Combining Visual Modelling with Visual Programming for CORBA Component Development

Frank Stefan Bühler

De Montfort University Leicester
Faculty of Computing Sciences and Engineering
De Montfort University, The Gateway
Leicester LE1 9BH
England

DE MONTFORT
UNIVERSITY

3 Volumes (Thesis and Appendices)

APPENDICES (VOLUME 3)

Dissertation submitted for the degree Doctor of Philosophy

May 2002

Appendix Q: Powerpoint Slides for Evaluation 1 and 2

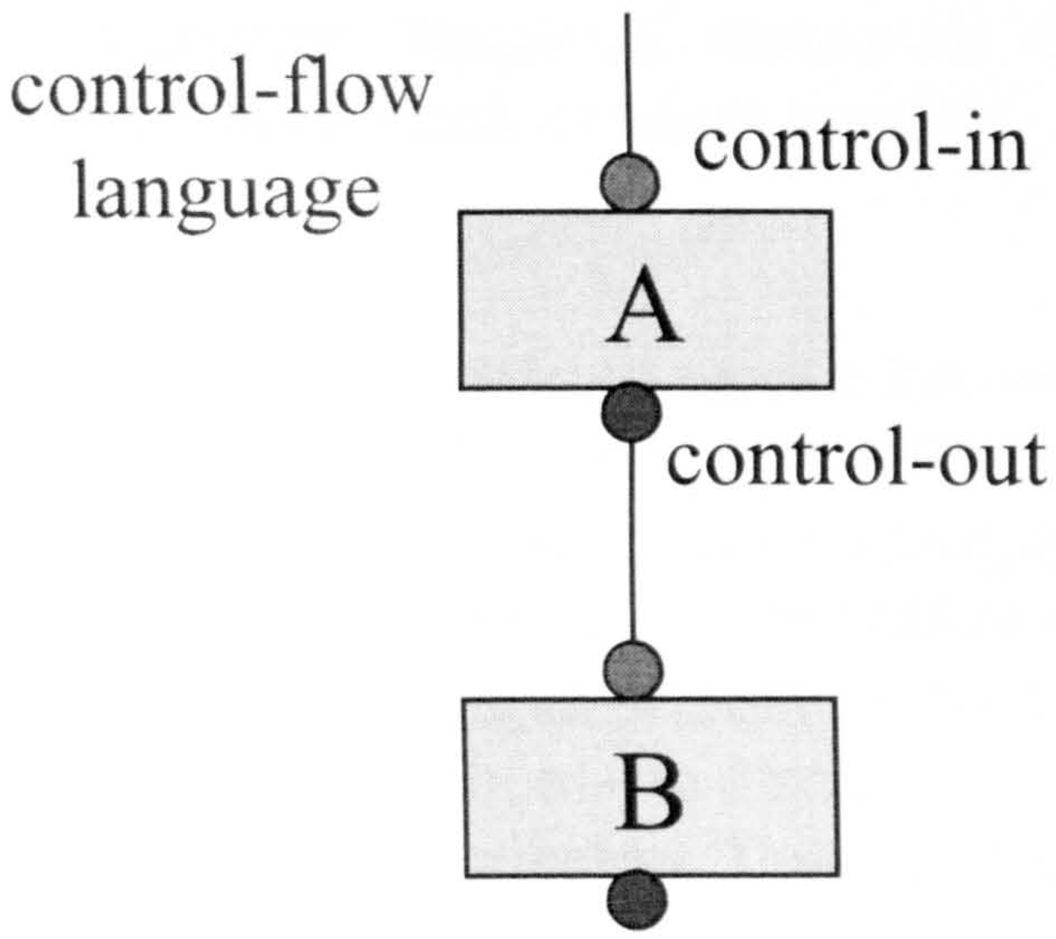
The Powerpoint slides used for both evaluations are presented in the following.

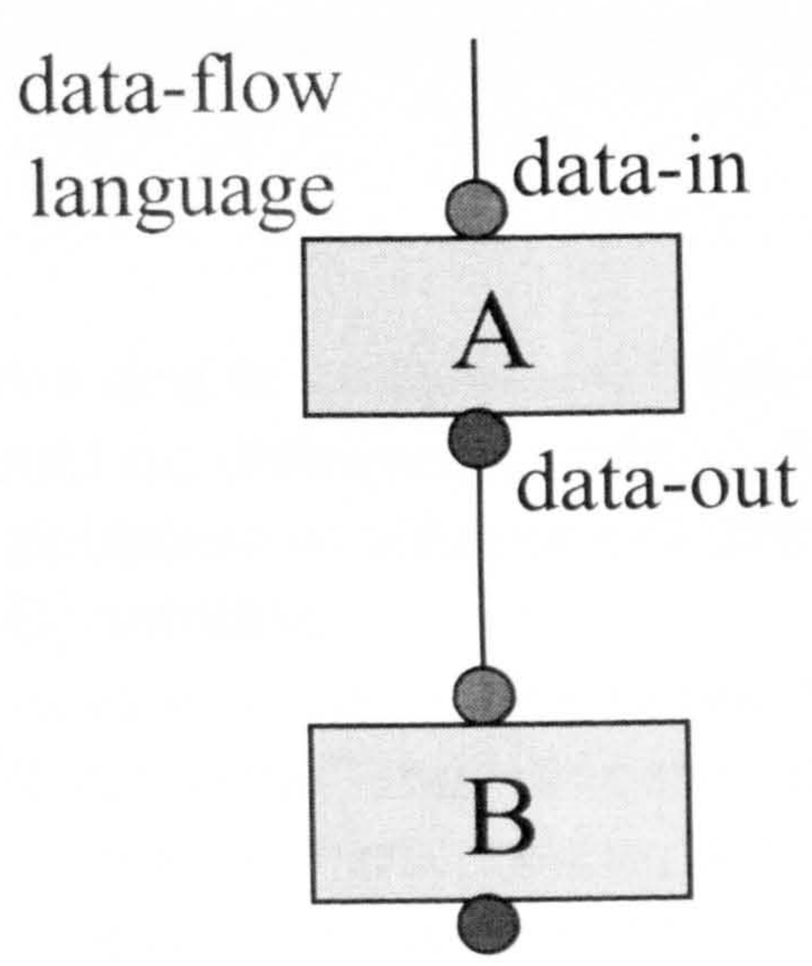
Evaluation 1

Duration : 2 hrs.

Planned date : 23/10/2000

Evaluator : Frank Buehler





User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Hypothesis

Creating a VPL program may be more efficient than using a textual language such as Java or C/C++.

Different paradadigms

Prograph is a commercially available VPL that is based on the **data-flow paradigm.**
Another academic VPL is VMB-C which is based on the **control-flow paradigm** and closely related to the language C.

Objective of the evaluation

The aim of the evaluation is to compare the efficiency of different approaches w.r.t. "readability" and "writability" of a program. "Negative" results for VPLs are expected because the visual environment is not involved in the trials (-> lack of "directness")!!

Trials/Tasks

"readability" : Students are asked to read a few simple programs and to write down what they think the programs do. (Prograph & Fortran : both should be unknown to the students)

"writability" : Students are asked to write a program (for the calculation of a fibonacci number > 2
- in (Prograph and/or C/C++/Java and/or VMB-C) notation.

Only a few students should be able to come close to a solution. Better results are expected for C or Java. Thus: the discussion whether a VPL is "better" than a text-based language is the "wrong" discussion. The visibility of structures/relationships/concepts is of more importance in order to increase the efficiency of program development!

Evaluation 1 (2 hrs.)

ca. 9-15 CS/SE students,
1 beamer for training, pencils for questionnaire & trials

step

1) Questionnaire (2 min.)
   including tests on "readability" of Prograph/Fortran programs (max. 13 min.)
2) Prograph Training (25 min.)
3) VMB-C training (25 min.)

   break (15 min.) -> building 3 groups

4a) Tests on "writability" (three groups, max. 20 min.)
   group A - Prograph program
   group B - C/C++/Java program
   group C - VMB-C program
4b) Tests on "writability" (three groups, max. 20 min.)
   group C - Prograph program
   group A - C/C++/Java program
   group B - VMB-C program

Questionnaire

Name of student: _____

Please note: all responses will be confidential!

This evaluation is part of a research project on Visual Languages, aiming to help design future generations of visual tools. Could you please spend a few minutes to answer the following questions. After this you are asked to describe the meaning of some code samples presented on the following pages.

Q1 : Which of the following programming languages do you know?

	C	: yes O no O
	C++	: yes O no O
	Java	: yes O no O
(other)	____	: yes O no O
(other)	____	: yes O no O

Q2 : How much experience of programming do you have?

Language C_____ High O Medium O Low O None O
Language C++_____ High O Medium O Low O None O
Language Java_____ High O Medium O Low O None O
Language _____ High O Medium O Low O None O
Language _____ High O Medium O Low O None O

Q3 : Do you have any knowledge of a graphical language (i.e. a programming language that makes use of icons or diagrammatic elements rather than textual code elements)?

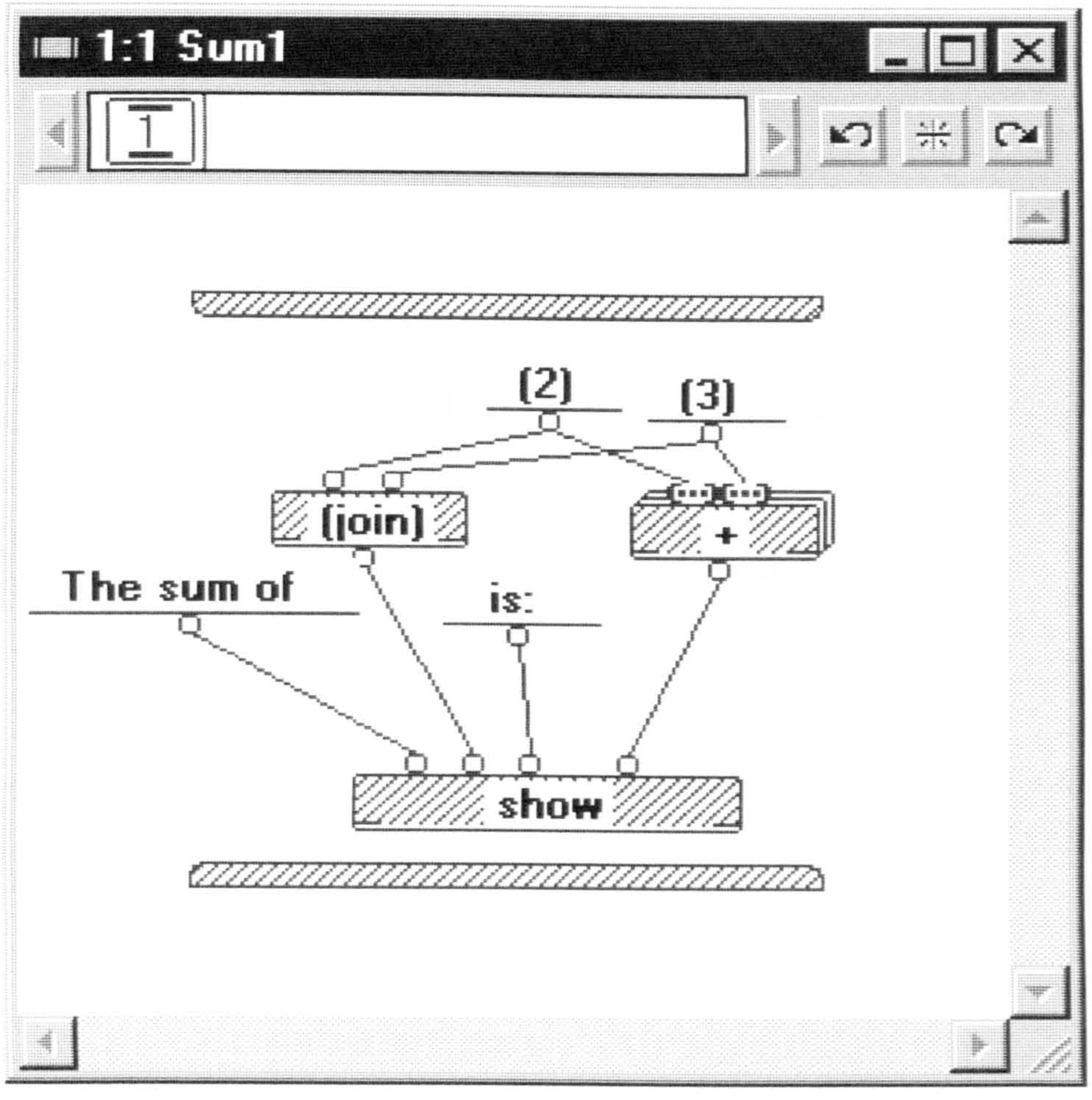
yes O no O

if yes, could you please state the kind of experience you have.

_____

_____

Questionnaire : "Readability" of VPL programs



Q4: Please describe the meaning of the Prograph program on the left.

A: _____

_____

_____

_____

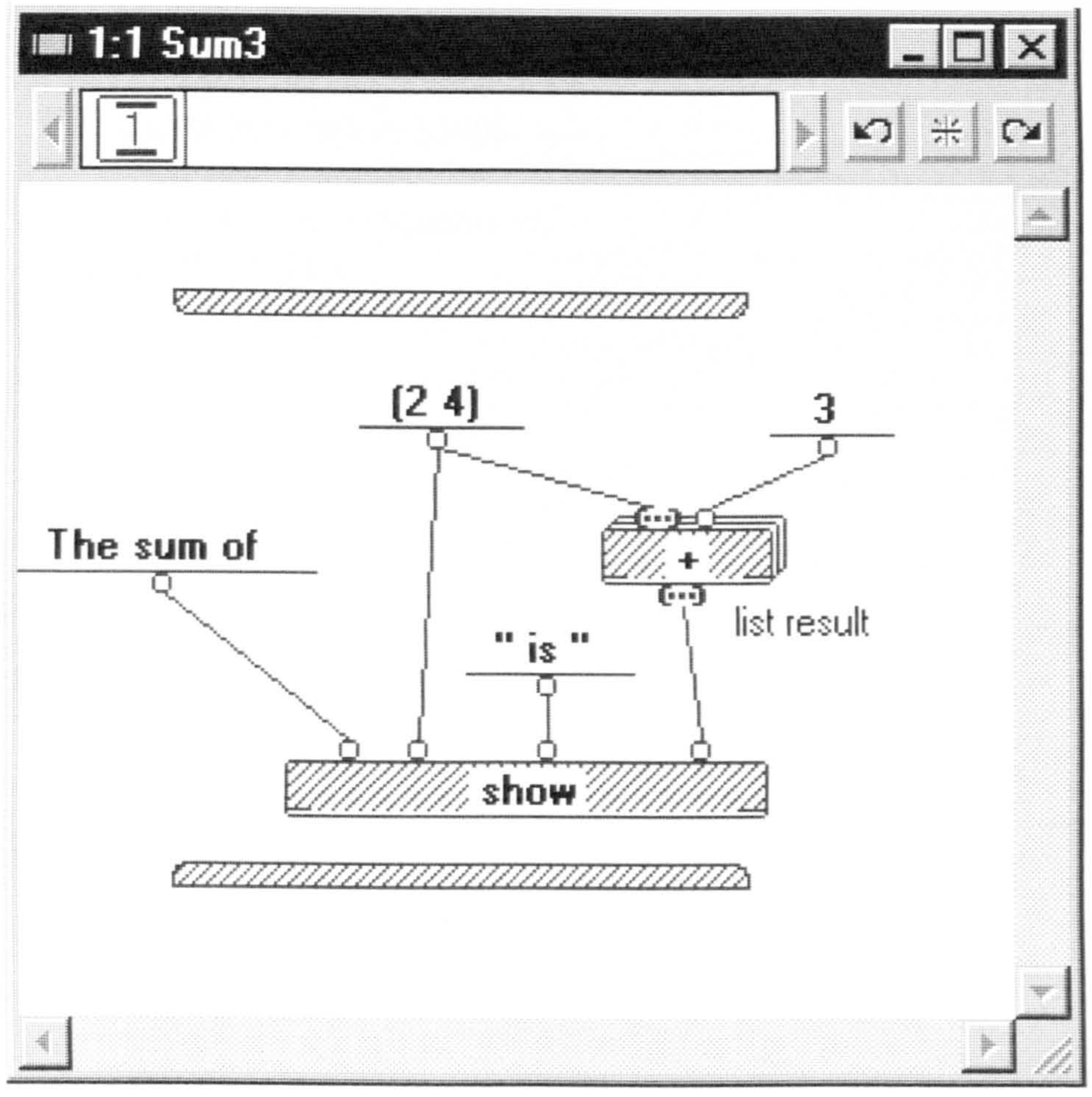
Screen output:

_____

Help:

+	Mathematical operation for adding numbers
(join)	Concatenates two or more lists.
show	Display terminals on the screen

Questionnaire : "Readability" of VPL programs



Q5: Please describe the meaning of the Prograph program on the left.

A: _____

_____

_____

_____

_____

Screen output:

_____

Help:

+	Mathematical operation for adding numbers
show	Display terminals on the screen

Questionnaire : "Readability" of text-based programs

```
INTEGER N, F1, F2, F3, IE, IO
PARAMETER (IE=0, IO=0)

WRITE (IO, '(A)') 'Number ='
READ (IE, 111) N

F1 = 1
F2 = 1

DO 10  I = 3,N,1
      F3 = F1 + F2
      F1 = F2
      F2 = F3
10  CONTINUE
    WRITE (IO, 111) F3
    STOP

111 FORMAT (I3)
```

...

Q6: Please describe the meaning of the Fortran program on the left.

A: _____

_____

_____

_____

Screen output for **N = 4**:

_____

_____

Help:

INTEGER	Data definition
PARAMETER	Constant definition
DO..CONTINUE	Loop
FORMAT	Read/Write format for I/O

Questionnaire : "Readability" of programs

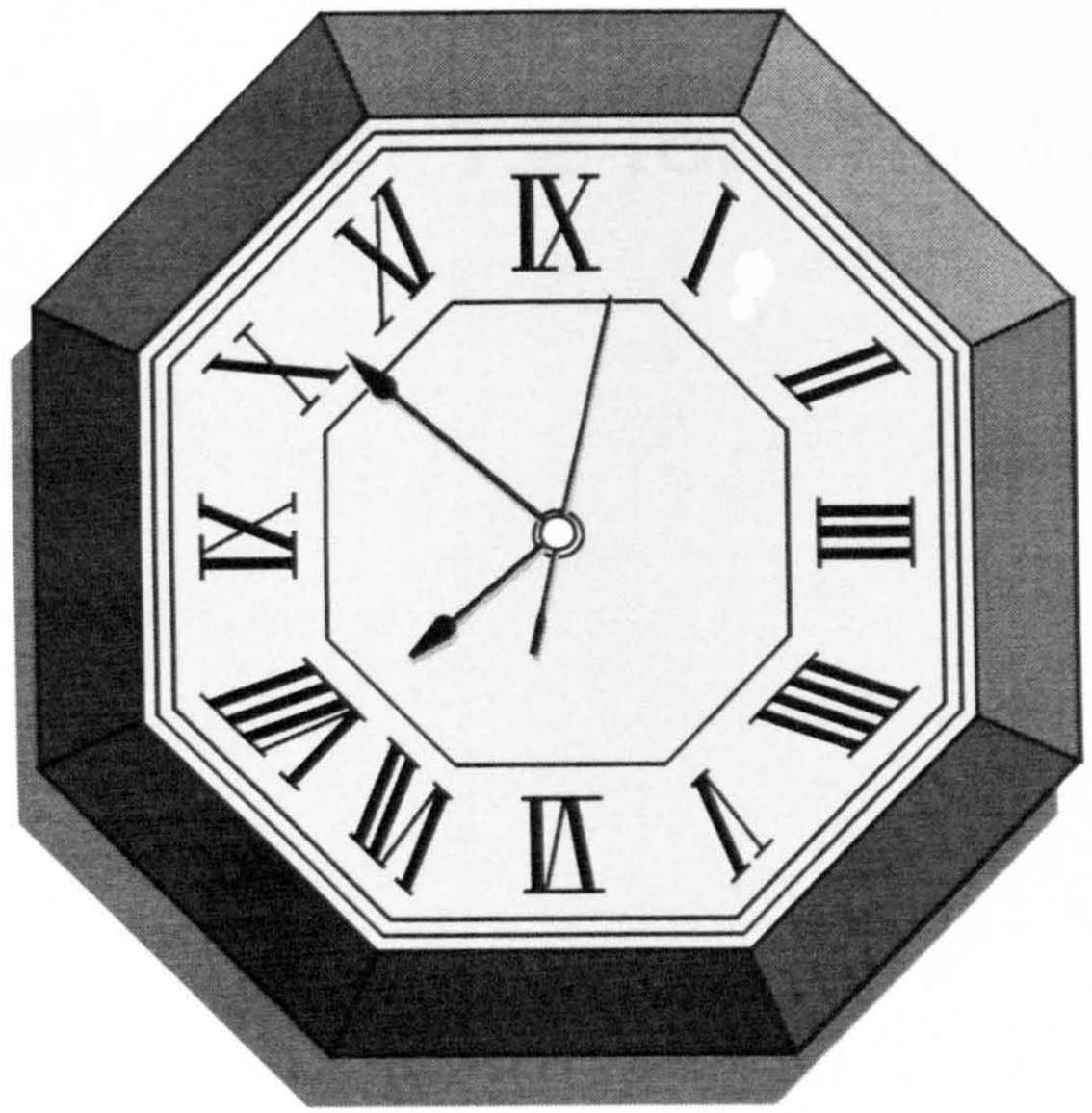
Q7: What **problems and experiences** did you encounter while trying to understand the Prograph/Fortan code?

A1: (comments related to Prograph)

A2: (comments related to Fortran)

End of Questionnaire

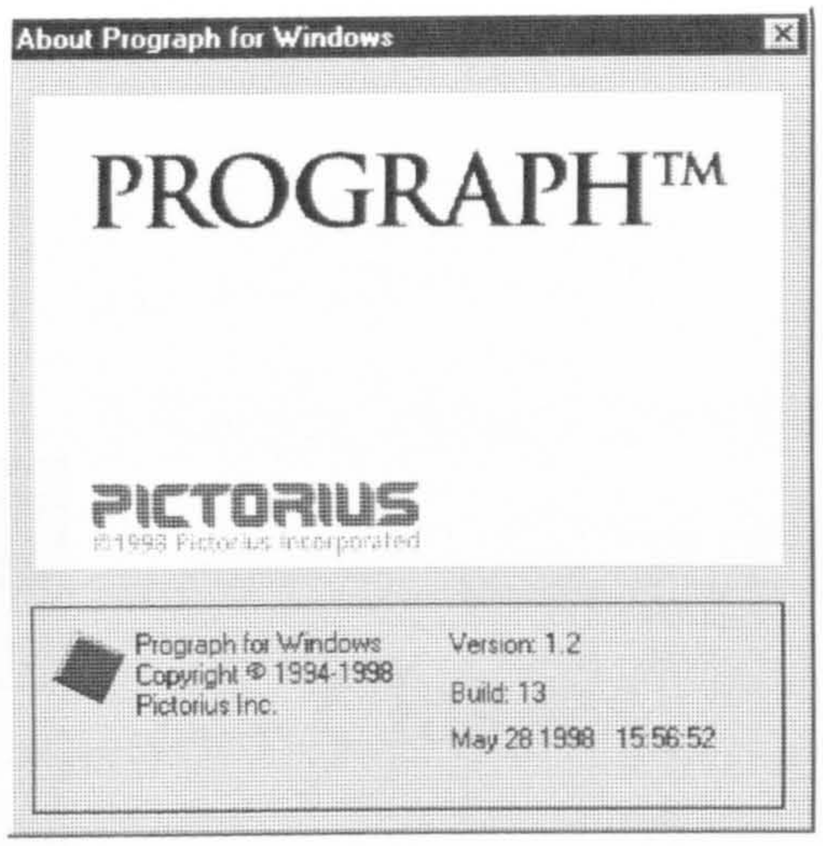
15 minutes



Thank you!

Introduction to Prograph 1.2 for Windows

Prograph for Windows 1.2



A Prograph application is tied together using a **project file**.

A project file contains a list of the **sections** that comprise that particular project.

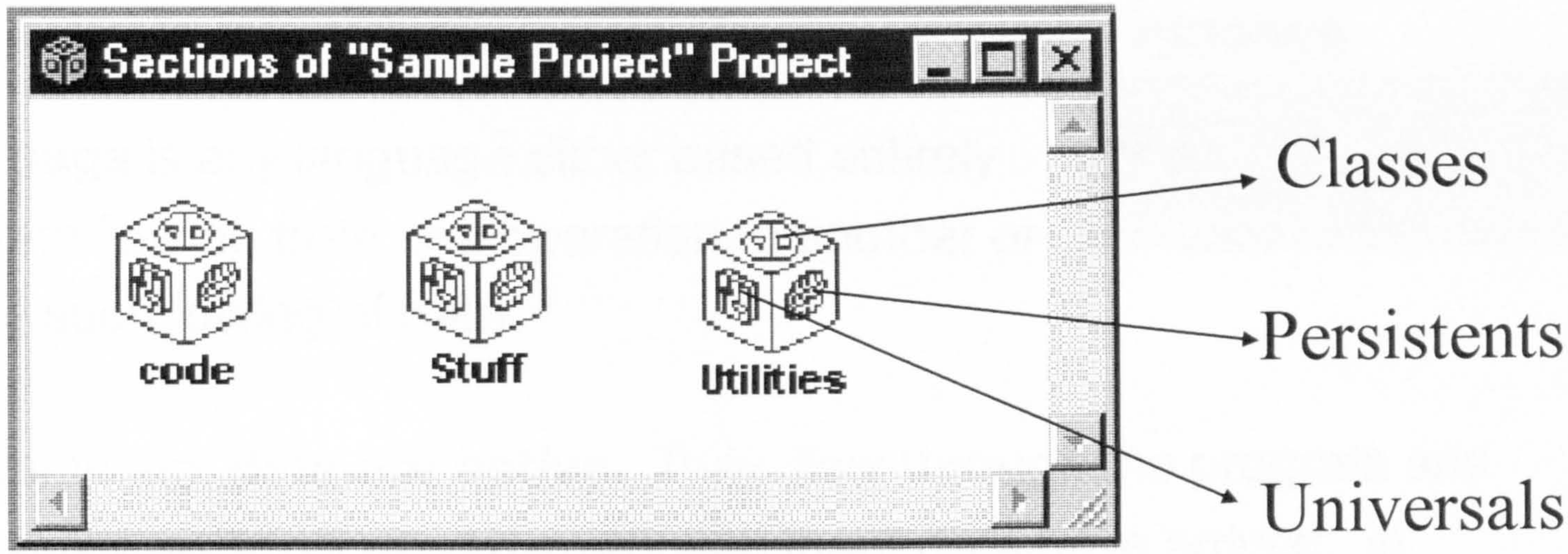
A section is an individual disk file that stores Prograph source code. Typically, a **set of functionally** related source is stored in a single section. For example, all code for helper functions might be stored in a single section.

Once a section has been added to a project, the program elements defined in that section are available for use in the application. **Program elements** can be universal methods, persistents, and/or classes.

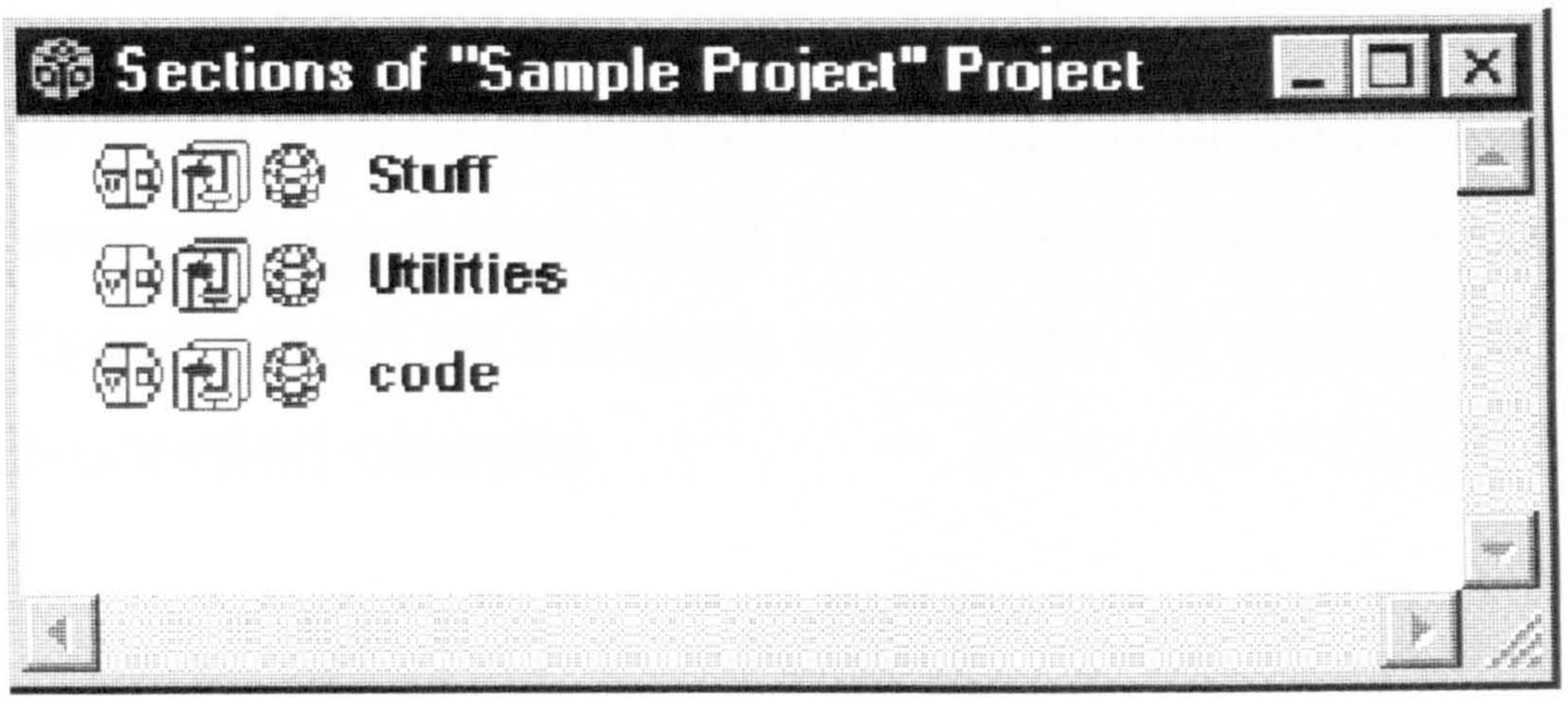
Prograph for Windows 1.2

Creating a section

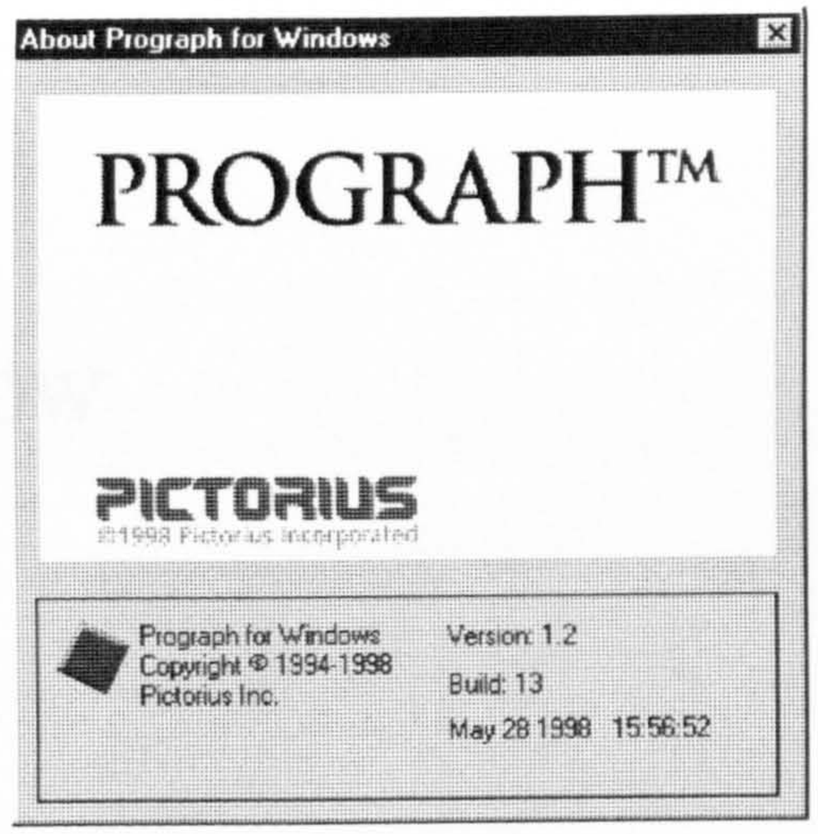
Iconic View



View by Name

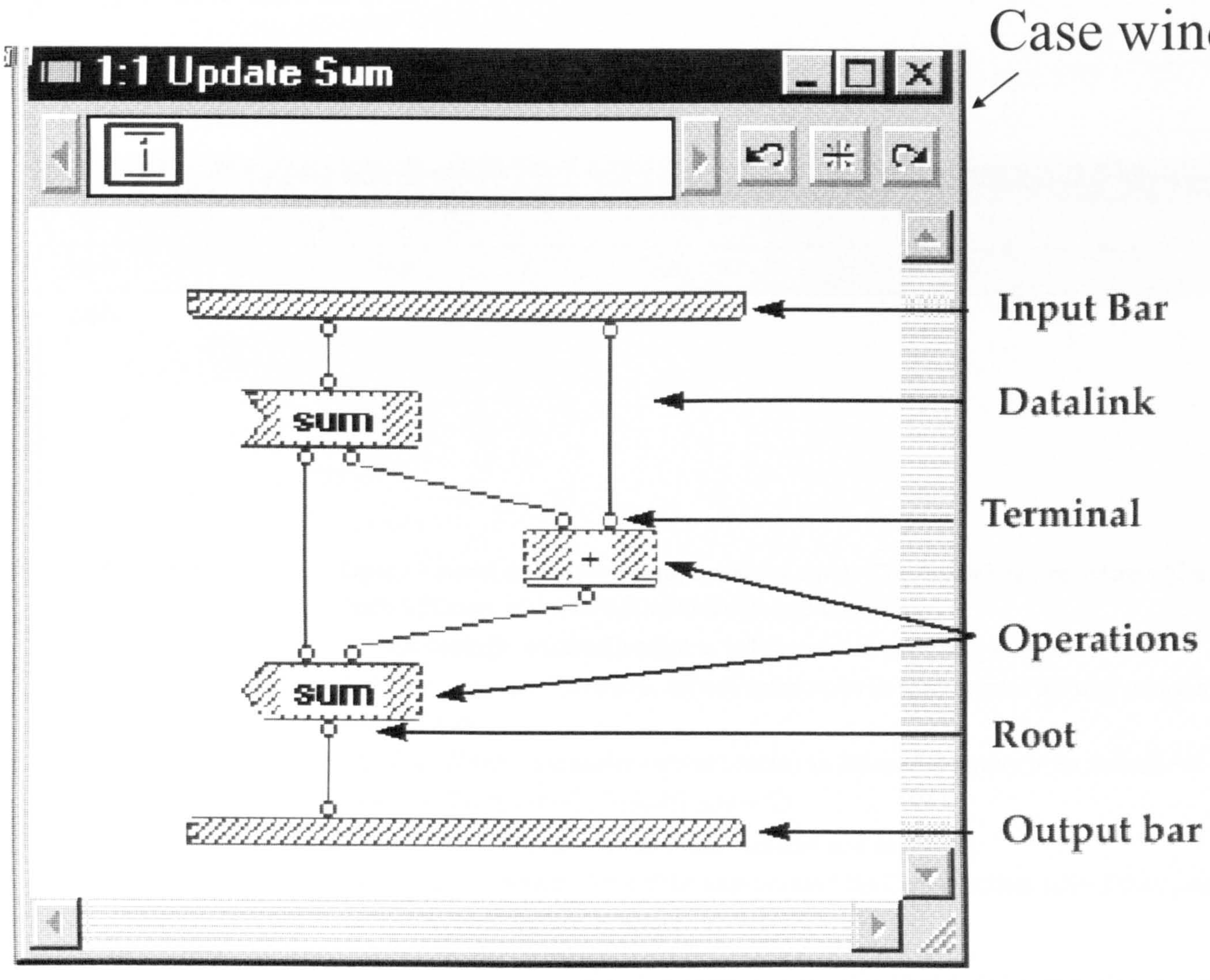


Prograph as a data flow language



A **data-flow language** is any language either based entirely on the notion of data flowing from one operation to another or directly supporting such owing of data.

In data-flow programming, **data are active**. They flow through the program and activate each operation as soon as all the required input data have arrived. In Prograph, operations can be anything from a simple system-supplied primitive, such as addition, to a call to a user-defined method that can be arbitrarily complex. Data traveling along datalinks in a Prograph program can be either **elementary data types**, such as integers or strings, **or complex objects**; instances of object-oriented classes.


Case window
1:1 Update Sum
sum
+
sum
Input Bar
Datalink
Terminal
Operations
Root
Output bar

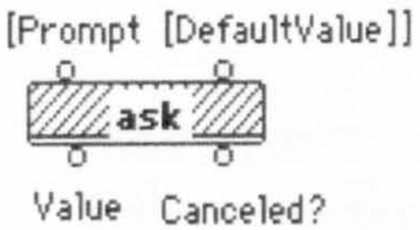
Prograph 1.2
Important operations

Prograph Help

Datei Bearbeiten Lesezeichen Optionen ?

