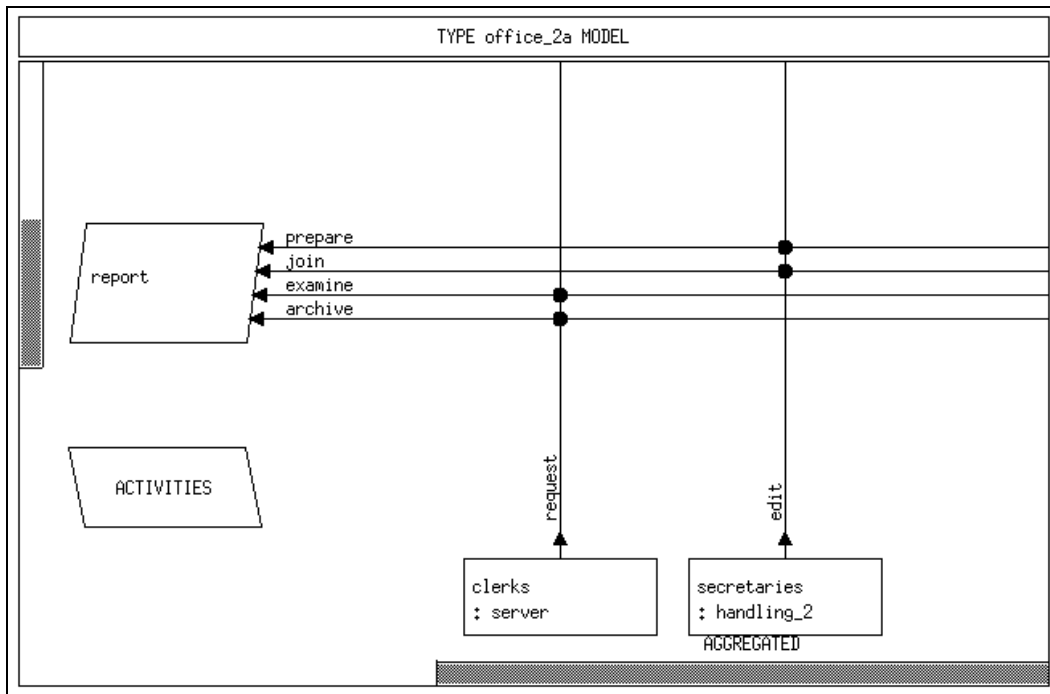


Figure 7.28: The Component Type “office\_2a” Using the Aggregate

After the *save & quit* operation, the refreshed type structure of the survey window changes as illustrated in Figure 7.29. In contrast to Figure 7.22, which displays the type structure of the model type “office\_2”, the line between the component types “handling\_2” and “server” disappears now.

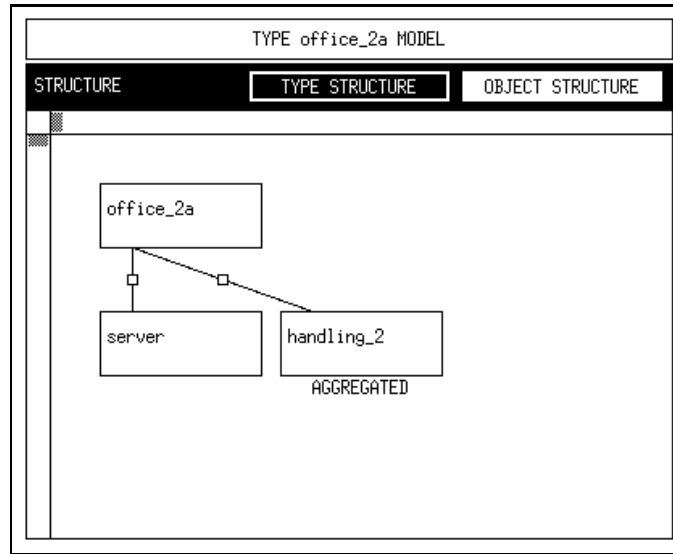


Figure 7.29: The Type Structure of “office\_2a”

The reason for this changed type structure is, that the substitute representation of “handling\_2” does not have the structure of the original type, the internals are hidden due to the aggregation.

Within the object structure of the survey window the aggregated component is displayed in the same way as in the component type graphic window. Again, the string “AGGREGATED” is displayed underneath the corresponding component symbol (cf. Figure 7.30).

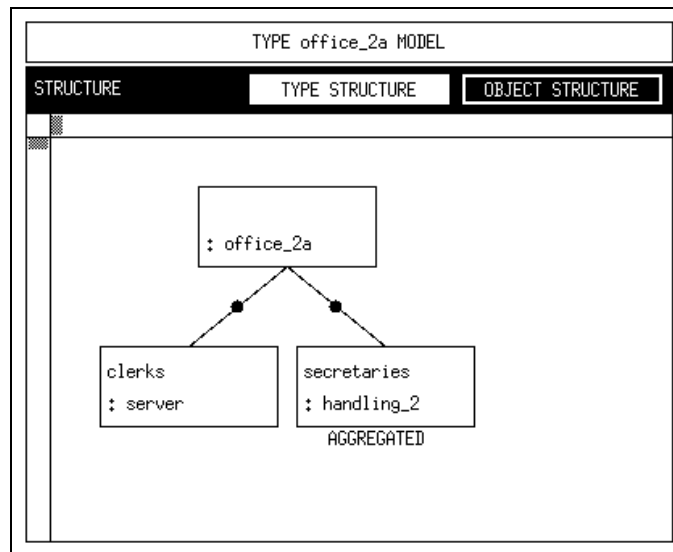


Figure 7.30: Object Structure of “office\_2a”

As explained within this section, the aggregation of a component type is performed in three steps:

1. Creation of the aggregation description; specification of the maximal population of each provided service within the aggregation description window.
2. Source code generation and execution of the aggregation with HIT via *transform* & *run*.
3. Integration of the aggregate within existing model or component types via *use aggregate* within the component type graphic window.

### 7.3.4 Model 3

The development of the third example starts with the modelling of a new component type “supervision\_3”, which will refine the clerks’ work. The final component type graphic window of type “supervision\_3” is depicted in Figure 7.31.

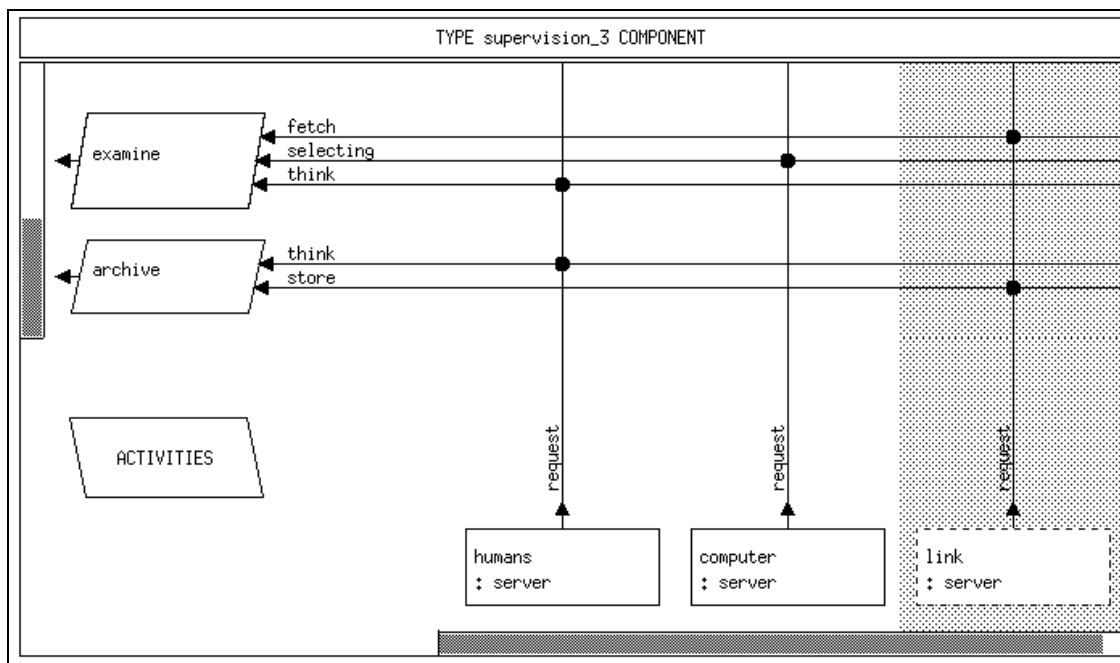


Figure 7.31: The Component Type “supervision\_3”

It shows, that two services are provided. The first one, “examine”, models the process of fetching a document, selecting some pages of it and thinking about these pages. The second provided service, “archive”, models the final storage of the document by use of the services “think” and “store”. The bodies of “examine” and “archive” should be filled according to this description.

The machinery within the component type “supervision\_3” consists of the components “humans”, “computer” and “link” of type “server”. At this point, another feature is demonstrated by the “link” component. It is drawn as a dashed box within a grey area, indicating that the component is *shared*. The “link” models a communication medium, which physically exists only once. For this reason the same “link” will be used within component types “supervision\_3” and “handling\_3” in contrast to the “humans” and “computer”, which are distinct and individual for each component type. So, applying the *shared* operation on the component “link” creates a reference

to a component “link” being instantiated somewhere in a higher component or model type.

To *share* the component “link” within the secretaries’ submodel, too, a new component type “handling\_3” is created. Since the only difference with respect to “handling\_2” (cf. Section 7.3.2) deals with the component “link”, “handling\_3” is created via a *copy* operation on “handling\_2” in the environment window. The shared mode is set for the component “link”.

Up to now two new component types, “handling\_3” and “supervision\_3” are available. The corresponding “clerks” and “secretaries” will be instantiated and used within the new model type “office\_3” as depicted in Figure 7.32 (the model type is not complete yet). You may create the type “office\_3” as a copy of “office\_2”, but then do not forget to delete the formal parameters of the used services “examine” and “archive” (directly within the editor window or using the function *set with transfer* provided at the crosspoint) and the actual parameters of calls of these used services in the service body, because they will no longer be bound to the “request” of a server.

The resulting type and object structures are given in Figures 7.33 and 7.34, respectively.

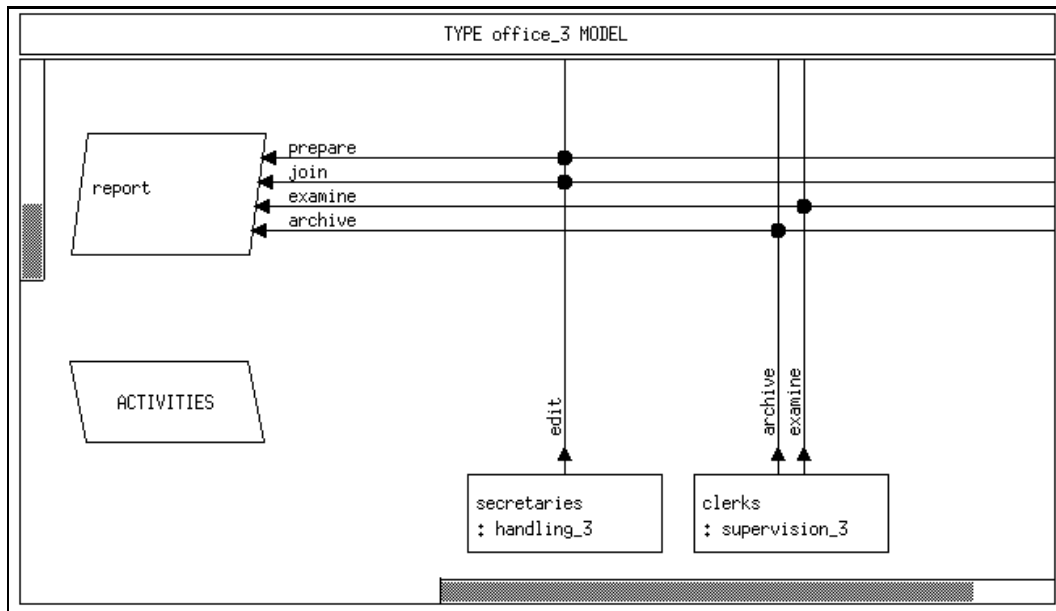


Figure 7.32: The Model Type “office\_3” (preliminary)

The object structure of the model type “office\_3” (as given in Figure 7.34) demonstrates, why the specification of the model type is not yet finished. Two dashed boxes entitled “link” are displayed, indicating that for each box no instantiation of a component “link” of type “server” was found in a higher component. So the shared components with name “link” are just references to uninstantiated components. For this reason it cannot be predicted, that both references point to an identical instantiation.

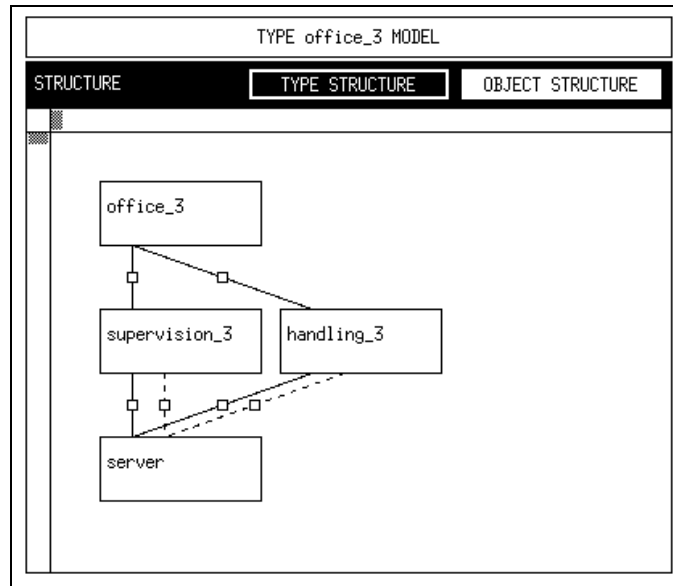


Figure 7.33: Type Structure of “office\_3” (preliminary)

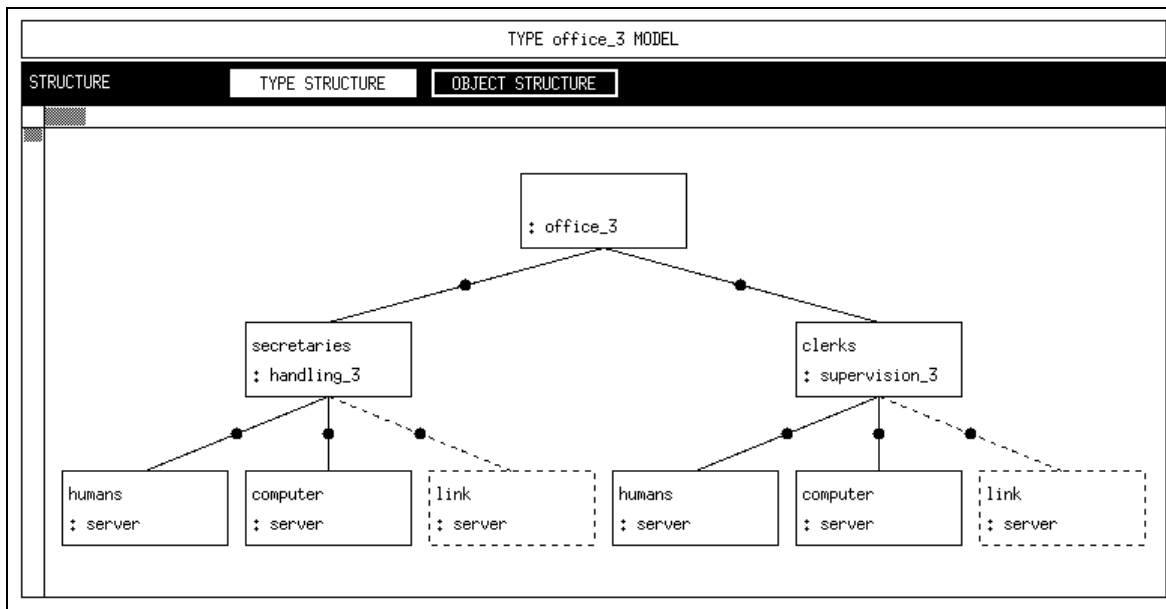


Figure 7.34: Object Structure of “office\_3” (preliminary)



To complete the model type “office\_3”, it is sufficient to instantiate a component “link” of type “server”. Remember to set the service discipline of this component to “processor sharing” via its control procedures (cf. Section 7.3.2, Figure 7.17). This component will not be used within the model type. The corresponding type and object structures in Figures 7.35 and 7.36 show the final “office\_3”.

Note that within the object structure no dot is displayed on the edge between “office\_3” and “link”, since no service of “link” is used. On the other hand, the dashed edges between “handling\_3” and “link” as well as between “supervision\_3” and “link” respectively contain a dot, since within these components the service “request” of “link” is used.

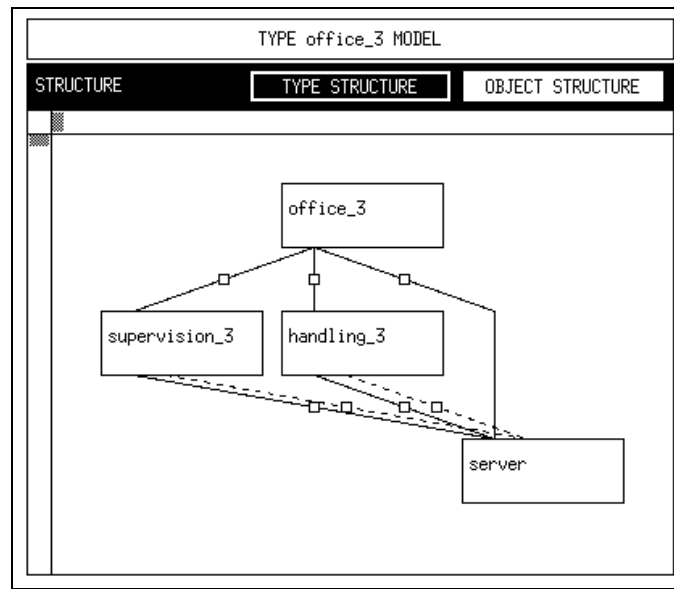


Figure 7.35: Type Structure of “office\_3” (final)

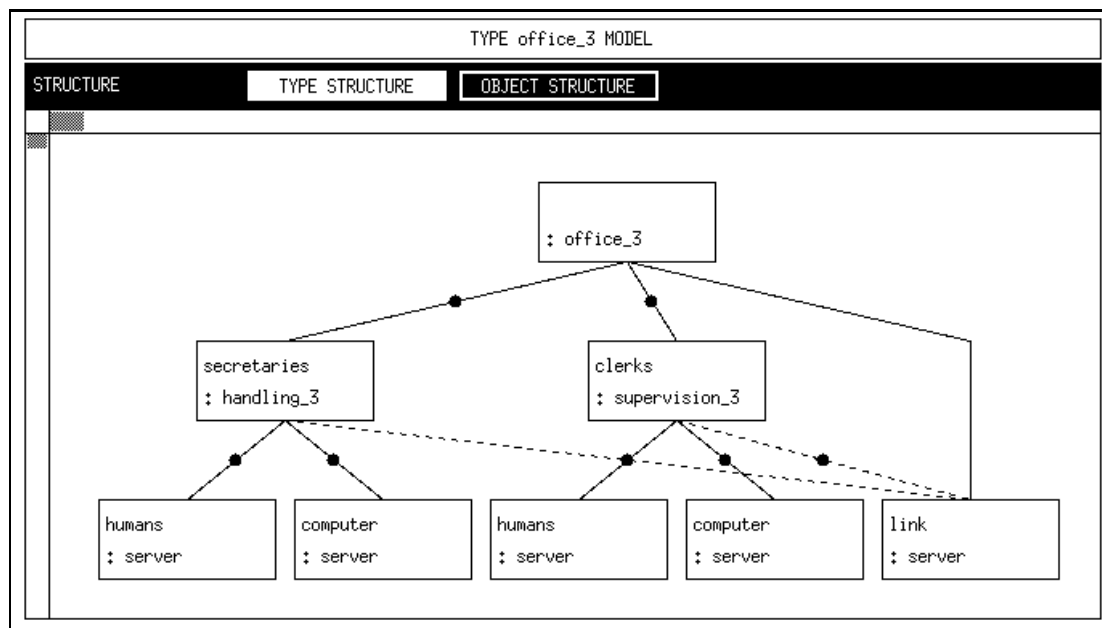


Figure 7.36: Object Structure of “office\_3” (final)

### 7.3.5 Model 4

The preceding sections reflect the top-down design. In contrast to this refinement method this section puts emphasis on the bottom-up modelling style.

Following the story of the office example, the company decides to take part in some foreign transactions. To avoid a mix of documents, a second office is installed to deal with all export transactions, whereas the first office works on import documents. From the view of modelling two (sub)offices that are structured as described in the preceding sections, must be united to form “the” office.

The application of bottom-up design principles within the given context is done on basis of the model type “office\_2” (cf. Section 7.3.2). The problem arising due to the paradigm of hierarchical modelling is, that only component types may be used to build “higher” component/model types. To allow the use of the model type “office\_2” as a platform of bottom-up modelling, it must be converted into a component type. This is done via the *convert* operation offered by the popup menu on “office\_2” within the environment window. Its application pops up an input box asking for the name of the (converted) component type. Entering the name “office\_2\_cp” and pressing the *OK* button starts the internal conversion from a model to a component type, which is finally displayed in the appropriate list.

Note that it is intended to use this component type later on. So it is necessary to provide the service “report” to make it accessible within higher components, to delete the CREATE statement in the activities of “office\_2\_cp” and to remove the lines containing “LOOP” and “END LOOP;” within the body of the service “report”.

After these preparing steps the modelling of the new office starts by creating the model type “office\_4”. Its internal structure, which refers to the component type graphic window, will be discussed next.

In contrast to the preceding models the modelling of the model type “office\_4” is presented in a different order. First, the machinery will be inserted. As mentioned above, the main part of the model type is already given (remember: bottom-up) by the component type “office\_2\_cp”. A component, named “import\_export” of this component type is specified in the well-known manner.

To declare two components (import and export documentation) the concept of component arrays is introduced as some kind of structured shorthand notation. Generally, it would be possible (and allowed) to declare two components of the component type “office\_2\_cp” to distinguish both offices. Even in the case of two components the modelling overhead would increase due to this procedure. A more structured way is to declare one component array with two objects, for instance “import\_export[1..2]”. This declaration provides the advantage, that specifications concerning the component “import\_export” are valid for all (internal) objects, so that they must be given only once. The (internal) objects within a component array are distinguished by their index as known from high level programming languages. Consequently, a component array does not provide single services but service arrays.

After this short introduction to the concept of component arrays (for detailed information see any standard documentation on HIT or HI-SLANG), the declaration of component arrays within the component type graphic window is discussed next. Selecting

the *array* entry provided by the popup menu on the component “import\_export” displays a box that allows for the specification of the array bounds mentioned above. Following the example, the array bounds are specified as “1” respectively “2” and are confirmed by clicking on the *OK* button. The representation of the component changes as follows: First, the range of the array ([1..2] in our example) is appended to the name of the component. Additionally, the component is displayed with a grey shade, representing its field dimensions graphically (cf. Figure 7.40).

If there was no additional machinery, each office (“import\_export[1]” and “import\_export[2]”) would use an own communication link internally. The component “link” of type “server” is declared within the scope of the component type “office\_2\_cp”, thus one “link” component would exist for each object of the component array. But the sample company has just started its expansion and money is running short. To model its decision of installing one communication link for both (sub)offices, the link must be declared shared within the component type “handling\_4”, which is a copy of “handling\_2” (the component “secretaries” within “office\_2\_cp” must of course be changed from type “handling\_2” to “handling\_4” by a *delete, new component* and some *set* operations). To allow access to the component “link” of type “server” from both (sub)offices, it is declared within the current model type. Component type “handling\_4” contains “link” as a shared component (see also model 3). Remember to set the dispatch control procedure of “link” in the model type to “shared”. The control procedure name “shared” is not related to the fact, that “link” is used as shared component. Finally, the specification of the machinery is completed.

The current specification covers the static structure of the model. It is known, where and in which way reports are to be handled. Unfortunately, up to now there are no reports at all; in other words, the load specification is still missing. To complete the specification of the model type, the service “contract” is specified, which will be based on the used service “documentation”. As mentioned above, the component array “import\_export[1..2]” provides the service “report” as a service array. For reasons of consistency, the used service “documentation”, which is to be bound to “report”, must be declared as an array, too. This is done via the *set array* operation of the used service. The next step is to specify the body of the service “contract”. The contents of the associated text editor is given in Figure 7.37. The figure also illustrates, in which way the use of service arrays is specified.

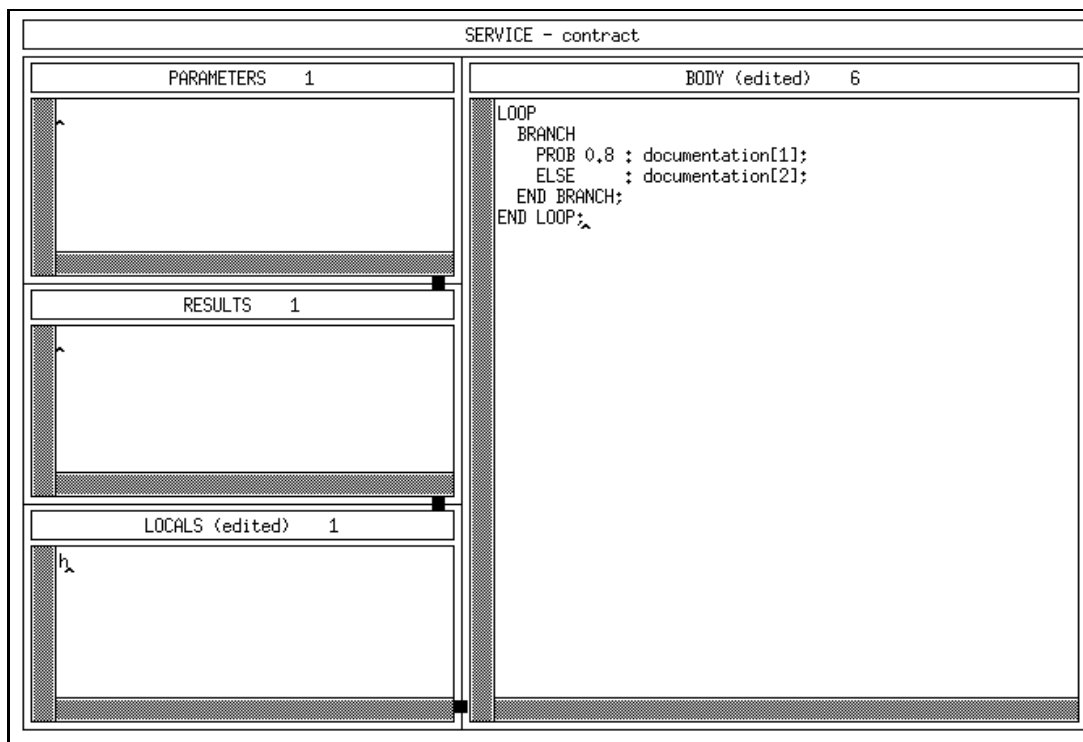


Figure 7.37: Body of “contract”

Finally, it is necessary to specify when and which processes are to be created. To allow a comparison with the former models, the load of the first office should be similar. Therefore, it is specified that 25 processes “contract” are to be created. Assuming that both offices work identically quick and because of the probability 0.8 of calling the used service “documentation[1]”, this would result in a mean number of 20 reports to be treated by the first office. The resulting structures of the completed model type “office\_4” are given in Figures 7.38, 7.39 and 7.40, respectively.