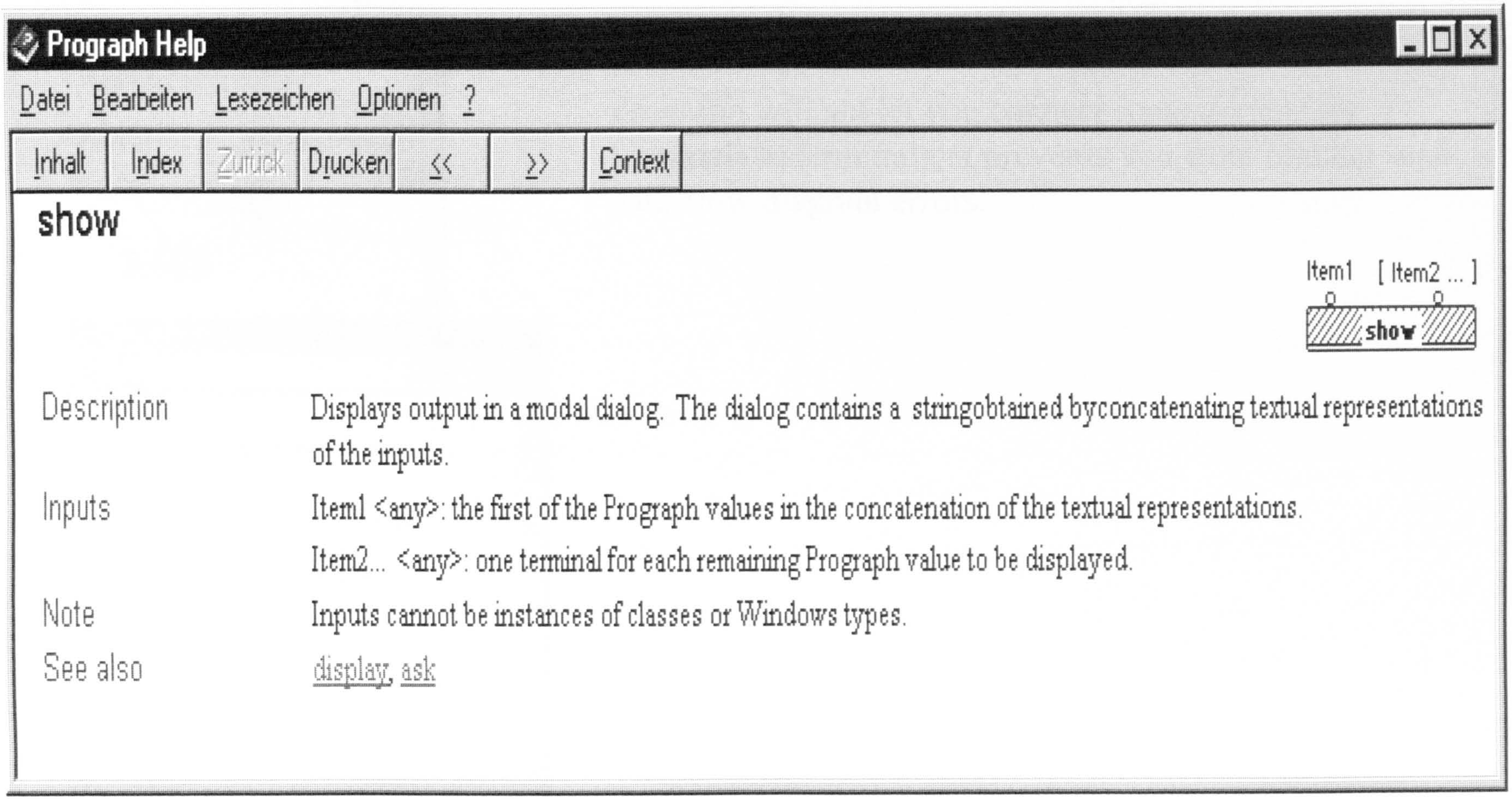
Inhalt | Index | Zurück | Drucken | << | >> | Context

ask



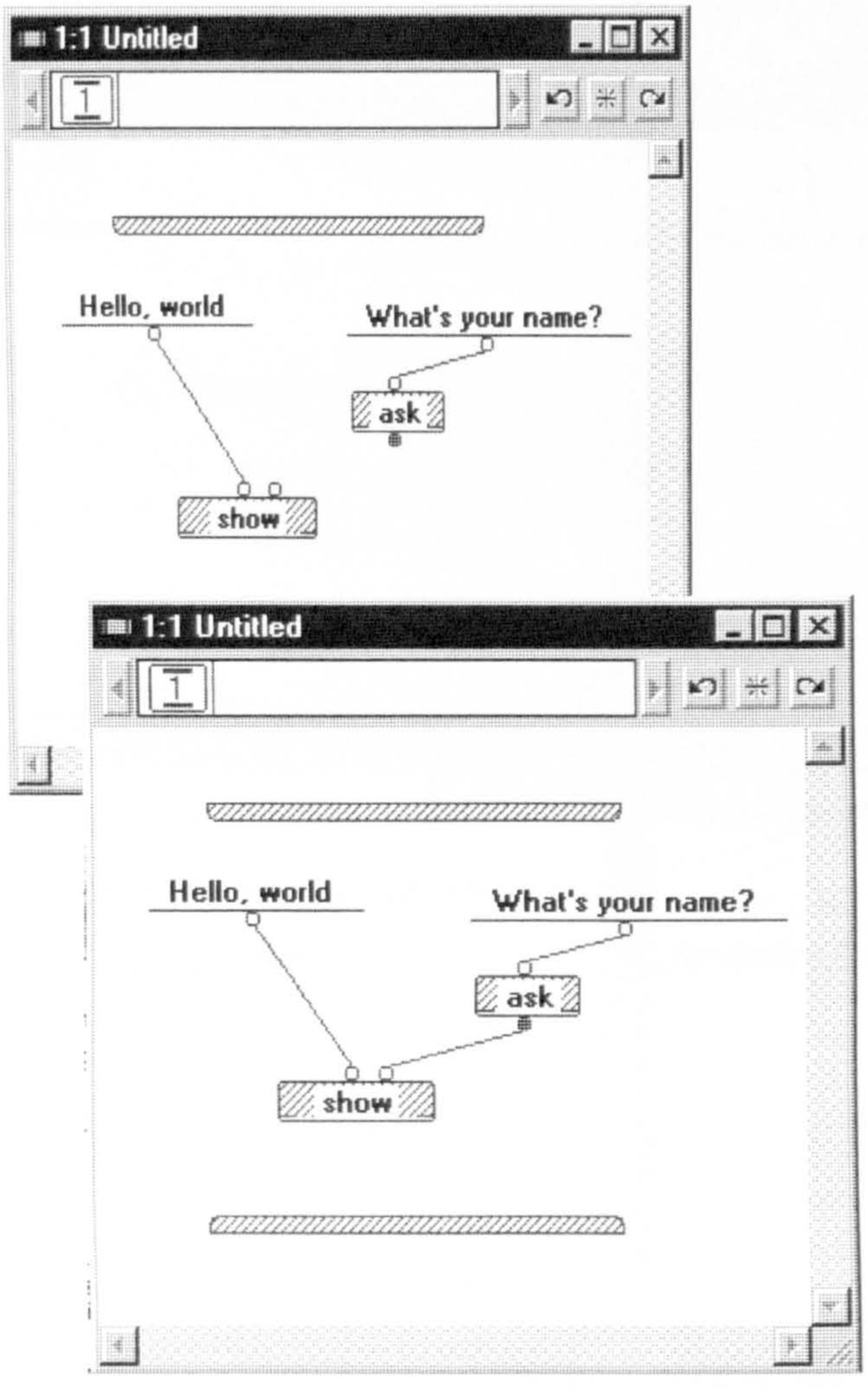
Description	Opens a modal dialog prompting a user for input. The dialog has two buttons (Cancel and OK), an editable area, a textual prompt, and a default value in the editable area.
Inputs	Prompt <string>: a textual prompt to aid the user in providing a value. DefaultValue <any>: an initial, default value to be used as input if the user presses OK without typing a value in the editable area.
Note	The DefaultValue parameter cannot contain an External structure or an instance of a class
Default(s)	Prompt = 'Enter valueÓ; DefaultValue = 'Ó
Outputs	Value <any>: Contains the last value entered and displayed. Canceled? <boolean> : True if the user pressed the Cancel button, false if they pressed the OK button.
See also	accept, answer, answer-v, select

Prograph 1.2
Important operations



Visual Programming with Prograph 1.2

Datalinks

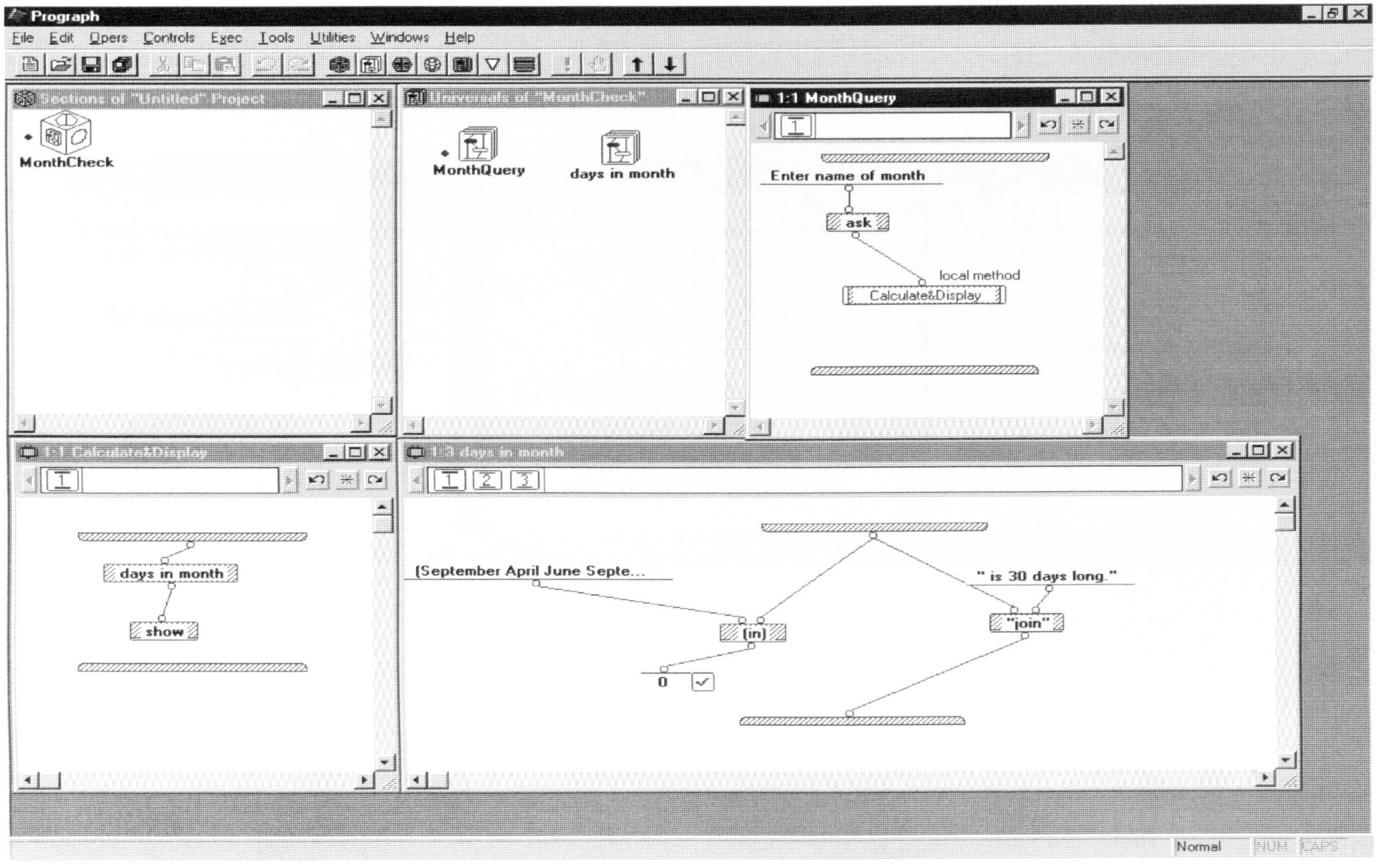


Prograph passes data between operations via terminals and roots, which are connected using **datalinks**.

Prograph will not let you connect roots to roots, or terminals to terminals; it prevents you from making such data-flow diagram errors.

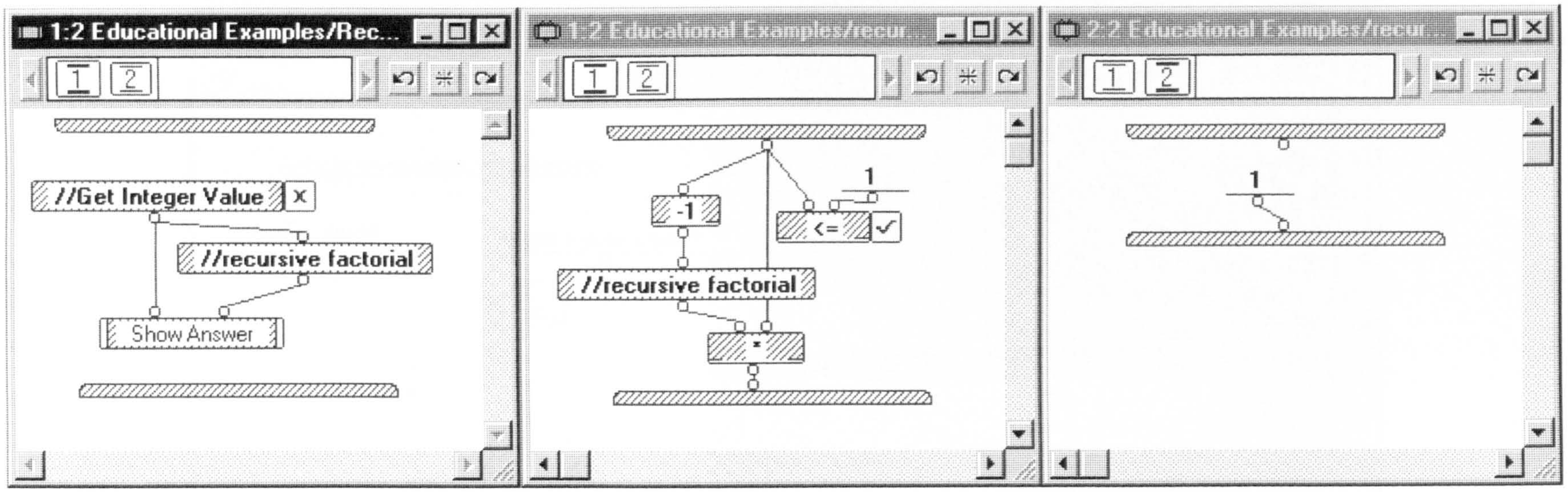
Visual Programming with Prograph 1.2

A sample with local operations



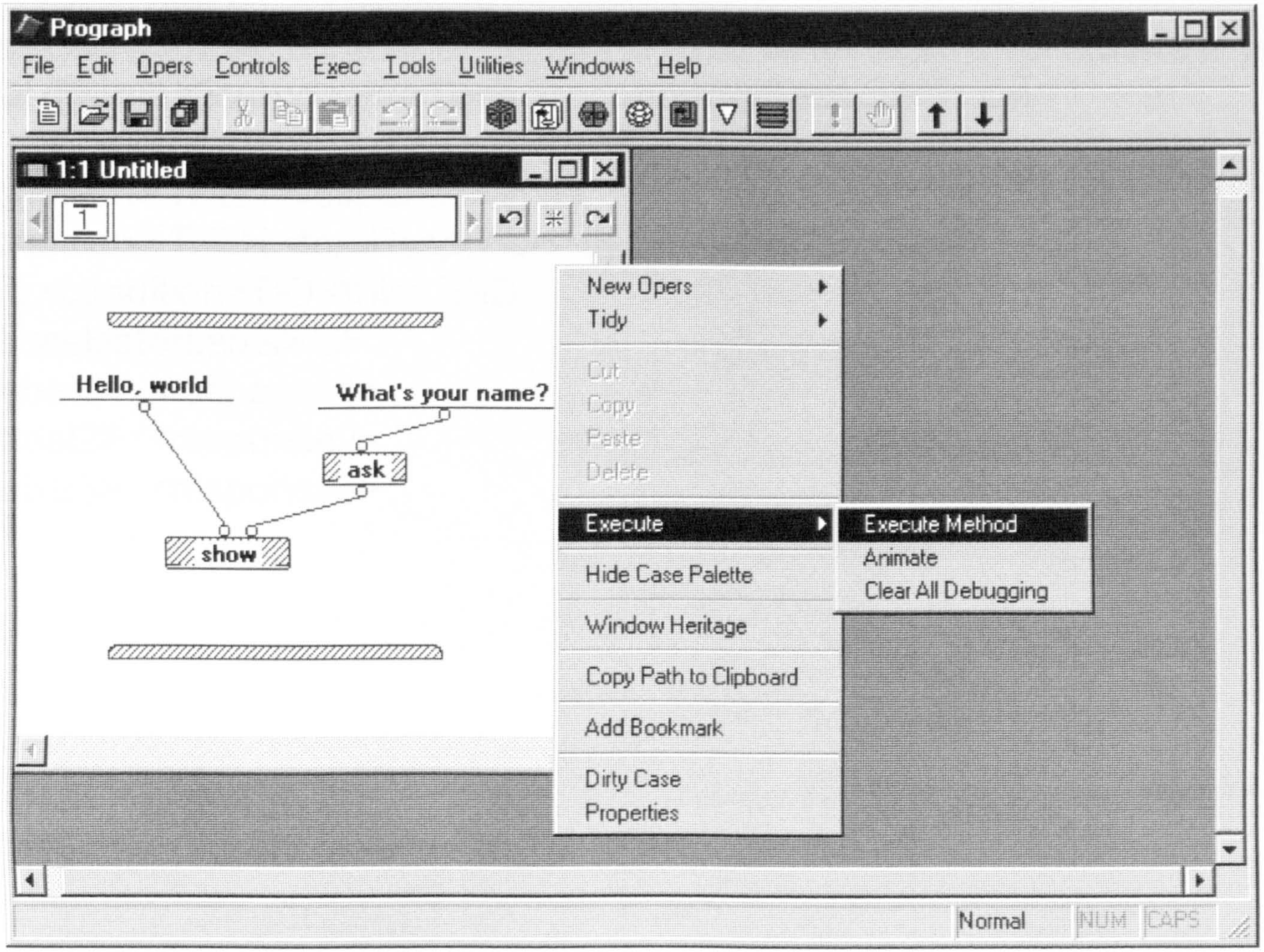
Visual Programming with Prograph 1.2

Recursion



Visual Programming with Prograph 1.2

Executing a method



Visual Programming with Prograph 1.2

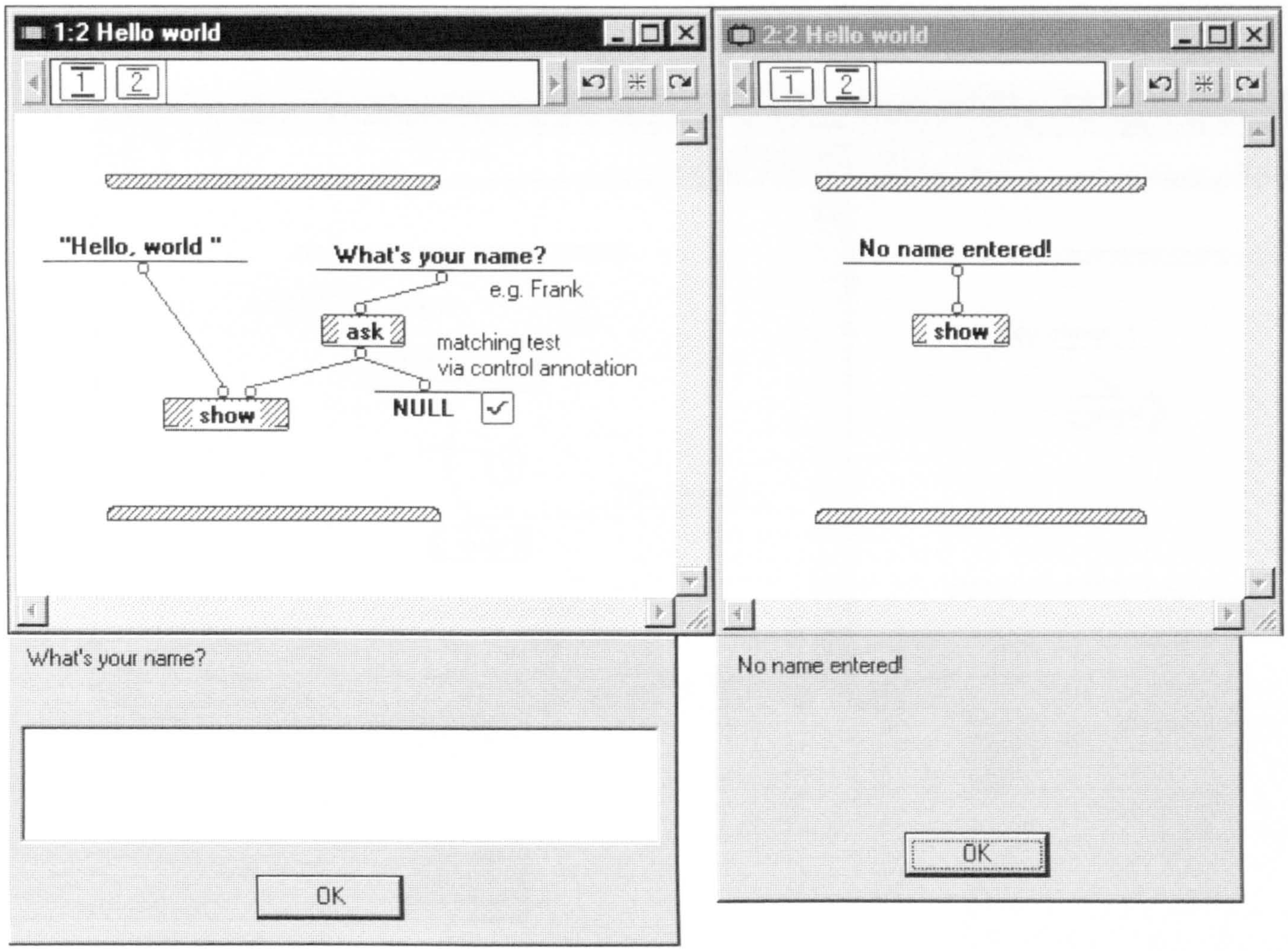
Conditional execution

In **text-based languages**, a particular syntax is used **to structure variations** in program flow. Typical grammatical constructions for conditional execution are:

- IF <condition> THEN <response> END
- IF <condition> THEN <trueResponse> ELSE <falseResponse> END
- WHILE <condition> DO <this> END
- CASE <selectorLabel> OF
  <label1> : <response1> ;
  <label2> : <response2> ;
  <label3> : <response3> ;
  END

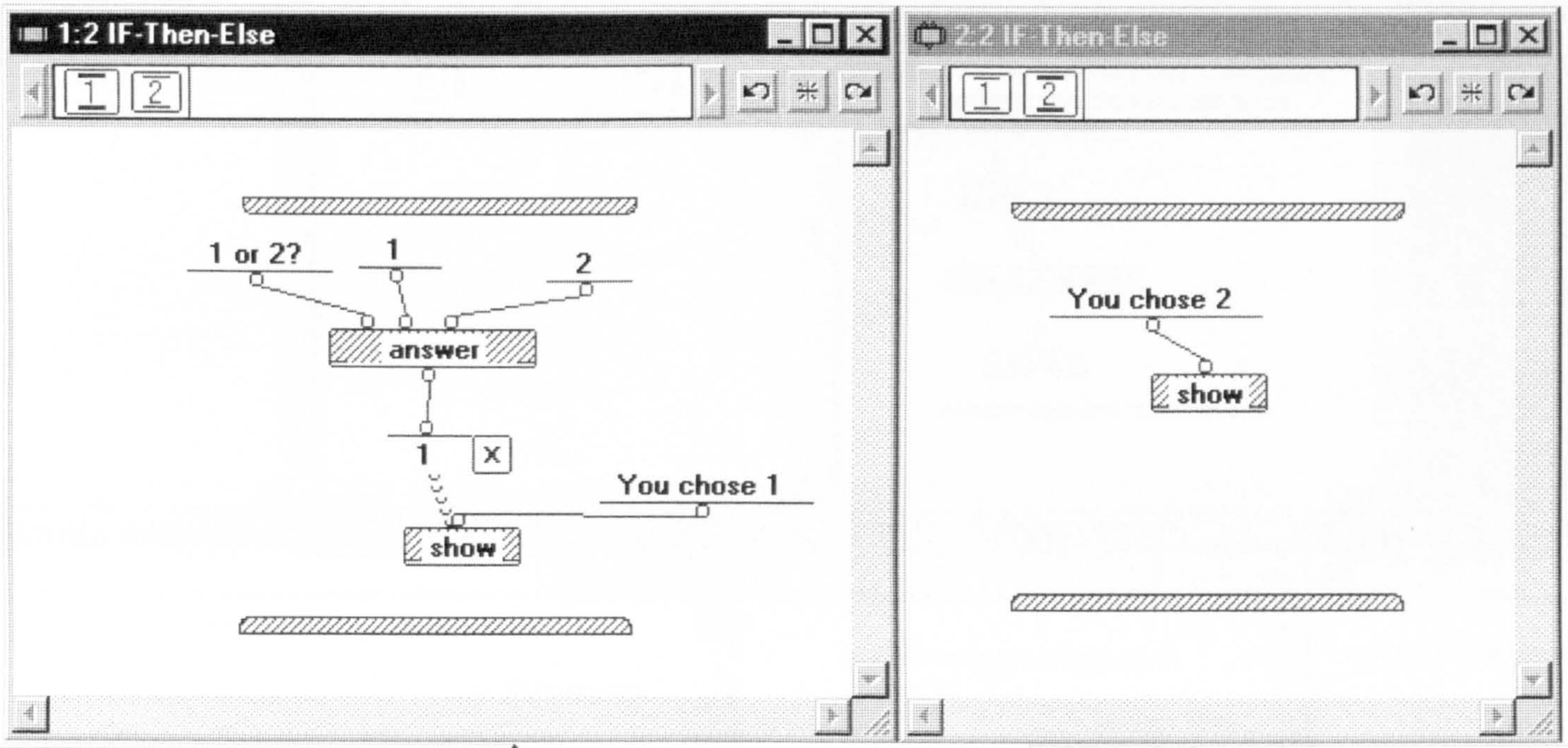
Visual Programming with Prograph 1.2

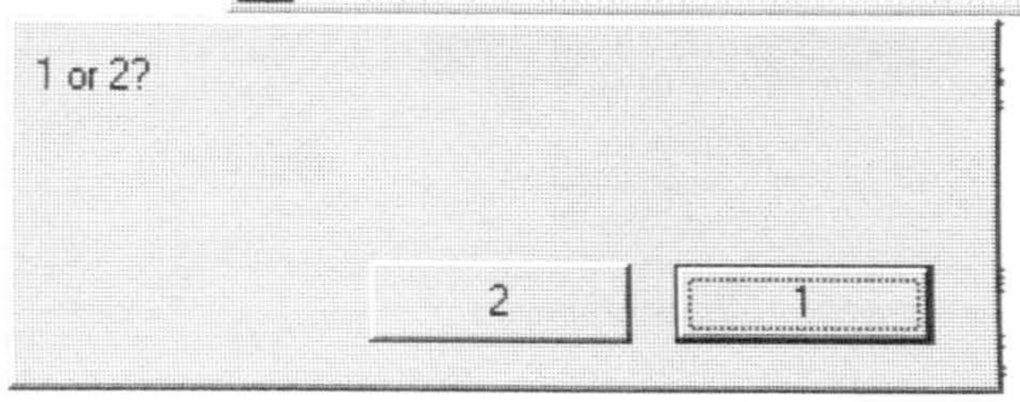
Case windows / data validation through matching tests

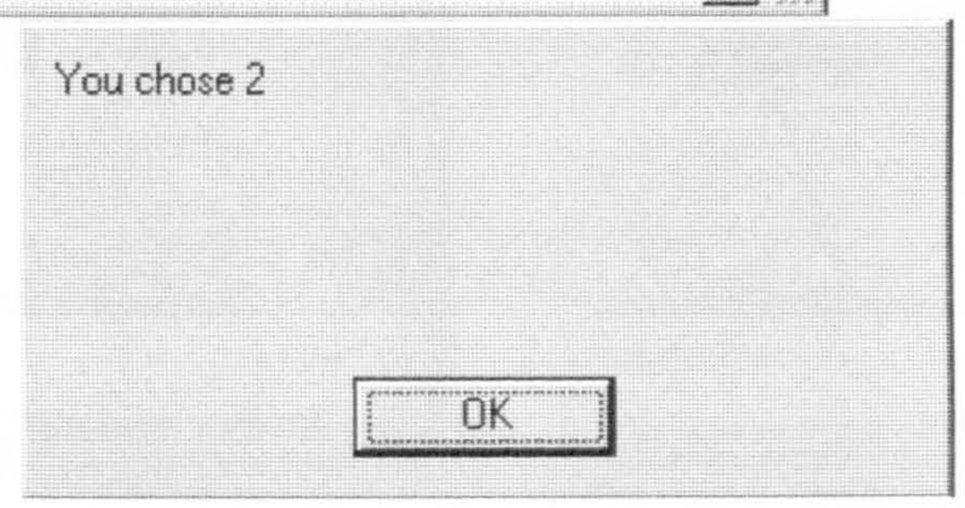


Visual Programming with Prograph 1.2

Case windows / If-then-else

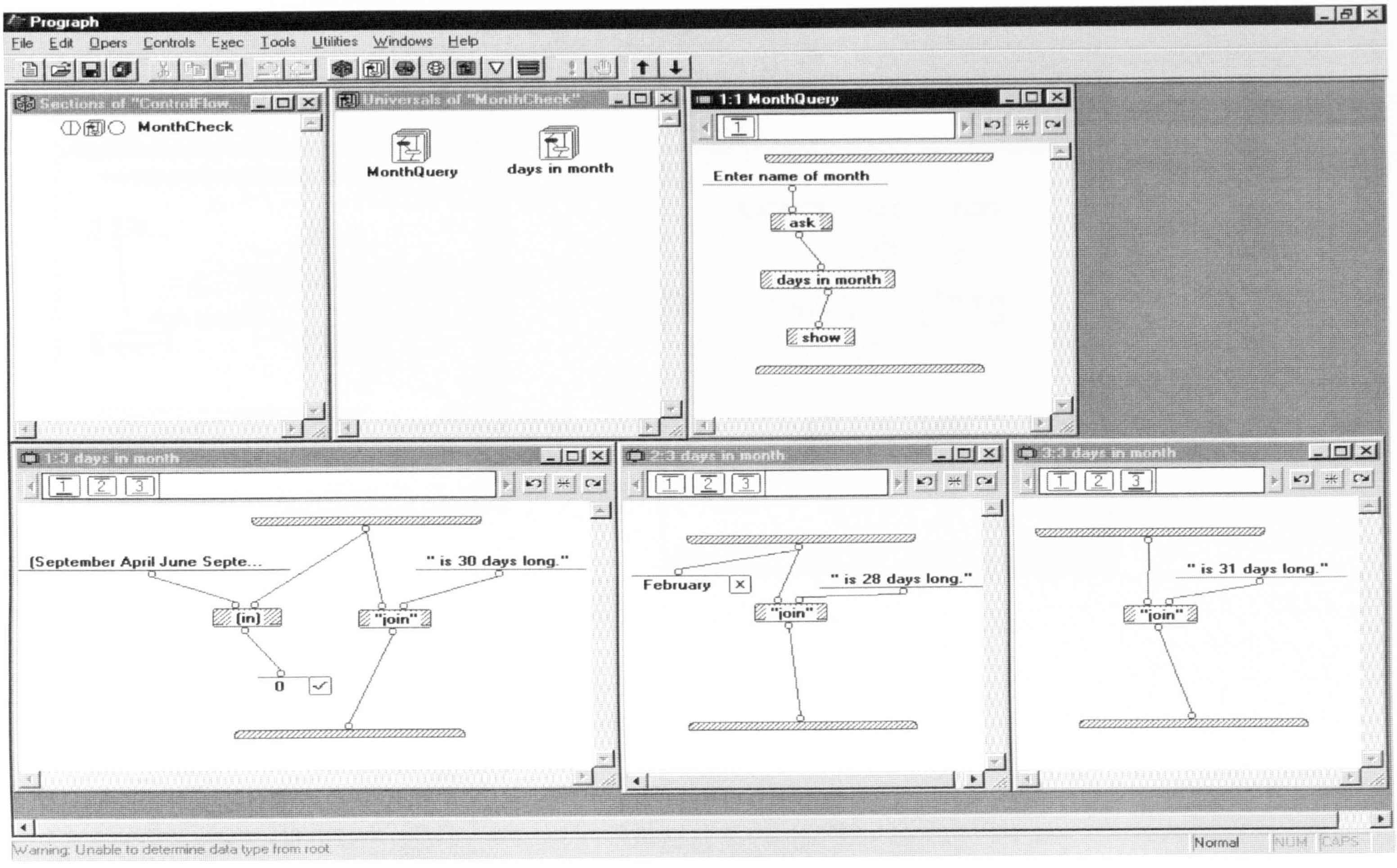






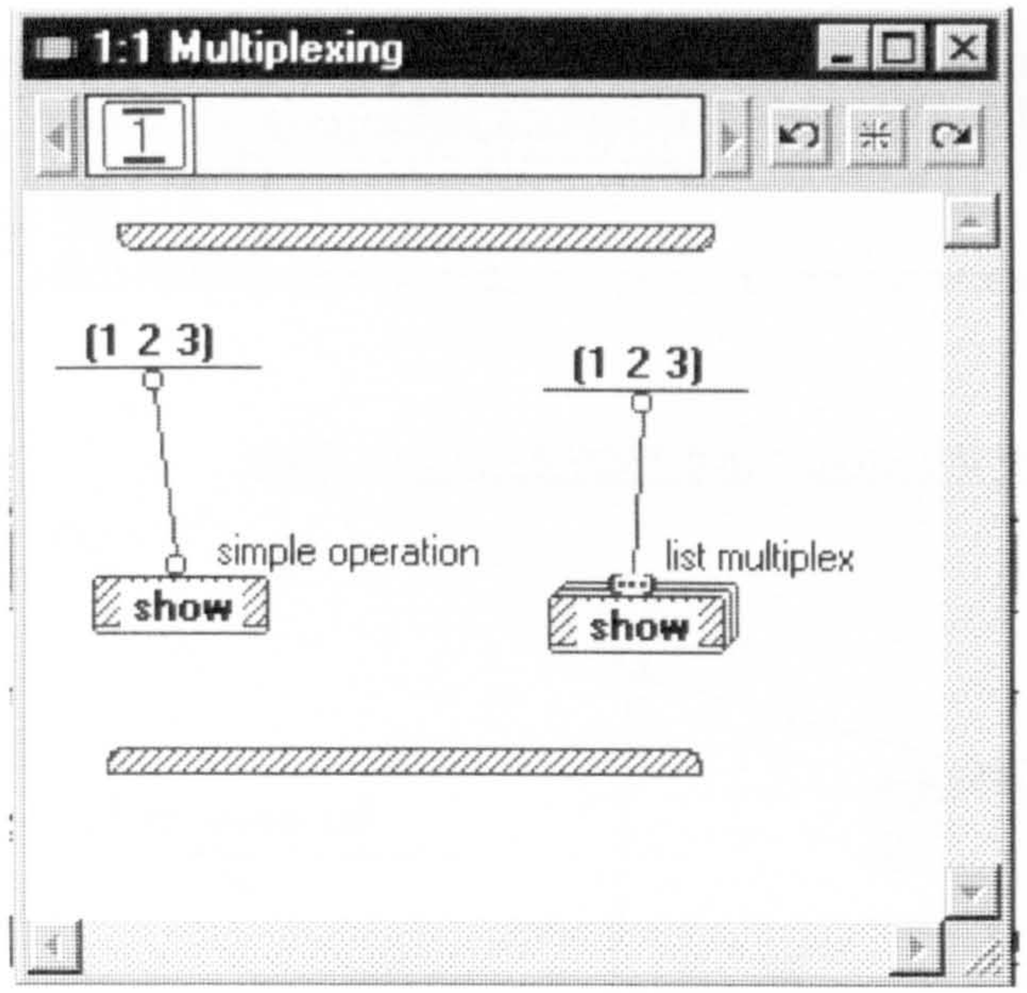
Visual Programming with Prograph 1.2

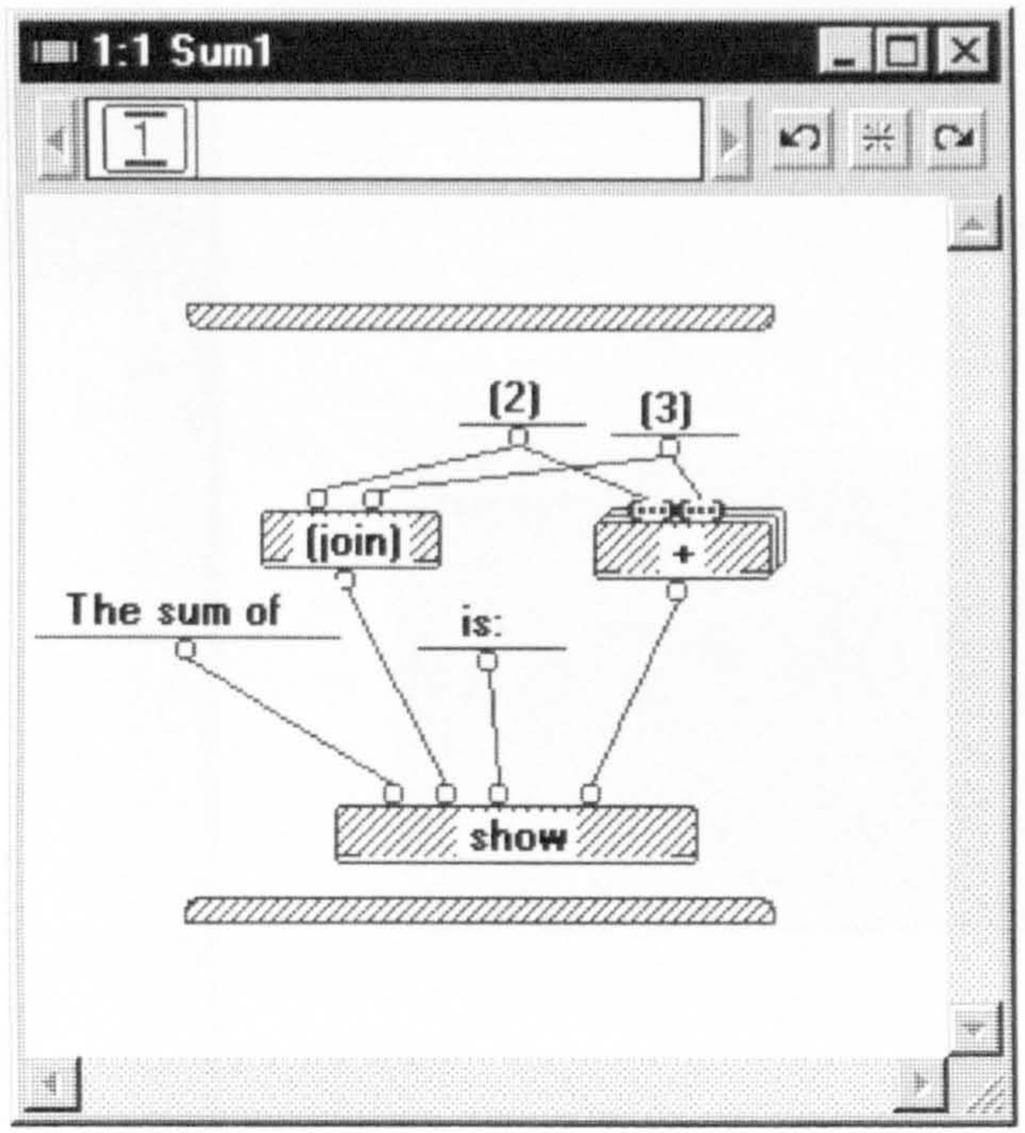
A complete sample: how many days has a month?



Prograph 1.2

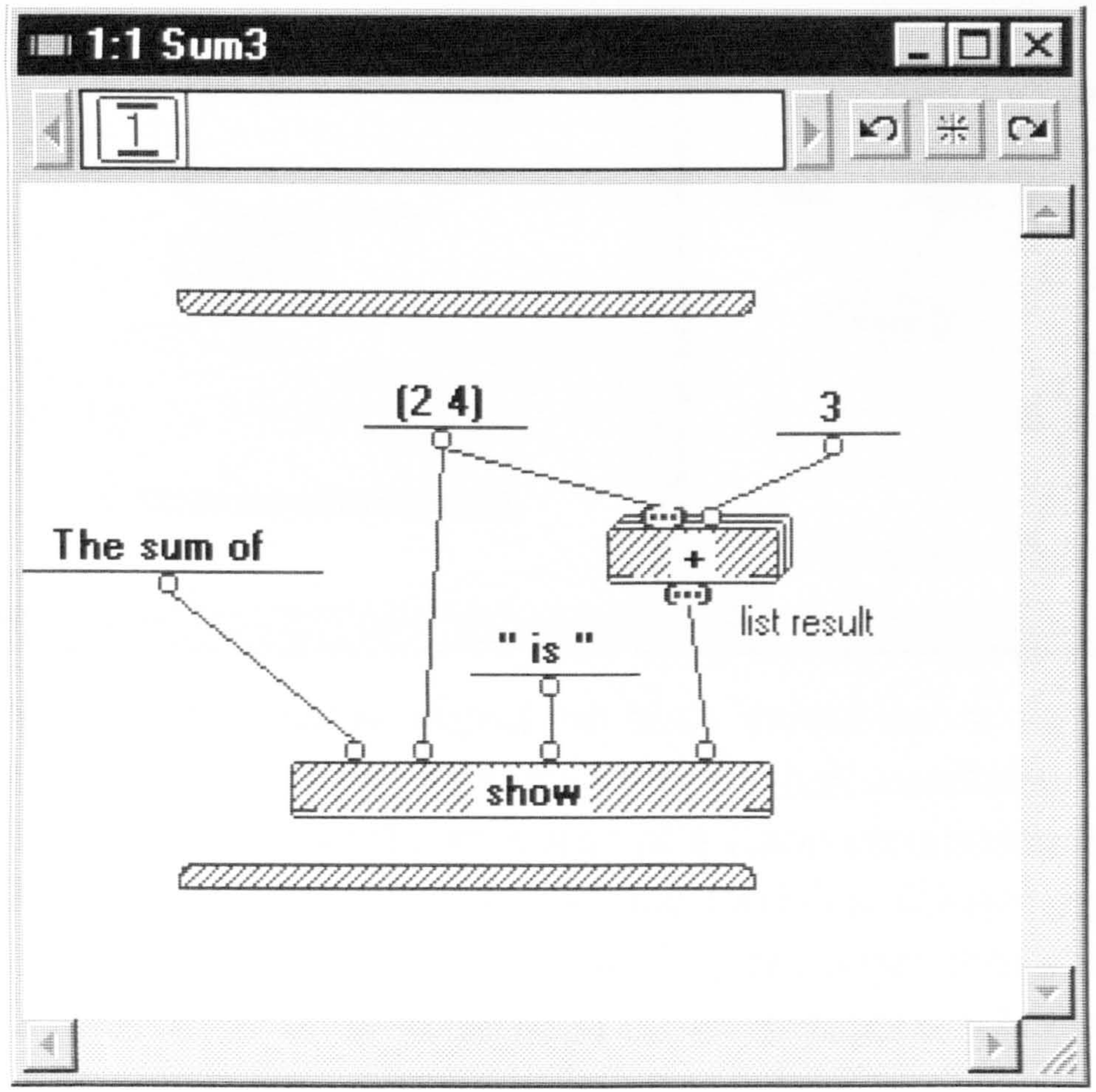
Important operations : list multiplexes

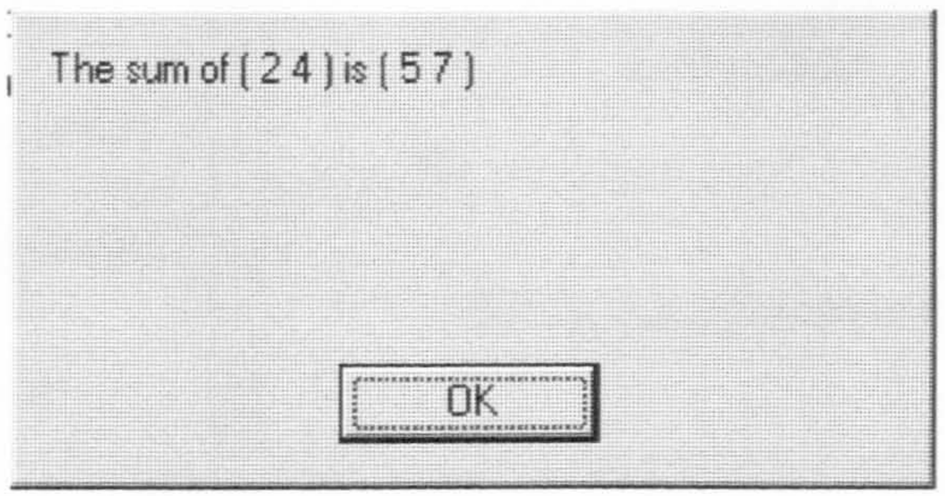




Prograph 1.2

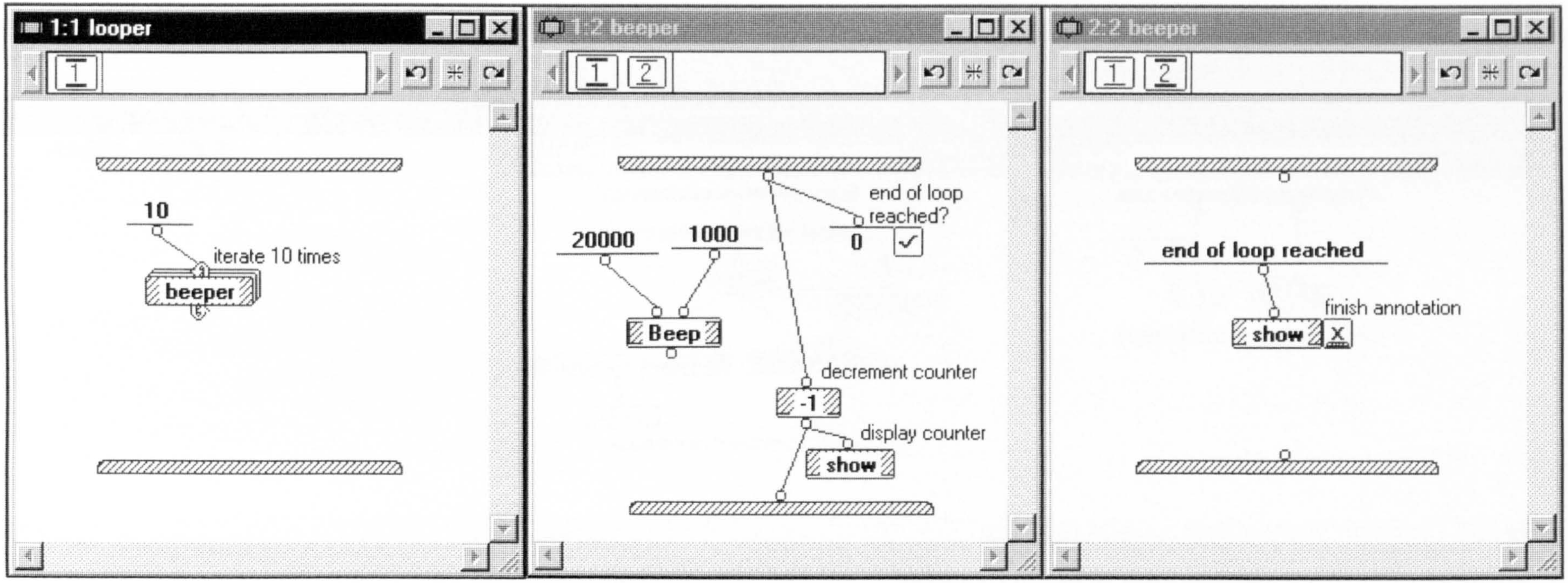
Important operations : list multiplexes





Prograph 1.2

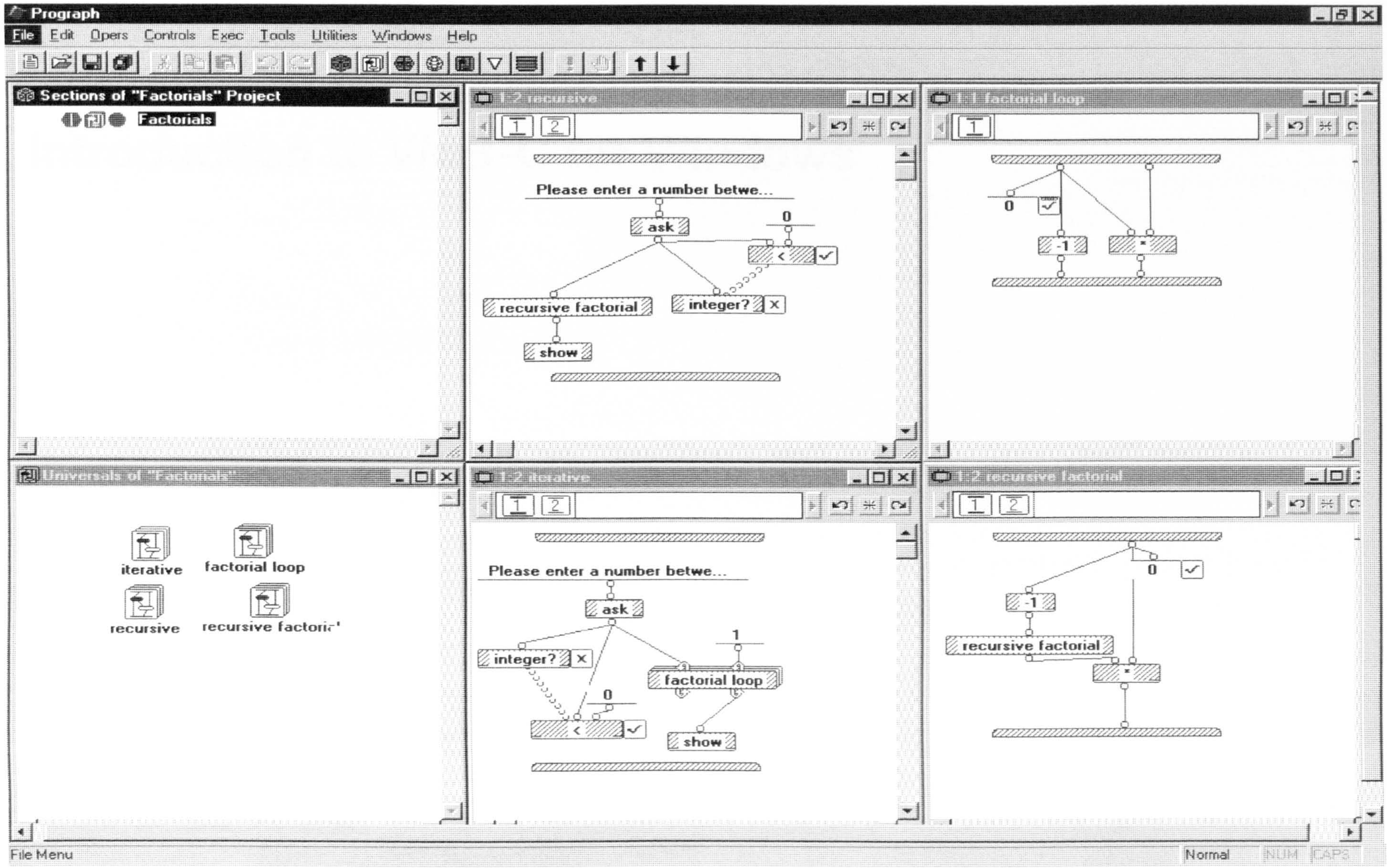
Important operations : loop multiplexes



Once a data value or object has been routed into a loop multiplex, it circles around within the loop until the switch is thrown, meaning the halt condition is met.
No data passes out the root side of a Loop annotation until the multiplex has completed its execution. When the halt condition is met, the switch is thrown and the root side of the Loop annotation functions just like an unannotated root passing its output data along any connected datalink. The value that passes out the Loop annotation root depends on whether the multiplex fails, terminates, or finishes .

Visual Programming with Prograph 1.2

A complete sample before terminating the Prograph training



Introduction to VMB-C for Windows

Introduction to VMB-C

A VMB-C application is tied together using a **project file**. A project file contains a list of the **project definitions** that comprise that particular project.

A project definition contains the source code of a **single application**. Typically, a **main function and further sub-functions** are defined in a single application.

Once a function has been added to an application, the program elements defined in any other function may call this function. **Program elements** can be any (complex) **processors** that represent a certain functionality (e.g. declaration processors, data processors, operation processors).

Introduction to VMB-C

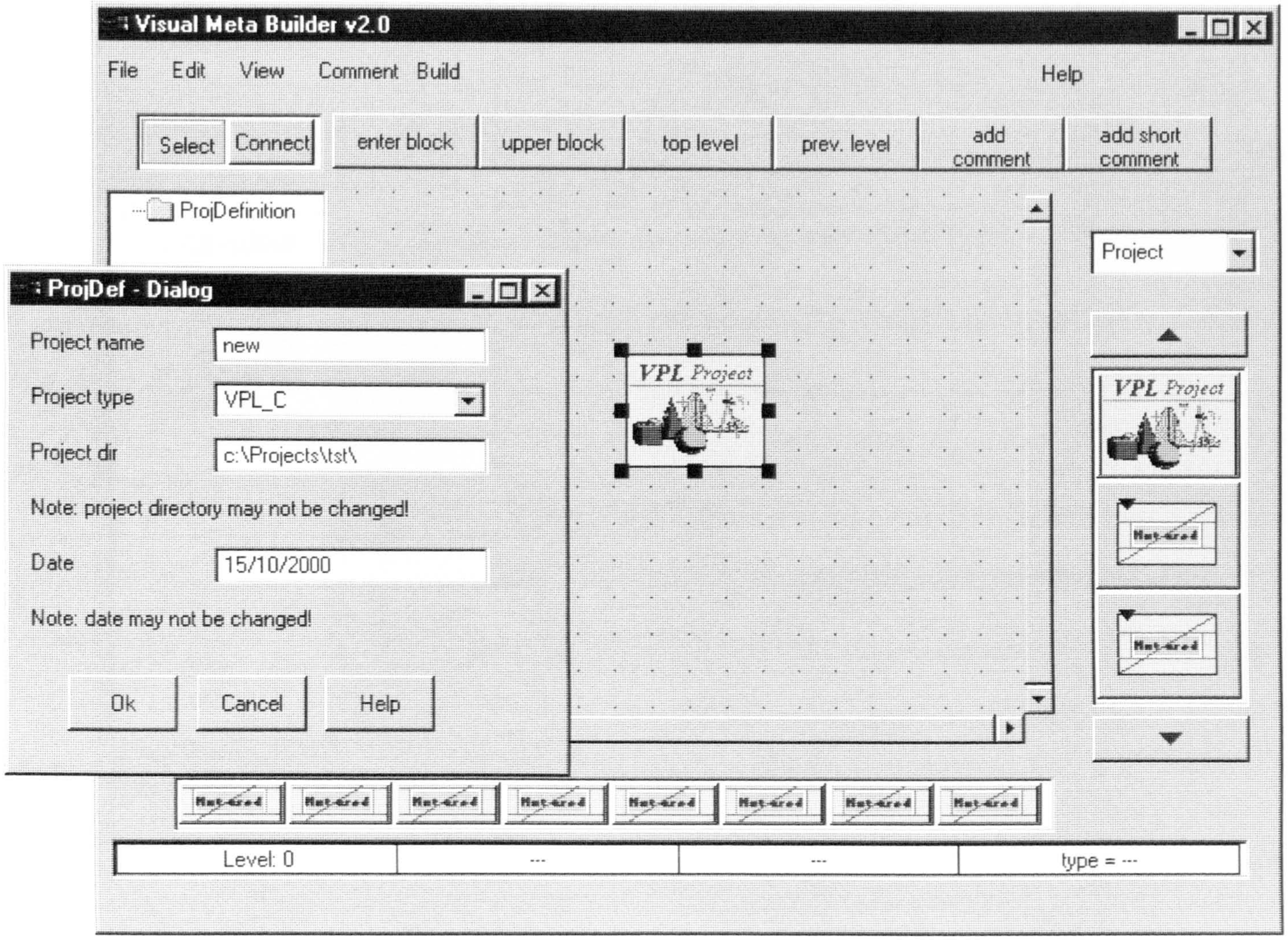
A **control-flow language** is any language based entirely on the notion of control flowing from one operation to another.

In control-flow programming, **data are inactive**. The flow through the program is completely determined through "control operations".

In VMB-C, **operations and data** are represented by so-called **processors and attributes** (i.e. processor properties) which can be anything from a simple system-supplied primitive, such as addition, to a call to a user-defined method that can be arbitrarily complex.