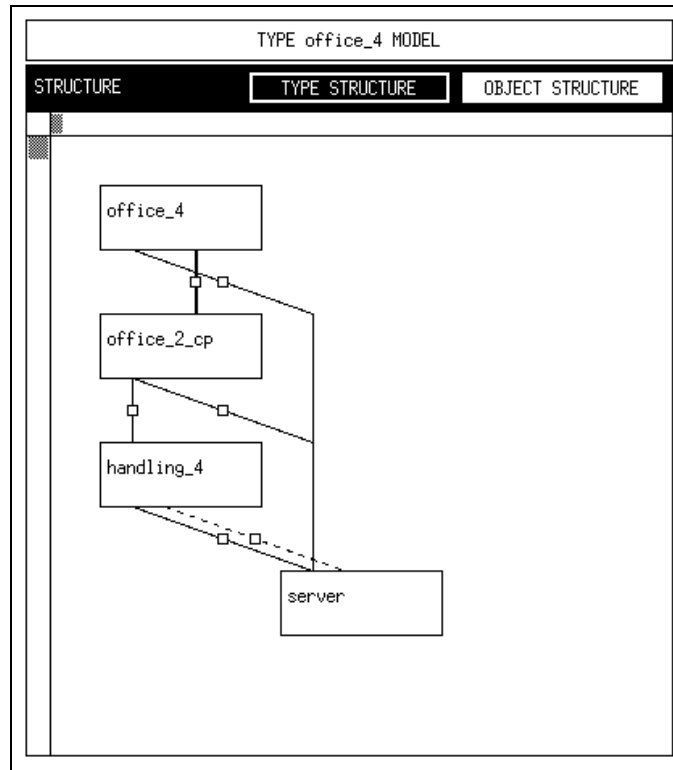


Figure 7.38: Type Structure of “office\_4”

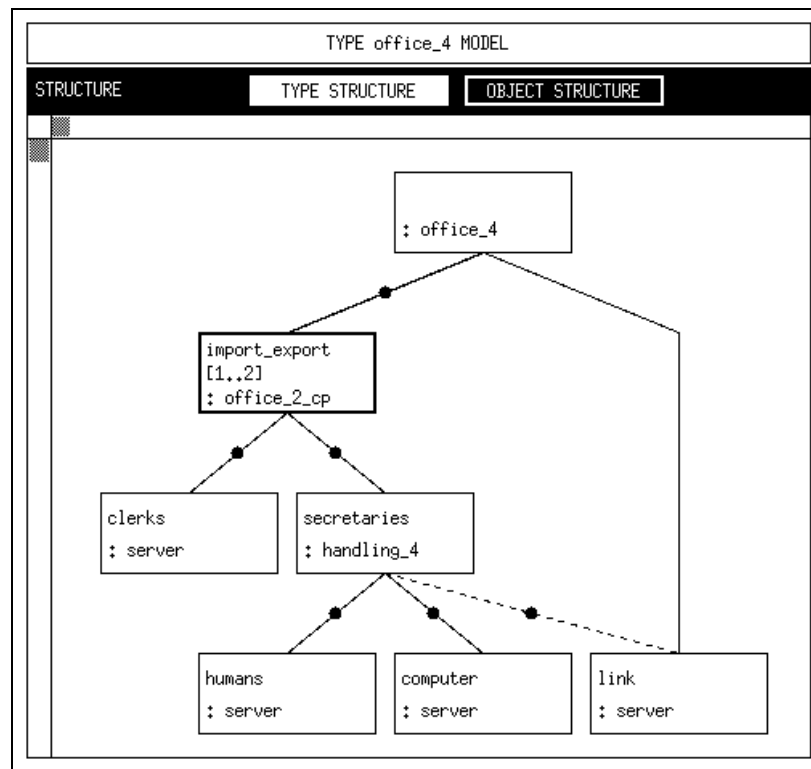


Figure 7.39: Object Structure of “office\_4”

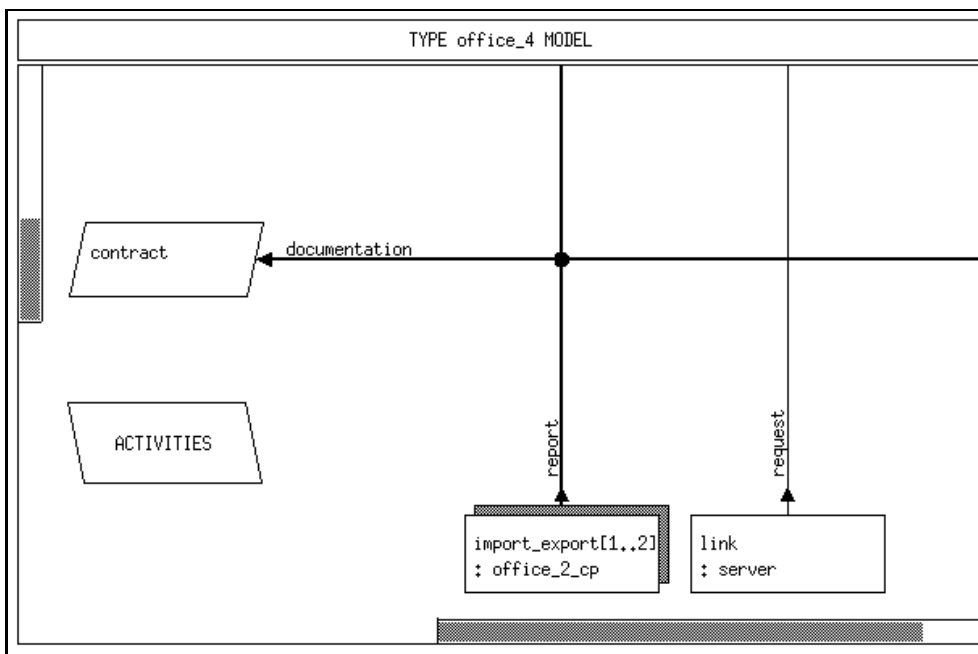


Figure 7.40: The Component Type Graphic Window of “office\_4”

## 7.4 Examples of Evaluation and Experiment Specification

After having introduced the models under study, it is possible now to address the specification of evaluations and experiments, respectively.

### 7.4.1 Evaluation 1

The first candidate for the specification of an evaluation is the introductory example of model 1. Recalling its hierarchical structure, two servers were declared and used within “office\_1” as depicted in Figure 7.14.

Suppose, that the goal of evaluation “eva\_1” is to determine the turnaroundtime of service calls to the server “clerks”.

To reach the stated goal, the first step is the creation of the evaluation “eva\_1” itself. This is done by performing the *new evaluation* operation on the label EVALUATIONS within the environment window, selecting the model type “office\_1”, entering the given name and confirming these steps by pressing the *OK* button in the dialog window. As expected, the evaluation “eva\_1” is inserted in the list of evaluations.

The next step is to open the evaluation window of “eva\_1” via the *open* operation. The result of this action is depicted in Figure 7.41.

So far, the basis for the evaluation specification is created and displayed.

The next step is to specify, where the measurements should be performed. This is done via the concept of evaluation objects, that are associated with the corresponding components. To create an evaluation object for the intended measurements, the

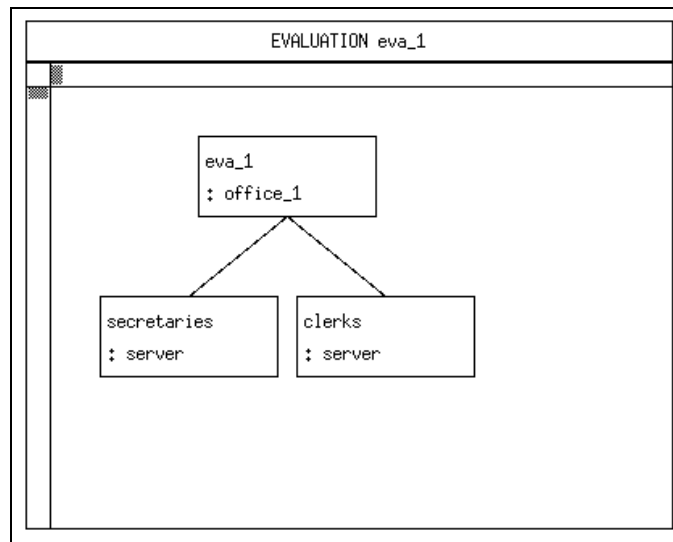


Figure 7.41: The Evaluation “eva\_1”

operation *new evaluation object* is selected from the popup menu provided by clicking on the component “clerks” with the right mouse button. After specifying the requested name of the evaluation object as “clerks\_1”, the display changes as shown in Figure 7.42.

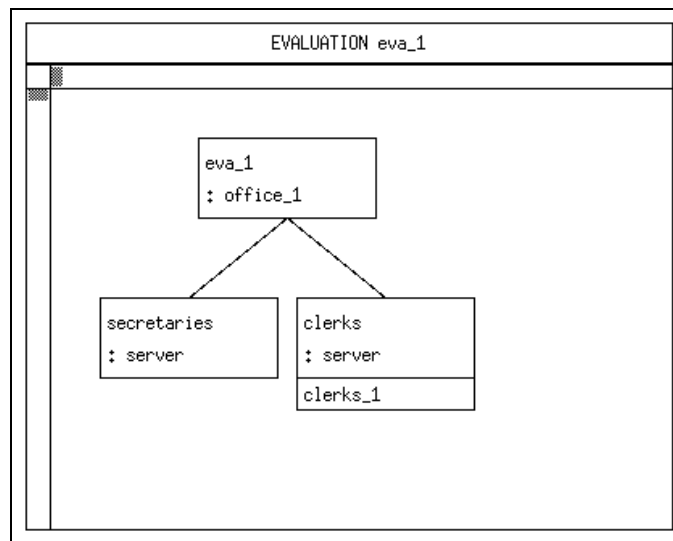


Figure 7.42: The Evaluation Object “clerks\_1” is Created

A small rectangular box is appended to the component for each created evaluation object, providing a popup menu. The *open* operation allows to pop up the evaluation object window, but before this, you are asked to confirm a save operation, because editing in the evaluation object window is only possible, if the evaluation object exists in the database. The initial evaluation object window of “clerks\_1” is given in Figure 7.43.

The evaluation object window serves for the specification of measurements that are to be performed.

Since the turnaroundtime is to be measured in our example, the TURNAROUND-

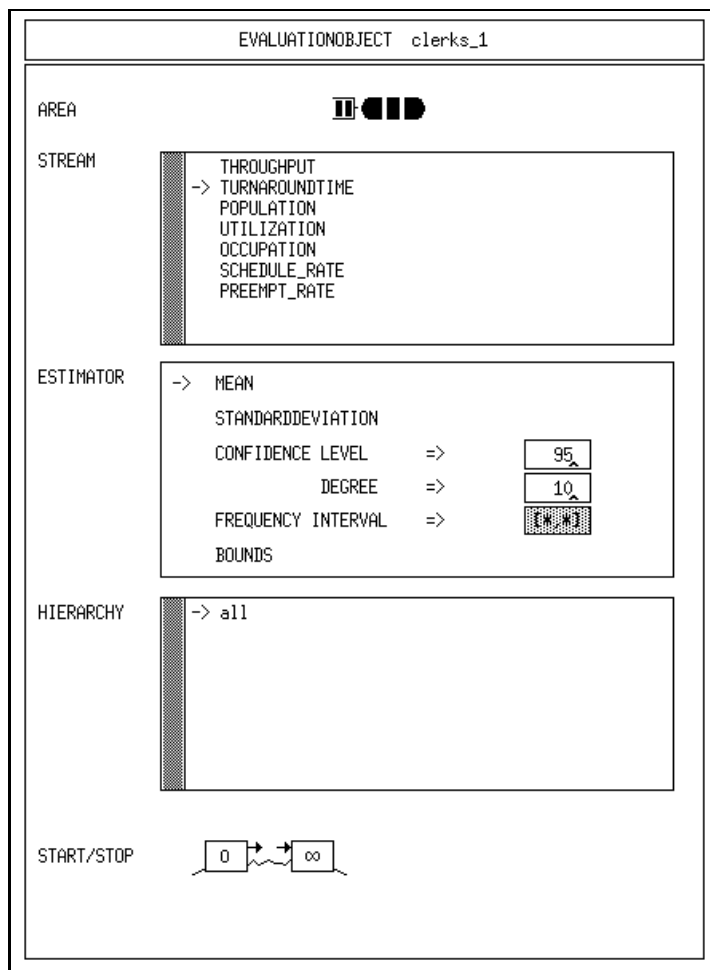


Figure 7.43: Inside the Evaluation Object “clerks\_1”

TIME entry is selected, the default POPULATION is deselected, both by clicking with the left mouse button on the streams.

To prepare a reasonable simulative analysis, the estimators STANDARDDEVIATION and CONFIDENCE LEVEL are selected, too. Note that in case of analytical experiments these estimators are ignored.



After these specifications the evaluation object window of “clerks\_1” is displayed as shown in Figure 7.44.