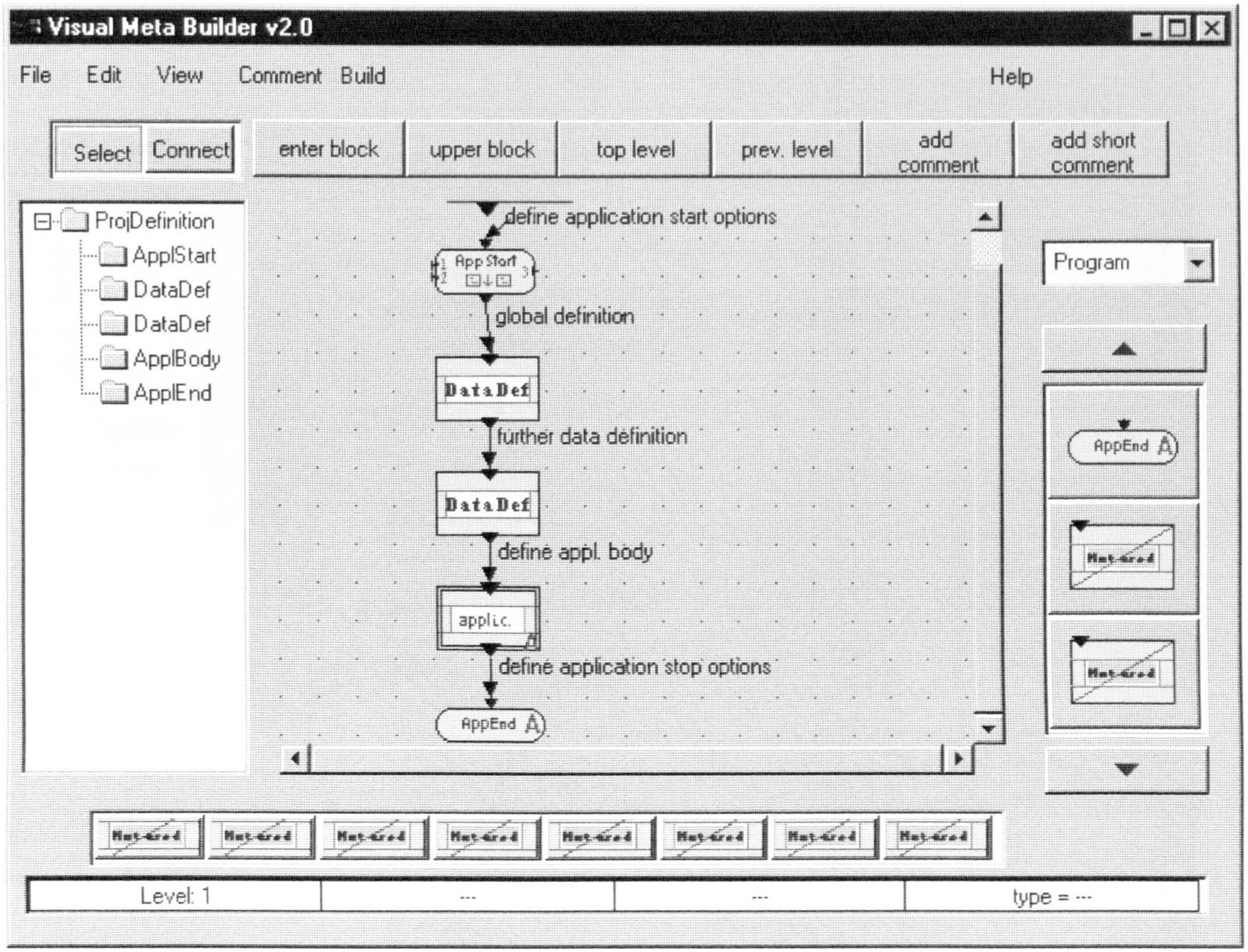
Visual Programming with VMB-C

Program structure



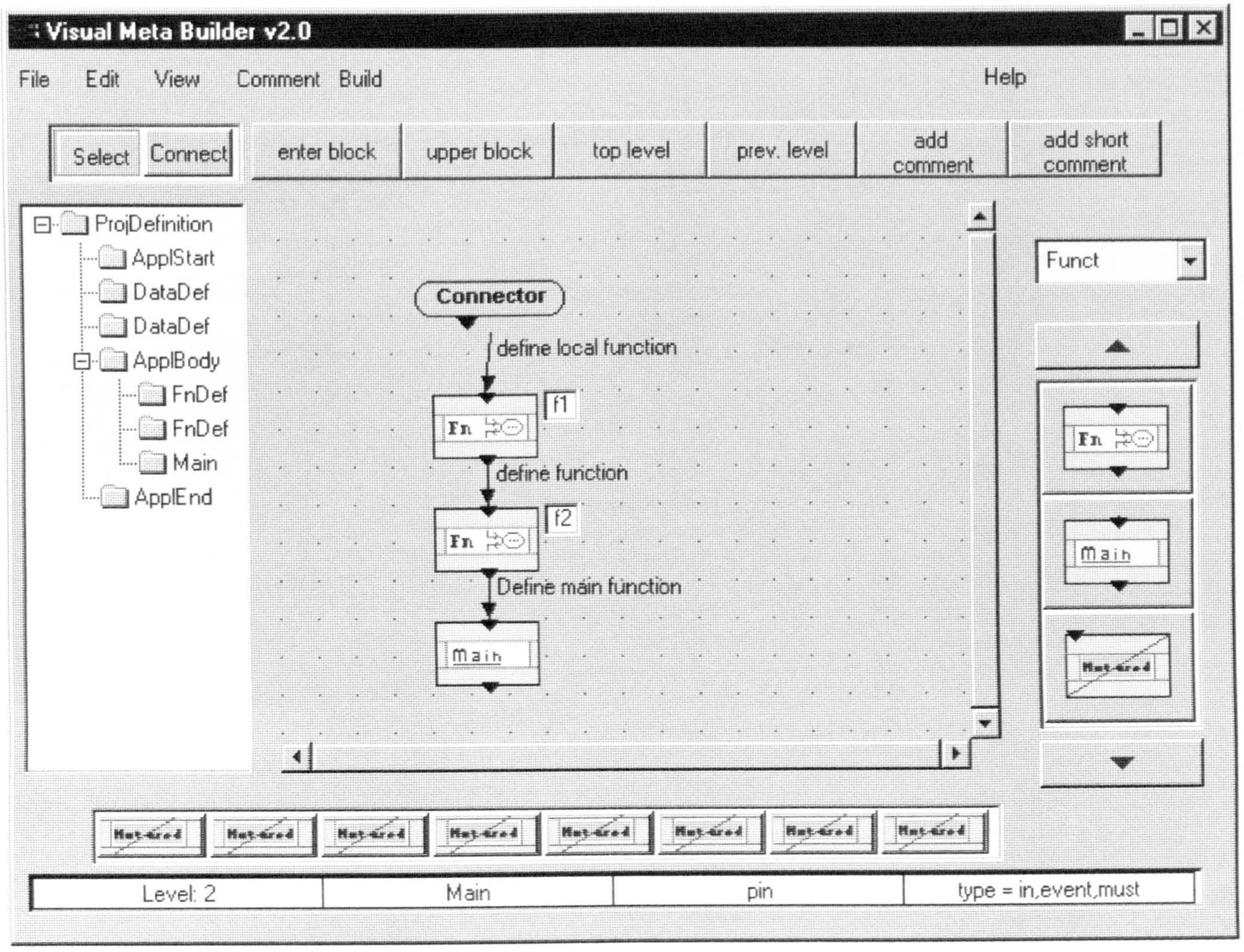
Visual Programming with VMB-C

Program structure



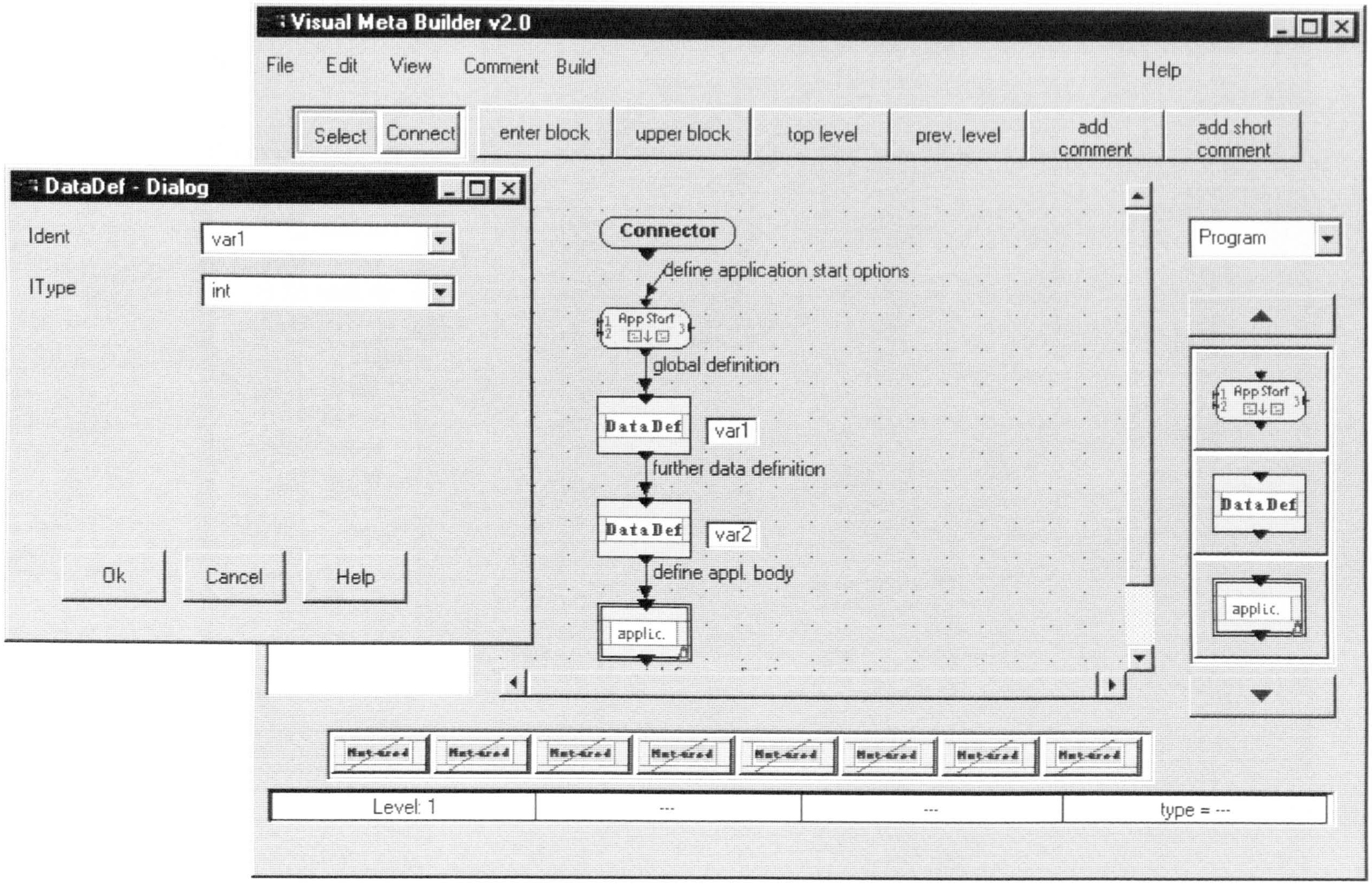
Visual Programming with VMB-C

Program structure



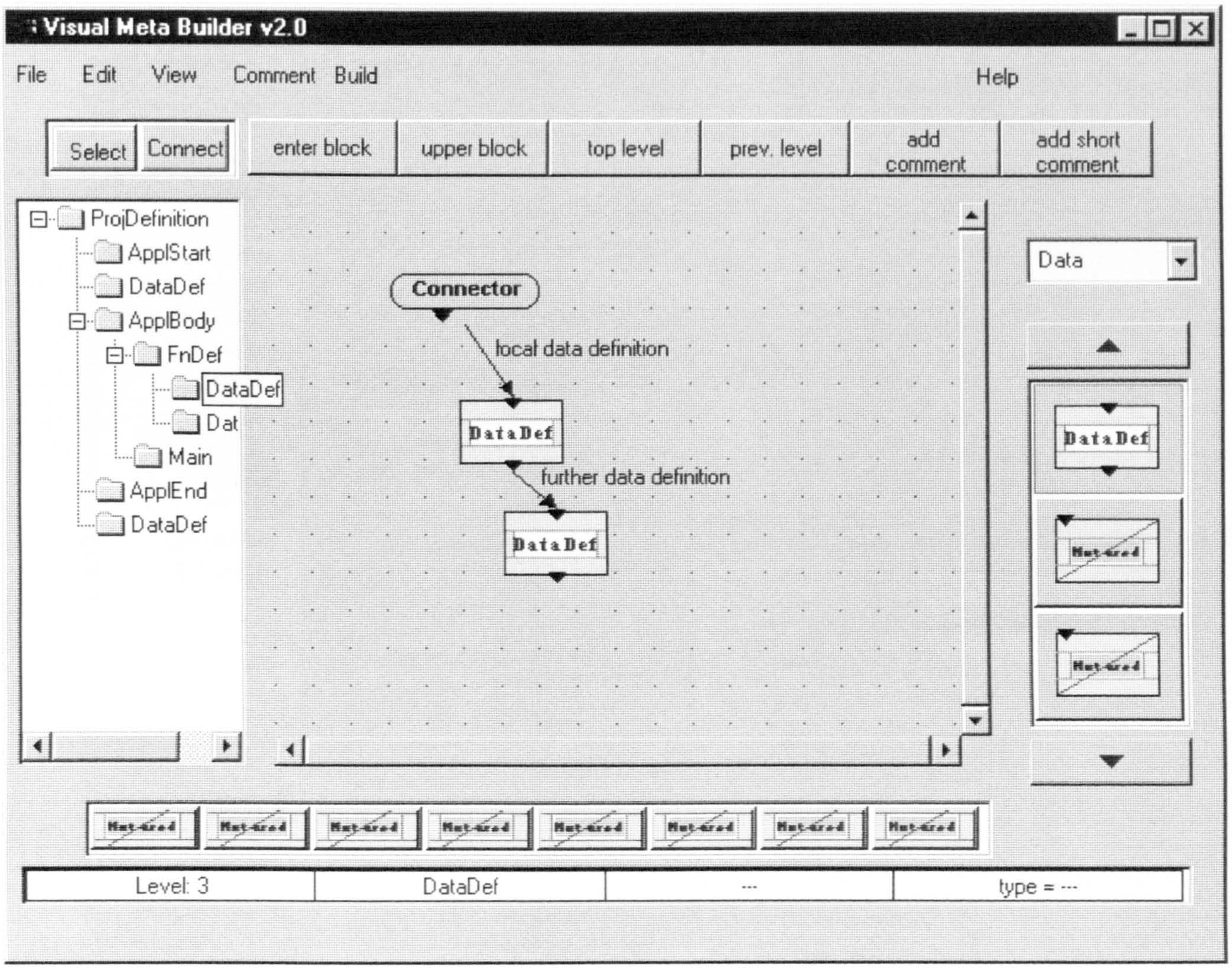
Visual Programming with VMB-C

Global Data Definition



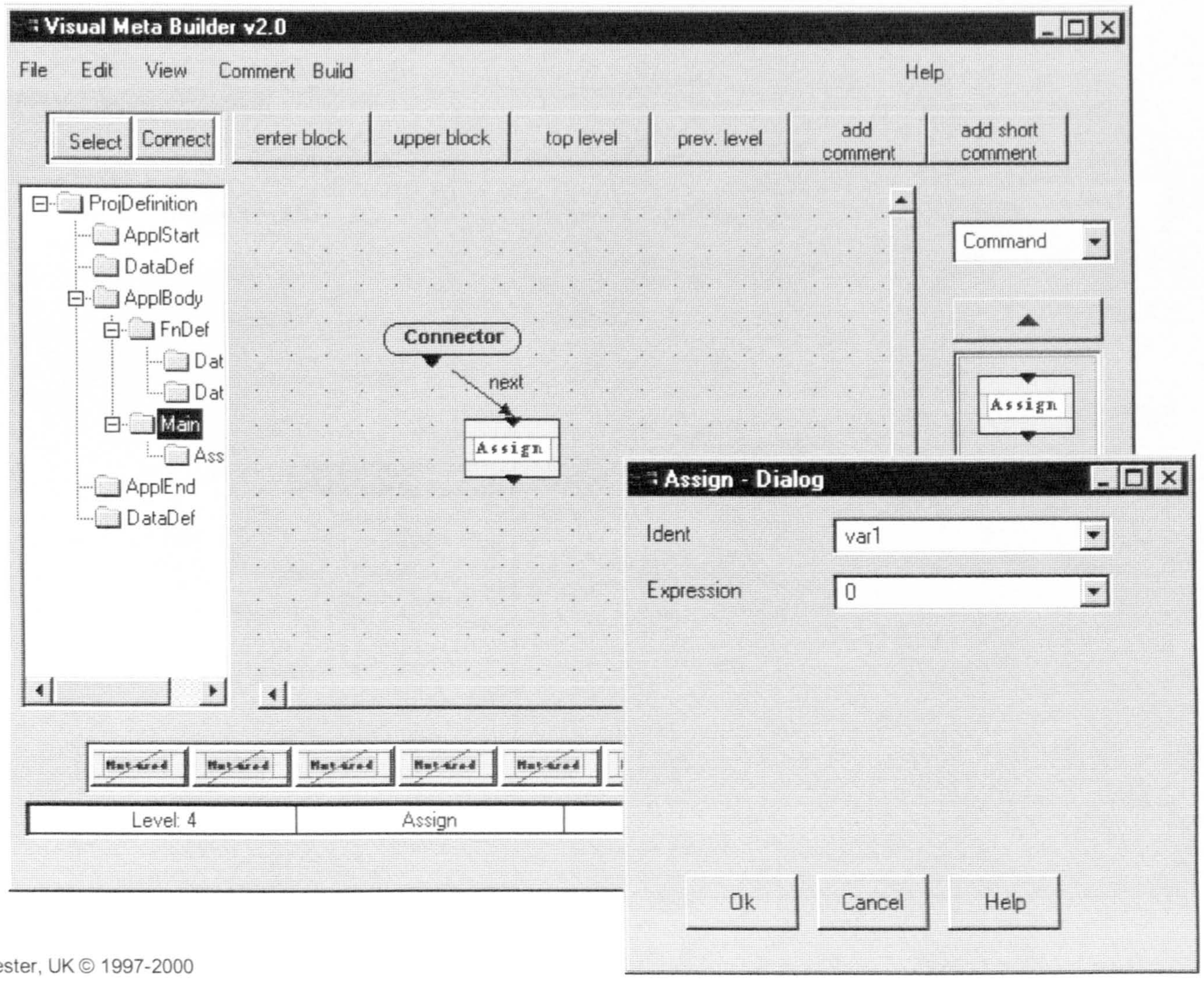
Visual Programming with VMB-C

Local Data Definition



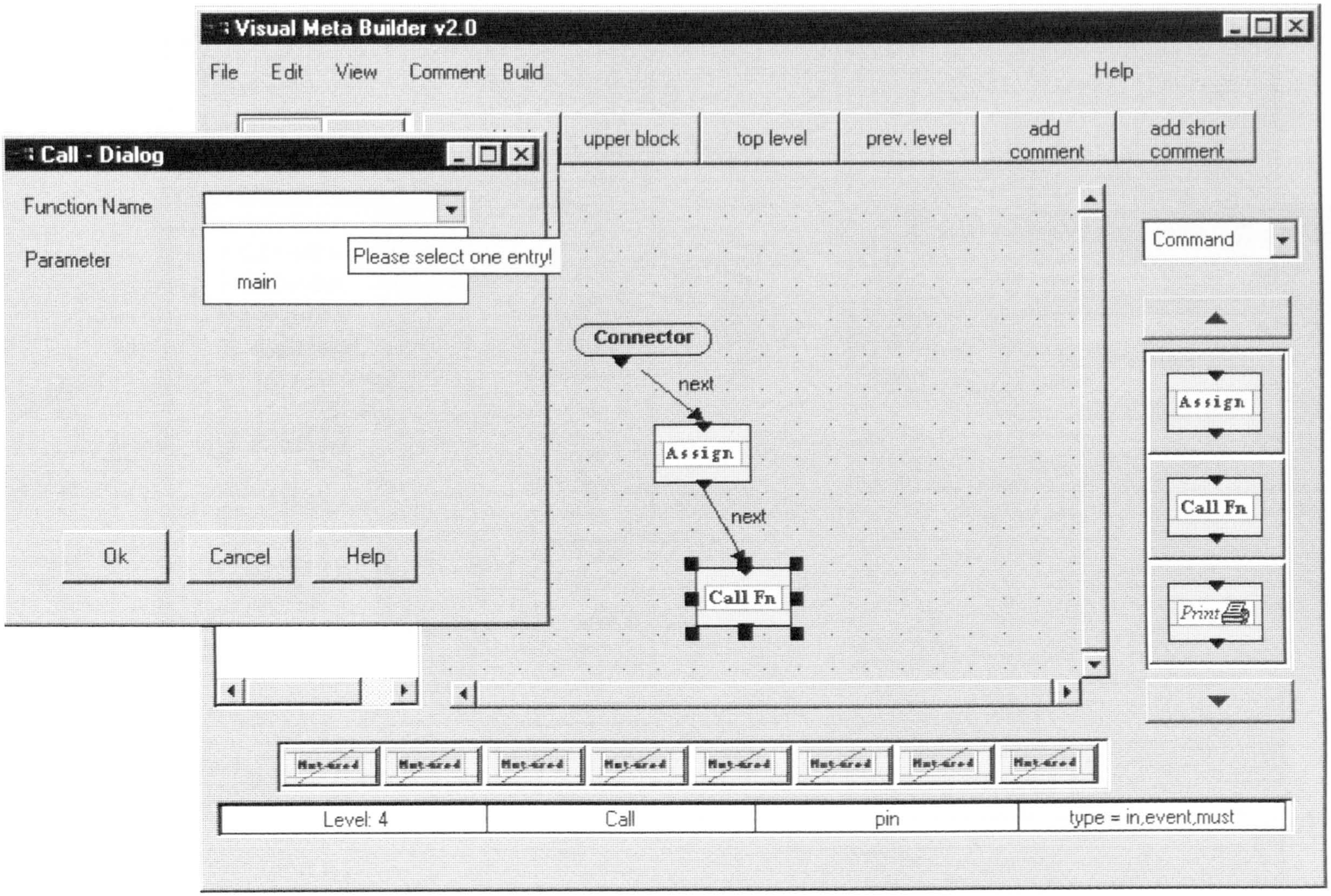
Visual Programming with VMB-C

Important Command Processors (system-defined primitives) : ASSIGN



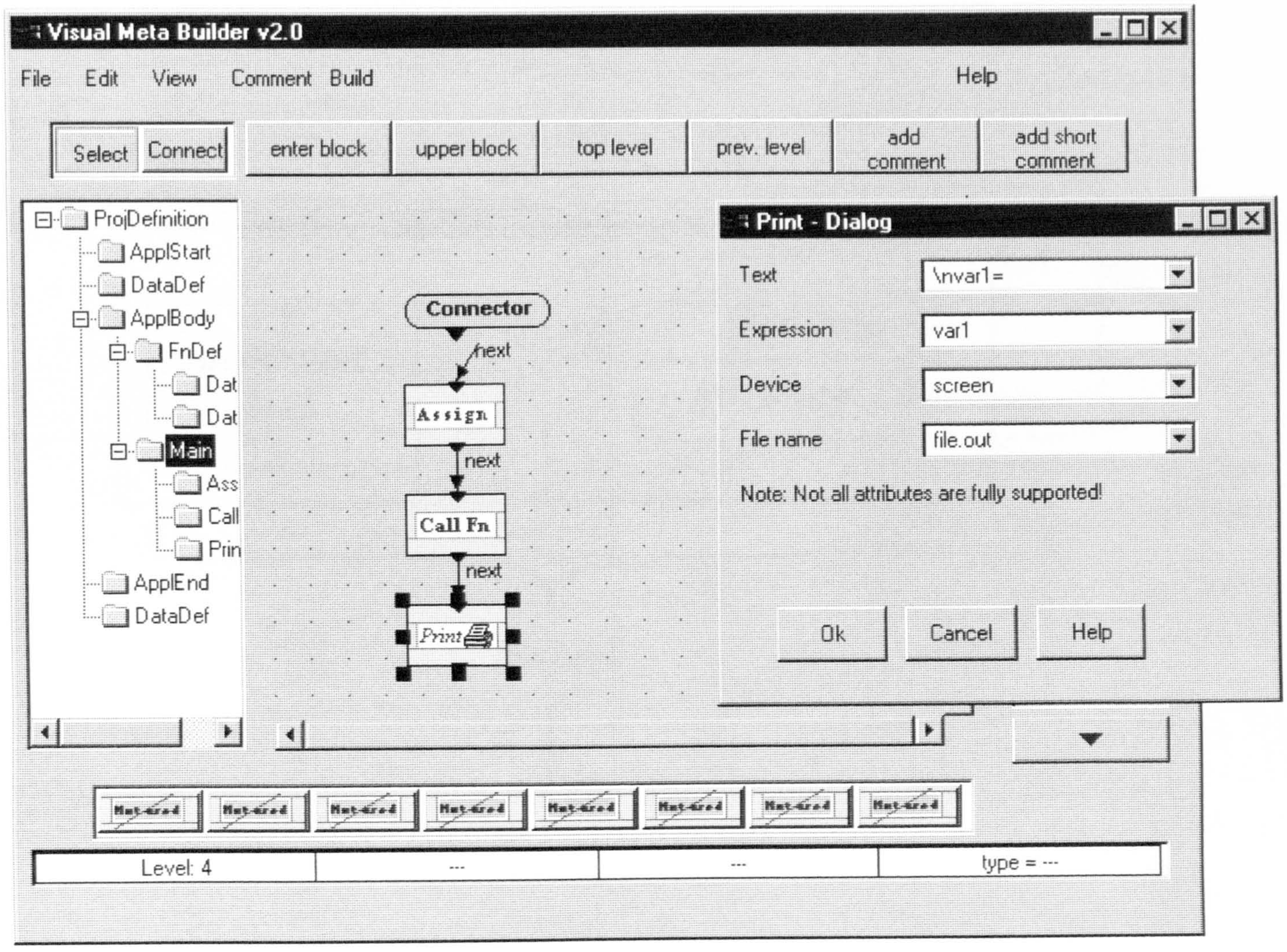
Visual Programming with VMB-C

Important Command Processors (system-defined primitives) : Function-CALL



Visual Programming with VMB-C

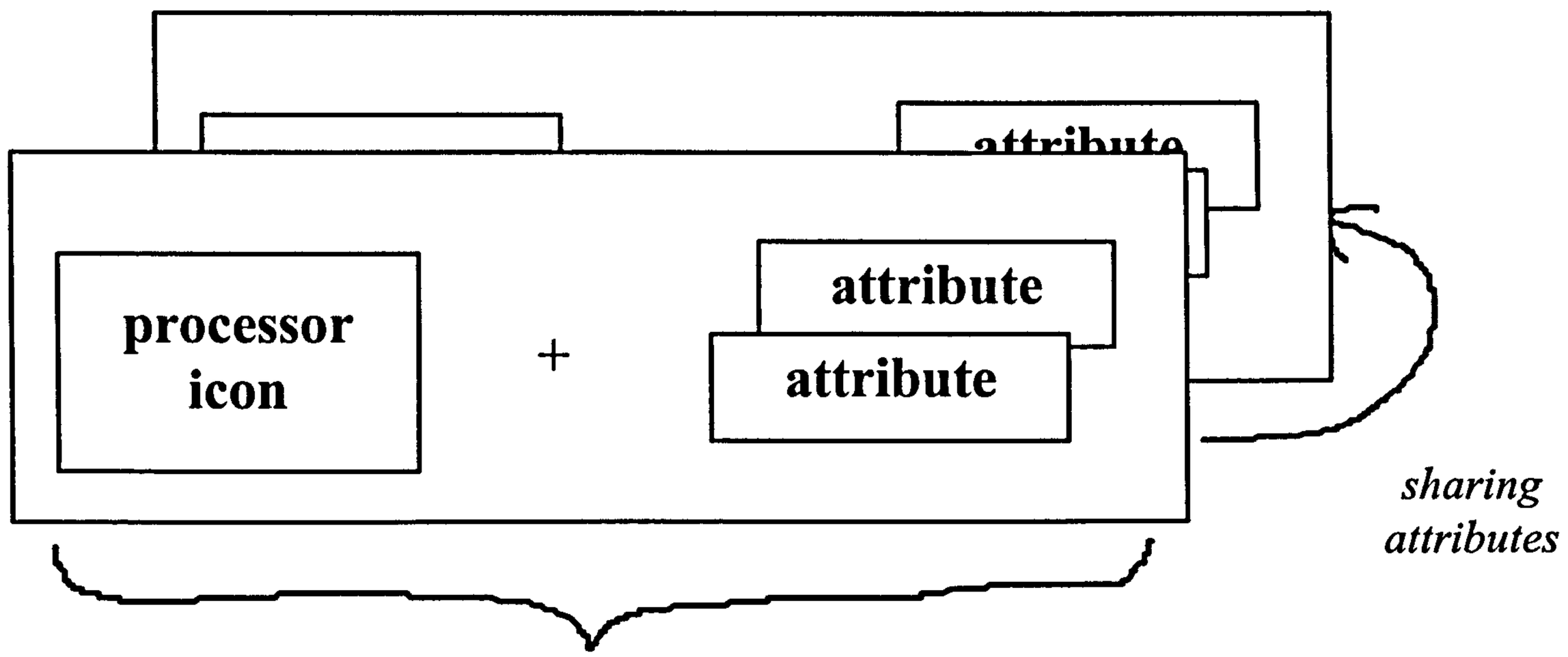
Important Command Processors (system-defined primitives) : PRINT



Visual Programming with VMB-C

A processor icon represents a function call

VPL-C



sharing attributes

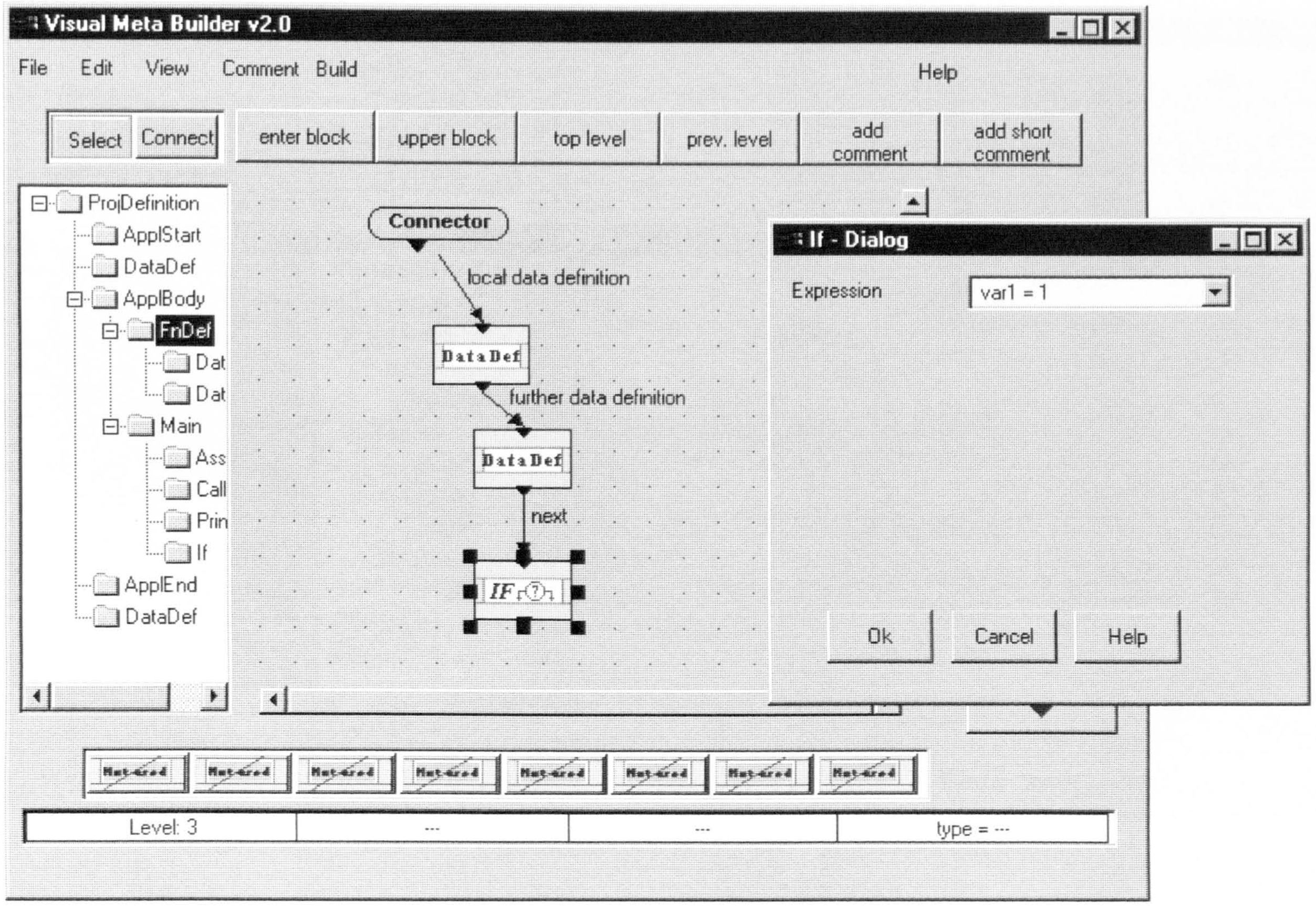
lexeme

↓

function call : fn(<parameter list>)

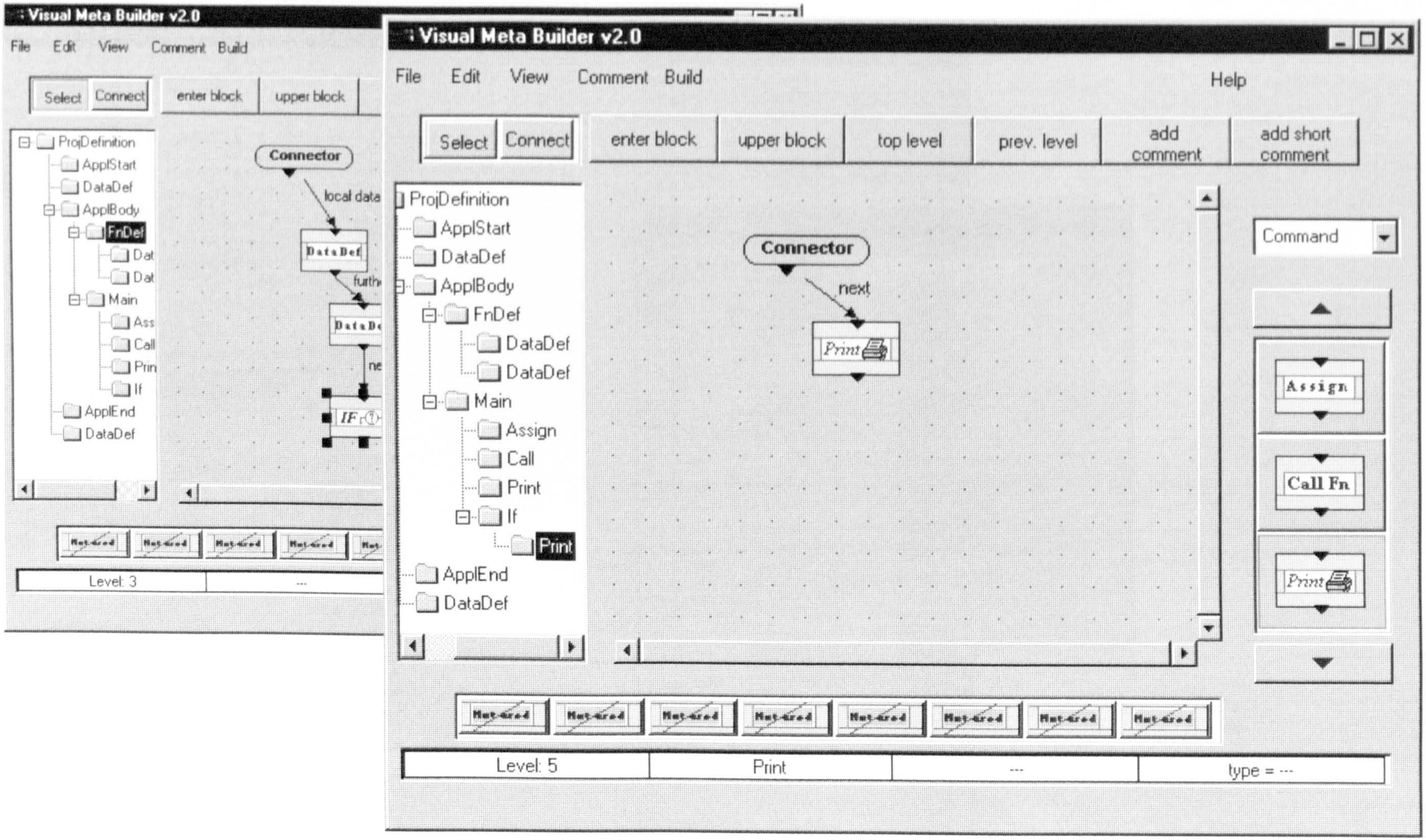
Visual Programming with VMB-C

Control Processors : IF



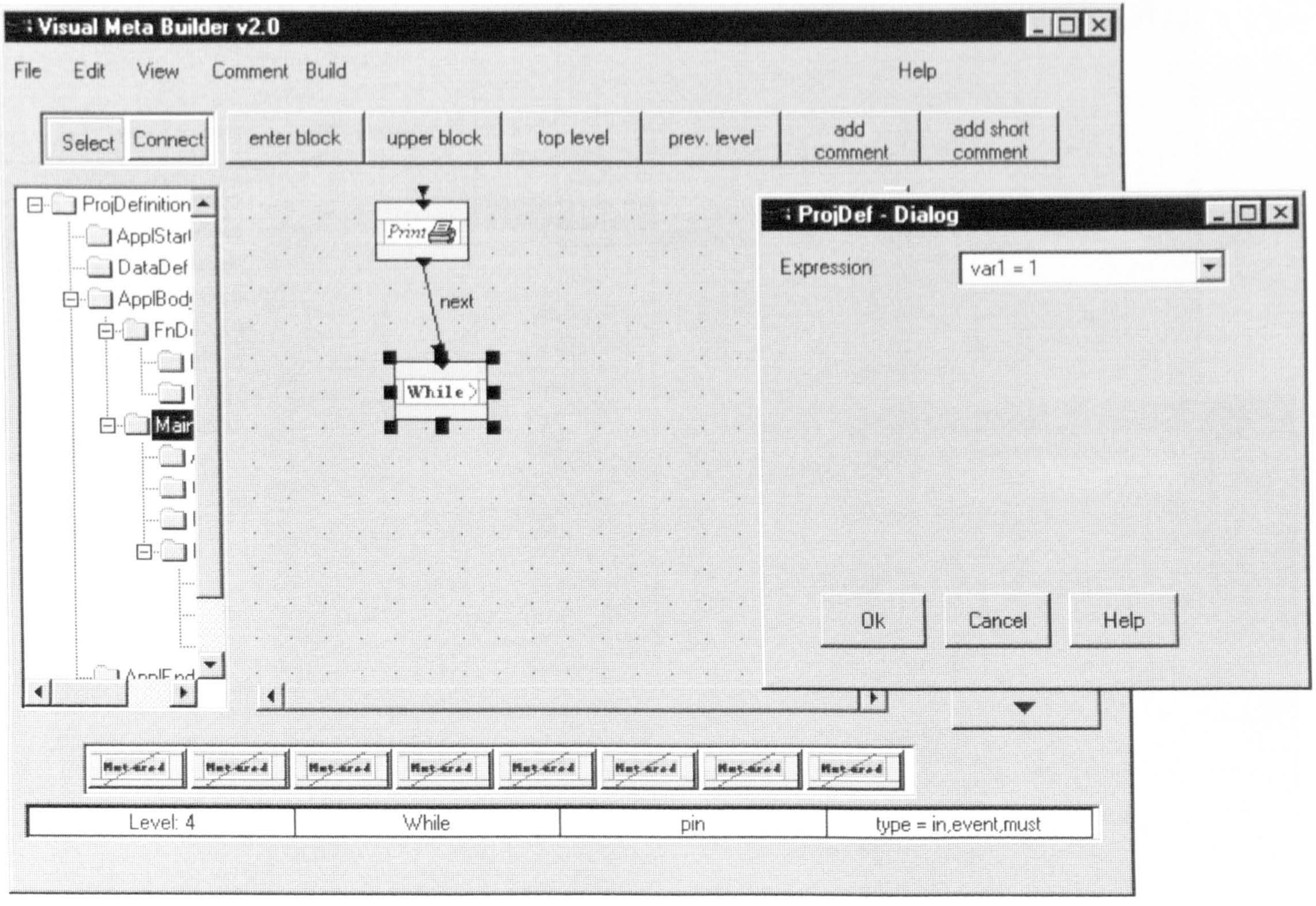
Visual Programming with VMB-C

Control Processors/Nested Commands : IF



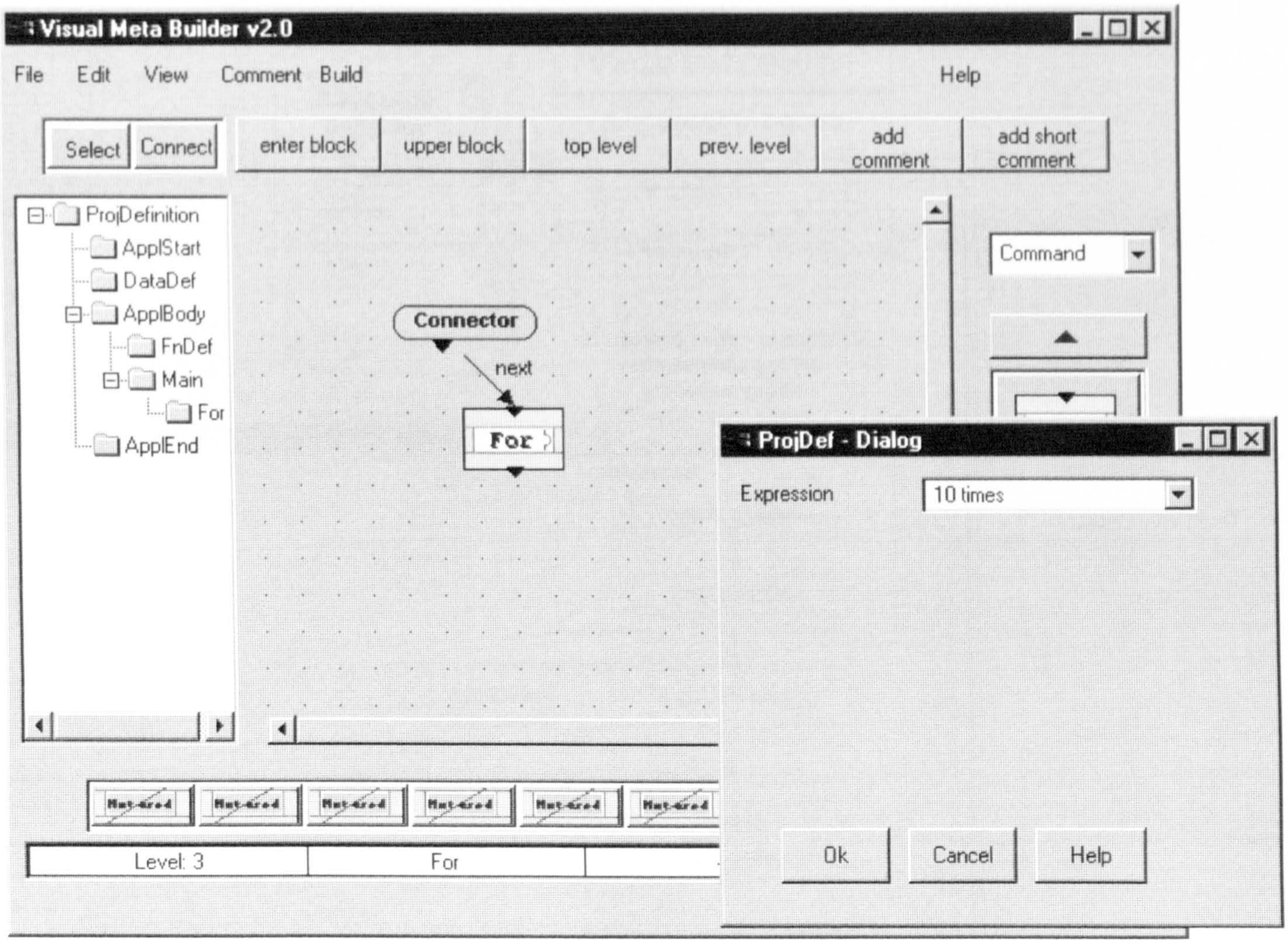
Visual Programming with VMB-C

Control Processors/Nested Commands : WHILE



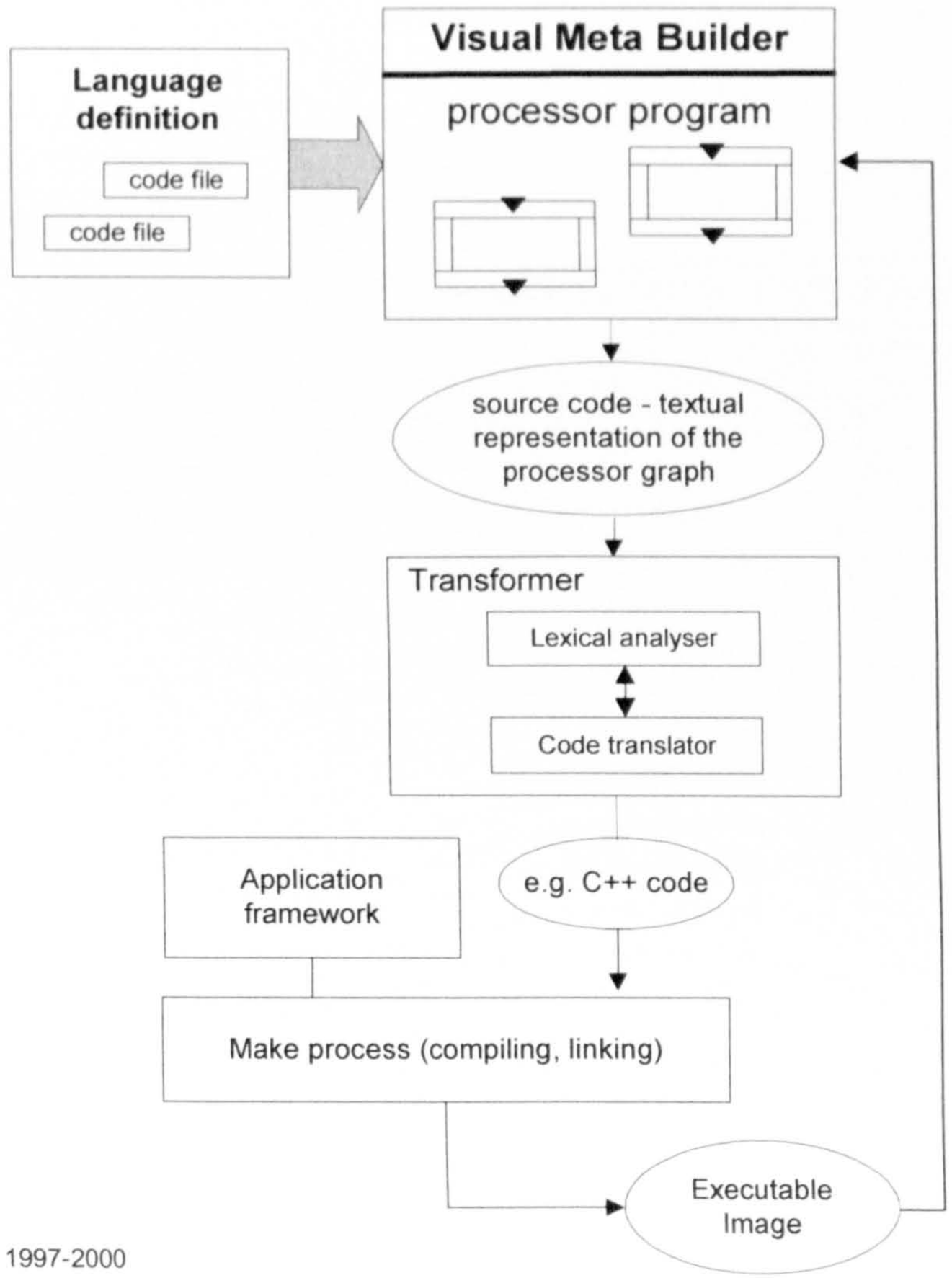
Visual Programming with VMB-C

Control Processors/Nested Commands : FOR



Visual Programming with VMB-C

General Architecture of the system



End of Training

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Task: Creation of a program for the calculation of "Fibonnaci numbers"

Background information

Rabbit populations are growing extremely fast and follow the rules below:

"At time $\mathbf{t_1}$ there exist one rabbit pair. This rabbit pair need 1 month ($\Delta\mathbf{t}$) in order to be able to create another pair. From time $\mathbf{t_2}$ onwards an existing rabbit pair creates one further pair each month. Thus, at time $\mathbf{t_3}$ there are two pairs, at time $\mathbf{t_4}$ there are 3 pairs, at time $\mathbf{t_5}$ there are 5 pairs and so forth. The number of possible pairs are called Fibonacci numbers."

The function F for calculating the number of pairs of rabbits at time $\mathbf{t_n}$ may be described by the following equation :

$$\mathbf{F(t_n) = F(t_{n-1}) + F(t_{n-2})} \quad \textbf{for all cases where } \mathbf{n > 2} \quad \textbf{with } F(t_1) = 1 \textbf{ and } F(t_2) = 1$$

The following table shows the number of rabbit pairs for the different times:

time t_n	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
$F(t_n)$ rabbit pairs	**1**	**1**	**2**	**3**	**5**	**8**	**13**	**21**

Here are the underlying calculations:

$$F(t_3) = F(t_2) + F(t_1) = 1 + 1 = 2$$
$$F(t_4) = F(t_3) + F(t_2) = 2 + 1 = 3$$
$$F(t_5) = F(t_4) + F(t_3) = 3 + 2 = 5$$
$$\textbf{etc.}$$

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Tasks Description

Your task is to write a program that may calculate the number of rabbit pairs
(i.e. the Fibonacci numbers) for times t_3, t_4 etc.

The following operations are required:

1) The user is asked to enter a time index (e.g. "Please enter number", user may enter 4).
2) The corresponding Fibonacci number is to be calculated.
3) The result should be presented to the user (e.g. "Result: 3")

Group A

You are asked to write the program using the Prograph notation.

Group B

You are asked to write the program using the C or C++ or Java notation.

Group C

You are asked to write the program using the VMB-C notation.

Please use the paper on the next pages!

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Your solution "Fibonacci"

1 [redacted]

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Your solution "Fibonacci" (continued)

2 [redacted]

[redacted]

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Your solution "Fibonacci" (continued)

3 [redacted]

[redacted]

Questionnaire : "Writability" of programs

Q: What **problems and experiences** did you encounter while trying to write the code? Are you confident that you did solve the problem?

A:

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Tasks Description

Your task is to write a program that may calculate the numbers of rabbit pairs (i.e. the Fibonacci numbers) for times t_3, t_4 etc.

The following operations are required:

1) The user is asked to enter a time index (e.g. "Please enter number", user may enter 4).
2) The corresponding Fibonacci number is to be calculated.
3) The result should be presented to the user (e.g. "Result: 3")

Group A

You are asked to write the program using the C or C++ or Java notation.

Group B

You are asked to write the program using the Prograph notation.

Group C

You are asked to write the program using the Prograph notation.

Please use the paper on the next pages!

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Your solution "Fibonacci_2"

```
1 [redacted]
```

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Your solution "Fibonacci_2" (continued)

```
2 [redacted]
```

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Your solution "Fibonacci_2" (continued)

3 [redacted]

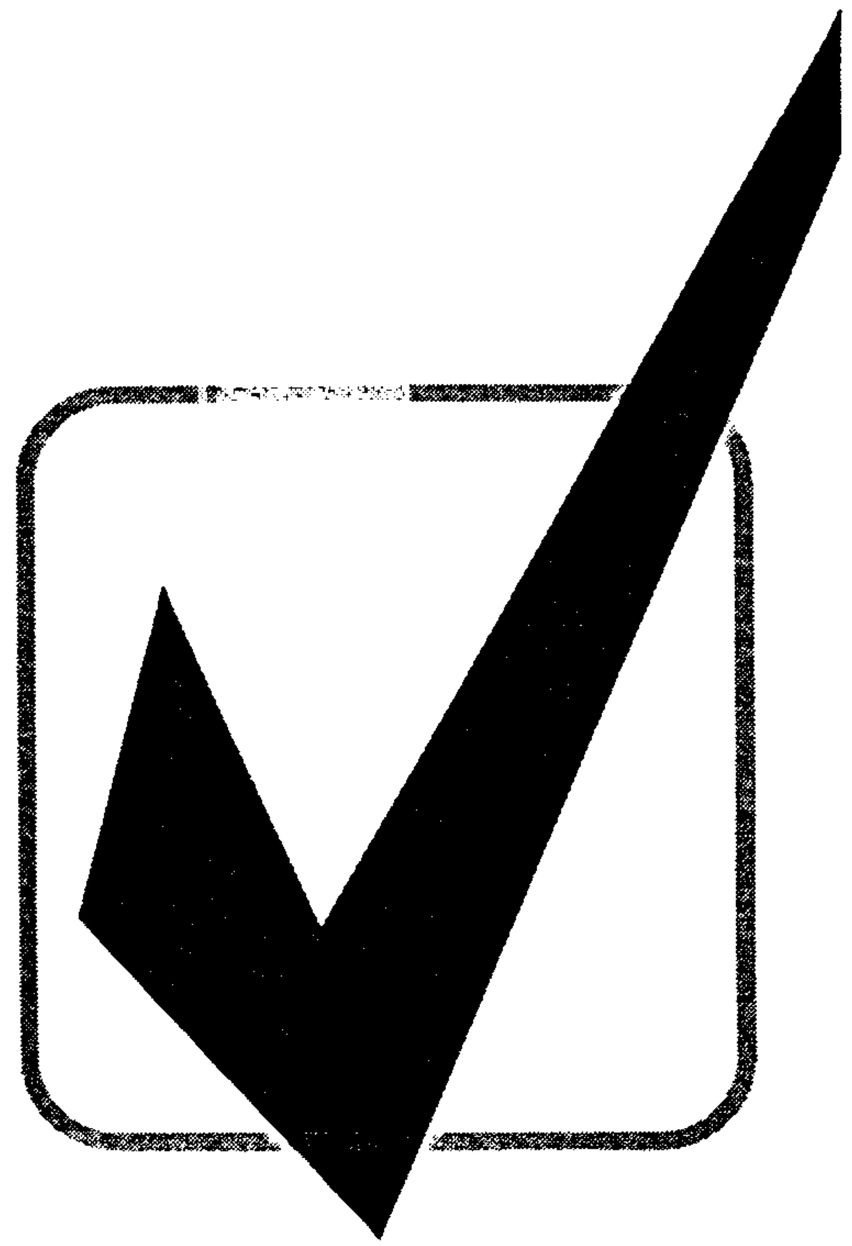
[redacted]

Questionnaire : "Writability" of programs

Q: What **problems and experiences** did you encounter while trying to write the code? Are you confident that you did solve the problem?

A:

End of Evaluation 1



Thank you for your help!

User Evaluations

Eval1: **Fundamental programming**

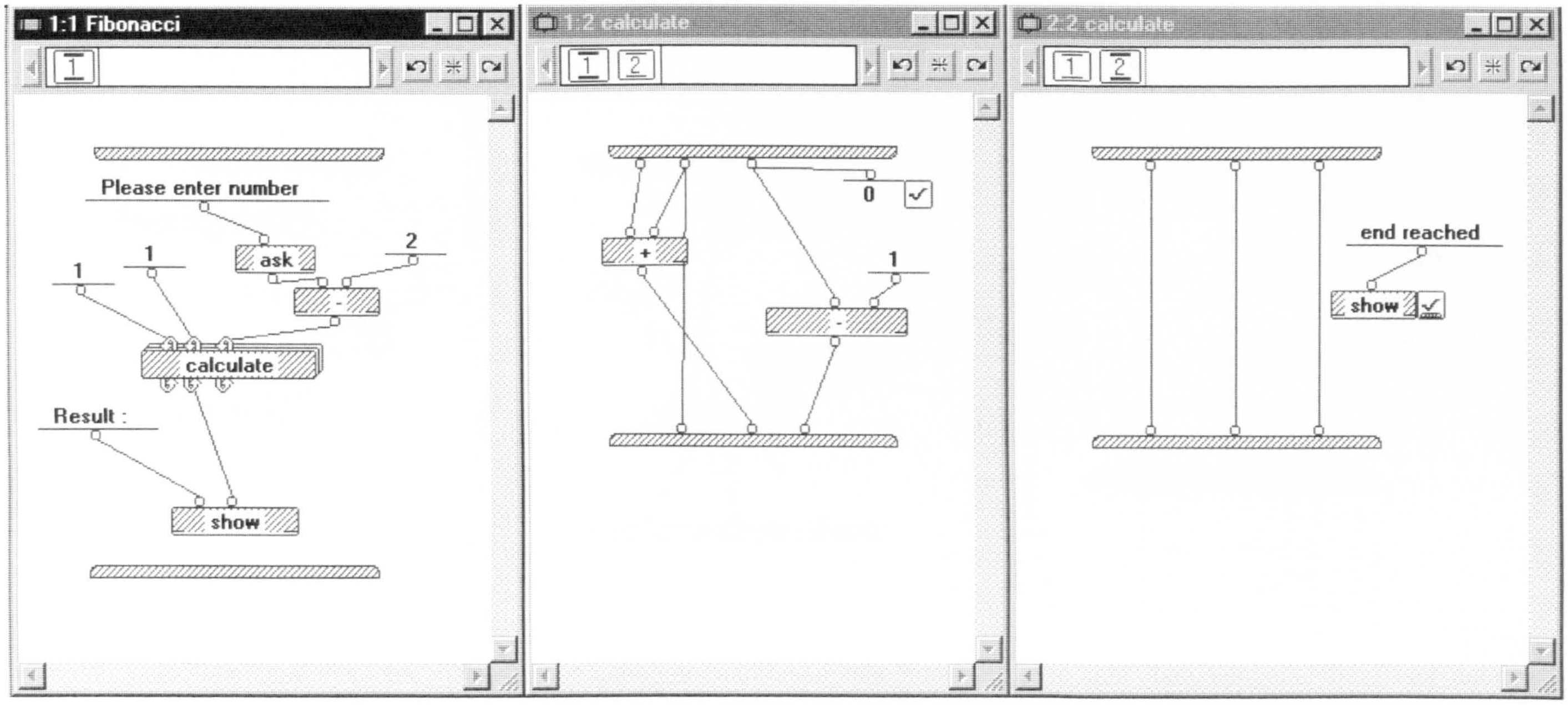
Solutions

Prograph

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Prograph: Iterative Solution "Fibonacci"

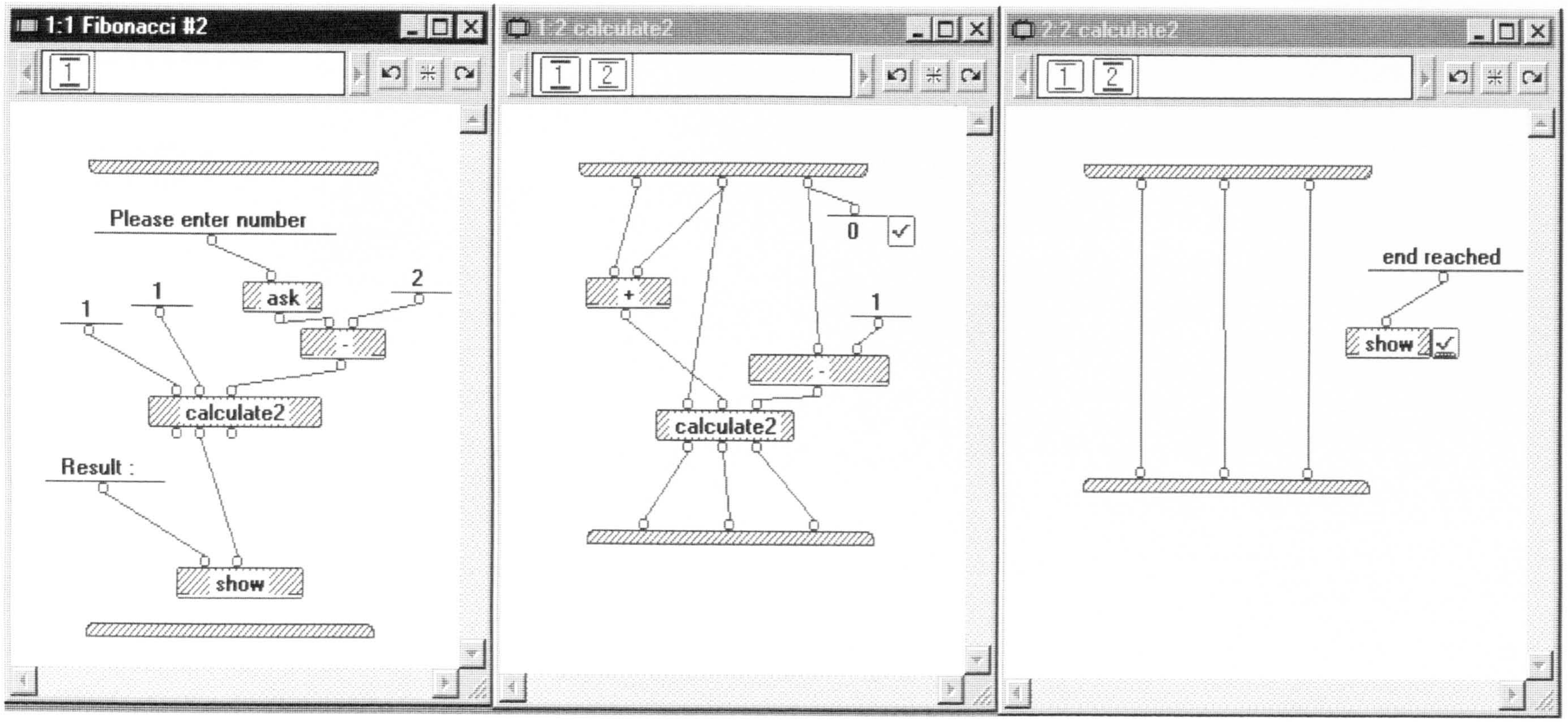


$$F(1) = 1;\ F(2) = 1;\ F(3) = 2;\ F(4) = 3;\ F(5) = 5;\ F(6) = 8;\ F(7) = 13$$

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

Prograph: Recursive Solution "Fibonacci"



F(1) = 1; F(2) = 1; F(3) = 2; F(4) = 3; F(5) = 5; F(6) = 8; F(7) = 13

User Evaluations

Eval1: **Fundamental programming**

Solutions

C/C++

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

C/C++: Iterative Solution "Fibonacci"

```
#include <stdio.h>


main()
{
int N, F1, F2, F3, IE, IO;

  printf("\n Number =");
  N = scanf("%d");

  F1 = 1;
  F2 = 1;

  for (int I=3; I <N; ++I)
  {
      F3 = F1 + F2;
      F1 = F2;
      F2 = F3;
  } /* for */

  printf("%d", F3);
  return 0;
}
```

User Evaluations on Fundamental Programming

Eval1: "Fibonacci numbers"

C/C++: Recursive Solution "Fibonacci"

```
#include <stdio.h>

calculate(int N, int I, int F1, int F2)
{

int F3;

      F3 = F1 + F2;
      F1 = F2;
      F2 = F3;
      I += 1;
      if (I <= N)
        F3 = calculate(N, I, F1, F2);
      return F3;
}
```

```
main()
{
int N, I, F1, F2, F3;

printf("\n Number =");
N = scanf("%d");
F1 = F2 = 1;
I = 3;
printf("%d", calculate(N, I, F1, F2));
return 0;
}
```

Evaluation 2

Duration : 2 hrs.

Planned date : 24/10/2000

Evaluator : Frank Buehler

CORBA programming:

CORBA (Common Object Request Broker Architecture) is a well-established platform- and programming-language-independent distributed object computing environment. It is of special interest how a visual tool like VOODE/VOOPL-1 for CORBA may help construct distributed object-oriented programs

(instead of just using text-based systems).

User Evaluations on Fundamental Programming

Eval2: "CORBA Programming"

Hypothesis

Creating a CORBA object may be more efficient in a complete visual environment (design A) than using a (visual) environment that uses a text-based language such as Java or C/C++ (design B).

Different paradadigms

Commercial environments do not include a VPL.

Objective of the evaluation

Comparison of the efficiency of different approaches (design A against design B) for the **extension of an existing sample (w.r.t. CORBA server** and **an interface method)**. The aim is to detect the number and severity of usability problems.

Trials/Tasks

The following tasks should be carried out by the subject (for both approaches/designs). For the sake of simplicity it is not necessary that the code is compiled and tested. The user should give a statement when he/she thinks that the task is completed. After each evaluation step a questionnaire is given to the subject.

1st task : change of existing method (1a: printf/include statement, 1b: if statement)
2nd task : definition of new method (1a: definition of new method "print", 1b: behaviour of "print")
3rd task : definition of a CORBA server mainline

Evaluation 2 (2 hrs.)

ca. 3-5 CS/SE students,
1 computer projector for training, 1 PC for trials

step

1) Questionnaire (2 min.)

2) Training (60 min.)

break (15 min.) -> building 2 groups A + B + scheduling

3) Trials (Task Analysis) : 2x30 min. = 60 min. for each student
Group A
design A : VOODE/VOOPL-1 for CORBA
design B : Orbix 2.3c + (Explorer + Textpad) or (MS Developer Studio)
Group B
design B : Orbix 2.3c + (Explorer + Textpad) or (MS Developer Studio)
design A : VOODE/VOOPL-1 for CORBA
for both designs:
3a) change of existing method (10 min.)
3b) definition of new method (10 min.)
3c) definition of a CORBA server mainline (10 min.)

4) Feedback Questionnaire

User Evaluations on Fundamental Programming

Eval2: "CORBA Programming"

Expected results

The training will provide basic information for both designs.

It is expected that the subjects will better remember the visual approach and perform better in task 2 and 3.

User Evaluations

Eval1: **User evaluation on CORBA programming**

Expected/Possible Problems for text-based approach

Task 1

1a : selection of correct source file grid_i.cpp
syntax problem with printf statement
missing include statement

1b : selection of correct source file grid_i.cpp
selection of grid constructor (not set method!)
syntax problem with if statement
no throwstatement

Task 2

2a : selection of interface source file grid.idl
add print interface method
execute generate.bat
add new method in source file grid_i.h **(user doesn't remember that there is a Wizard in MS-DevStudio)**
add missing code in source file grid_i.cpp

2b : edit method in source file grid_i.h
edit missing code in source file grid_i.cpp

Task 3

selection of source file Srv_Main.cpp
add code: wrong syntax/semantic/unknown function calls

User Evaluations

Eval1: **User evaluation on CORBA programming**

Expected/Possible Problems for visual approach

Task 1

1a: selection of context "CORBA_Module_Methods\Set"
drag&drop of print processor
configuration of print processor

1b: selection of context "CORBA_Module_Methods\OnConstruction"
drag&drop of if processor
configuration of if processor

Task 2

2a: selection of Rational Rose and grid interface clas
addition of print method
start of VOODE/VOOPL-1 editor
drag&drop of print processor
configuration of print processor

2b: edit context "CORBA_Module_Methods\Print"
drag&drop of for processor
configuration of for processor

Task 3

selection of server mainline
add processor code: ComponentConstruction and ServerInit processors

Questionnaire

Name of student: _____

Please note: all responses will be confidential!

This evaluation is part of a research project on Visual Languages, aiming to help design future generations of visual tools. Could you please spend a few minutes to answer the following questions. After a training you are asked to carry out several tasks which concentrate on specific CORBA programming issues.

Q1 : Which of the following languages do you know?

C++ : yes O no O Java : yes O no O
UML : yes O no O (other) ____ : yes O no O

Q2 : How much experience of programming do you have?

Language C++_____ High O Medium O Low O None O
Language Java_____ High O Medium O Low O None O
Language UML_____ High O Medium O Low O None O
Language _____ High O Medium O Low O None O

Q3 : Do you have any knowledge of a graphical language (i.e. a programming language that makes use of icons or diagrammatic elements rather than textual code elements)?

yes O no O
if yes, could you please state the kind of experience you have.

_____

Q4 : Do you have any knowledge of CORBA?

yes O no O
if yes, could you please state the kind of experience you have.

_____

Basic Training (60 min.)

- **OO/CBD, UML/Rose**
- **C++ concepts/syntax (MS Developer Studio)**
- **CORBA principles, “Grid” sample**
- **Visual Programming (VOODE/VOOPL-1)**

Introduction to OO

- **Object-oriented programming** (OOP) is the act of modelling systems in terms of objects. The basic concepts of object-oriented programming are data abstraction, instantiation, composition, and specialisation [BOOCH-1994].

- As Rumbaugh and others claim, **"object-oriented modeling and design** promote better understanding of requirements, cleaner designs, and more maintainable systems" [RUMBAUGH-1991].

Introduction to OO

- Data abstraction : programmer-defined data type that can be manipulated in a manner similar to predefined data types. It corresponds to a set of legal data values and a number of functions that can be performed on these values.

- Classes/Objects/Instantiation : A class corresponds directly to an abstract data type (ADT). An object is an instance of a class and has specific values.

- Composition : An application is composed of objects.

- Generalisation/Specialisation : These OO mechanisms help organise (or structure) classes and objects to share the same code. It is implemented in a concrete programming language (such as C++) as a inheritance relationship.

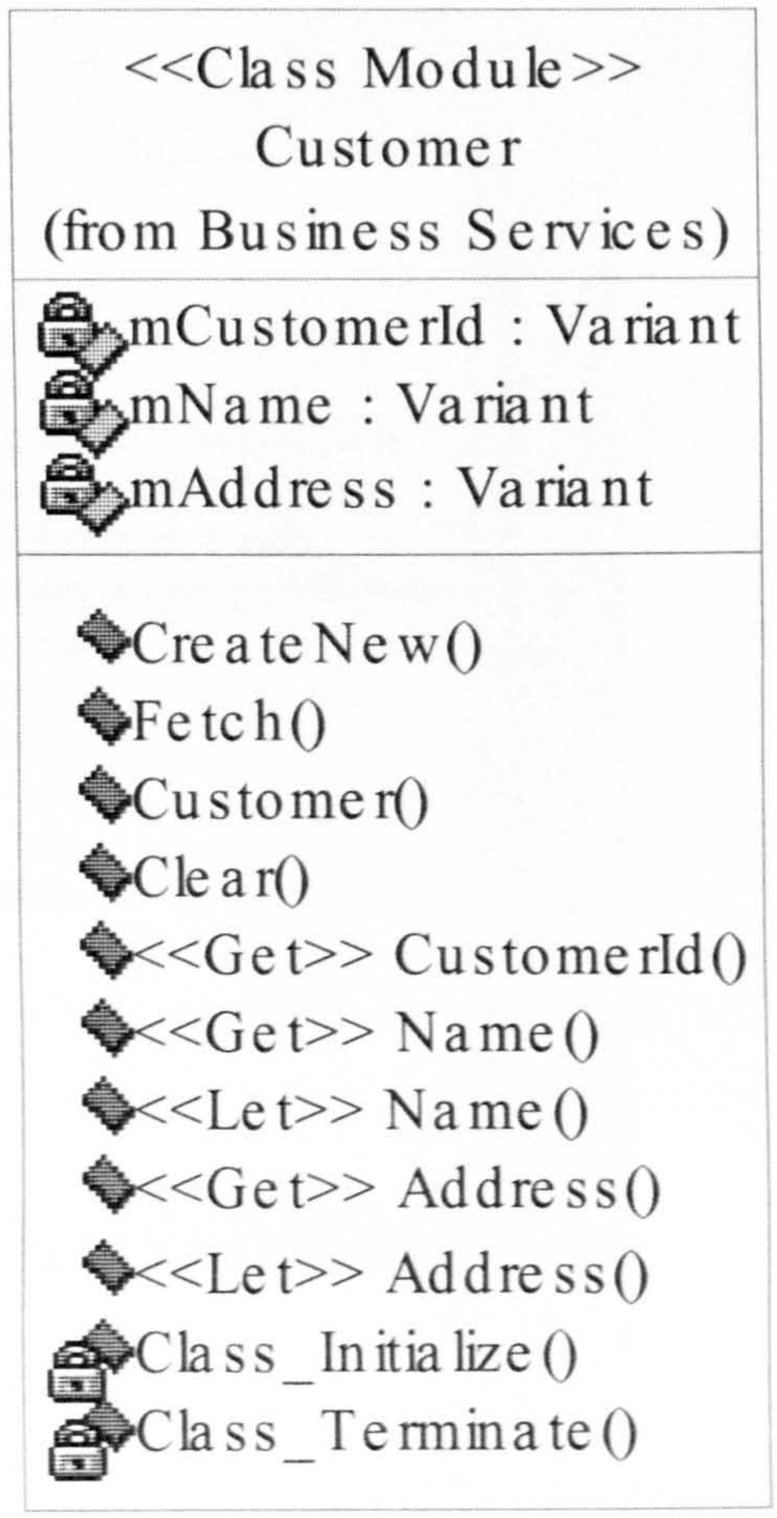
UML (Unified Modeling Language) is the key notation used within nowadays software development projects. The UML is a result on the effort of a common concept developed by the world's most prominent methodologists.

(N.B.: The OMG Specifications for analysis and design include the UML, the repository standard Meta-Object Facility (MOF), and XMI.)

Introduction to OO

What is a class?

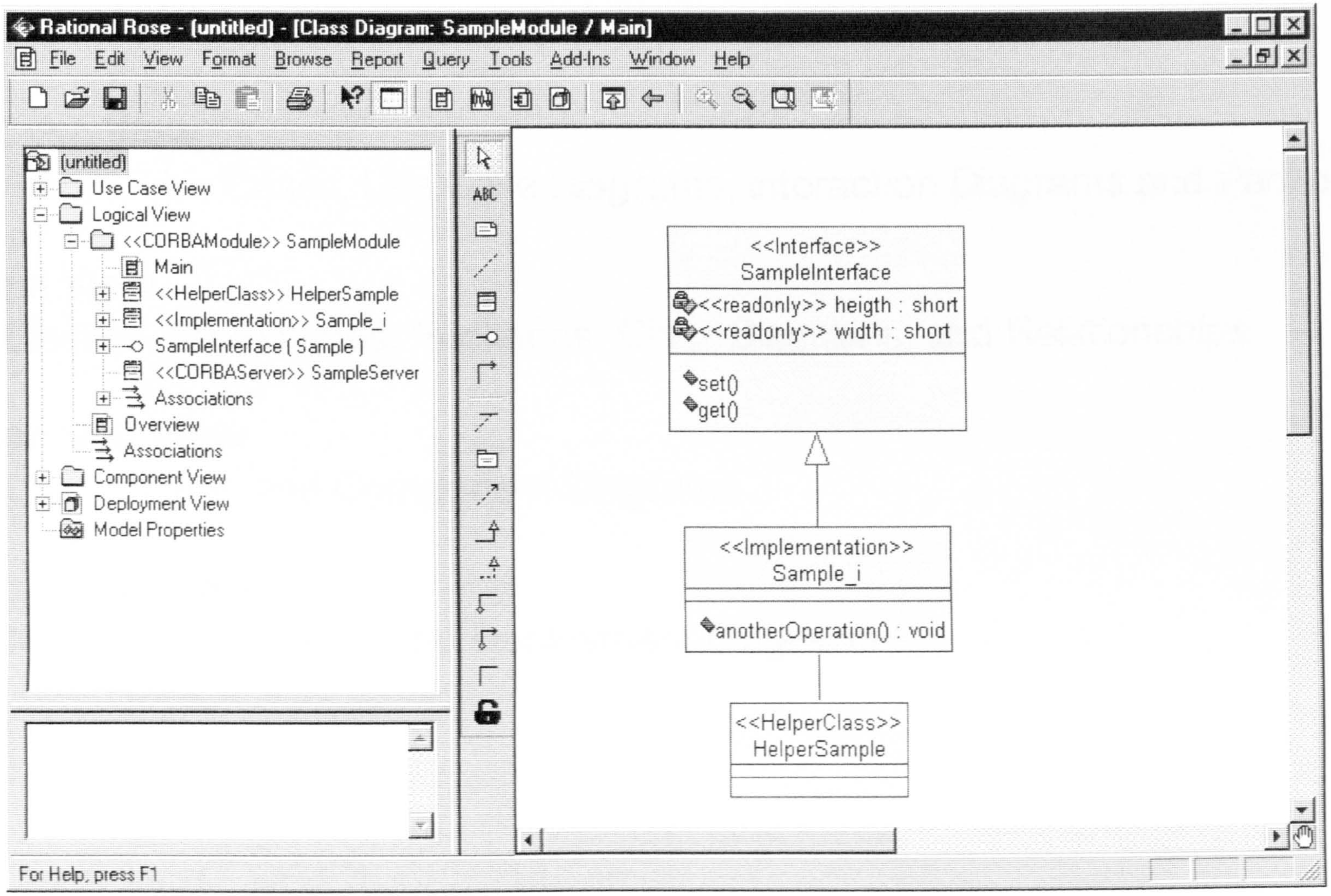
class



object

Introduction to OO

Inheritance



Introduction to OO

Major Modelling Elements in UML/Rose

Use Case View

Actors, Use Cases, Use Case Diagrams, Interaction Diagrams and Packages

Logical View

Classes, Stereotypes, Packages, Class Diagrams, and Relationships

Component View

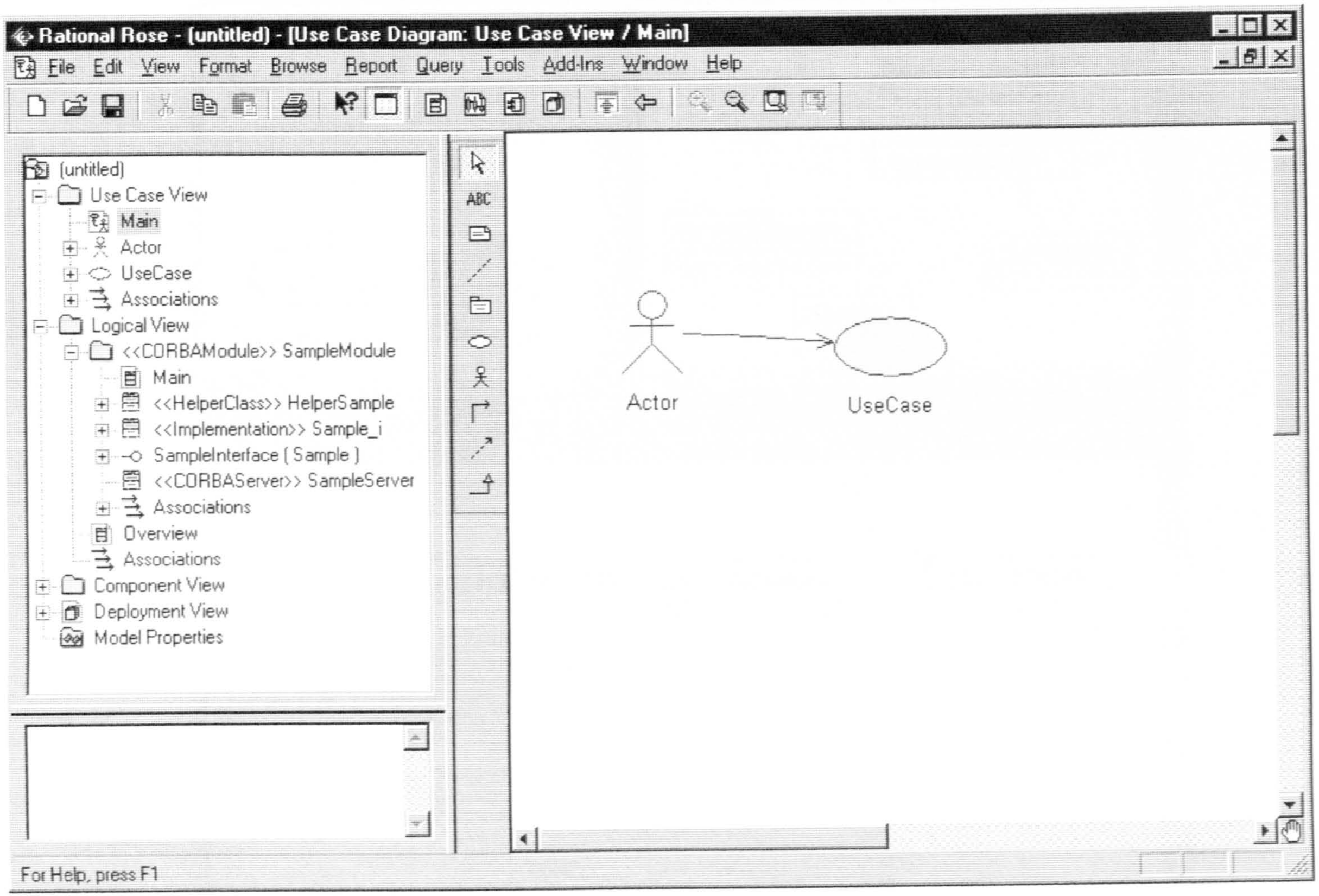
Components and Component Diagrams

Deployment View

Nodes, Connections and Deployment Diagrams

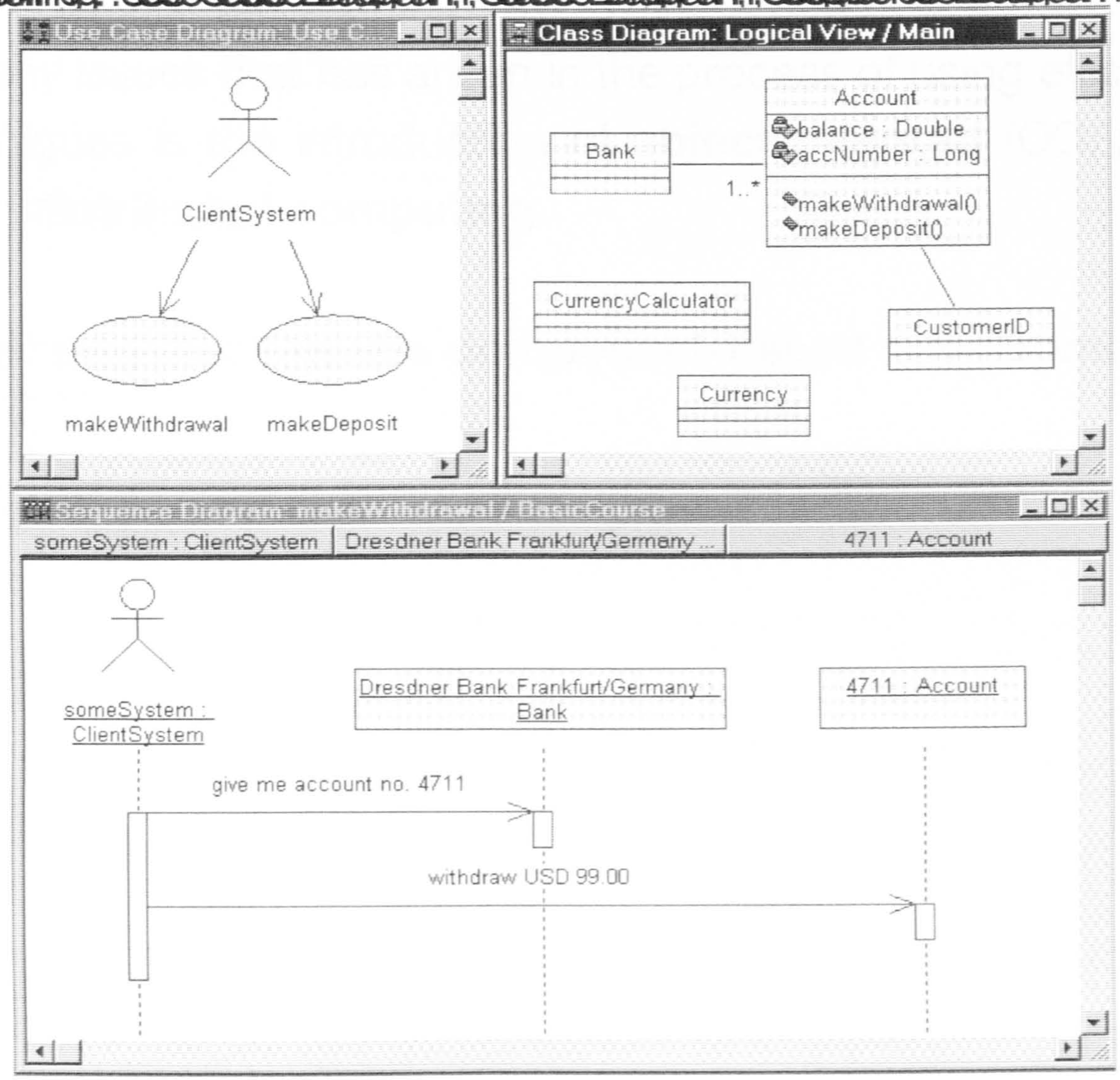
Introduction to OO

UML/Rose modelling : Use Case Diagram



Introduction to OO

UML/Rose modelling : Use Case Diagram, Class Diagram, Sequence Diagram



Introduction to CBD

• One of the many issues that has arisen in the process of using efficient and state-of-the art techniques is the introduction of **object-oriented** (OO) **programming methods** within **distributed computing**.

• This is currently extended towards **component-based development**.

• The leading server-site component technology is **CORBA** (and in the near future EJB).

Introduction to CBD

What is a (software) component?

A **component** is

- ... NOT a complete application!
  Components must be combined with others to form complete applications.
  A component may contain from 5 to 200 business-meaningful classes.
  A component is built for composition and collaboration with other components.
- .. a self-contained (marketable) entity that has a defined use.
- .. accessed via an (network-addressable) interface.
- .. interoperable : technology is hidden

Thinking in component terms is the best known way to master the **complexities** of large-scale distributed system development.

Introduction to CBD

Component Categories

Small Granularity

- User Interface Components : GUI controls (e.g. JavaBeans, ActiveX controls)

Medium Granularity

- Business Components : "Handlers", "Managers", "Engines" (e.g. CORBA, EJB)
- Infrastructure Components : Print Service, Security Service
- Enterprise (Distributed) Components : Address system

(Very) large Granularity

- Large Business Components : Information System, Invoice Management System (e.g. MQSeries)
- Legacy System Components : Wrapped software system with clear interfaces (e.g. CORBA wrapper)
- System-level Components : Invoice Management System (could consist of enterprise distributed components)

Medium and large granularity components (i.e. enterprise components) which match "business concepts" are the kind of components we are interested in.

Introduction to CBD

What is Component-Based Development?

Ability to **build new systems** by assembly of pre-existing components. CBD is however also beneficial if these components have to be built as part of the project.

Herzum, Sims [Herzum-1999, p11]:
"Component-based development is a **software development approach** where **all aspects and phases of the development lifecycle**, including requirements analysis, architecture, design, construction, testing, deployment, the supporting technical infrustructure, and also the project management, **are based on components**".

The **object-oriented approach** (i.e. the concepts of encapsulation, inheritance and polymorphism and a common standardised modelling language such as UML) can be seen as an important predecessor of component-based development.

Distributed System Development != **Component-Based Development.**
EJB helps reduce cost and development time for distributed systems but doesn't automatically include that the system build is constructed in an component-oriented way.

Introduction to C++

- C++ was invented by Bjarne Stroustrup (and was derived from C)
- C++ is an OOPL
- class definition : class x {
  };
- Access rights : private, public, protected
- Datatypes : char, int, double, ...
- Control statements : for, while, switch, if, ...
- Type definitions : Typedef,
- Enumerations : enum
- Pointers : char*, int*, double*, ...

- Access to many library functions
- #include
- Main function : main()

Introduction to C++

```
#include "helper.h"

class grid: public gridS {
        int m_height;    // store the height
        int m_width;     // store the width
        long **m_a;      // a 2-D array to store the grid data itself
public:
        // ctor
        grid(int, int);

        // dtor
        virtual ~grid();

        // Class method
        virtual int width();

private:
        // Class method
        virtual int calculate(int d);
};
```

Introduction to C++

```
#include <stdio.h>

// ctor
grid::grid(int h, int w){
   m_height=h;  // set up height
   m_width=w;   // set up width

   // now allocate the 2-D array: as an array of pointers to 1-D arrays.
   m_a = new long* [h];
   for (int i = 0; i < h; i++ )
         m_a[i] = new long[w];
}

// dtor
grid::~grid(){
   // free the individual 1-D arrays:
   for (int i = 0; i < m_height; i++)
         delete[] m_a[i];
   // then free the overall array:
   delete[] m_a;
}

...
```

CORBA - Introduction

What is CORBA?

CORBA (Common Object Request Broker Architecture) is a well-established platform- and programming-language-independent distributed object computing environment.
It is based on OMG/ISO Interface Definition Language (OMG IDL) and the Internet Inter-ORB Protocol (IIOP).

CORBA - Introduction

Objectives of CORBA

CORBA - Integration platform

- Industry standard for distributed object-oriented applications
- Language and platform independent
- Promotes component architecture (e.g. CORBA components)
- Integration of legacy systems
- There are more than 70 ORB vendors plus hundreds of related products [Herzum-1999].

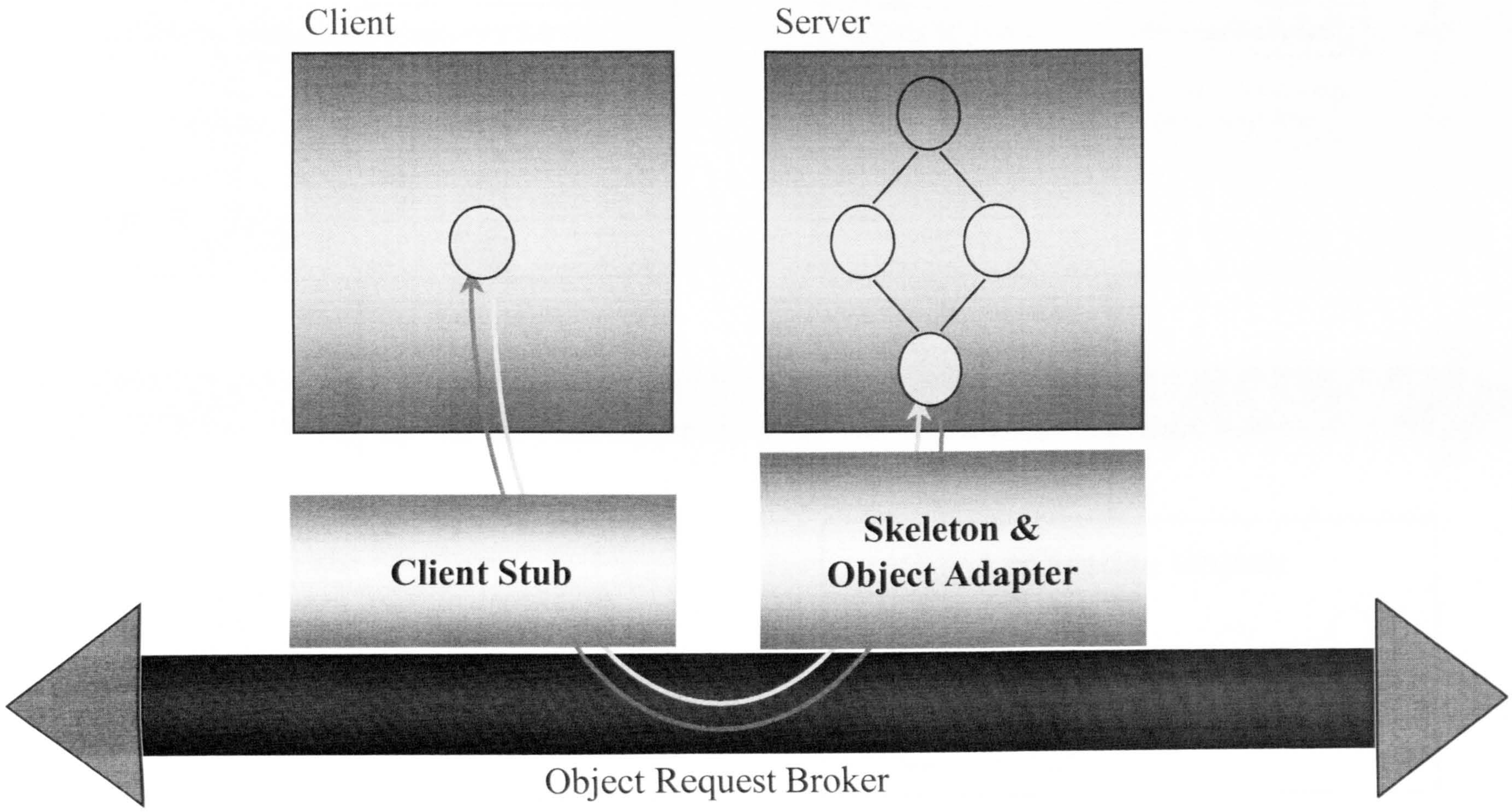
CORBA (Common Object Request Broker Architecture)

- 1989: Foundation by Object Management Group (OMG) with 8 companies (HP, Sun, Philips, Unisys, 3Com, American Airlines, Canon, Data General)
- 1993: CORBA 1.0
- 1996: CORBA 2.0
- 2000: CORBA 3.0

  (Component Model, quality-of-service control, messaging invocation model, integration with Internet, EJB and Java)

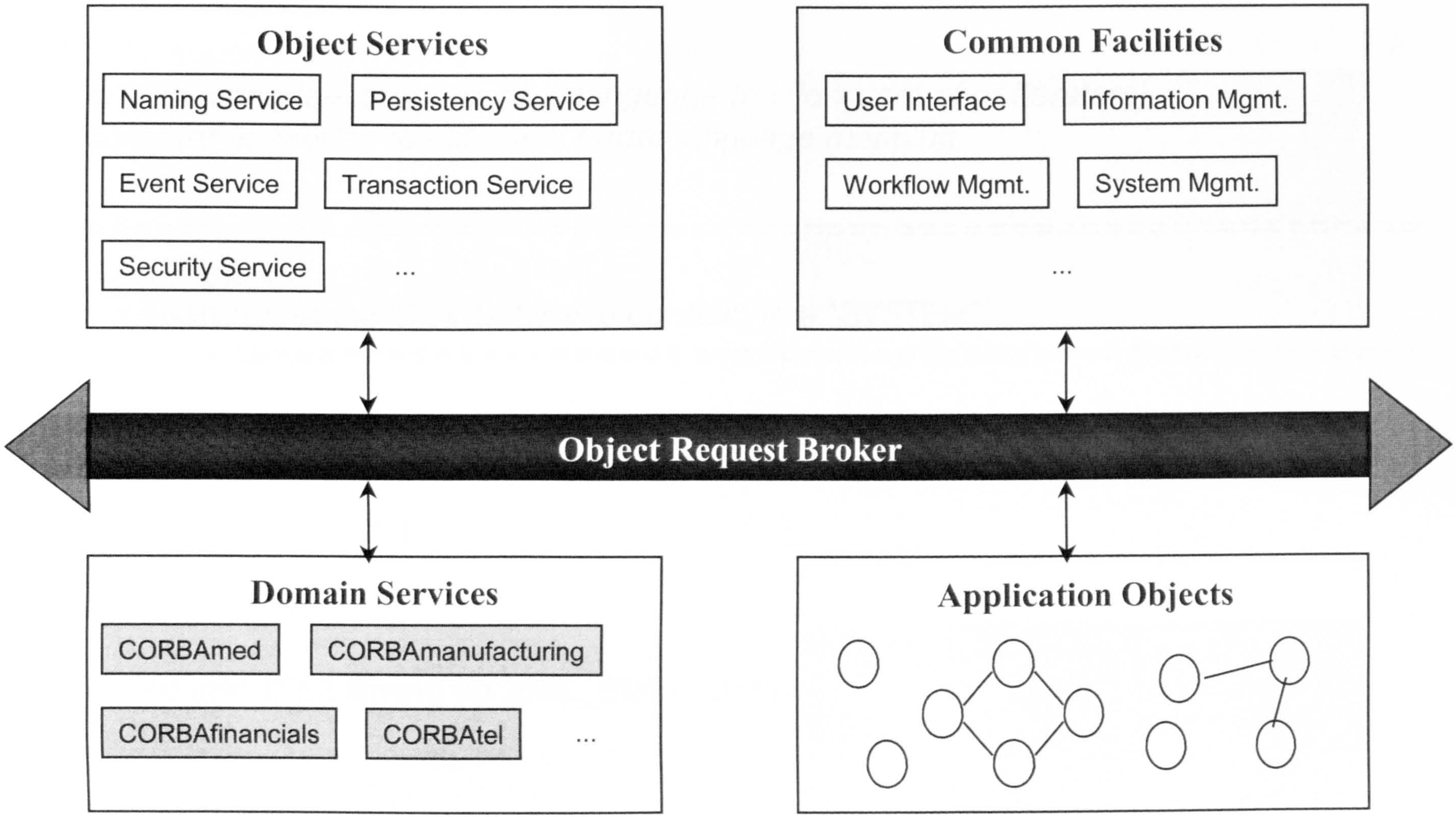
CORBA - Introduction

Client-Server communication



CORBA - Introduction

OMA (Object Management Architecture)



CORBA - Introduction

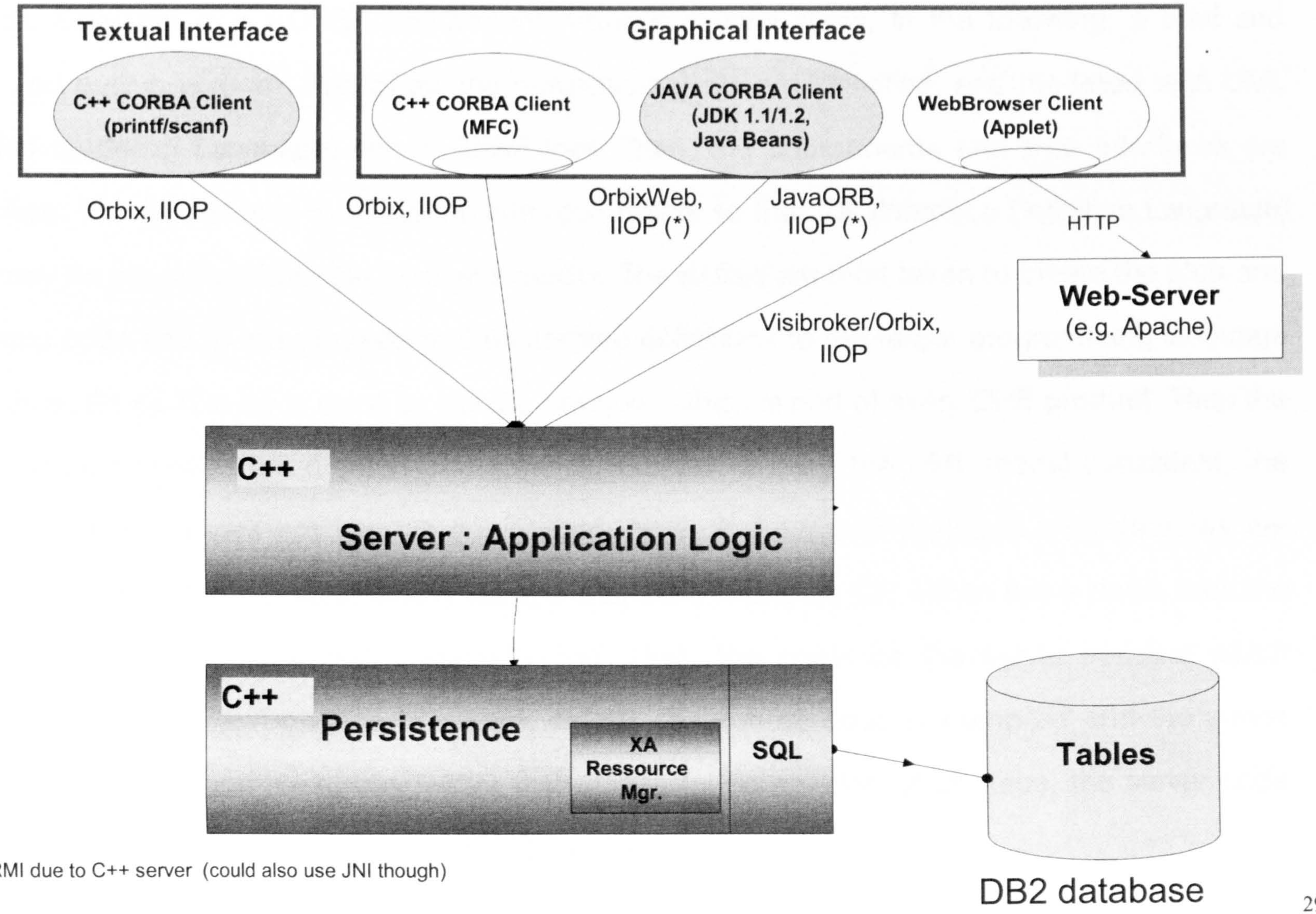
CORBA IDL - the object contract language

- *IDL is language independent*
- *IDL is not a complete programming language (no flow control or iterators)*
- *IDL compiler is needed to establish target language mapping*

```
// ==========================================================
// file:      sample.idl
// task:      Definition of the IDL interface to connect to a "SYSTEM"
// ==========================================================
```

```
module CORBA_Module
{
   typedef string XML_String;

   interface I_SYSTEM
   {
         void sendXMLString (in XML_String data);
   };
};
```

CORBA Programming : application architecture



(*) no RMI due to C++ server (could also use JNI though)

How to create a CORBA object?