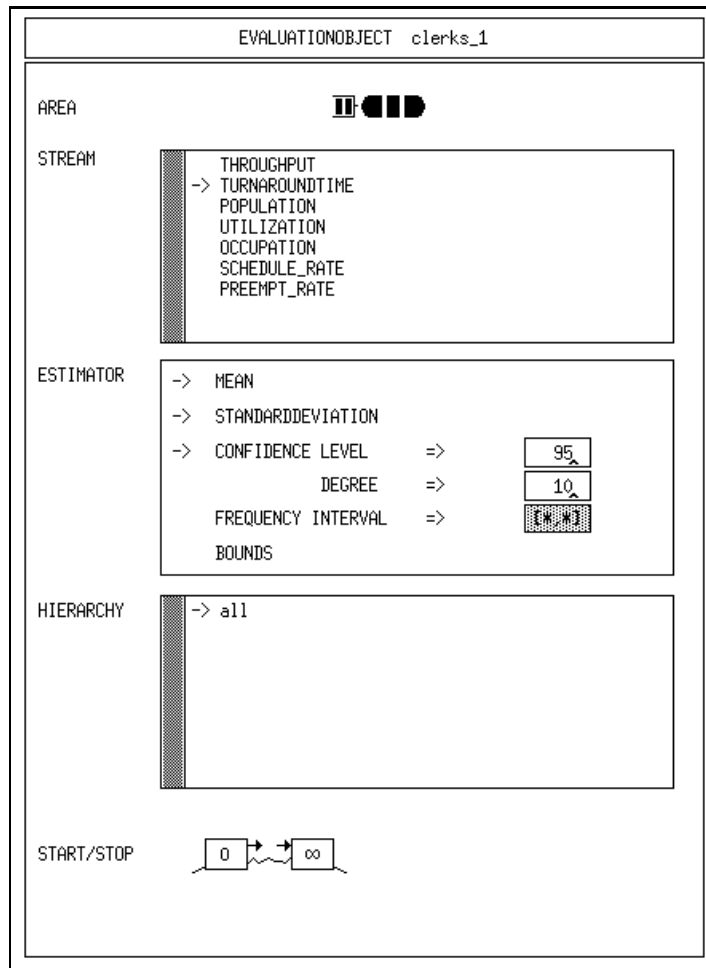


Figure 7.44: The Intermediate Evaluation Object “clerks\_1”

Up to now, the specification of the evaluation object is nearly completed. With respect to the yet unknown analysis technique of the experiment, it is necessary to specify when the measurement should start (the so-called **measurement start control**) and when the simulation should be stopped (**simulation stop control**). This information is reasonable for simulative analysis only; it ensures, that statistically relevant results are obtained. Note that for analytical experiments this part of the specification is ignored.

In order to specify the measurement start control, a start/stop window is opened by performing the *open* operation on the corresponding symbol on the left side. The sign “0” inside the symbol is automatically deleted to express that a user defined start control exists.

The start/stop window and its initial contents are depicted in Figure 7.45.

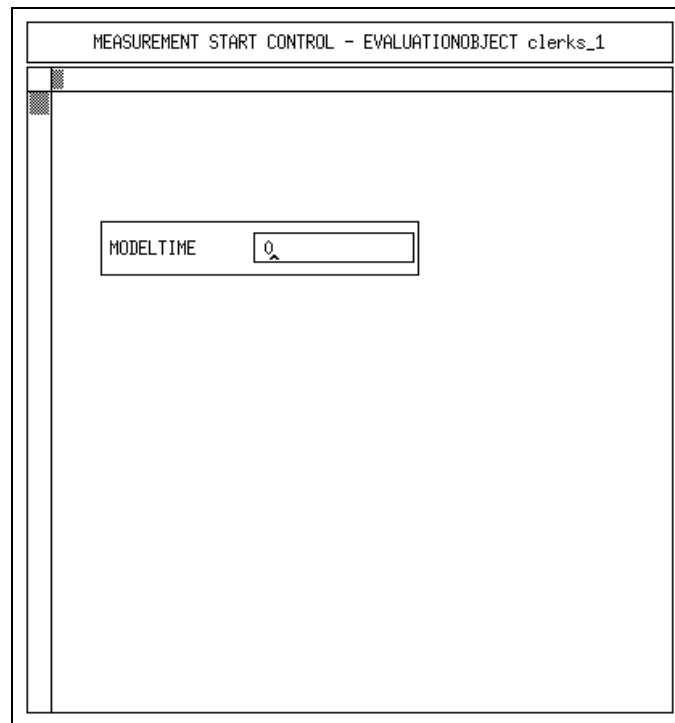


Figure 7.45: The Initial Measurement Start Control of “clerks\_1”

The specification within this window is as follows:

A **MODELTIME** condition allows a precise specification of the start control. From the framework of the model under study it is estimated, that a **MODELTIME** of 10000 seconds would fit to reach a kind of steady state, i.e., the influence of the initial state is sufficiently reduced and measurement should start after that time. The result of typing “10000” into the text field is shown in Figure 7.46.

The specification of the start control is finished by applying the *save & quit* operation.

The next step is the specification of the simulation stop control. We want to stop the simulation when the confidence interval is smaller than an interval of  $\pm 10\%$  around the measured mean value. Therefore, the item *simulation STOP* in the popup menu on the box containing the infinity symbol (“ $\infty$ ”) is selected. The simulation stop condition can be directly specified within the evaluation object window. A box

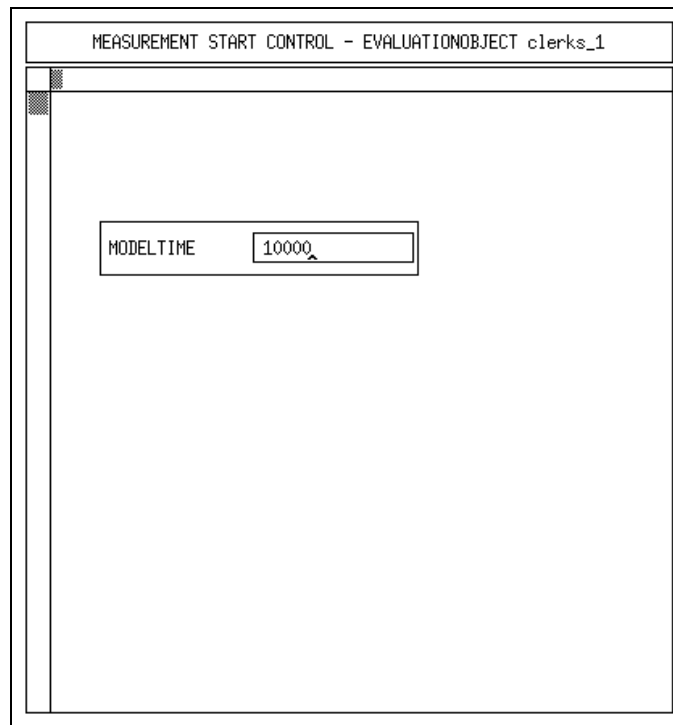


Figure 7.46: The Final Measurement Start Control of “clerks\_1”

appears with input fields for WIDTH and UPDATES, see Figure 7.47. A number of at least 1000 updates on the stream is the initial stop condition. You change it by the selection of *width* in a popup on the background of the box with the right mouse button. The default value of 10 is already correct, so there is no need to edit it.

The final result is given in Figure 7.47.

At this point, the specification of the evaluation “eva\_1” is completed. The single steps that were performed are listed as a brief summary of this section.

1. Creation of the evaluation in the environment window.
2. Determination where the measurement is to be performed by creation of the evaluation object within the evaluation window.
3. Specification of measurements themselves via the evaluation object window.

Optional steps, being meaningful only for simulation, are:

4. Specification of measurement start and stop controls within start/stop windows.
5. Finally, specification of the simulation stop control.

The next step, the specification of the experiment, is presented in the following section.

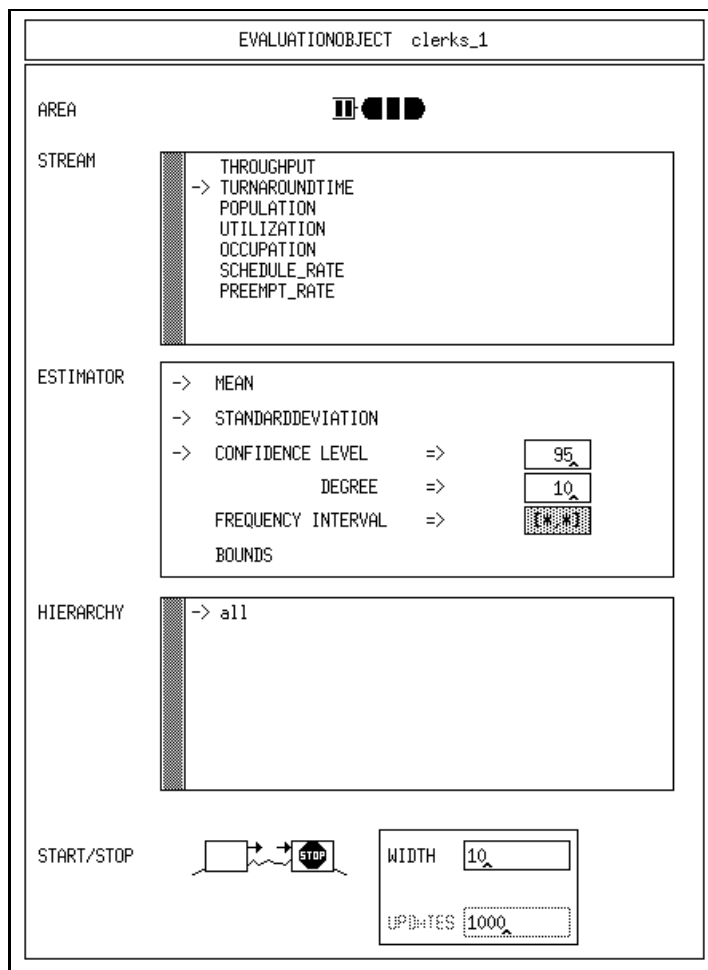


Figure 7.47: The Final Evaluation Object Window of “clerks\_1”

## 7.4.2 Experiment 1

In this section the specification of experiment “exp\_1” is presented, that will allow the analysis of evaluation “eva\_1”. Starting with the environment window, the experiment is created by selecting *new experiment* on the title bar of the experiment list. The name “exp\_1” is typed into the dialog box and the experiment is inserted into the list of experiments after pressing the *OK* button. Then the *open* operation is selected from the popup menu of the experiment, and the experiment window is opened (cf. Figure 7.48).

Remember, that a simulative analysis is intended, which is the default method. Two specifications can be made for the method *SIMULATIVE* (see the preceding section, too). The *CPUTIME* is set to 10005 seconds, which is the time the experiment should run at most. The *MODELTIME* is set to the very large value of 1e10, a value we do not expect to reach. Alternatively, a value of 0 could be used, which leads to the complete ignoration of the *MODELTIME* stop criterion.

To enter evaluation “eva\_1”, the *insert evaluation* operation of the title bar menu is activated, which causes a message box to be displayed. After selecting evaluation “eva\_1” within the environment window and pressing the *OK* button, “eva\_1” is inserted and displayed in the subwindow *EVALUATIONS*. Additionally you are asked

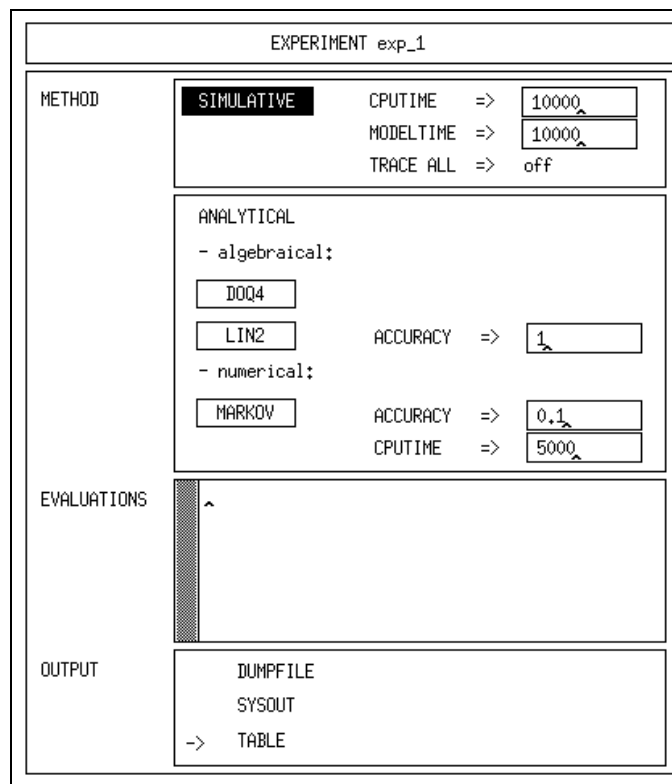


Figure 7.48: The Initial Experiment Window of “exp\_1”

whether an EVALUATE statement for this evaluation should be added to the experiment body. You confirm this with an *OK*. Note that only the evaluations being displayed in this subwindow may be analysed within the current experiment.

To specify the experiment structure, an editor is provided for manipulation of the so-called experiment body. It is opened via the *open* operation on the title bar menu. The specification of the experiment body of the current example is rather simple. Evaluations, that are to be analysed, are simply given in EVALUATE statements. Note that only imported evaluations may be referenced within the experiment body. The state of the experiment body is given in Figure 7.49. In our case the EVALUATE statement is already inserted, so there is no need for a change. In general, multiple evaluations forming an experiment are possible.