The development of a CORBA component requires several steps. In the following, a brief and simplified outline is given. First of all, the business objects are identified, and modelled with UML (Unified Modeling Language) and a CASE tool. Then, the components and their interfaces are specified. The next step is to refine the interface model so that IDL (Interface Definition Language) files may be generated from the business model. The idl files are then taken to create the stub and skeleton code and to map/transform the interface definitions to the target programming language (e.g. Java, C++). This job is done by the IDL compiler which is part of every ORB product. Then the implemention classes are created and refined. In order to keep the UML model consistent, the implementation classes are reverse engineered. Now, the required methods and attributes are designed and further specified, ideally using the CASE tool and an IDE. When this is done, then the behaviour of the components is implemented. Next, the code for the server mainline which instantiates the CORBA objects is written. Finally, the source code is compiled and the server executable is built and registrated. After the successful completion of all steps, the server code may be tested and client applications may be written.

How to create a CORBA object in a run-time environment?

- Usage of a CASE tool to model the business objects (opt.)
  (identification of required objects and public interfaces)

- Creation of IDL file <name>.idl (from business model, or editing via text editor)

- Running IDL compiler to create stub and skeleton code
  (idl -B <name>.idl :: <name>C.cpp, <name>S.cpp, <name>.hh)

- Creation of implementation classes and refining them
  (idl -S <name>.idl :: <name>.ih, <name>.ic :: renamed <name>i_.h, <name>_i.c)

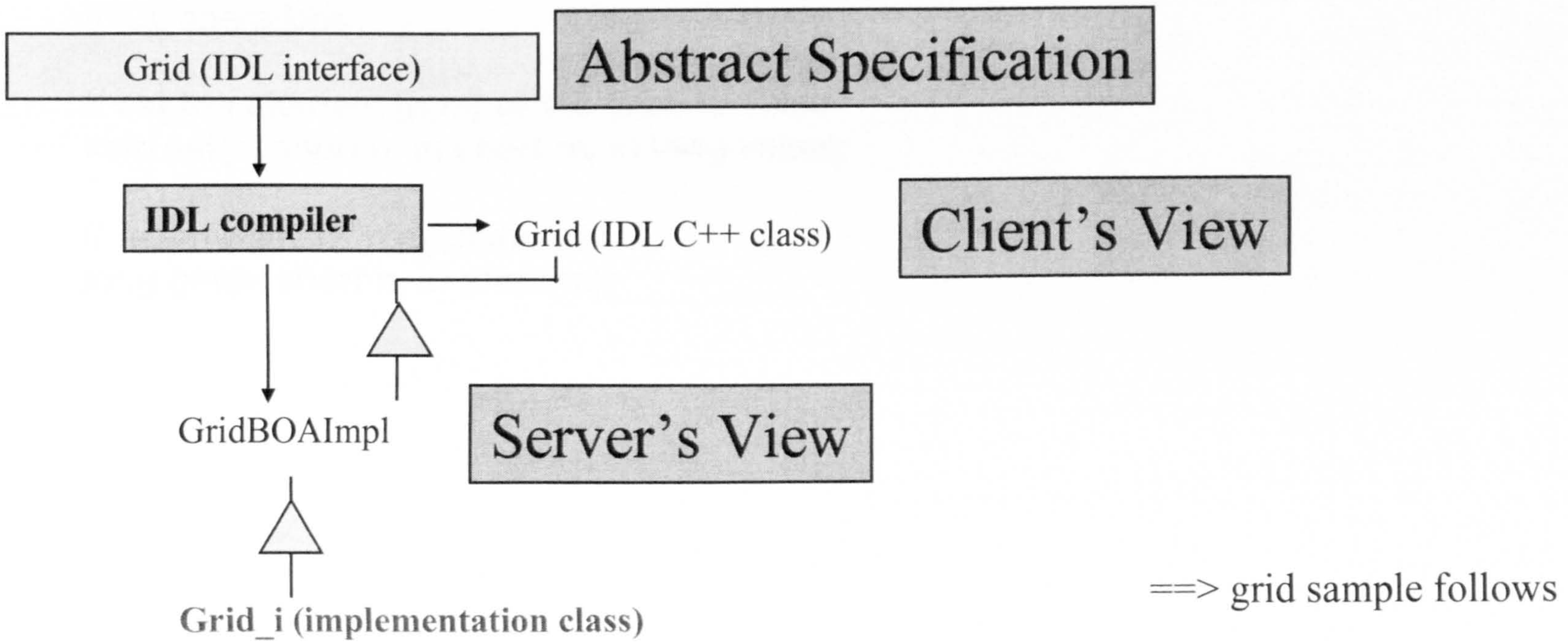
- Reverse engineer implementation classes into CASE tool and designing/specifying methods&attributes

- Implementation of the IDL file (e.g. with C++ classes) and server

- Compiling and building the server then running the server
  (<name>S.cpp, <name>_i.cpp, <server>.cpp), nmake (Makefile)

CORBA IDL to C++ mapping - BOAImpl approach

IDL module -> C++ namespace
IDL interface -> C++ class
IDL operation -> C++ member function
IDL attribute -> C++ member function
...



==> grid sample follows

CORBA Programming

Creating the Interface Specification (IDL file)

```
interface grid {
    readonly attribute short height;  // height of the grid
    readonly attribute short width;   // width of the grid

    // IDL operations

    // set the element [n,m] of the grid, to value:
    void set(in short n, in short m, in long value);

    // return element [n,m] of the grid:
    long get(in short n, in short m);
};
```

CORBA Programming
Implementing the Grid class

```
#include "grid_i.h"

// ctor
grid_i::grid_i(CORBA::Short h, CORBA::Short w){
   m_height=h;  // set up height
   m_width=w;   // set up width
   // now allocate the 2-D array: as an array of pointers to 1-D arrays.
   m_a = new CORBA::Long* [h];
   for (int i = 0; i < h; i++ )
            m_a[i] = new CORBA::Long[w];
}

// dtor
grid_i::~grid_i(){
   // free the individual 1-D arrays:
   for (int i = 0; i < m_height; i++)
            delete[] m_a[i];
   // then free the overall array:
   delete[] m_a;
}

// implementation of the function which reads the height attribute
CORBA::Short grid_i::height(CORBA::Environment &)
#if defined(IT_RAISE_NAT_EH) && !defined(PCWORLD)
            throw (CORBA::SystemException)
#endif
{
   return m_height;
}
```

```
// implementation of the function which reads the width attribute
CORBA::Short grid_i::width(CORBA::Environment &)
#if defined(IT_RAISE_NAT_EH) && !defined(PCWORLD)
            throw (CORBA::SystemException)
#endif
{
   return m_width;
}

// implementation of the set operation:
void grid_i::set(CORBA::Short n, CORBA::Short m, CORBA::Long value,
CORBA::Environment &)
#if defined(IT_RAISE_NAT_EH) && !defined(PCWORLD)
            throw (CORBA::SystemException)
#endif
{
   m_a[n][m] = value;
}

// implementation of the get operation:
CORBA::Long grid_i::get(CORBA::Short n, CORBA::Short m,
CORBA::Environment &)
#if defined(IT_RAISE_NAT_EH) && !defined(PCWORLD)
            throw (CORBA::SystemException)
#endif
{
   return m_a[n][m];
}
```

CORBA Programming
Developing an Orbix Server Program

```
#include <iostream.h>
#include <stdlib.h>
#include "grid_i.h"

int main()
{
  // create a grid object - using the implementation class grid_i
  grid_i  myGrid(100,100);

  try {
    // tell Orbix that we have completed the server's initialisation:
    CORBA::Orbix.impl_is_ready("grid");
  }
    catch (CORBA::SystemException &sysEx) {
    cerr << "Unexpected system exception" << endl;
    cerr << &sysEx;
    exit(1);
  } catch (...) {
    // an error occured calling impl_is_ready() - output the error.
    cout << "Unexpected exception" << endl;
    exit(1);
  }

  // impl_is_ready() returns only when Orbix times-out an idle server (or an error occurs).
  cout << "server exiting" << endl;

  return 0;
}
```

CORBA Programming
Developing a Client Application

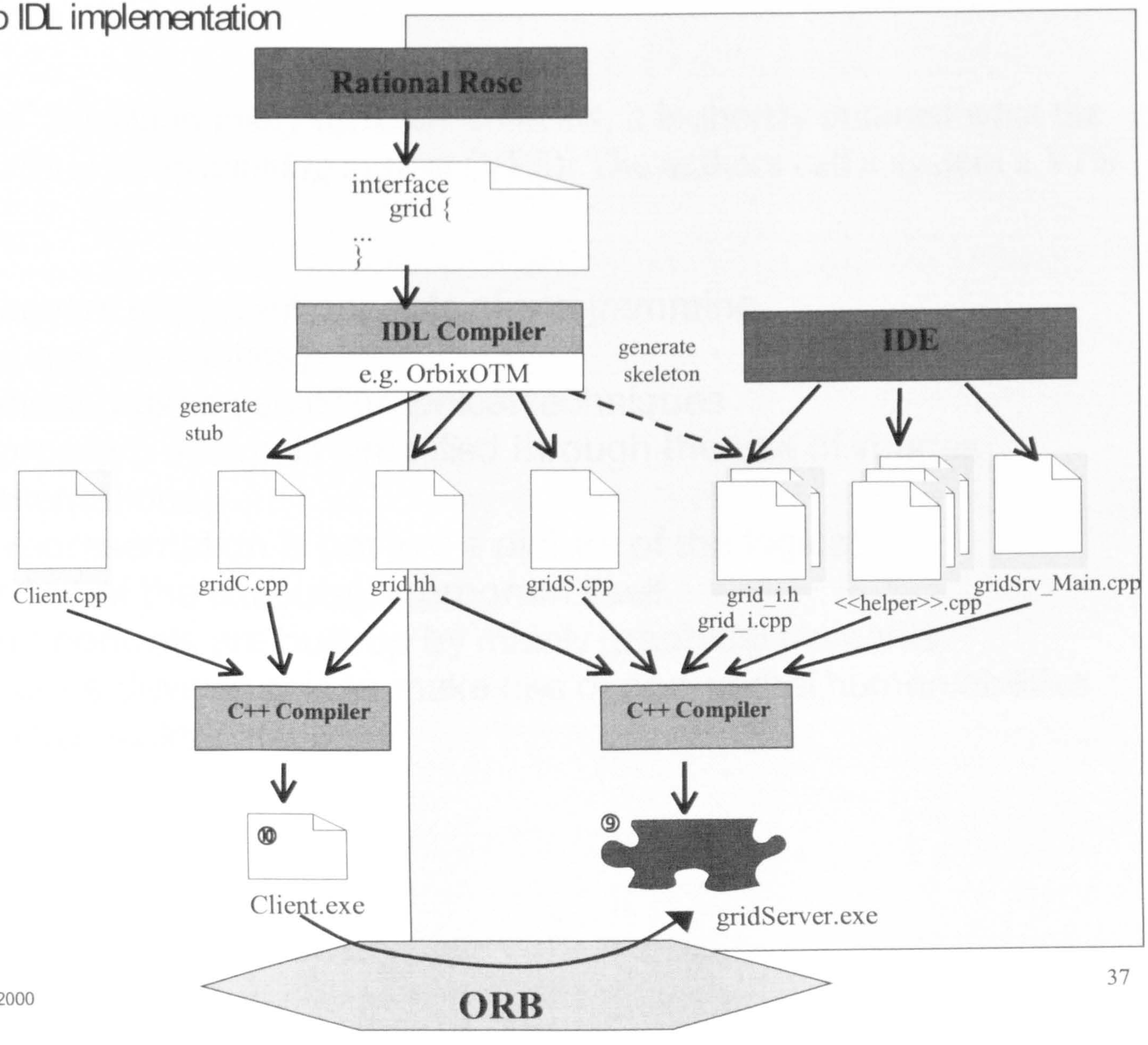
```
#include "grid.hh"
#include <iostream.h>
#include <stdlib.h>


int main (int argc, char **argv) {
  grid_var gridVar;   // pointer the grid object that will be used.
  CORBA::Short h, w;
  CORBA::Long v;

  if (argc < 2) {
    cout << "usage: " << argv[0] << " <hostname>" << endl;
    exit (-1);
  }
  try {
    // First bind to the grid object.
    // argv[1] has the hostname (if any) of the target grid object;
    // The default is the local host:
    gridVar = grid::_bind("", argv[1]);
  } catch (CORBA::SystemException &sysEx) {
    cerr << "Unexpected system exception" << endl;
    cerr << &sysEx;
    exit(1);
  } catch(...) {
    // an error occurred while trying to bind to the grid object.
    cerr << "Bind to object failed" << endl;
    cerr << "Unexpected exception " << endl;
    exit(1);
  }

  try {
    // try to read the height and width of the grid:
    h = gridVar->height();
    w = gridVar->width();
  } catch (CORBA::SystemException &sysEx) {
    cerr << "Unexpected system exception" << endl;
    cerr << &sysEx;
    exit(1);
  } catch(...) {...
  }

  // no problem reading the height and width:
  cout << "height is " << h << endl;
  cout << "width  is " << w << endl;

  try {
    // try to set element [2,4] of the grid - to value 123
    gridVar->set(2, 4, 123);
    // then read back what we have just set:
    v = gridVar->get(2, 4);
  } catch ...

  // no problem setting and getting the elememt:
  cout << "grid[2,4] is " << v << endl;

return 0;
}
```

CORBA Programming

From IDL generation to IDL implementation

Grid sample

The client side may be implemented via any CORBA compliant tools.



Visual Languages & Visual Programming

As the term "visual" is used in many different contexts, it is shortly outlined what the authors mean by a *visual programming system* (VPS). The authors call a system a VPS if:

- the included VPL covers important aspects of programming (i.e. a visual syntax and semantics).
- the visual environment makes use of graphical techniques (for instance, the program design is simplified through the use of images and pictorial representations).
- ideally, the visual representation is not just a picture of the logical program structure but of the executable program itself.
- the user interaction controls are built up by mainly graphical elements.
- the aim of the system's developer is to make use of non-verbal human abilities (i.e. "right hemisphere" tasks).

What is a Visual Programming Language?

A few representative answers [http://www.faqs.org/faqs/visual-lang/faq/]:

(a) Visual Programming (VP) refers to any system that allows the user to specify a program in two-(or more)-dimensionsional fashion. [...] conventional textual languages are not considered two dimensional since the compilers or interpreters process them as long, one-dimensional streams. [Myers90a]

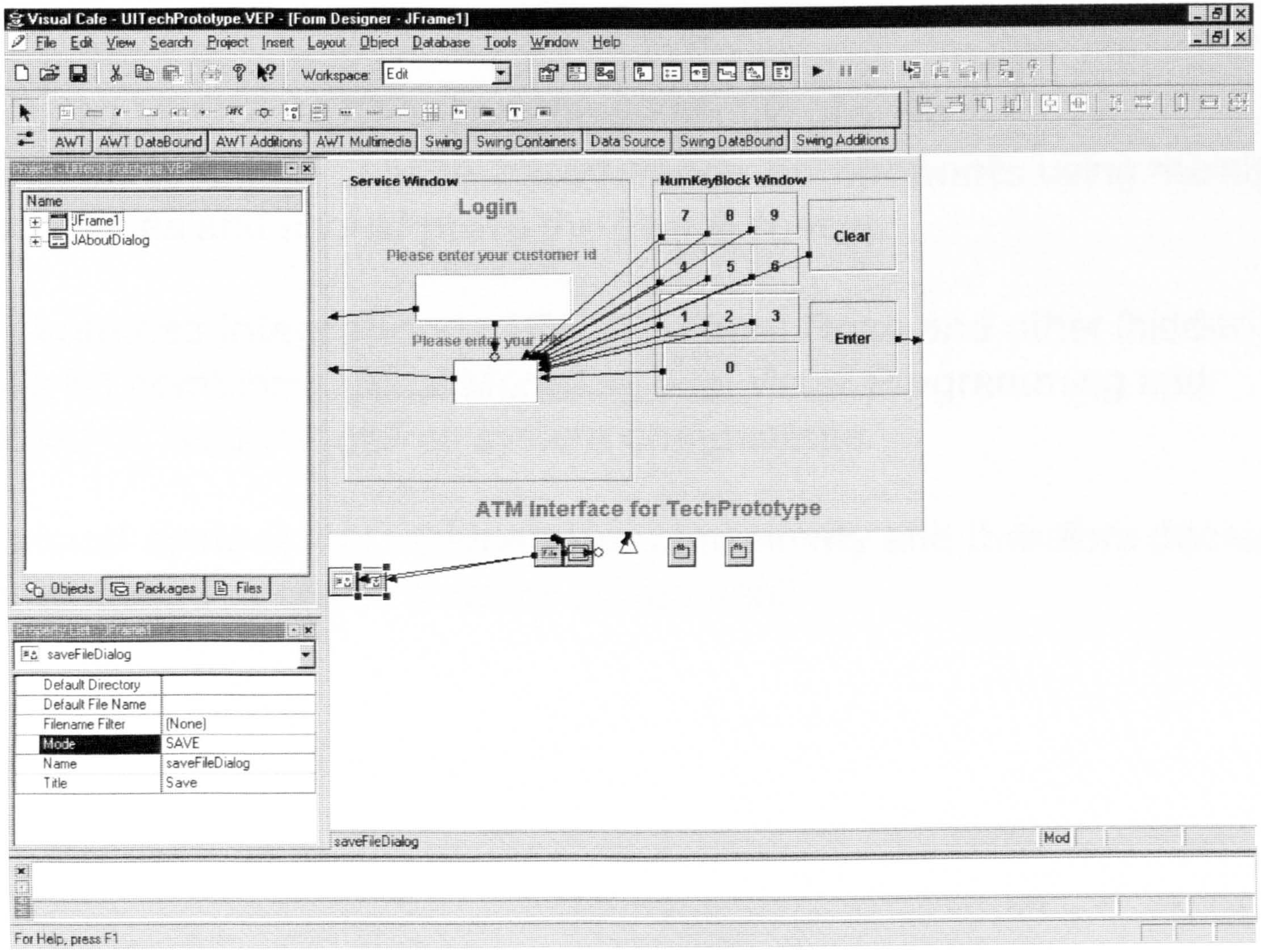
(b) A Visual Language manipulates visual information or supports visual interaction, or allows programming with visual expressions. The latter is taken to be the definition of a visual programming language. Visual programming languages may be further classified according to the type and extent of visual expression used, into icon-based languages, form-based languages and diagram languages. Visual programming environments provide graphical or iconic elements which can be manipulated by the user in an interactive way according to some specific spatial grammar for program construction. [Golin90b]

(c) Visually transformed languages are inherently non-visual languages but have superimposed visual representations. Naturally visual languages have an inherent visual expression for which there is no obvious textual equivalent. [Burnett89]

(d) Visual programming is commonly defined as the use of visual expressions (such as graphics, drawings, animation or icons) in the process of programming. These visual expressions may be used in programming environments as graphical interfaces for textual programming languages; they may be used to form the syntax of new visual programming languages leading to new paradigms such as programming by demonstration; or they may be used in graphical presentations of the behavior or structure of a program. [McIntyre&Burnett]

(e) A visual language is a set of spatial arrangements of text-graphic symbols with a semantic interpretation that is used in carrying out communication actions in the world.

IDE: Visual Café 3.0a (Visual Bean Support)



VOODE/VOOPL-I for CORBA

Objectives of the Prototype

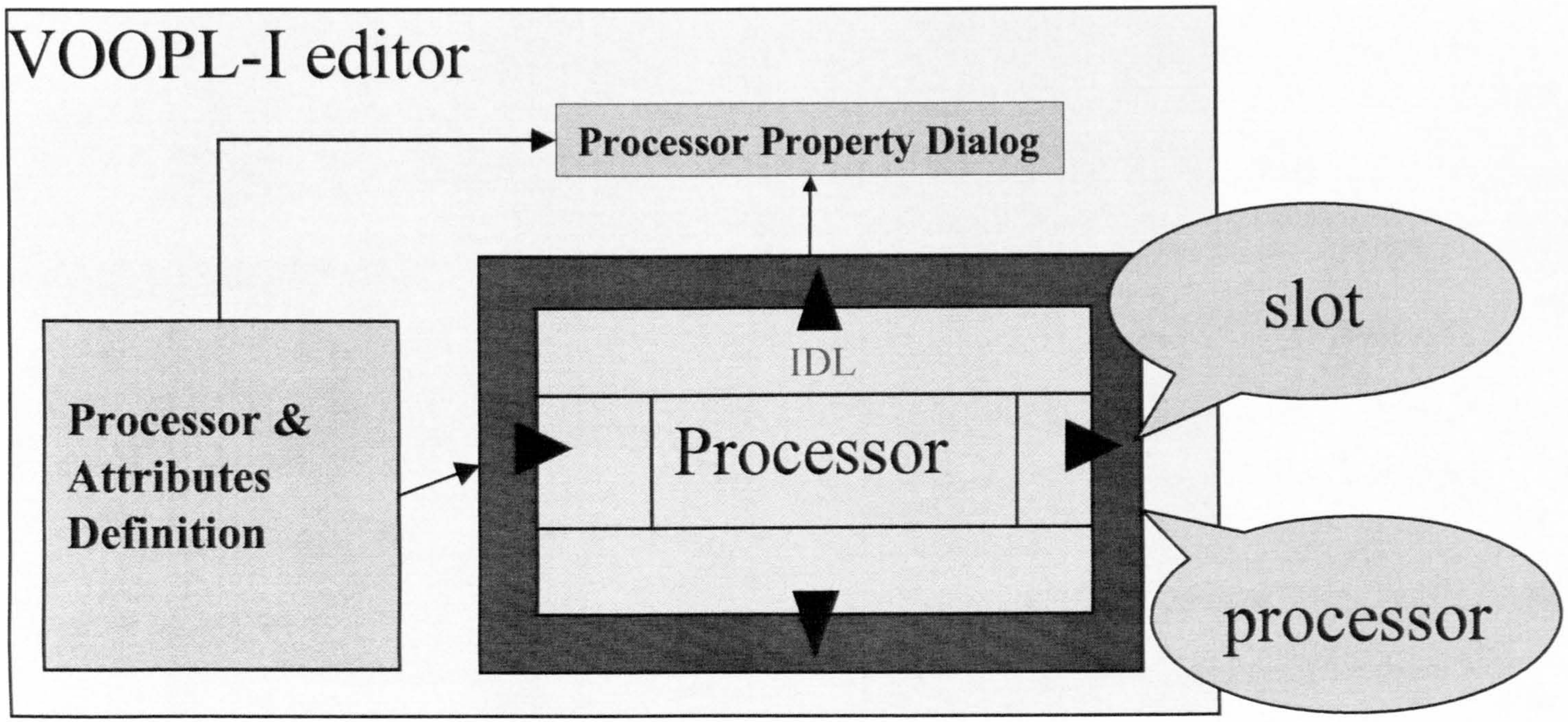
The intended VOODE/VOOPL-I prototype should provide :

- a means to create CORBA-based **server components** using mainly visual techniques and should prove the overall concept.

- a **seamless integration in UML/Rational Rose** and other (hidden) tools. This is : combining Visual Modelling with Visual Programming and reuse of "helper tools" as system components.

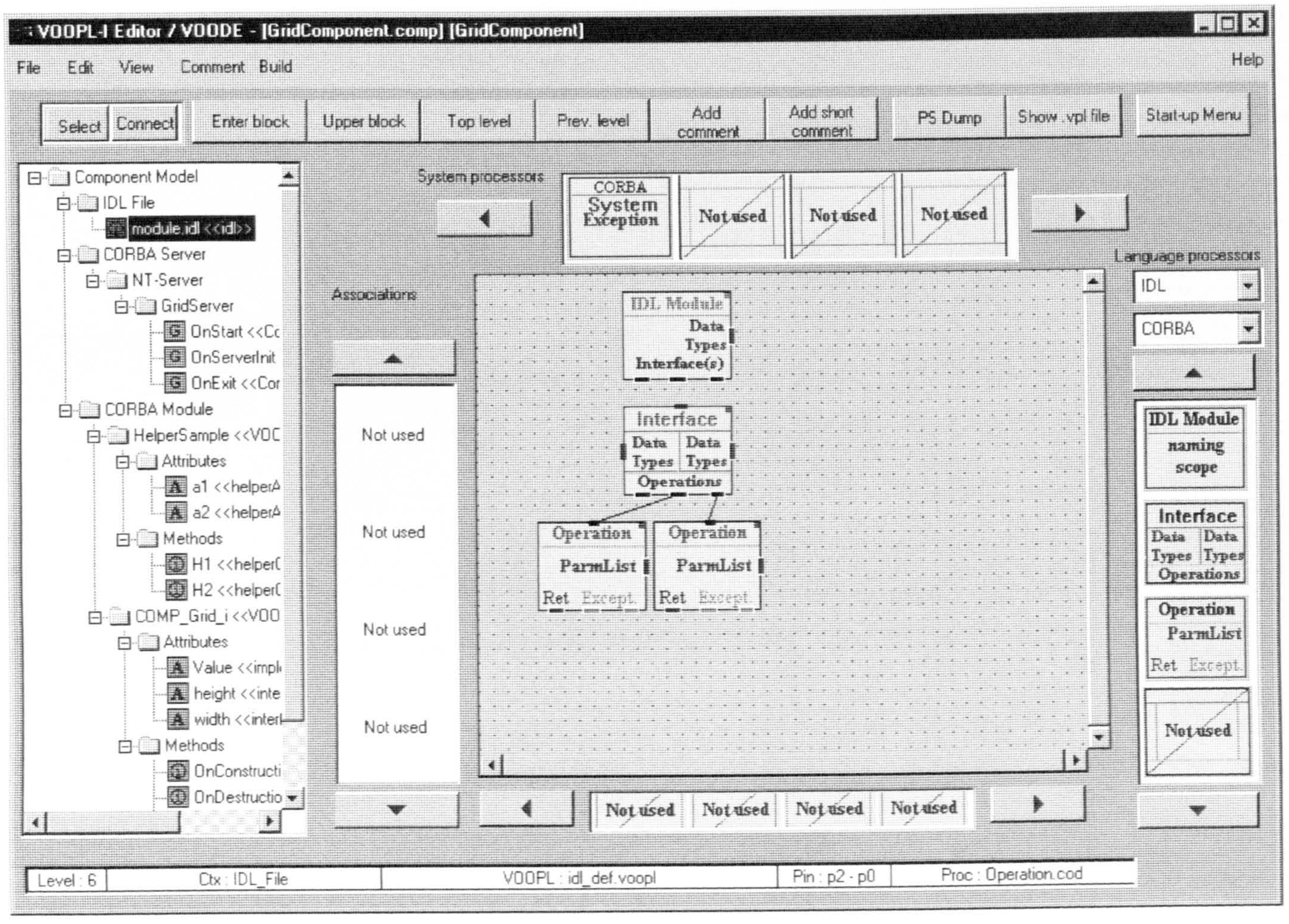
- a **visual system** that **reduces the complexity** and therefore decreases the time needed to create a server component.

The visual metaphor



A **processor** represents an action which is carried out at a certain time. A processor consists of different **slot** types (e.g. events, data, parameters) and contains **attributes** which specify the arguments of the action to be carried out.

IDL Definition



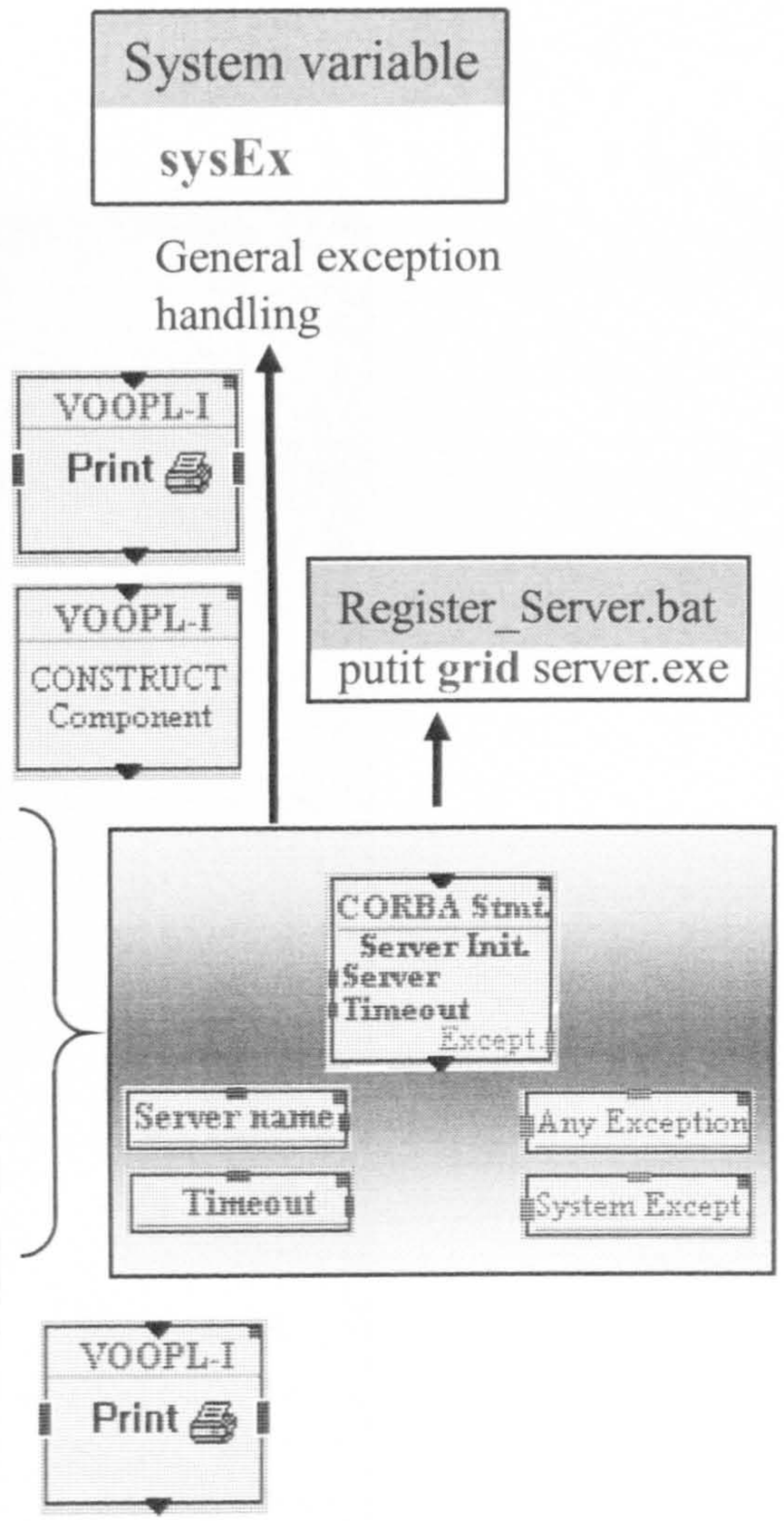
Visual CORBA Programming

Developing an Orbix Server (grid sample)

automatically generated!	#include <iostream.h> #include <stdlib.h> #include "grid_i.h" int main() {
OnStart	cout << "server started" << endl;

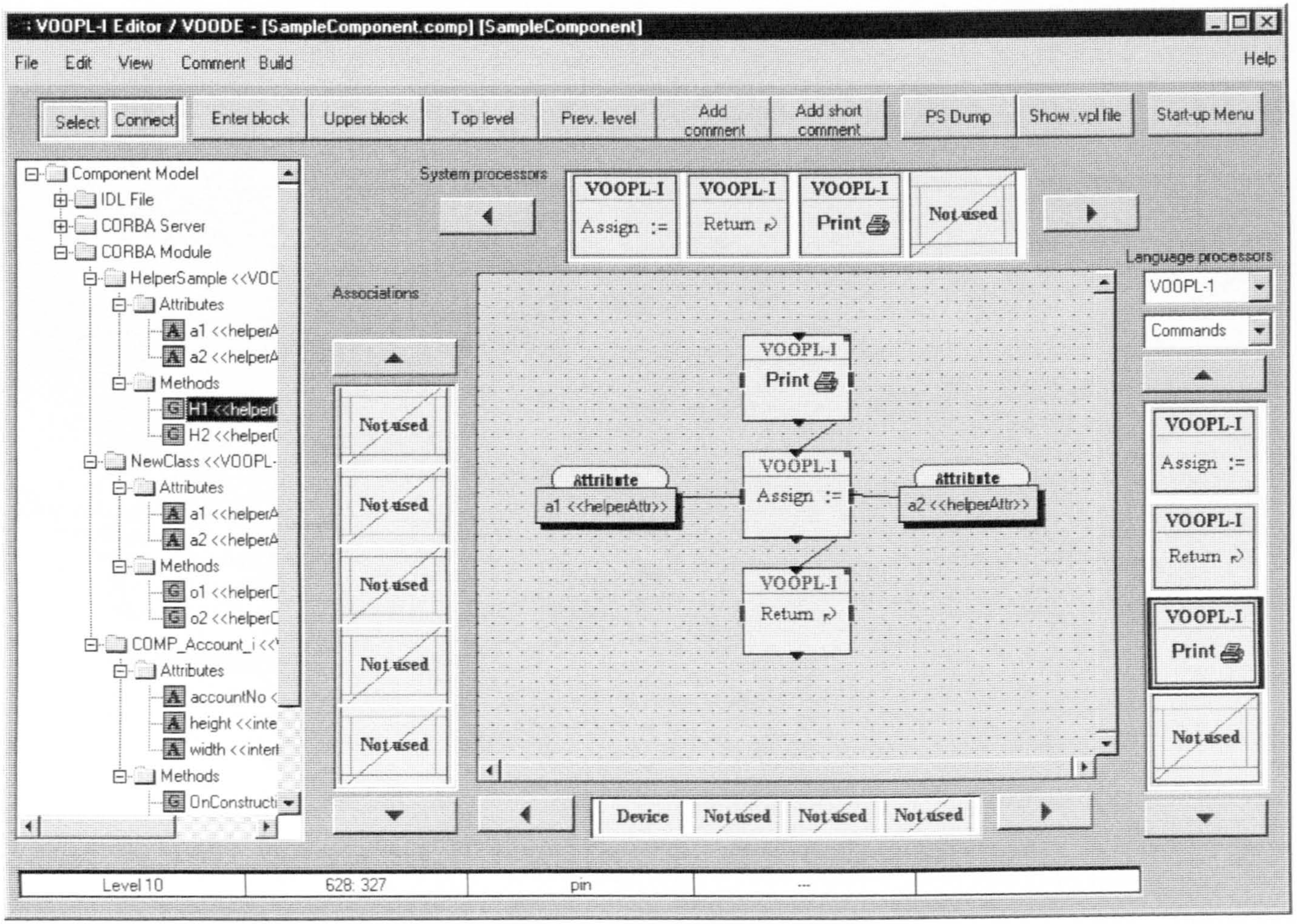
OnServerInit	// create a grid object - using the implementation class grid_i
	grid_i myGrid(100,100);
	try {
	// tell Orbix that we have completed the server's initialisation:
	CORBA::Orbix.impl_is_ready("**grid**");
	}
	catch (CORBA::SystemException &sysEx) {
	cerr << "Unexpected system exception" << endl;
	cerr << &sysEx;
	exit(1);
	} catch (...) {
	// an error occured calling impl_is_ready() - output the error.
	cout << "Unexpected exception" << endl;
	exit(1);
	}

OnExit	// impl_is_ready() returns only when Orbix times-out an idle server (or an error occurs). cout << "server exiting" << endl;
automatically generated!	return 0; }



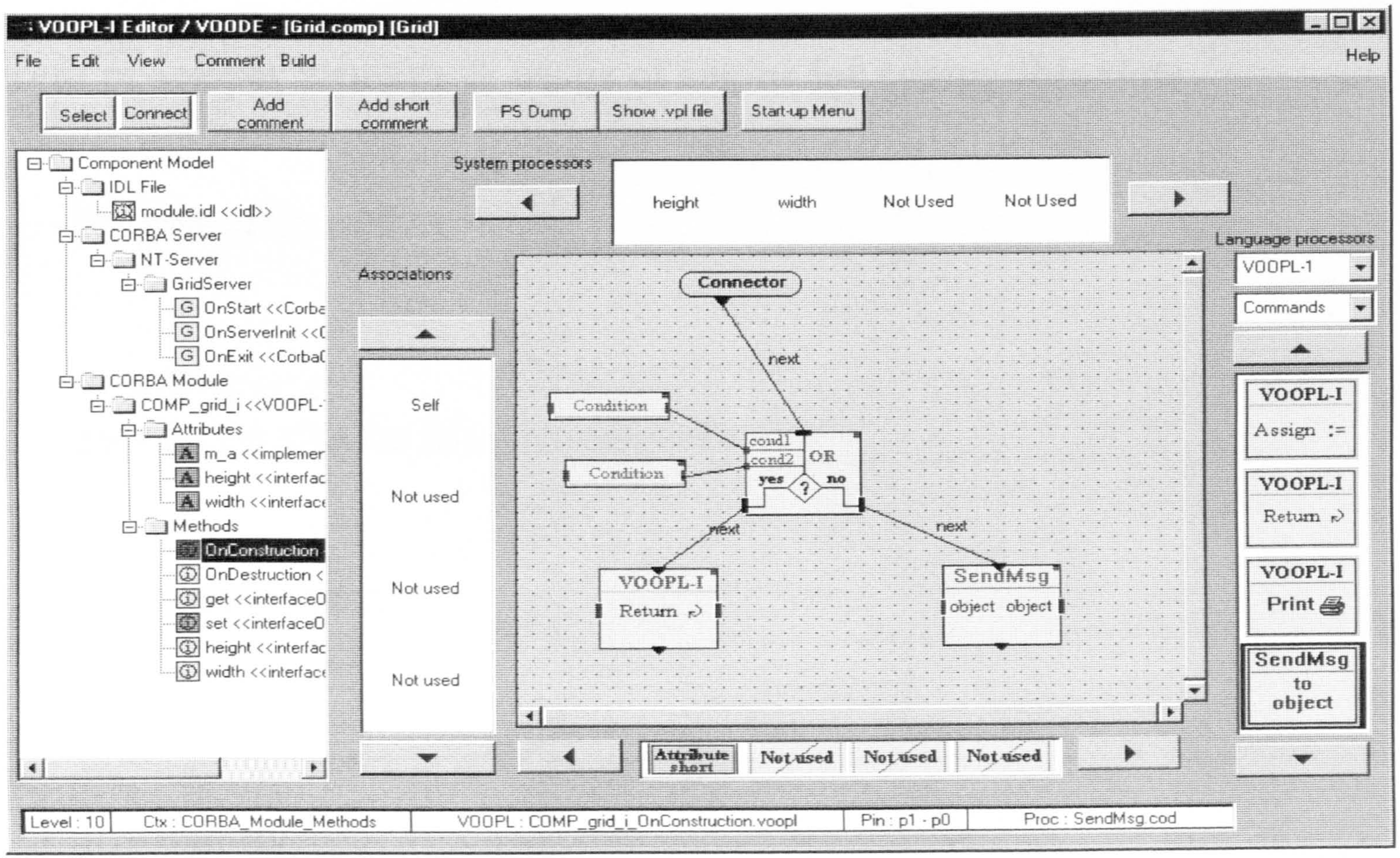
Visual CORBA Programming

Implementing the Grid class



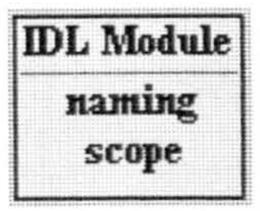
VOODE/VOOPL-I for CORBA

Implementing the Grid class

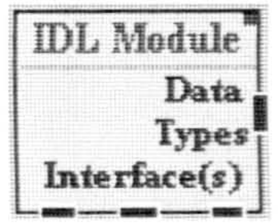


"Traffic light" metaphor

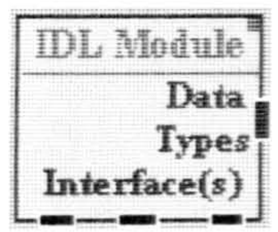
Different processor bitmaps & usage of colour for compatible slots



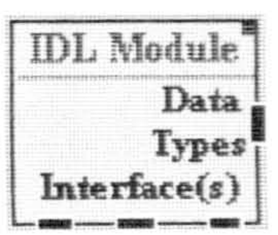
- for selection matrix



for workplace
- syntax/configuration ok/complete

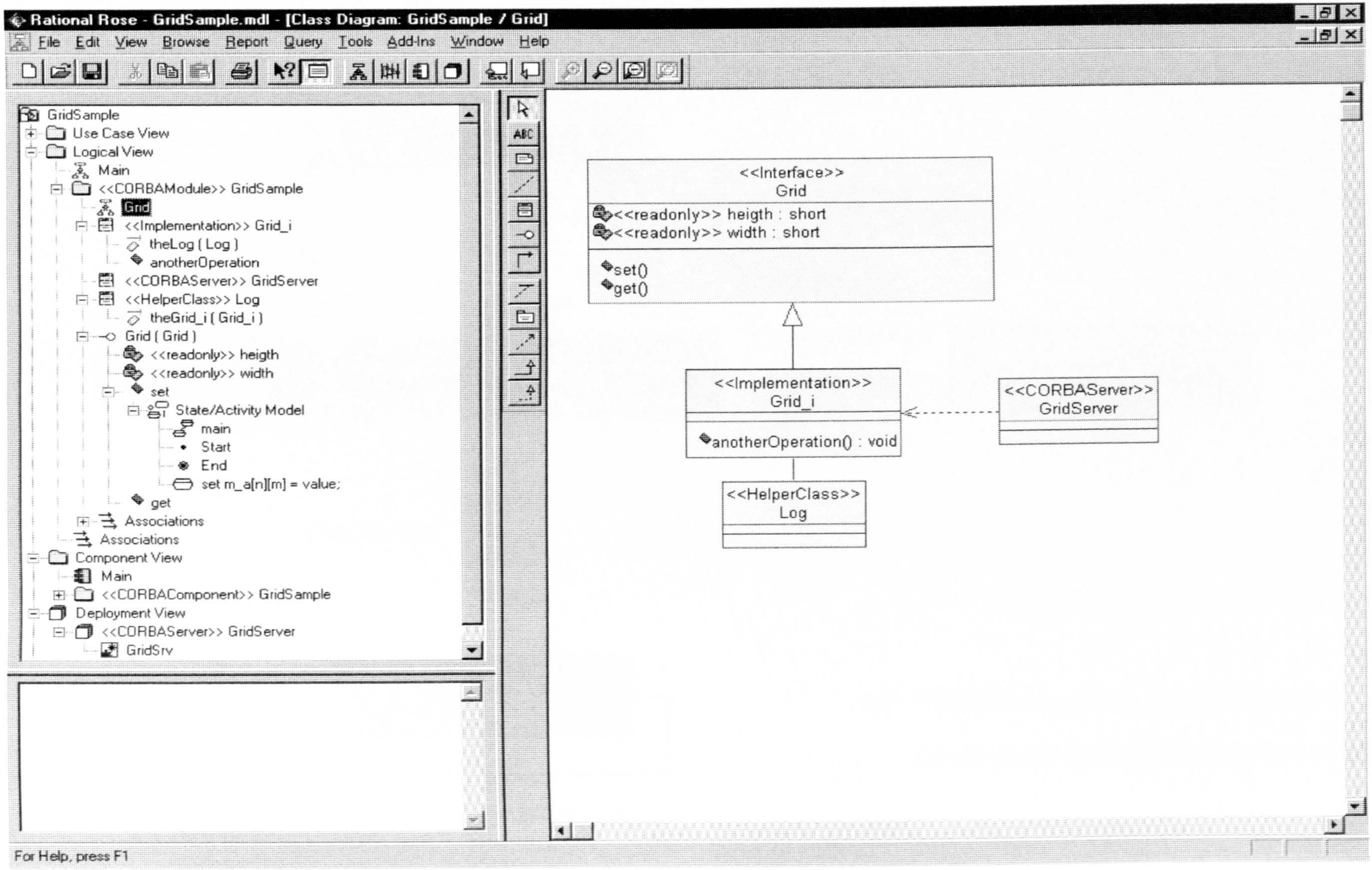


- connection error



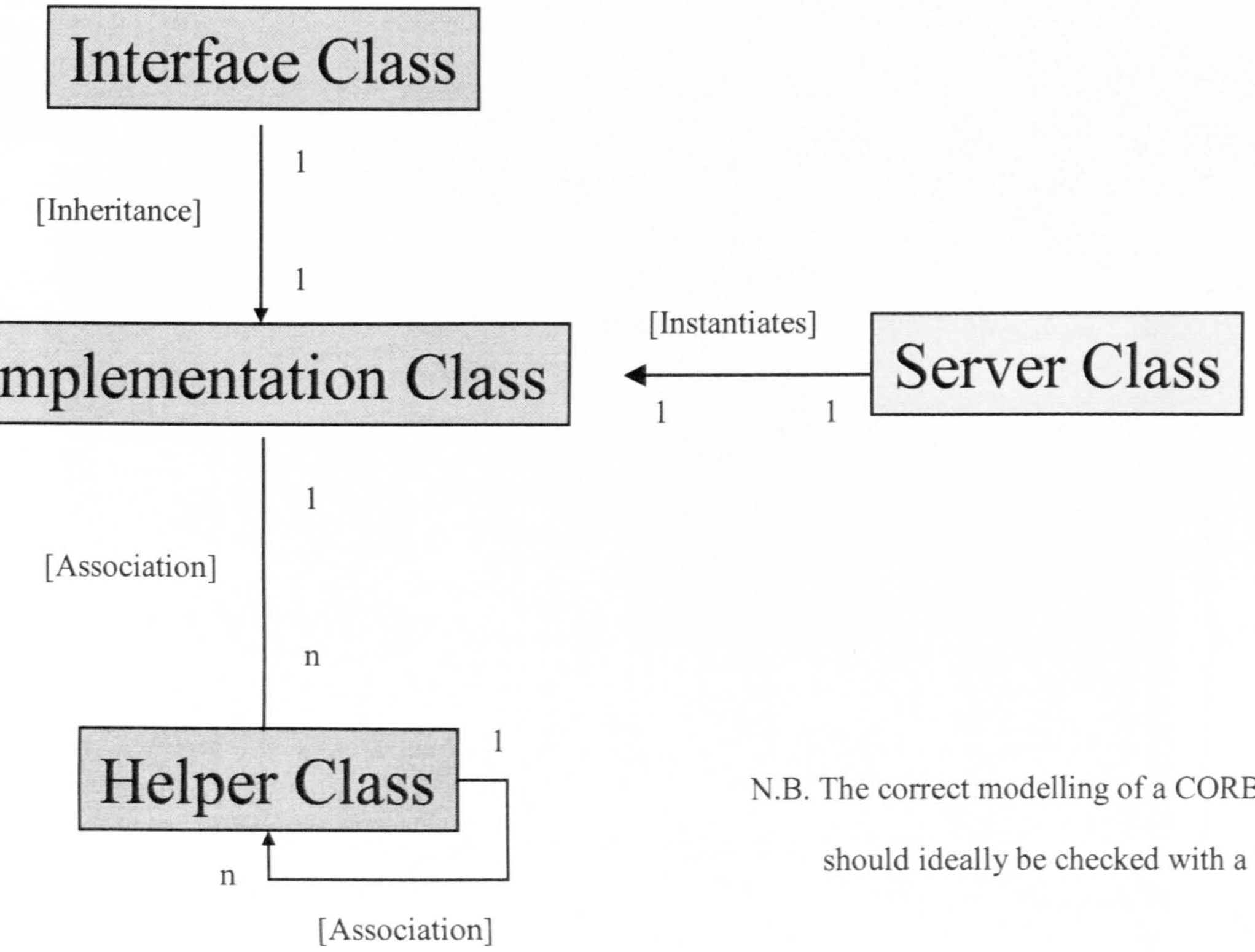
- configuration error (attribute values are not correct)

VOODE/VOOPL-I : sample model of CORBA component



Visual Modelling of CORBA Components

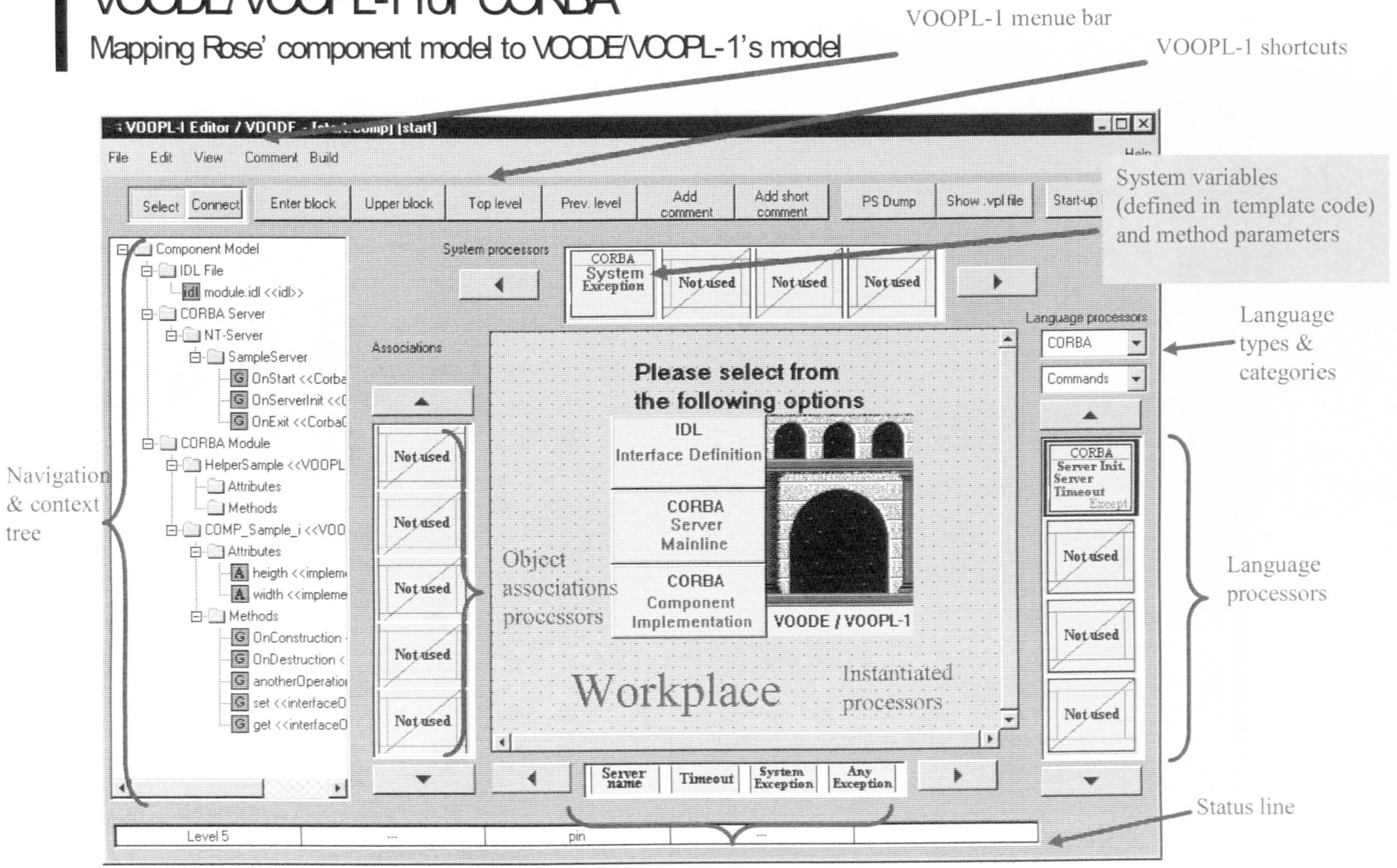
Meta model (simplified view)



N.B. The correct modelling of a CORBA object should ideally be checked with a Rose script!

VOODE/VOOPL-I for CORBA

Mapping Rose' component model to VOODE/VOOPL-1's model



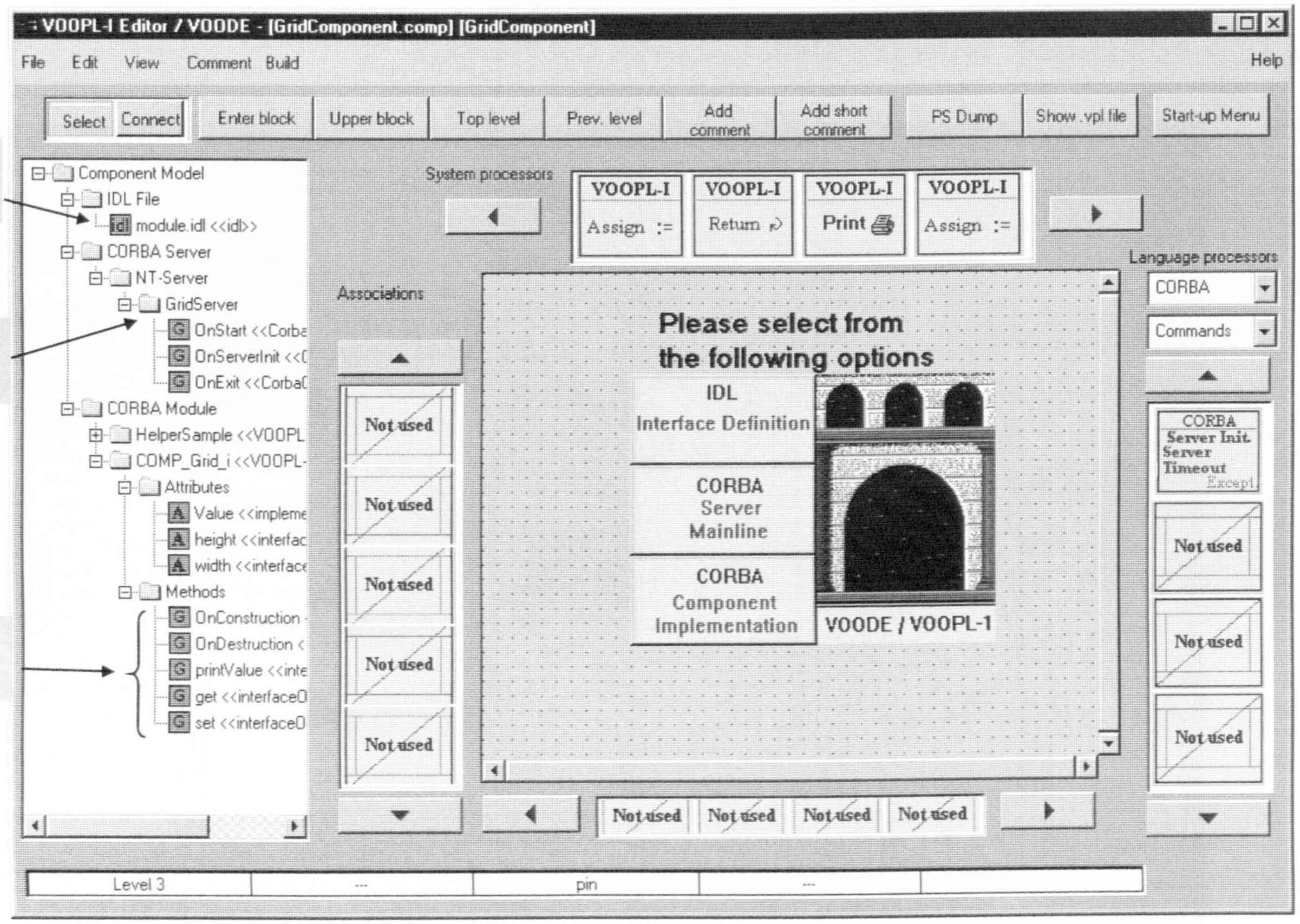
VOODE/VOOPL-I for CORBA

Mapping Rose' component model to VOODE/VOOPL-1's model (Grid sample)

visual specification

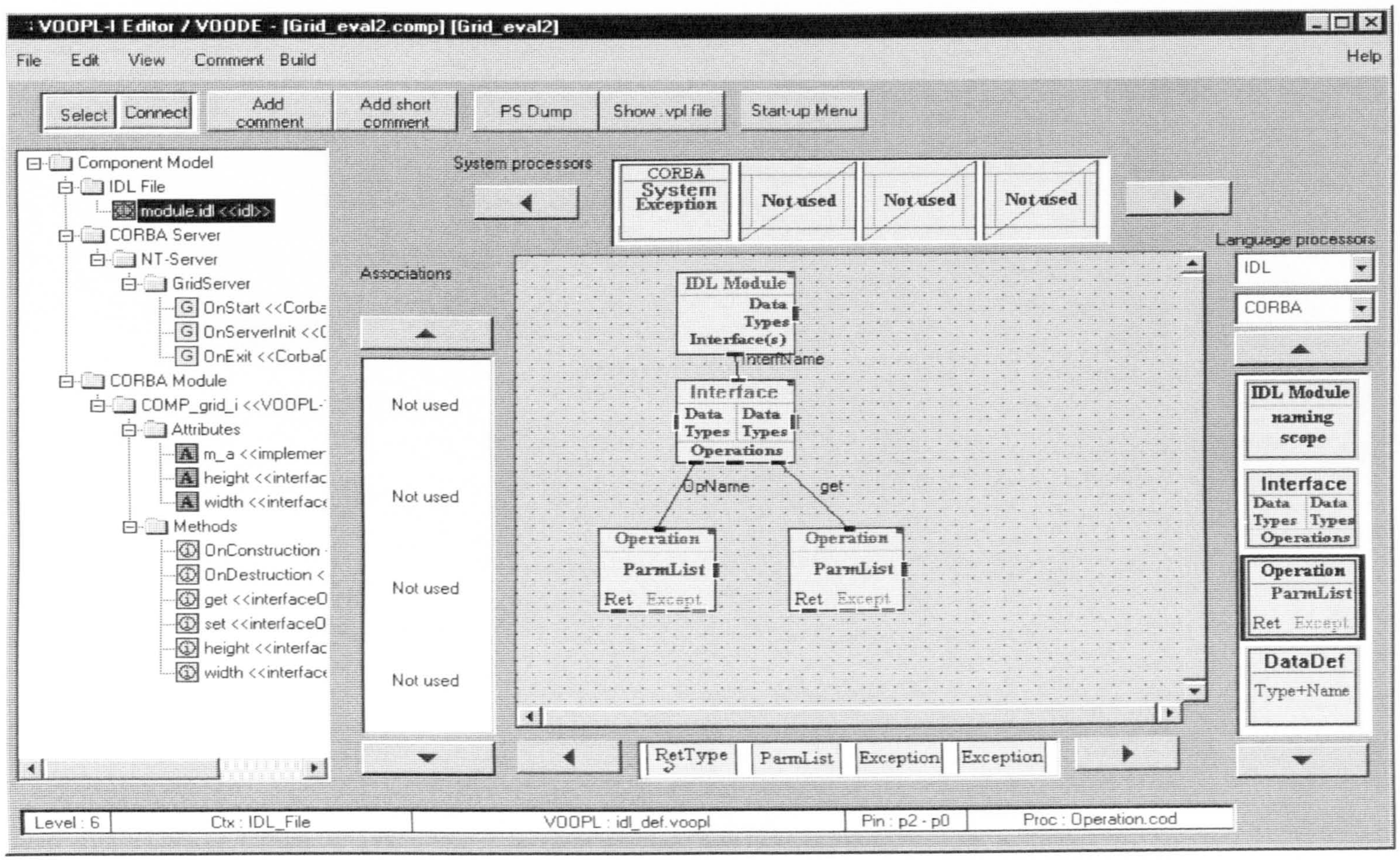
visual configuration

visual programming



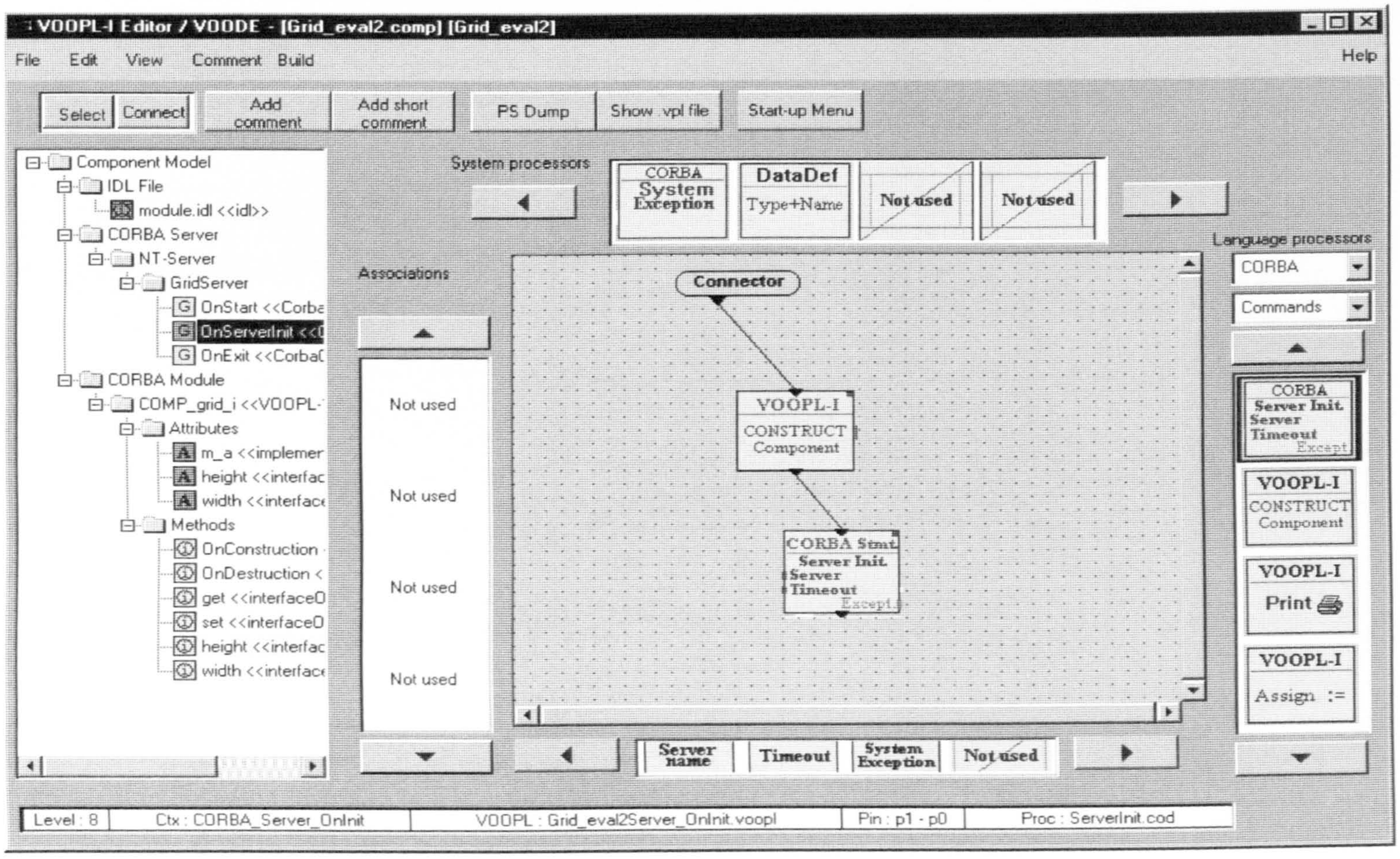
VOODE/VOOPL-I for CORBA

Grid sample



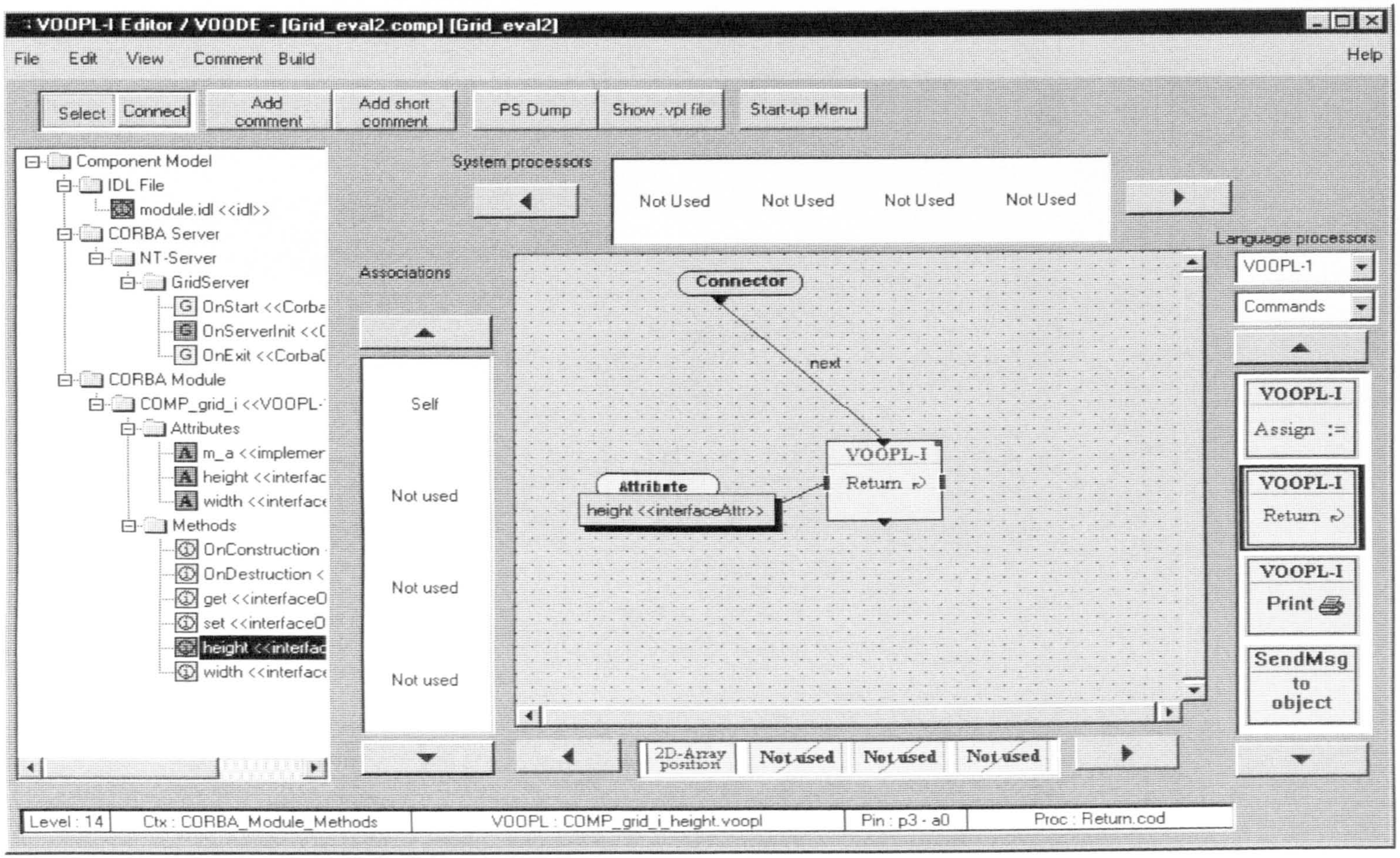
VOODE/VOOPL-I for CORBA

Grid sample



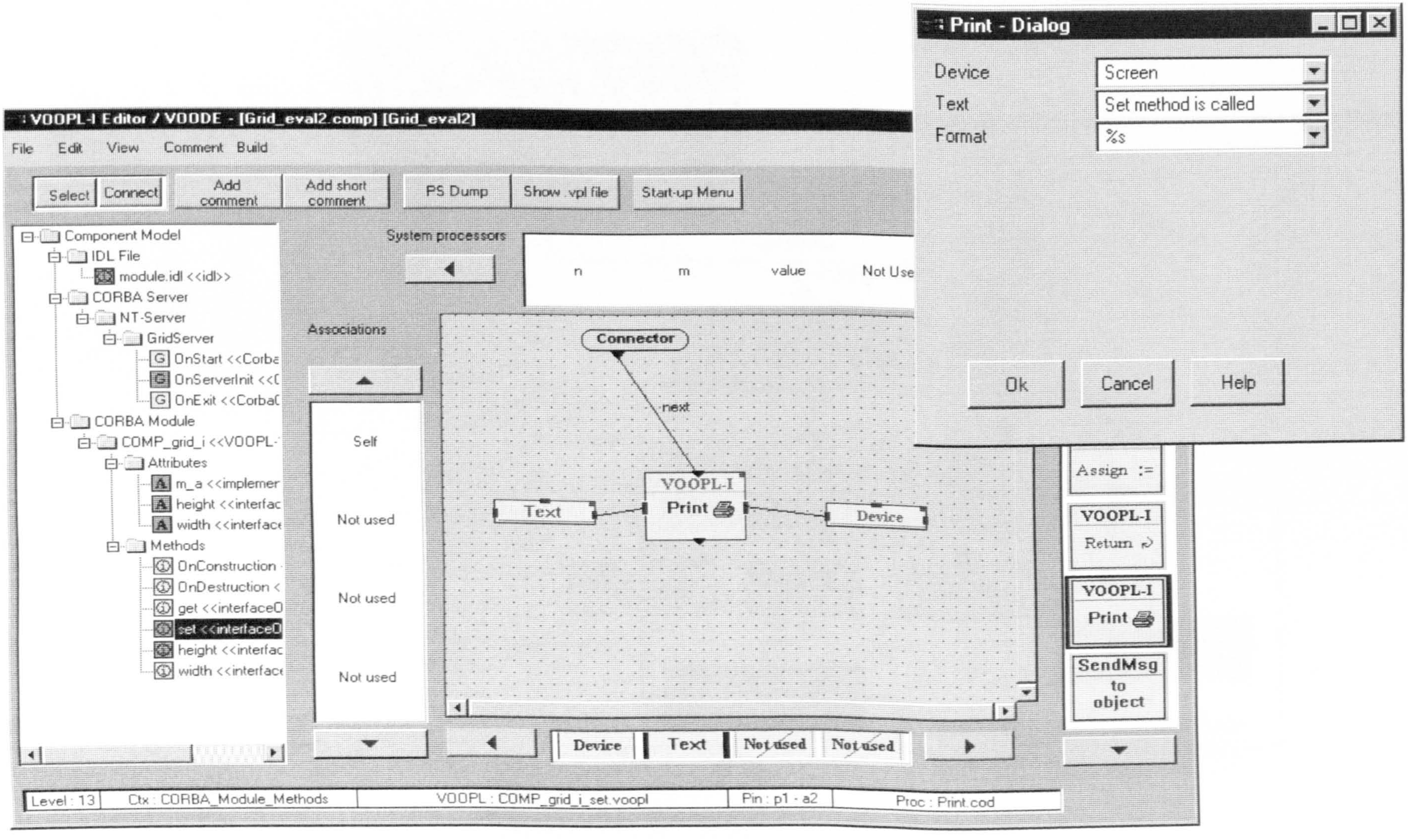
VOODE/VOOPL-I for CORBA

Grid sample



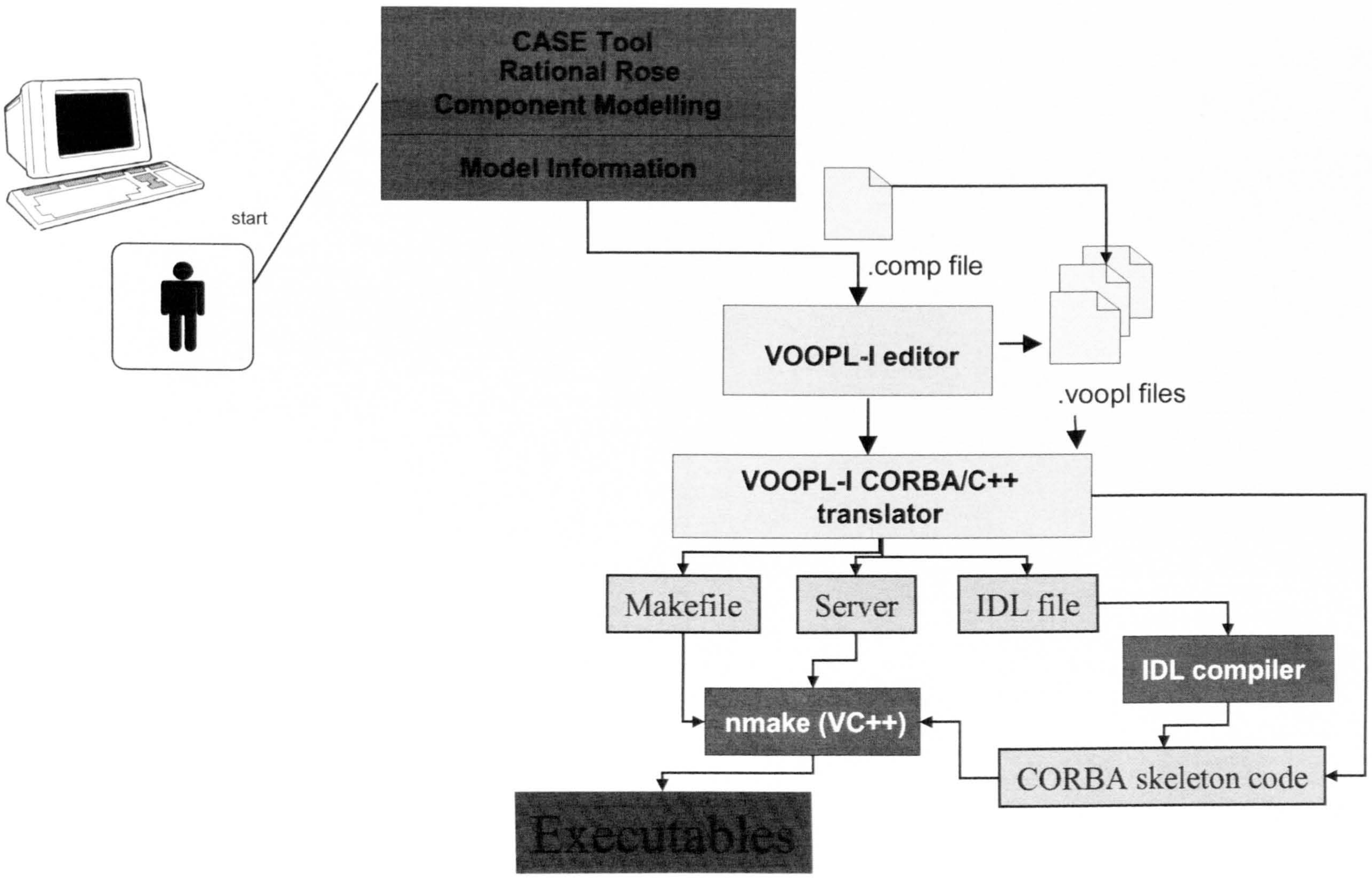
VOODE/VOOPL-I for CORBA

Grid sample



VOODE/VOOPL-I for CORBA

System Architecture (old version)



End of training

Evaluation 2

Start of evaluation

User Evaluations

Eval2: **User evaluation on CORBA programming**

Background information

During system development user requirements may change several times. This causes both the change of existing code and the extension of new code. Under these circumstances you should be able to understand and change the existing code quite easily.