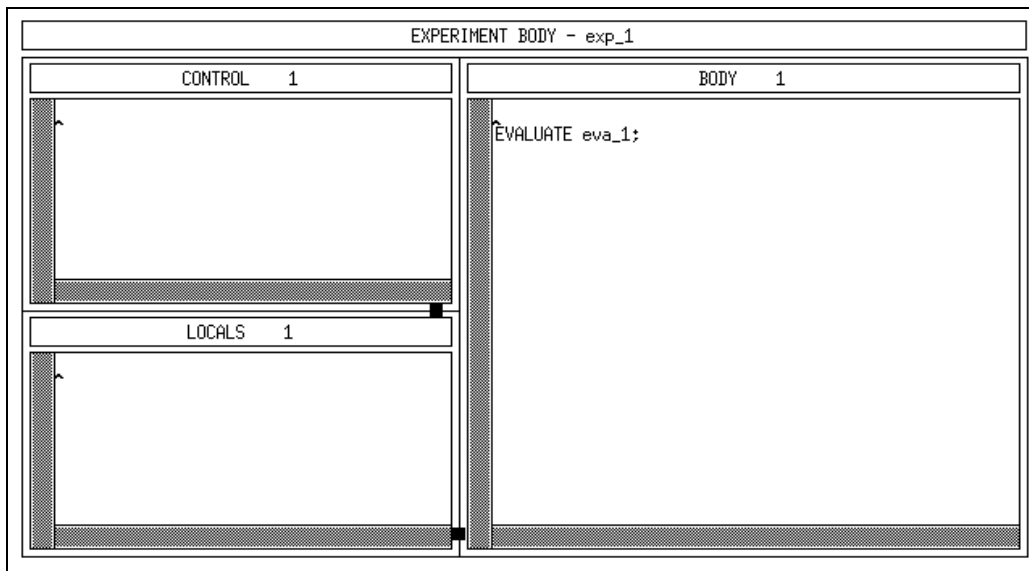


Figure 7.49: The Body of “exp\_1”

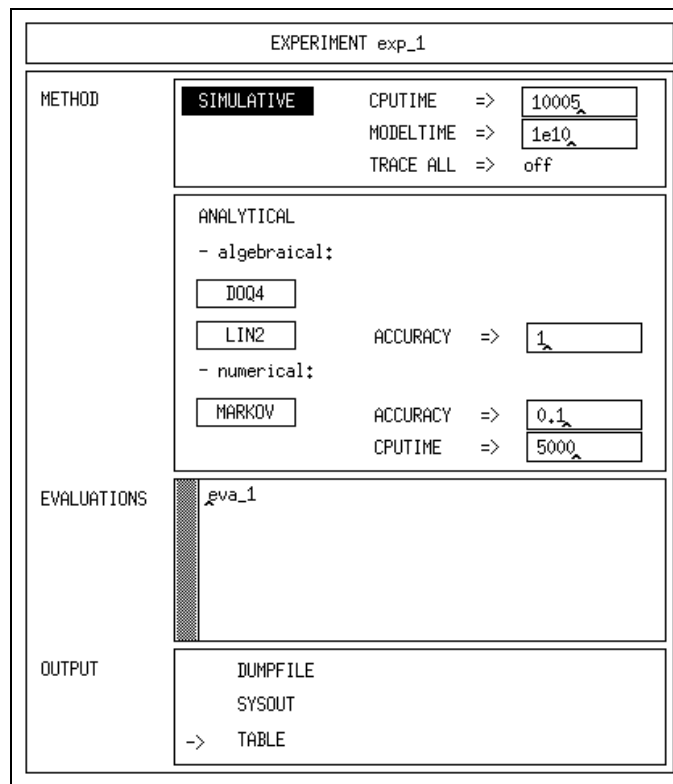


Figure 7.50: The Final Experiment Window of “exp\_1”

The experiment specification is finished now. The distinct steps of an experiment specification are summarized as follows:

1. Creation of the experiment within the environment window.
2. Selection of the analysis METHOD and specification of its parameters.
3. Insertion/import of evaluations to be analysed.

4. Selection of the OUTPUT form of the results.
5. Specification of the experiment body.

The specification work is finished now. The model type, the evaluation and the experiment are stored in the database. The last step towards analysis execution is the transformation of the former specifications into HI-SLANG source code, which is the input of HIT. To perform this transformation, it is necessary to close all subwindows of the environment window, because all modelling objects related to the experiment may not be changed during the transformation, and to select the *transform & run* operation of the menu on “exp\_1”. After a successful transformation the run control is displayed. Selecting *run* from the title bar menu finally starts HIT.

The results of the simulative experiment will be discussed in Section 7.4.4.

### 7.4.3 Evaluation 2/2a

The specification of the evaluations “eva\_2” and “eva\_2a” is almost identical. Since the steps of specification needed throughout this section were discussed in the preceding section, just a summary is given instead of a detailed operation description.

Recalling the object structures of “office\_2” (Figure 7.23) and “office\_2a” (Figure 7.30) shows, that the model types include an identical component “clerks” of type “server”. To allow a comparison of the experimental results as well as the runtime needed for simulation (“office\_2a” contains an aggregated component type), this component is (again) selected for measurement purposes.

As in “eva\_1”, the TURNAROUNDTIME is specified as the stream of interest and will be estimated by a CONFIDENCE LEVEL of 95% with a width of 10%. The measurement start condition and the simulation stop control are specified as in evaluation “eva\_1”.

### 7.4.4 Experiment 2/2a

The specification of the experiments “exp\_2” and “exp\_2a” is done analogously to “exp\_1”.

Again, the method SIMULATIVE is selected with the same CPUTIME and MODELTIME values as in “exp\_1”.

Instead of repeating the specification procedure a result discussion of the experiments 1, 2 and 2a is presented. The models under study reflect the office example on different stages of abstraction and model development.

A short characterization of the model types is given to recall their structure.

“**Office\_1**” is the most abstract model type; it just contains two “server” components.

“**Office\_2**” contains the refined and more detailed component type “handling\_2”; it is the most complex model type under study.

“Office\_2a” is almost identical to “office\_2”, but it contains the aggregated component type “handling\_2a”.

The results are written in dumpfiles, tables (the default) or the protocol and can be accessed via the show operation on the experiment in the environment window. To allow a comparison of the experiments’ results, the probabilities and time consumptions were chosen appropriately. The results of the experiments performed on a workstation are given in the following table.

Results			
Experiment	Mean	CONF. LEVEL 95 %	Runtime
exp_1	1257.4	± 9.51 %	3.0 sec.
exp_2	1178.0	± 9.50 %	165.4 sec.
exp_2a	1209.0	± 9.83 %	3.5 sec.

Table 7.1: Experiment Results

The table presents some interesting measurements, that may be viewed from two points.

The measurements of minor interest within this discussion are the pure analysis results.

The more interesting results are given in the last column. Obviously, “exp\_2” took the very most time to compute the results, whereas the other experiments run in much lower times. Recalling the model type structure, these relations could be expected. Model type “office\_2” is the most detailed candidate. In contrast to the other model types, it contains an additional submodel and therefore consumes additional runtime. The runtimes of experiments “exp\_1” and “exp\_2a” are nearly identical compared to “exp\_2”.

On another workstation or even in repeated experiment runs you may get different runtimes, but the relations between the runtimes of the experiments remain the same.

It should be noted that by selecting DOQ4 as the analysis method, the exact results can be obtained with even less runtime. Of course simulation has been selected here to demonstrate the additional features available especially for simulation.

### 7.4.5 Evaluation 4

The most complex evaluation specification is addressed within this section. New issues arising during the specification of “eva\_4” are constrained to the use of a shared component and a component array in “office\_4”.

The evaluation window of “eva\_4” is displayed according to the object structure as given in Figure 7.39.

The first step is the complete specification of an evaluation object at the component “clerks” of type “server”. As done in the previous sections, the operation *new evaluation object* is selected from the corresponding popup menu. At this point the first



problem arises. After entering the name for the evaluation object (“clerks\_4\_at\_1”), the window is displayed as given in Figure 7.51.

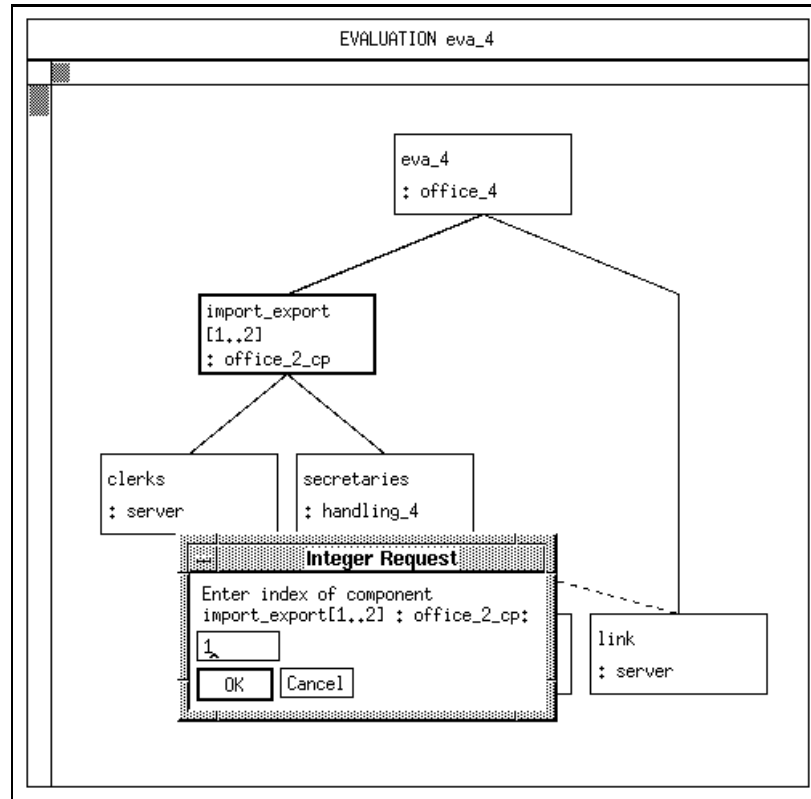


Figure 7.51: Path Specification

The explanation of the box asking for an “index of component `import_export[1..2] : office_2_cp`” is simple. The declaration of component “clerks” is within the scope of the component “import\_export” of type “office\_2\_cp”, which is declared as a component array with two objects. From this point of view, there are two components “clerks”, one declared within “import\_export[1]” and another one declared within “import\_export[2]”. It is necessary for the creation of an evaluation object at the component “clerks” to specify which of both components is meant. By accepting the index “1” in the dialog box of Figure 7.51, the selection of the first component is done. This specification is the so-called component declaration path specification, because the path from the component of interest up to the model is specified along the main declarations until it is unique.

The status after the creation of the evaluation object “clerks\_4\_at\_1” and a subsequent *save* operation is depicted in Figure 7.52.

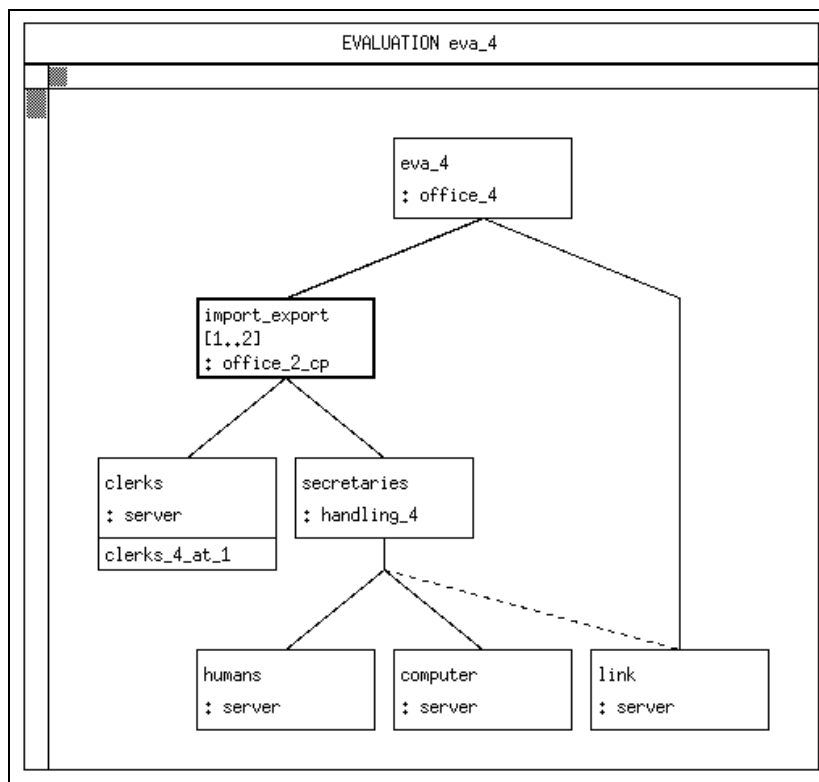


Figure 7.52: The Evaluation Object “clerks\_4\_at\_1” is Created

Note that the selected index is not displayed in the evaluation window directly. To control or recall the path of a component object, the operation *show path* may be used, which is provided on evaluation objects. The result of its application on “clerks\_4\_at\_1” is shown in Figure 7.53.

The point to be addressed next is the concept of hierarchies and their specification, respectively.

So far only the standard hierarchy “all” has been used within the evaluation object window (cf. Figure 7.43), which means, that any load approaching the evaluation object should be measured. Now it is focussed on means of explicit and detailed specification of (load filtering) hierarchies. Load flow into a component is filtered according to its origins to allow distinguished measurements.

Within HITGRAPHIC, hierarchies are declared at their ending component (**target component**) via the hierarchy survey and hierarchy window and may be used within all evaluation objects at this component. Note that even in case of nested component arrays load filtering can distinguish between their components. Later on in this section, it will be discussed how this can be specified.

To access hierarchies the operation *open hierarchies* is performed on the component “clerks”, which displays an empty hierarchy survey window. The hierarchy survey window provides the user with basic operations on hierarchies. After twice performing the *new hierarchy* operation of the title bar and thus creating the hierarchies “clerks\_archive” and “clerks\_examine”, the hierarchy window of “clerks” is displayed as given in Figure 7.54.

The specification of a hierarchy is demonstrated for “clerks\_archive”. The *open* opera-

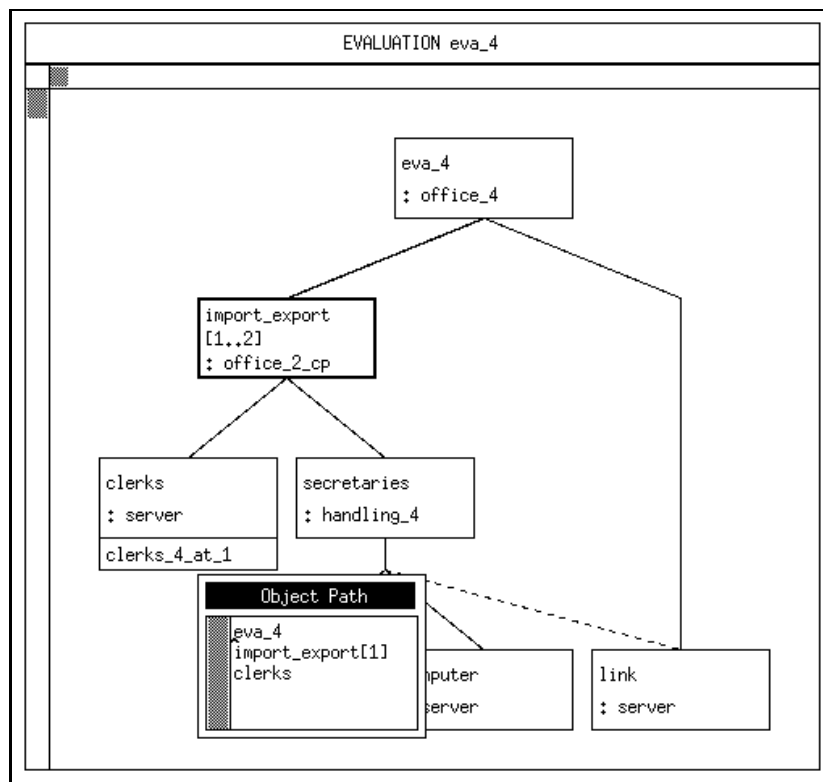


Figure 7.53: Object Path of “clerks\_4\_at\_1”

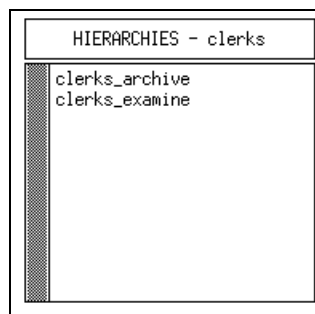


Figure 7.54: The Hierarchy Survey Window of “clerks”

tion is performed on this hierarchy and after confirming the save request the hierarchy window without any load path is displayed (cf. Figure 7.55).

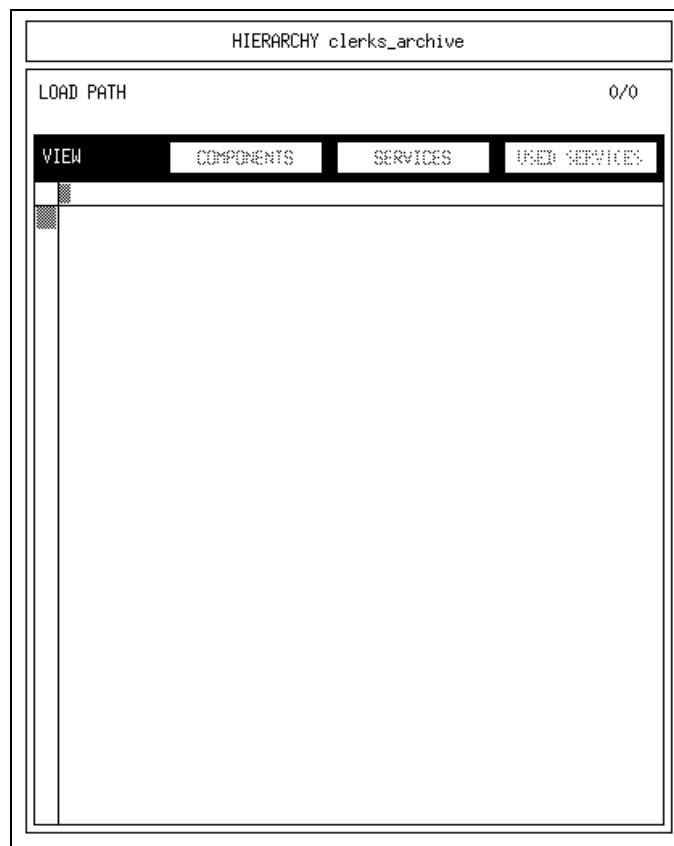


Figure 7.55: The Initial Hierarchy Window of “clerks\_archive”

The hierarchy window allows for the specification of a set of load paths on distinct levels of abstraction. The combination of the load paths forms the entire load filtering hierarchy.

The sample specification of a load path is done in several steps. The first step is the creation of a load path via the *new load path* operation on the title bar popup menu which changes the displayed load path subwindow as shown in Figure 7.56.

Three notable changes follow this operation:

1. A small black rectangle is drawn next to the string “LOAD PATH” and the numbering “1/1” is displayed additionally. This area serves as a kind of album. Each load path is represented by a small box, the actual (displayed) load path is inverted (filled black). The user may switch between load paths by clicking on their symbols with the left mouse button. The string of the form “n/m” at the right-hand side denotes, that the n-th of m load paths is currently displayed.
2. Within the graphic area a part of the object structure/evaluation structure is displayed. It is exactly the path from the model (the root component) down to the target component (that component, which is associated with the current hierarchy/load path). Note that the index of the component array “import\_export” is denoted by an asterisk. This symbol represents a so-called bound index, which can neither be specified nor manipulated. Bound indices occur exactly at those positions, that need to be specified during the generation of evaluation objects within the evaluation window. When a hierarchy is used

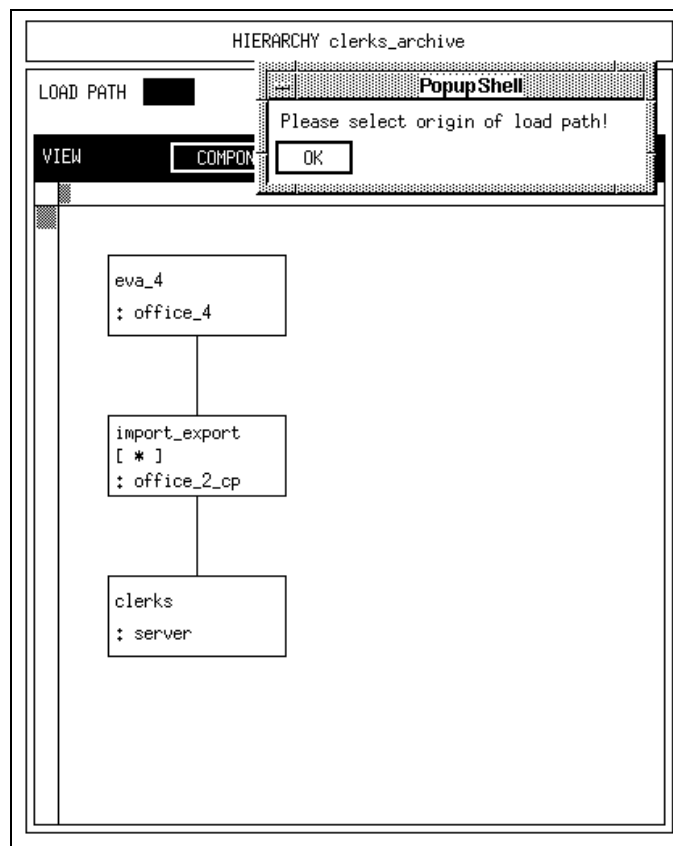


Figure 7.56: The First Load Path of “clerks\_archive” is Created

in combination with an evaluation object, the bound index is replaced by the associated index from the component declaration path of the evaluation object for this usage.

3. A message box with the text “Please select origin of load path!” is displayed. The “originator” is the component, at which the load path should start, i.e., the load generated by CREATE statements in this component is filtered by this load path. It is selected by performing the *select* operation within the symbol.

In the current model the load is generated (via a CREATE statement) within the top component “eva\_4”. Selecting this component and pressing the *OK* button completes the specification at the COMPONENTS level, since the path between the model and the target component “clerks” is unique. After the *save* operation provided by the background menu the display automatically switches to the SERVICES view, which is depicted in Figure 7.57.

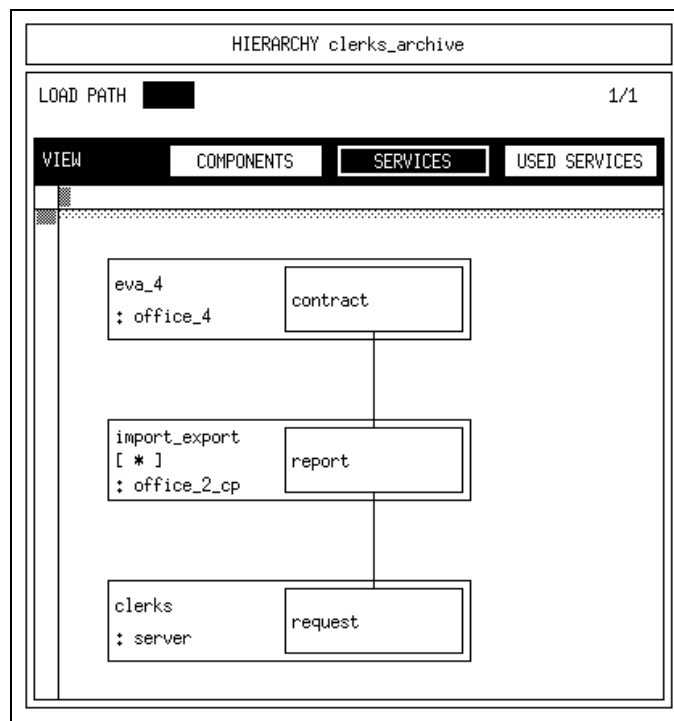


Figure 7.57: The SERVICES View

The SERVICES view is based on the unique path of the COMPONENTS view. For each level the corresponding component is drawn justified to the left side. Within each component symbol all services are visible, that somehow allow a load flow into the target component and are used by still visible higher levels. So, the only load flow within the current example is from service “contract” of component “eva\_4” via service “report” of component “import\_export” to the service “request” of the component “clerks”.

Since there is a unique path again (just one possible service at each component), the switch to the USED SERVICES view is performed by pressing the USED SERVICES button with the left mouse button (cf. Figure 7.58).

As indicated by its name, the USED SERVICES view is based on the SERVICES view extended by information about the associated used services. Again, only those used services are of interest, that are on the actual path from the originator to the target component.

In this sample specification it is obvious now for the first time why these specifications are necessary in certain cases. As illustrated by Figure 7.58, two used services are visible initially within the service “report” of the component “import\_export”. As a consequence, two distinct paths are possible at the current level of detail, both following the same components/services.

If the specification of the hierarchy “clerks\_archive” is finished at this point (one load path covering two alternative routes), this hierarchy would produce the same load filtering as the standard hierarchy “all” for this evaluation object. But, as illustrated by the name “clerks\_archive”, this hierarchy should specify the path to the target component “clerks” via the used service “archive”. To complete this specification, the used service “archive” is selected via the *select* operation. The result of this selection

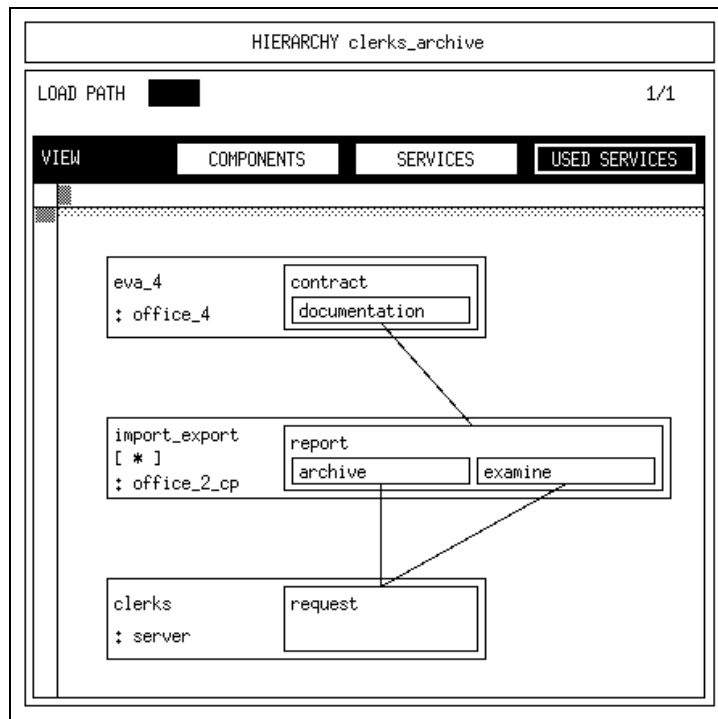


Figure 7.58: The USED SERVICES View

on the USED SERVICES view, which is *saved* from the menu on the background of the scrollable region and thereby finishes the specification of this hierarchy, is given in Figure 7.59.

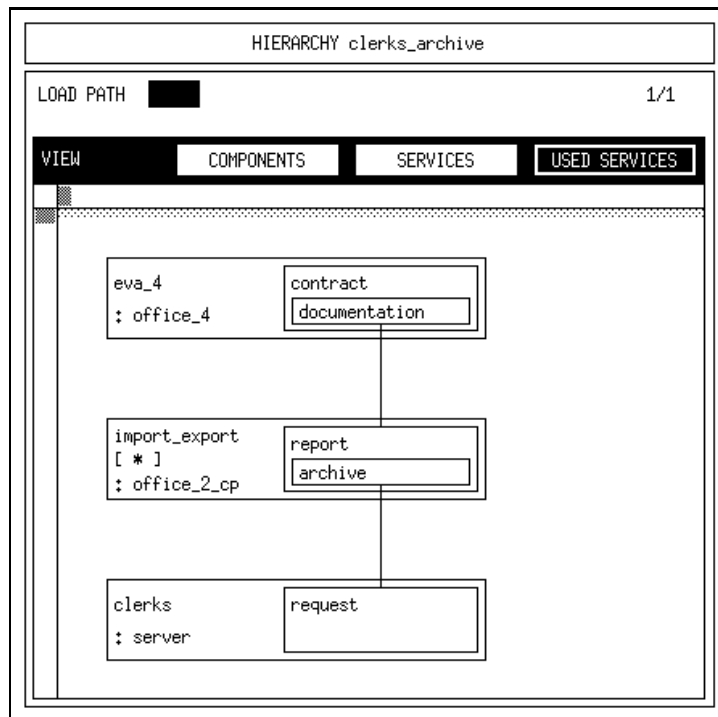


Figure 7.59: The Final Hierarchy “clerks\_archive”

Almost the same procedure of specification is executed for the second created hierarchy “clerks\_examine”. Obviously, the used service “examine” must be selected at the USED SERVICES level (cf. Figure 7.58).

Returning to the evaluation window (cf. Figure 7.52), the next step is to specify the evaluation object “clerks\_4\_at\_1” itself. The *open* operation displays the corresponding evaluation object window as illustrated in Figure 7.60.

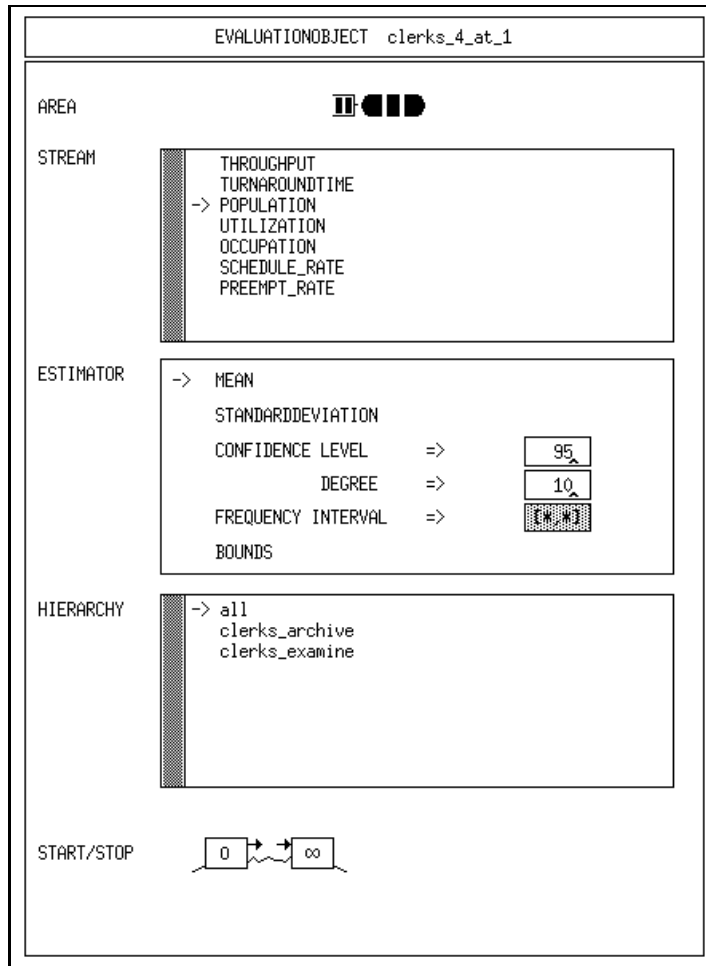


Figure 7.60: The Evaluation Object “clerks\_4\_at\_1”

Note that the initial state of the evaluation object window has changed in comparison to the previous examples. Additionally to the hierarchy “all”, the previously specified hierarchies “clerks\_examine” and “clerks\_archive” (specified at the target component “clerks”) are provided in the HIERARCHY list. Hierarchies are selected or deselected by a click with the left mouse button.

The completed specification is given in Figure 7.61.



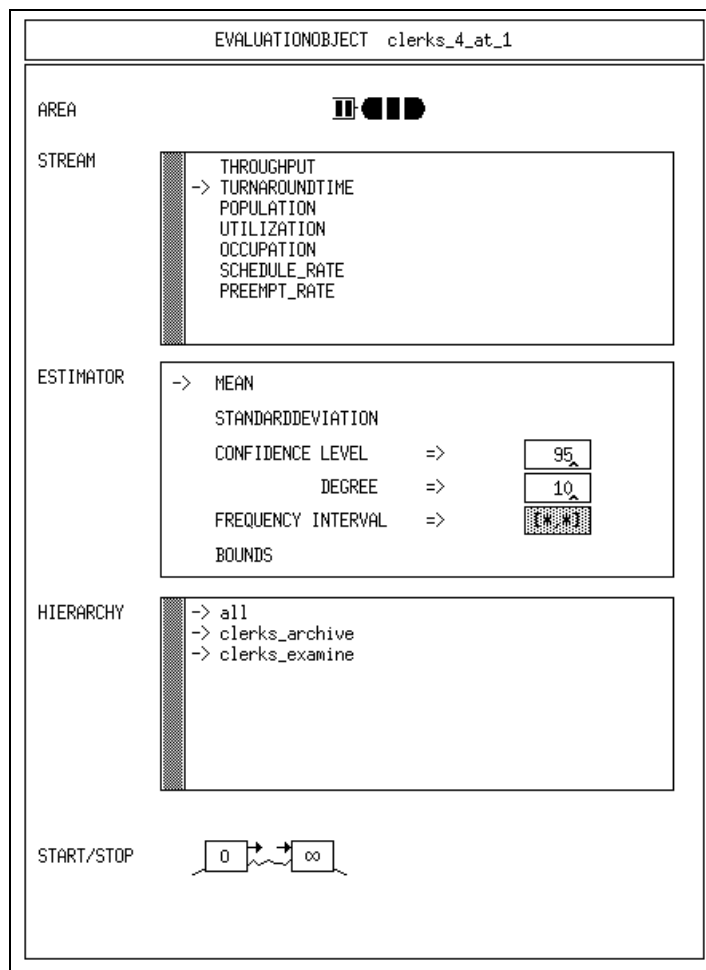


Figure 7.61: The Completely Specified Evaluation Object “clerks\_4\_at\_1”

The specification of the evaluation object “clerks\_4\_at\_1” as given in Figure 7.61 includes two aspects, that should be mentioned.

1. From the specification of ESTIMATOR (only MEAN is selected) and START/STOP (no control specification) it is obvious, that an analytical experiment run is intended. To perform a simulative analysis, the additional specification of the CONFIDENCE LEVEL and start/stop controls is recommended.
2. The second point of interest is, that the self specified hierarchies “clerks\_examine” and “clerks\_archive” are selected.

The next step is the creation of a second evaluation object. To demonstrate some new aspects, the evaluation object “link\_4” is installed at the component “link” of type “server” (cf. Figure 7.62). During this procedure, no additional specification of array indices is necessary, since the component is declared within the model type as a single component.

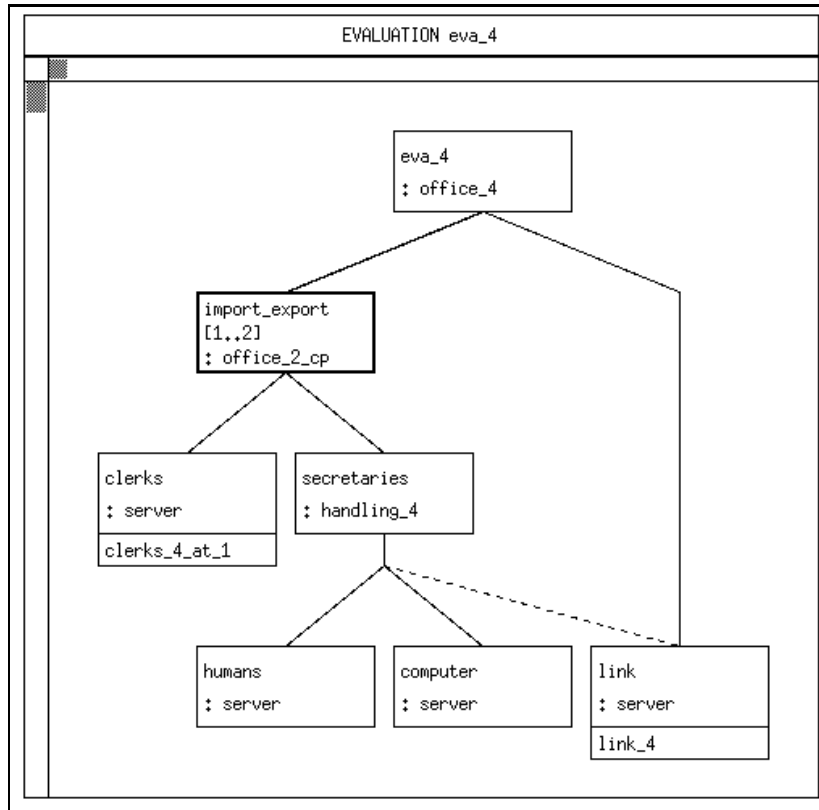


Figure 7.62: The Evaluation Object “link\_4” is Created

The hierarchies “secr\_1\_prep” and “secr\_1\_join” are created next (specified at the component “link”). Figure 7.63 shows the resulting hierarchy survey window.

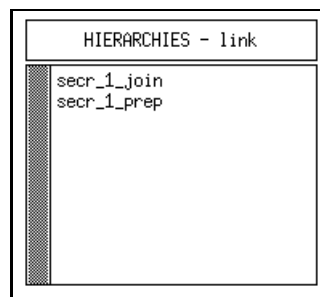


Figure 7.63: The Hierarchies at Evaluation Object “link\_4”

The load path specification is discussed for the hierarchy “secr\_1\_prep”. The hierarchy window is opened and a new load path is created. The result of this specification is summarized in Figure 7.64.

Note that the direct path from the model type “office\_4” to the component “link” is not displayed. The reason for this restriction of the graphics is obvious. The

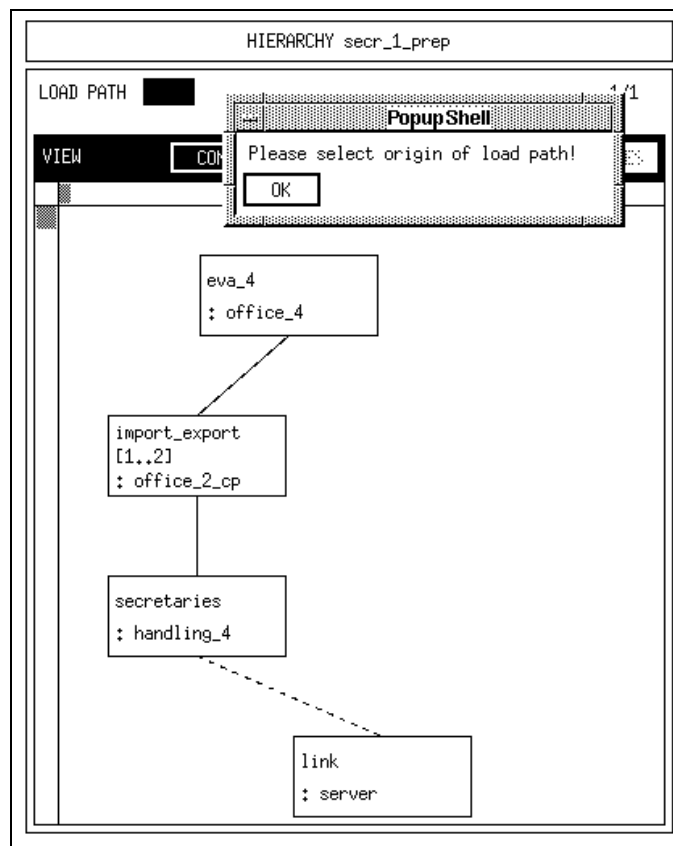


Figure 7.64: The Initial Load Path of Hierarchy “secr\_1\_prep”

component “link” was declared but not used within the model type “office\_4” (no connections between used and provided services were set). Consequently, no direct load flow is possible between these components.

Returning to the scenario of Figure 7.64 the model “eva\_4” is selected as the originator (only components/models with CREATE statements are useful, otherwise the load path is empty). In comparison with the load path specification as given in Figure 7.56, the array bounds of the component “import\_export” must be specified now (they are not written as an asterisk). The array indices of components, that cannot be derived from the declarations of the evaluation objects the hierarchy may be applied to, are called free indices. Their specification is necessary, whenever the component path of a load path becomes unique.

In the given example the index “1” is selected for the current load path. After the *save* operation the SERVICES view is displayed as depicted in Figure 7.65.

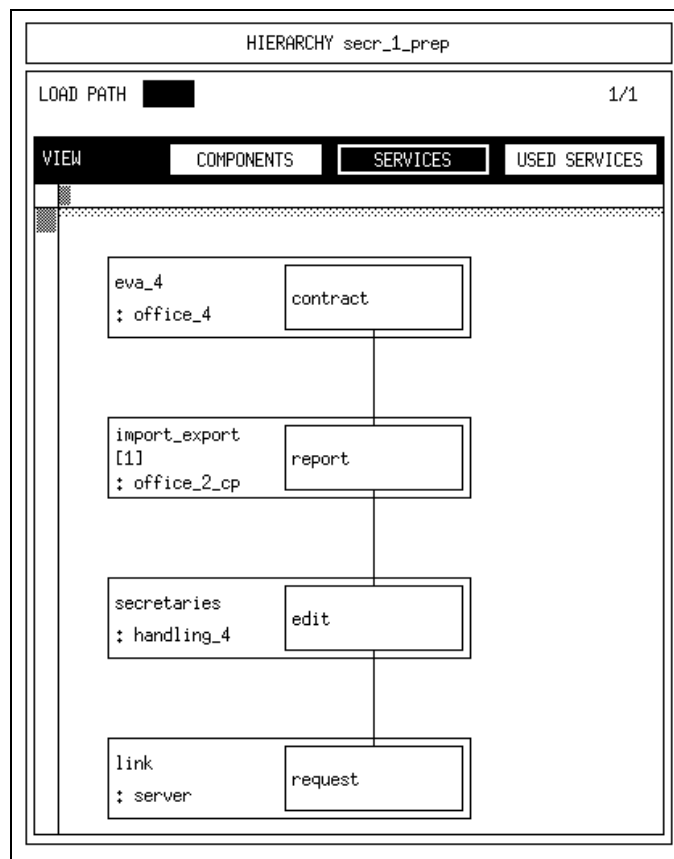


Figure 7.65: The SERVICES View of “secr\_1\_prep”

Since no additional specifications are possible on this level, the display is switched to the USED SERVICES level. A snapshot of the actual scenario is given in Figure 7.66.

At the USED SERVICES level four used services are presented, whose selection would cause an effect on the graphics (“prepare” and “join” of “report” on the second level; “fetch” and “store” of “edit” on the third level). If any of these used services would be selected, the other used service on the same level would be hidden, but the other levels would not be changed.

Following the intentions as indicated by the name of the hierarchy, the used service “prepare” is selected. The displayed structure changes as given in Figure 7.67 and is saved as depicted.

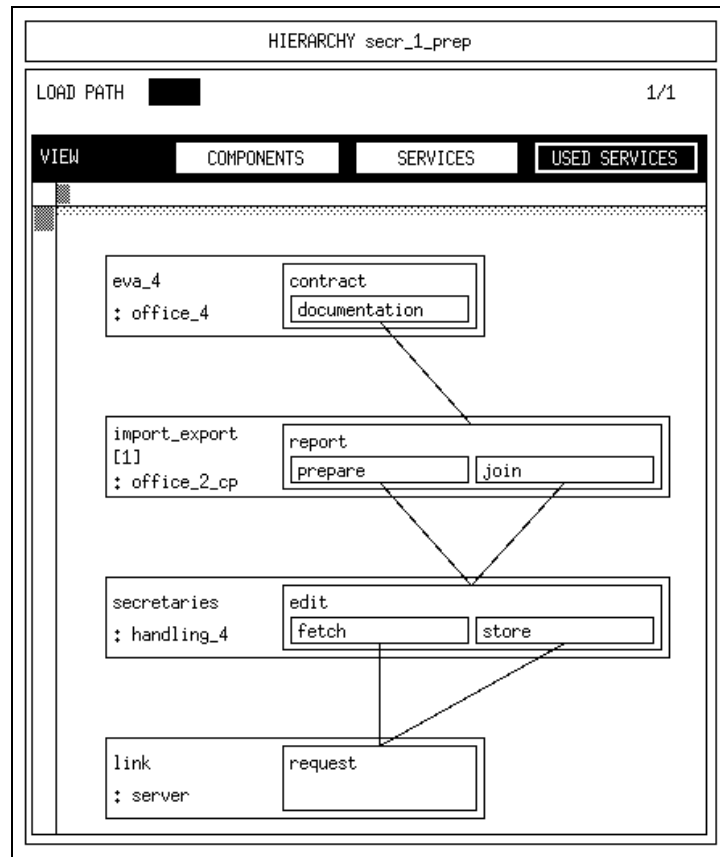


Figure 7.66: The USED SERVICES View of “secr\_1\_prep”

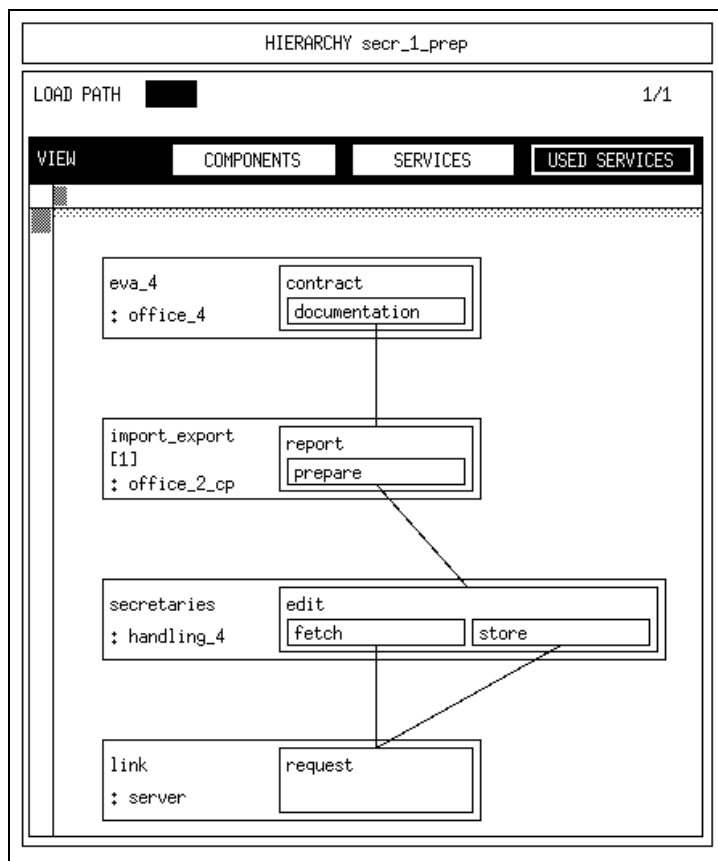


Figure 7.67: The Completed USED SERVICES View of “secr\_1\_prep”

The specified load path covers any load at the target component “link”, that is caused by the service “contract” at the top level and is directed via the used service “prepare” of the component “import\_export[1]”.

The hierarchy “secr\_1\_join” is created and specified analogously to the discussed specification of “secr\_1\_prep” with the exception, that the used service “join” is selected in the last step.

To complete the evaluation specification the specification of the evaluation object “link\_4” is discussed briefly. The streams THROUGHPUT and TURNAROUND-TIME are selected for all available hierarchies. The final specification is given by the corresponding evaluation object window in Figure 7.68.

Again, the specification is done preparing an analytical analysis by only selecting the ESTIMATOR MEAN and omitting the specification of start/stop controls. To demonstrate the concept of self defined hierarchies, “all” is selected as well as “secr\_1\_prep” and “secr\_1\_join”.

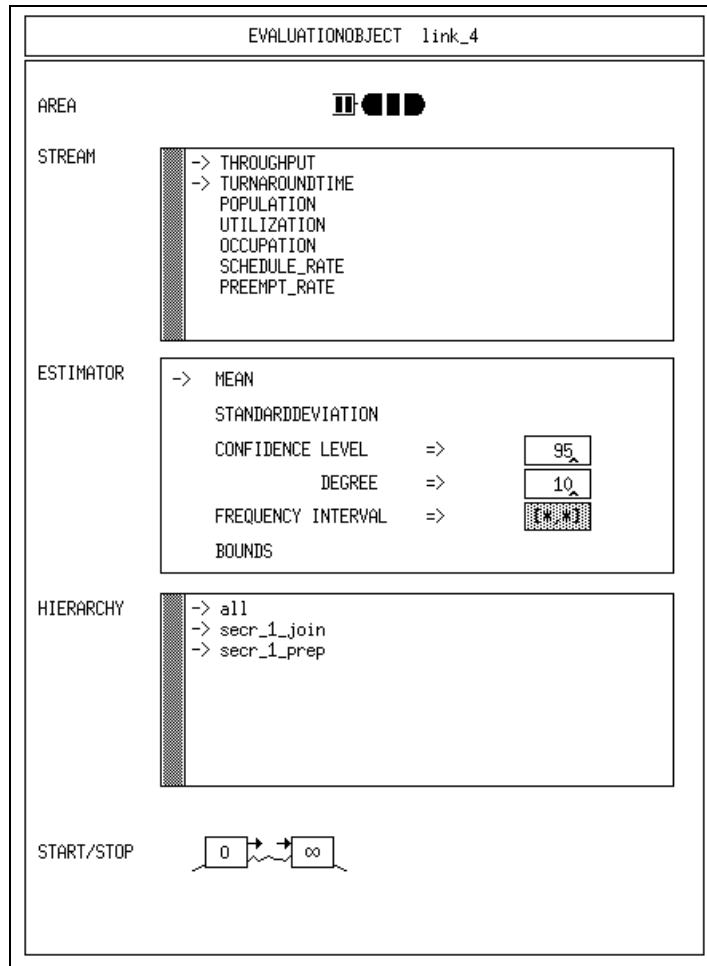


Figure 7.68: The Complete Evaluation Object "link\_4"

## 7.4.6 Experiment 4

The specification of the experiment “exp\_4” introduces the usage of analytical methods for the first time. Instead of SIMULATIVE the method ANALYTICAL DOQ4 is selected. For this method no additional specifications are necessary.

The import of evaluation “eva\_4” and the specification of the body are not discussed, because they are similar to the previous experiment specifications. Confirm Figure 7.69 for the complete specification.

EXPERIMENT exp\_4

---

**METHOD**

SIMULATIVE    CPUTIME =>

                  MODELTIME =>

                  TRACE ALL => on

ANALYTICAL

- algebraical:

DOQ4

LIN2            ACCURACY =>

- numerical:

MARKOV            ACCURACY =>

                  CPUTIME =>

**EVALUATIONS**

eva\_4

**OUTPUT**

DUMPFIL

SYSOUT

-> TABLE

Figure 7.69: The Completely Specified Experiment “exp\_4”

The results you get after performing *transform & run* are summarized in Table 7.2.

Results			
Evaluation object	Hierarchy	Turnaroundtime (Mean)	Throughput (Mean)
clerks_4_at_1	all	1200.00	
	clerks_archive	600.00	
	clerks_examine	1800.00	
link_4	all	9.38	0.25
	secr_1_join	9.38	0.05
	secr_1_prep	9.38	0.15

Table 7.2: Results of Experiment 4



## 7.5 Epilogue

If you have worked through the tutorial up to its end you have reached some kind of familiarity with the usage of this tool. Besides this the main ideas of model structuring and evaluation and experiment description have been demonstrated.

You are prepared to start your first real modelling project with HIT. But surely you extensively will have to use the HIT documentation and other references, e.g., for aspects of solution methods, details of HI-SLANG or the meaning of some menu items. Modelling is too complex to be done just by intuition.



# Index

- accept, 70, 73
- accuracy, 126
- activities, 81
- actual parameter, 79
- aggregation, 160
- aggregation description, 29, 35
- aggregation description window, 18, 92
- all, 103
- area, 104
- area of a component, 102
- array, 47, 59
  
- batch, 37, 43
- bind services, 153
- body, 80, 82
- busy, 134
  
- call HITGRAPHIC, 8
- change environment, 27
- check, 88
- component, 152
- component array, 170
- component type, 28, 33, 54
- component type graphic window, 15, 50
- confidence level, 112
- confirm box, 13
- connection, 51, 68
- control, 82, 124, 137
- control part, 82, 124, 137
- control procedure, 52, 156
- control procedure window, 16, 70
- copy environment, 26
- CPUTIME, 126, 136
  
- database conversion, 8
- db\_admin\_prg, 9
- DB\_DIR, 7
- debugging code, 139
- declaration, 76
- delete environment, 27
- dialog, 37, 43
  
- dialog box, 12
- dispatch, 70, 75
- DOQ4, 125
- dumpfile, 45
  
- editor window, 17, 76
- environment, 23
- environment window, 14, 24
- estimator, 103, 105
- evaluation, 30, 39, 174
- evaluation object, 95, 104, 174
- evaluation object window, 19, 102
- evaluation window, 18, 95
- events, 112
- experiment, 30, 41, 122, 180
- experiment series, 138
- experiment window, 22, 122
  
- formal parameter, 79
- free, 129
- frequency, 82, 105
  
- get, 57
- globals, 79
  
- HI-SLANG, 1, 36, 42, 78, 139
- HI-SLANG control procedure, 70, 81, 135
- hierarchy, 103, 106, 186
- hierarchy survey window, 20, 114
- hierarchy window, 21, 117
- HIT, 1, 36, 42
  
- information, 78
- input box, 12
  
- LIN2, 126
- load, 148
- load path, 21, 117, 188
- locals, 80
- locked, 130
  
- machine, 152

MARKOV, 126  
measurement start control, 106, 178  
measurement stop control, 106, 136  
menu, 11  
message box, 12  
method, 122  
mode, 129  
model type, 28, 31, 52  
modelling object, 23, 129  
modetime, 111, 126, 136  
  
normal, 47, 50, 59  
  
object structure, 48  
observer, 141  
offer, 70, 75  
output, 123, 127  
  
PostScript, 131  
print, 131  
provided procedure, 62  
provided service, 50, 62, 152  
put, 61, 65  
  
refine, 155  
released, 130  
result parameter, 79  
  
schedule, 70, 73  
search, 88  
seed, 137  
selection, 11  
self defined component type, 28  
service, 148  
service array, 170, 171  
shared, 47, 50, 59, 166, 168  
simulation stop control, 96, 136, 178  
simulative, 125  
single, 47, 59  
standard component type, 28  
start/stop control, 103, 106  
start/stop window, 20, 108  
startup window, 24  
stream, 52, 67, 102, 104  
survey window, 14, 46  
  
table, 45  
text editor, 17  
trace, 45  
type default, 71  
type structure, 47  
undefined result, 141  
unset\_busy, 9, 134  
updates, 107  
used procedure, 66  
used service, 51, 66, 149  
user database, 23  
  
width, 107