The aim of the following trials is to compare the efficiency of two different approaches.

The tasks which you are asked to carry out are based on the "Grid" sample used in the training before. There are three main tasks for the evaluation. The first task evaluates the difficulties when changing an existing method, the second task concentrates on the definition of a new method and the third task is about the creation of a CORBA server.

For the sake of simplicity it is not necessary that the code is compiled and tested.

If you have any questions than please ask now. All tasks are carried out with no further help. After each evaluation step a questionnaire is given to you so that you are able to give a statement about your experiences and problems.

You should give the evaluator a statement when you think that the task is completed.

Preparation by evaluator : a) start of Visual Studio (or Explorer and Textpad), MS-Dos command window
b) start of Rational Rose and VOODE/VOOPL-1 for CORBA

Visual Studio (or Explorer and Textpad)

User Evaluations

Eval2: **User evaluation on CORBA programming**

Task 1: Changing an existing method

Tasks Descriptions

The existing Grid sample which was introduced in the training shall be extended.

1a) The **set** method of class **grid_i** should be changed so that the additional **trace message**
"set method is called"
is displayed on the screen if the method is called by a client application.

(max. 4 minutes)

1b) After testing the code the problem was found that two-dimensional arrays may be created with dimensions [0][1] or [1][1]. It was decided that only arrays with **dimensions [n][n] (n>= 2)** are acceptable. Please change the code so that this requirement is fulfilled.

(max. 6 minutes)

Questionnaire : CORBA programming

Q1: Are you confident that you did solve the problem? **A1**: yes ☐ no ☐

Q2:: How do you think have you completed the tasks?
1 (very good) 2 3 4 5 6 (poor) **A2:** _____

Q3: What general **experiences** did you encounter while trying to write the code?
A3:

_____

_____

_____

Q4: What have been the major problems for you? Please state three problems if possible.

A4: 1 _____

2 _____

3 _____

User Evaluations

Eval2: **User evaluation on CORBA programming**

Task 2: Definition of a new method

Tasks Descriptions

The existing Grid sample which was introduced in the training shall be extended.

2a) A new interface method **print** should be added to the interface class. No arguments are needed in order to call the method. No value is to be returned to the calling method. The **initial behaviour** of the method shall be that the message "print method is called" is displayed on the screen.

(max. 10 minutes)

2b) The new behaviour for the method **print** should now be coded. The behaviour of the method shall be as follows:

The first 5 elements of the array should be printed out on the screen.

(max. 10 minutes)

Questionnaire : CORBA programming

Q1: Are you confident that you did solve the problem? **A1**: yes ☐ no ☐

Q2:: How do you think have you completed the tasks?
1 (very good) 2 3 4 5 6 (poor) **A2:** _____

Q3: What general **experiences** did you encounter while trying to write the code?
A3:

_____

_____

_____

Q4: What have been the major problems for you? Please state three problems if possible.

A4: 1 _____

2 _____

3 _____

User Evaluations

Eval2: **User evaluation on CORBA programming**

Task 3: definition of a CORBA server mainline

Tasks Descriptions

The existing Grid sample which was developed in the previous trials is to be extended.

3) Before a client application may use a CORBA server object it has to be created by a so-called CORBA server. The final task is to write the necessary mainline code which instantiates the server object.

   The following code is required:
   - create the **grid object**
   - "inform" Orbix that the object has been instantiated and **parameterise** the **server information** (i.e. name of server and timeout time)
   - if an error occurs the standard **system exception** handling should be implemented

(max. 10 minutes)

Questionnaire : CORBA programming

Q1: Are you confident that you did solve the problem? **A1**: yes ☐ no ☐

Q2:: How do you think have you completed the tasks?
1 (very good) 2 3 4 5 6 (poor) **A2:** _____

Q3: What general **experiences** did you encounter while trying to write the code?
A3:

_____

_____

_____

Q4: What have been the major problems for you? Please state three problems if possible.

A4: 1 _____

2 _____

3 _____

Rational Rose and VOODE/VOOPL-1 for CORBA

User Evaluations

Eval2: **User evaluation on CORBA programming**

Task 1: Changing an existing method

Tasks Descriptions

The existing Grid sample which was introduced in the training shall be extended.

1a) The **set** method of class **grid_i** should be changed so that the additional **trace message**
"set method is called"
is displayed on the screen if the method is called by a client application.

(max. 4 minutes)

1b) After testing the code the problem was found that two-dimensional arrays may be created with dimensions [0][1] or [1][1]. It was decided that only arrays with **dimensions [n][n] (n>= 2)** are acceptable. Please change the code so that this requirement is fulfilled.

(max. 6 minutes)

Questionnaire : CORBA programming

Q1: Are you confident that you did solve the problem? **A1**: yes ☐ no ☐

Q2:: How do you think have you completed the tasks?
1 (very good) 2 3 4 5 6 (poor) **A2:** _____

Q3: What general **experiences** did you encounter while trying to write the code?
A3:

_____

_____

_____

Q4: What have been the major problems for you? Please state three problems if possible.

A4: 1 _____

2 _____

3 _____

User Evaluations

Eval2: **User evaluation on CORBA programming**

Task 2: Definition of a new method

Tasks Descriptions

The existing Grid sample which was introduced in the training shall be extended.

2a) A new interface method **print** should be added to the interface class. No arguments are needed in order to call the method. No value is to be returned to the calling method. The **initial behaviour** of the method shall be that the message "print method is called" is displayed on the screen.

(max. 10 minutes)

2b) The new behaviour for the method **print** should now be coded. The behaviour of the method shall be as follows:

The first 5 elements of the array should be printed out on the screen.

(max. 10 minutes)

Questionnaire : CORBA programming

Q1: Are you confident that you did solve the problem? **A1**: yes ☐ no ☐

Q2:: How do you think have you completed the tasks?
1 (very good) 2 3 4 5 6 (poor) **A2:** _____

Q3: What general **experiences** did you encounter while trying to write the code?
A3:

_____

_____

_____

Q4: What have been the major problems for you? Please state three problems if possible.

A4: 1 _____

2 _____

3 _____

User Evaluations

Eval2: **User evaluation on CORBA programming**

Task 3: definition of a CORBA server mainline

Tasks Descriptions

The existing Grid sample which was developed in the previous trials is to be extended.

3) Before a client application may use a CORBA server object it has to be created by a so-called CORBA server. The final task is to write the necessary mainline code which instantiates the server object.

   The following code is required:
   - create the **grid object** (1st dimension 10, 2nd dimension 20)
   - "inform" Orbix that the object has been instantiated and **parameterise** the **server information** (i.e. name of server and timeout time)
   - if an error occurs the standard **system exception** handling should be implemented

(max. 10 minutes)

Questionnaire : CORBA programming

Q1: Are you confident that you did solve the problem? **A1**: yes ☐ no ☐

Q2:: How do you think have you completed the tasks?
1 (very good) 2 3 4 5 6 (poor) **A2:** _____

Q3: What general **experiences** did you encounter while trying to write the code?
A3:

_____

_____

_____

Q4: What have been the major problems for you? Please state three problems if possible.

A4: 1 _____

2 _____

3 _____

Feedback Questionnaire: Catalin

Q1: After conducting the trials are you motivated to use VOODE/VOOPL-1?
A1: yes ■ no □

Q2:: Once trained, how would you think is VOODE/VOOPL-1 used?
A2:

Would use it for server-side development, client-side point of view not in a visual way since this is better controllable by me (partly usage).
General statement: Visual interfaces tend to slow down advanced users.

Q3: Do you think that it has been a reasonable evaluation?
A3: yes ■ no □

Q4: What should be improved, if any?
A4:

Code may also be optionally entered in a text-mode - client and server code!

Node appearance should be made better : identification of slots/possible connections.

Feedback Questionnaire: Catalin

Evaluator comments
This evaluation indicates that there should be a mixed edit mode.

There are a lot of problems with environments - training proved to lead to much better evaluation results.

There are many problems finding the right code processor.

Feedback Questionnaire: Neil

Q1: After conducting the trials are you motivated to use VOODE/VOOPL-1?
A1: yes ■ no □

Q2:: Once trained, how would you think is VOODE/VOOPL-1 used?
A2:

I would use VOODE/VOOPL-1 for complex tasks. For simple tasks I would prefer keypad-typing, especially when I remember the code&syntax. I would also see a migration to smaller tasks if I am more used to it. Of course, this depends on time constraints.

Q3: Do you think that it has been a reasonable evaluation?
A3: yes ■ no □

Q4: What should be improved, if any?
A4:

It has been a fair evaluation.

Feedback Questionnaire: Neil

Evaluator comments

This evaluation indicates that there should be a mixed mode like in Together/J (or other systems...) where a user may work in the visual environment as well as in the textual environment in parallel (whatever suits best when conducting a task). This could be achieved within VOODE/VOOPL-1 by various means, e.g. an additional text window when selecting a processor, or an associated text window in the property dialog.

There are a lot of problems with environments - training is also needed for Rose etc. No environment was intuitive to the subjects.

There are many problems finding the right place to add the code. This also indicated a better integration with Rose. Only one “mental model” for the solution is needed.

Feedback Questionnaire: Paul Reeves

Q1: After conducting the trials are you motivated to use VOODE/VOOPL-1?
A1: yes ☐?? no ☐

Q2:: Once trained, how would you think is VOODE/VOOPL-1 used?
A2:

Might use it if it helped to solve the tasks, concerns when doing lots of things .> screen efficiency

Q3: Do you think that it has been a reasonable evaluation?
A3: yes ☐xx second time tasks have been clearer how is it done the tasks no ☐

Q4: What should be improved, if any?
A4:

change order of tools ...

Feedback Questionnaire: <subject name>

Q1: After conducting the trials are you motivated to use VOODE/VOOPL-1?
A1: yes ☐?? no ☐

Q2:: Once trained, how would you think is VOODE/VOOPL-1 used?
A2:

Might use it if it helped to solve the tasks, concerns when doing lots of things .> screen efficiency_____

_____

_____

Q3: Do you think that it has been a reasonable evaluation?
A3: yes ☐xx second time tasks have been clearer how is it done the tasks no ☐

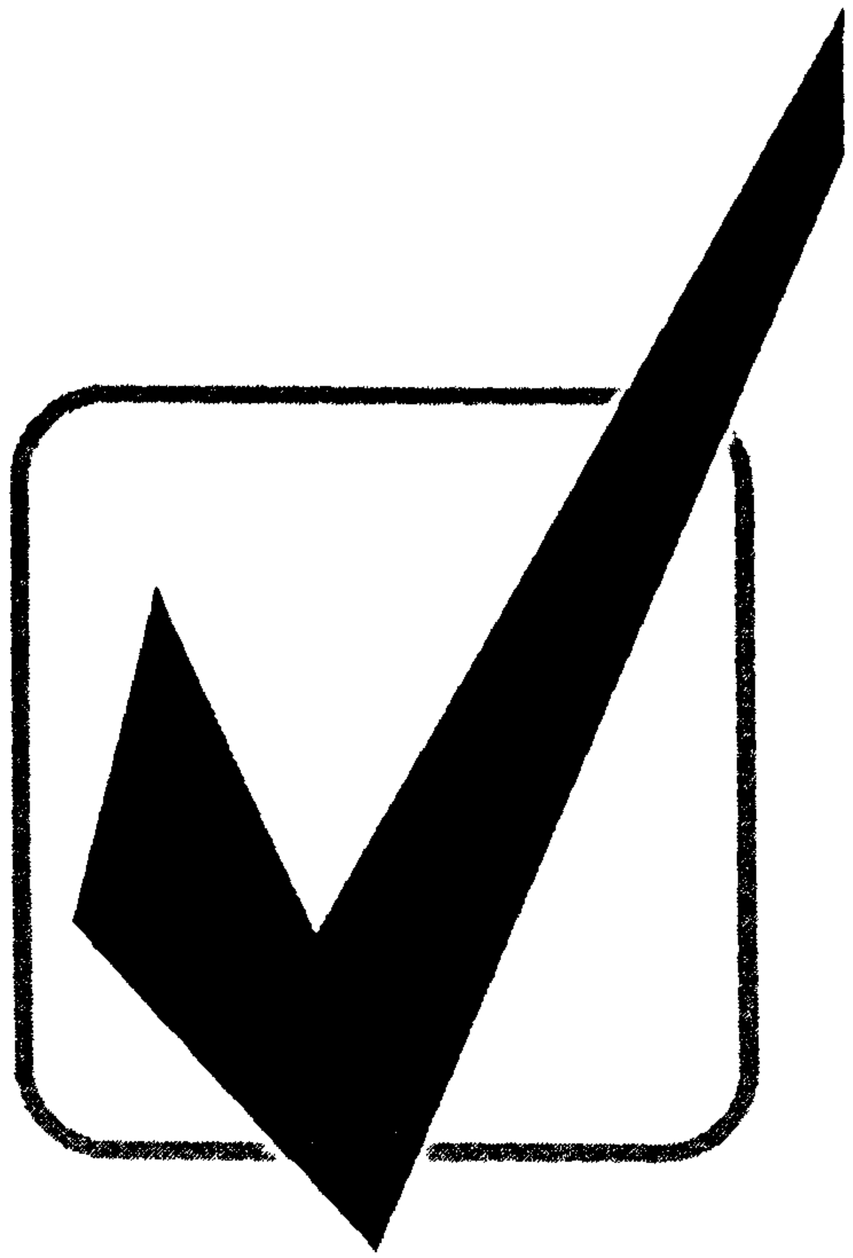
Q4: What should be improved, if any?
A4:

change order of tools ...

_____

_____

End of Evaluation 2



Thank you for your help!

Solutions

Visual Studio (or Explorer and Textpad)

Solution Task 1a

Overview of actions

1a) edit file **grid_i.cpp**

method : grid_i::set

- add print message (-> set method called) to method grid_i::set
- add #include "stdio.h" (otherwise printf is not compilable!)

```
#include "stdio.h"
// implementation of the set operation:
void grid_i::set(CORBA::Short n, CORBA::Short m, CORBA::Long value, CORBA::Environment &e)
#if defined(IT_RAISE_NAT_EH) && !defined(PCWORLD)
    throw (CORBA::SystemException)
#endif
{
printf("\n grid_i::set is called!\n");
print(n, m, e);
 m_a[n][m] = value;
}

    ...
```

Solution Task 1a

output is
on new window

Solution Task 1b

Overview of actions

1b) edit file **grid_i.cpp**
method grid_i::grid_i - add if statement

```
grid_i::grid_i(CORBA::Short h, CORBA::Short w) {
    // Check array dimension
    if ((h < 2) || (w < 2))
    {
       printf("\n wrong index!");
       return;
    } /* if */
    ...
```

Solution Task 2a

Overview of actions

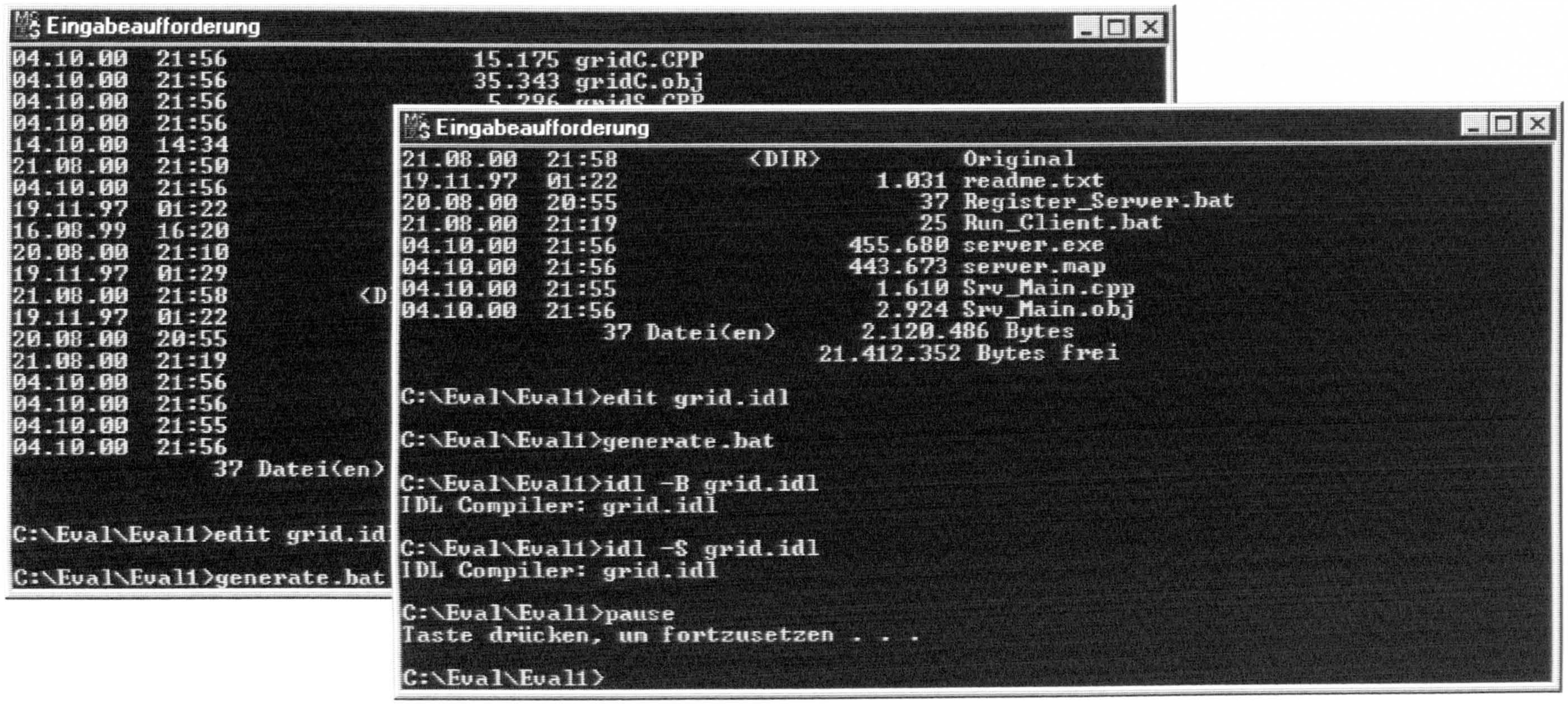
2a) edit file **grid.idl**
add **print** interface method (no specific IDL syntax check is needed)

```
// file : grid.idl
interface grid {
  readonly attribute short height;  // height of the grid
  readonly attribute short width;   // width of the grid

  // IDL operations

  // set the element [n,m] of the grid, to value:
  void set(in short n, in short m, in long value);

  // return element [n,m] of the grid:
  long get(in short n, in short m);

  // new method
  void print(in short n, in short m);
};
```

Solution Task 2a

Overview of actions

2a) execute **generate.bat**



Solution Task 2a

Overview of actions

2a) add new method in source file **grid_i.h**

```
virtual void print(CORBA::Short n, CORBA::Short m,
                           CORBA::Environment &env)
#if defined(IT_RAISE_NAT_EH) && !defined(PCWORLD)
    throw (CORBA::SystemException)
#endif
;
```

add missing code in source file **grid_i.cpp**

```
// implementation of the get operation:
void grid_i::print(CORBA::Short n, CORBA::Short m, CORBA::Environment &)
#if defined(IT_RAISE_NAT_EH) && !defined(PCWORLD)
    throw (CORBA::SystemException)
#endif
{
    printf("\n print method is called");
    return;
}
```

Solution Task 2b

Overview of actions

2b) add missing code in source file **grid_i.cpp**

```
// implementation of the get operation:
void grid_i::print(CORBA::Short n, CORBA::Short m, CORBA::Environment &)
#if defined(IT_RAISE_NAT_EH) && !defined(PCWORLD)
    throw (CORBA::SystemException)
#endif
{
int i;

  for (i = 0; i < 5; ++i)
    printf("\n %d", m_a[0][i]);

  return;
}
```

Solution Task 3

Overview of actions

3) add missing code in source file **Srv_Main.cpp**

```
#include <iostream.h>
#include <stdlib.h>
#include "grid_i.h"

int main() {
  // create a grid object - using the implementation class grid_i
  grid_i myGrid(10,20);

  try {
  // tell Orbix that we have completed the server's initialisation:
  CORBA::Orbix.impl_is_ready("grid");
  }
  catch (CORBA::SystemException &sysEx) {
    cerr << "Unexpected system exception" << endl;
    cerr << &sysEx;
    exit(1);
  } catch (...) {
    // an error occured calling impl_is_ready() - output the error.
    cout << "Unexpected exception" << endl;
    exit(1);
  }
// impl_is_ready() returns only when Orbix times-out an idle server
// (or an error occurs).
cout << "server exiting" << endl;
return 0;
}
```

Solutions

Rational Rose and VOODE/VOOPL-1 for CORBA

Solution Task 1a

Overview of actions

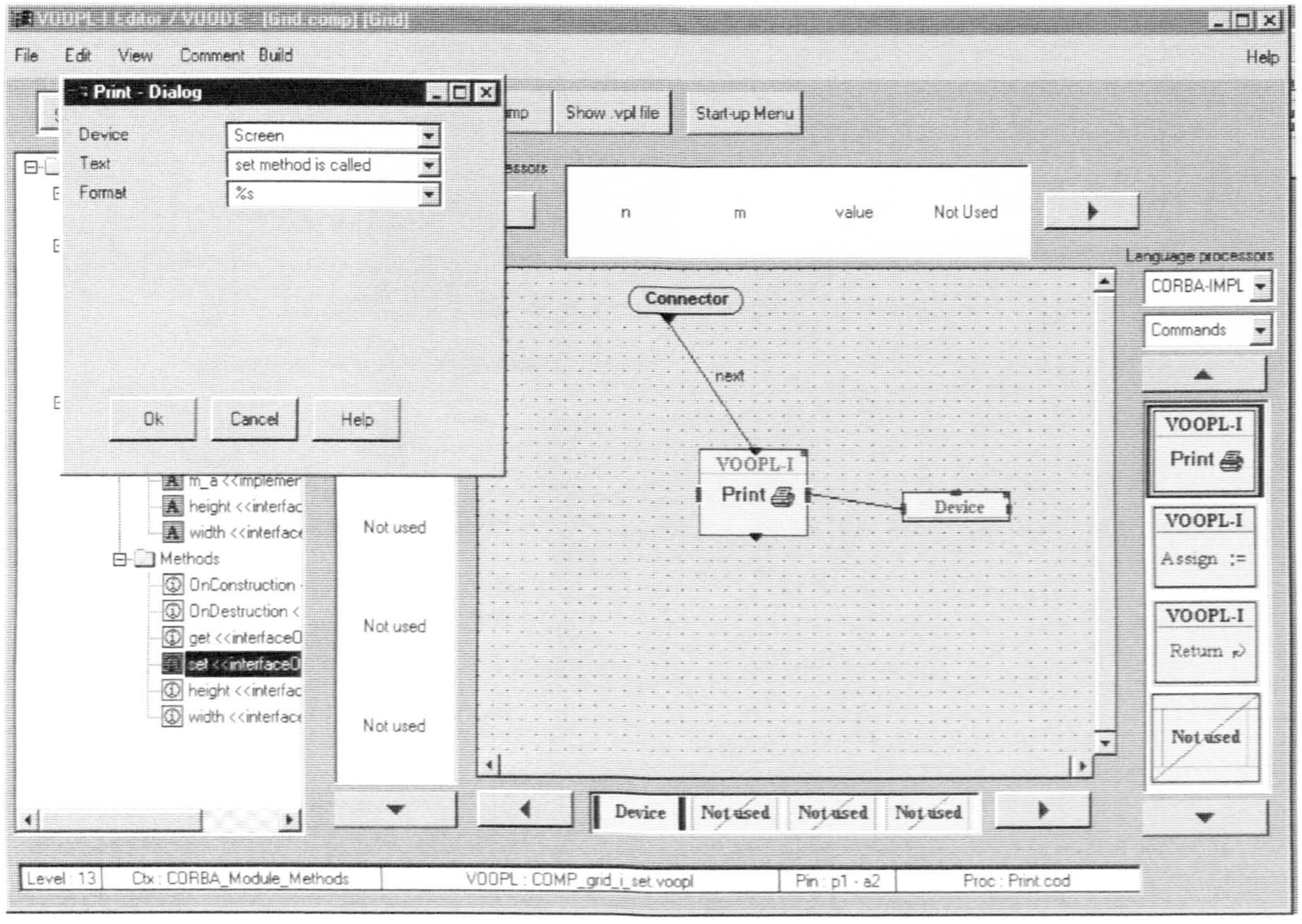
1a) edit file **grid.mdl (Rational Rose)** & start **VOODE/VOOPL-1 editor**
select method Component Model\CORBA Module\COMP_grid_i\Methods\set
- add print processor (CORBA-IMPL\Commands\Print)
- connect processors (Connector.0-Print.0)

alternative 1
- add device attribute processor (CORBA-IMPL\Commands\Print.Attribute(Device))
- connect processors (Print.0-Device.2)

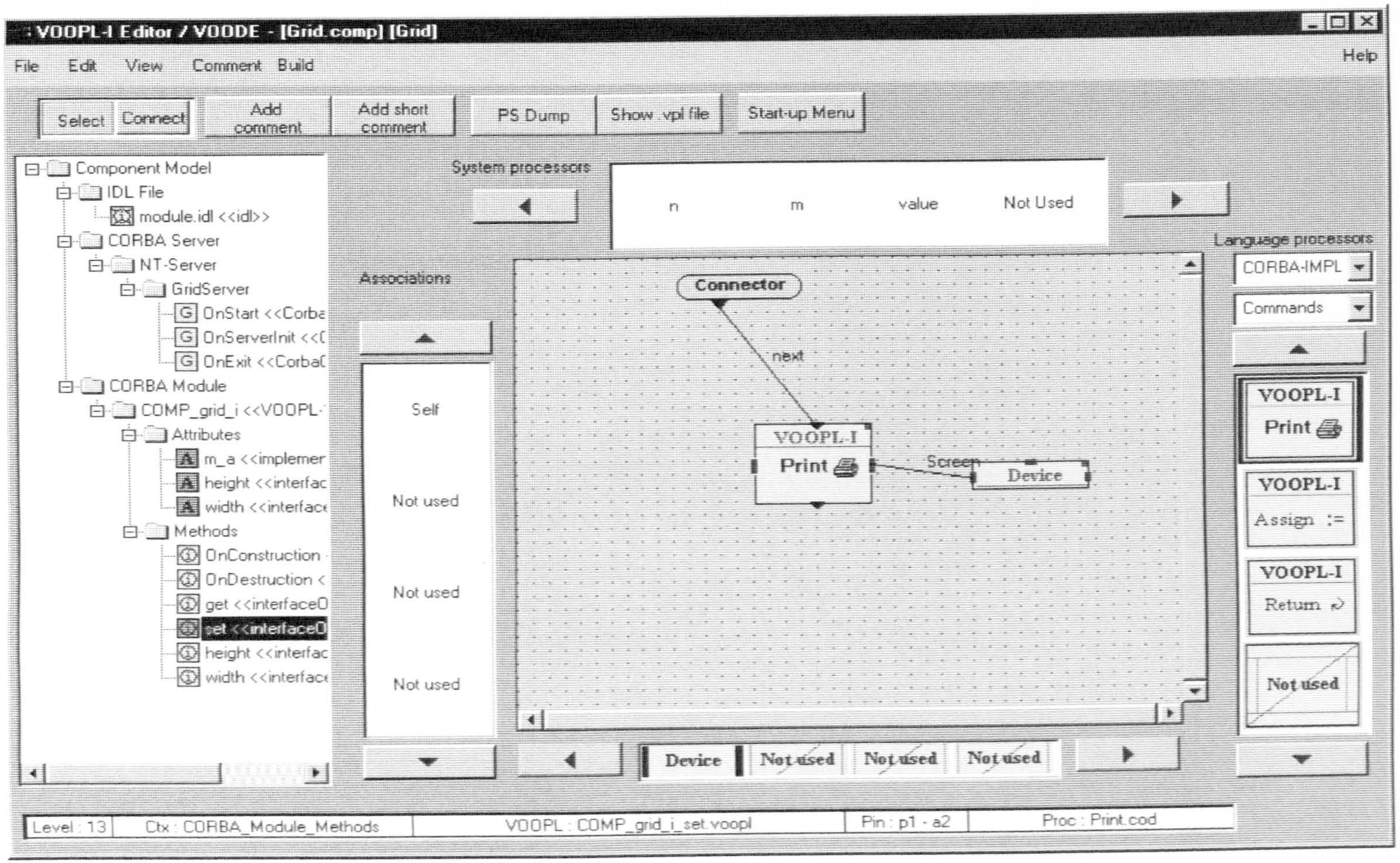
alternative 2
- open print processor dialog and select device attribute “screen”

open print processor dialog and enter text string “set method is called”

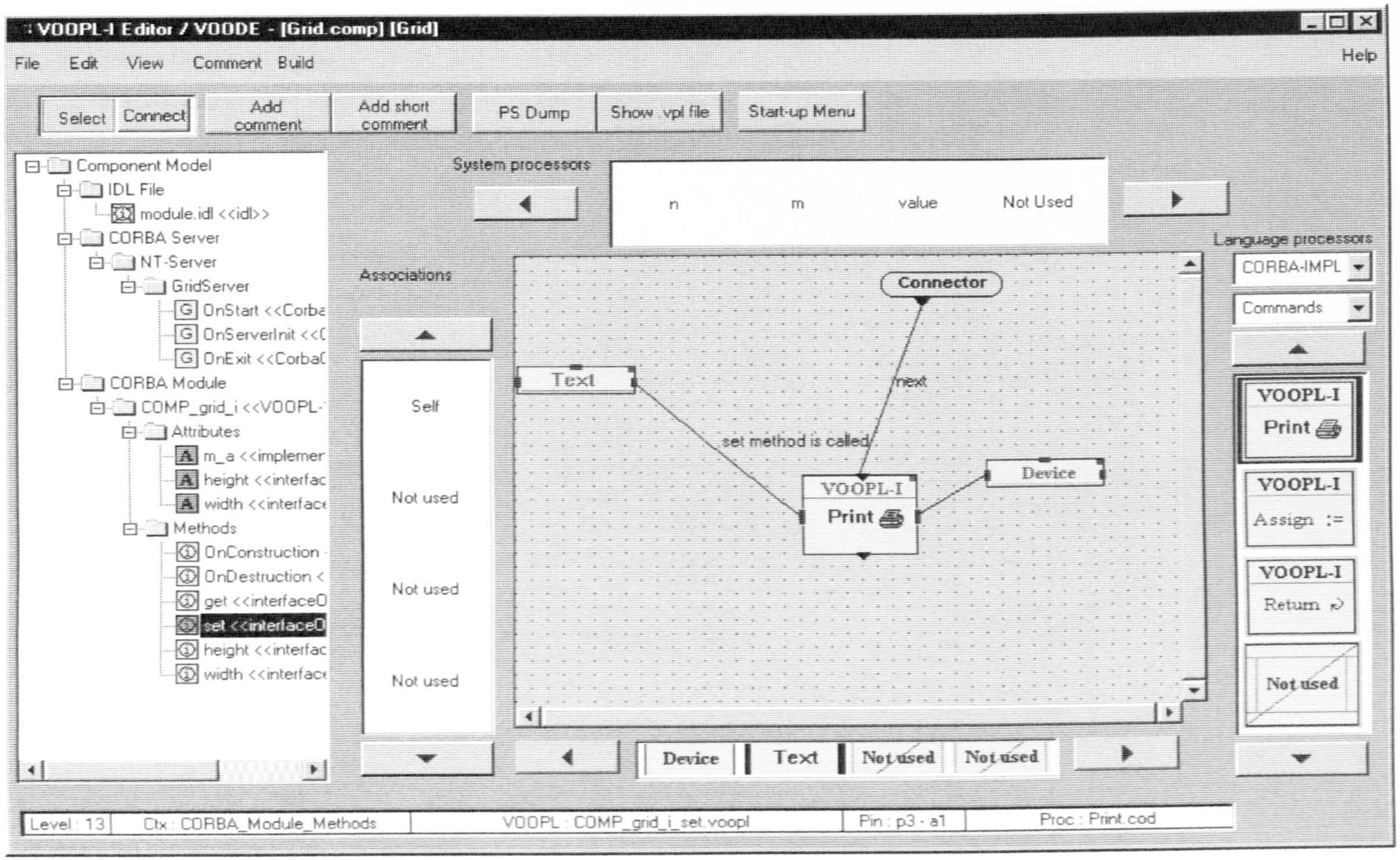
Solution Task 1a



Solution Task 1a



Solution Task 1a

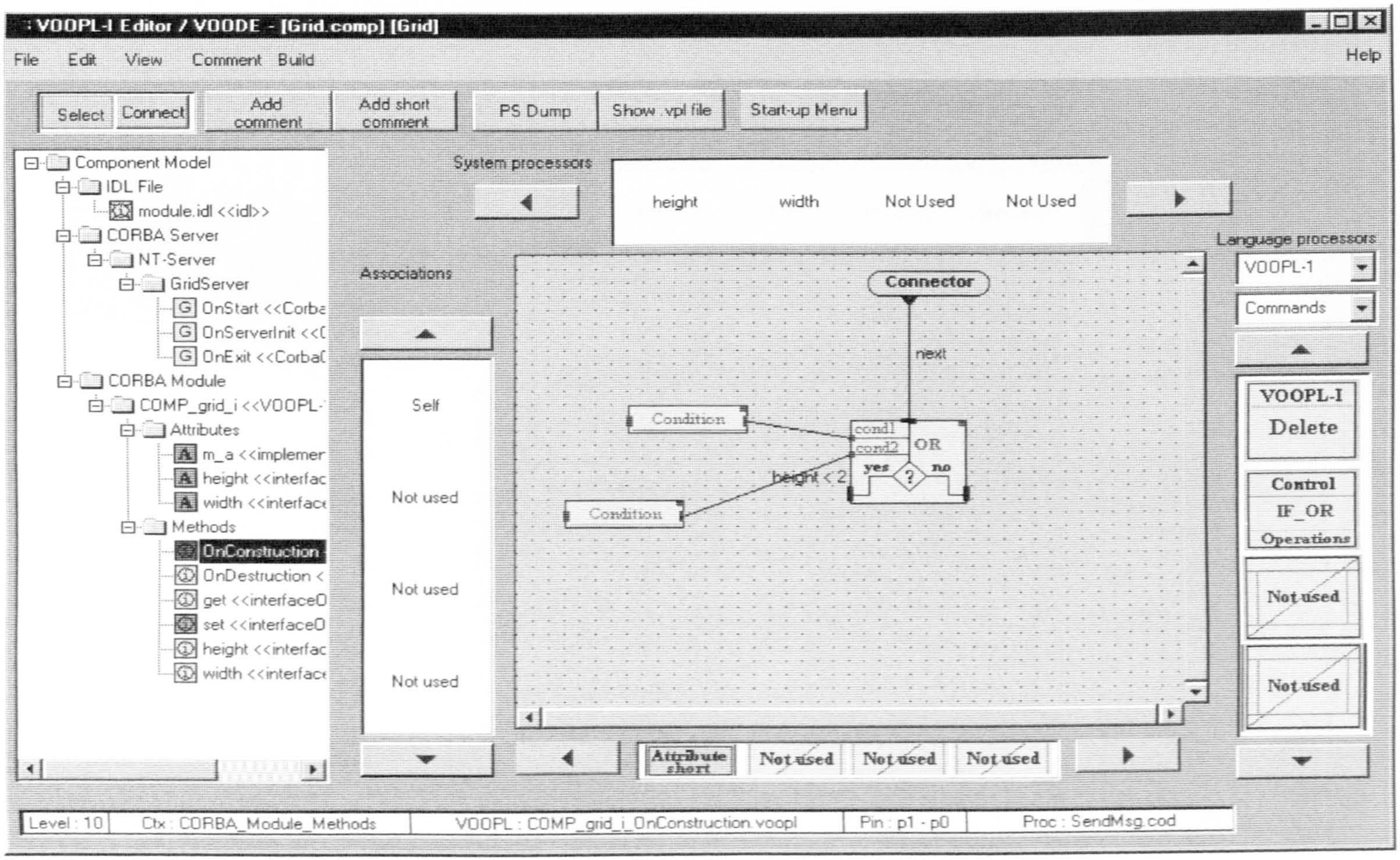


Solution Task 1b

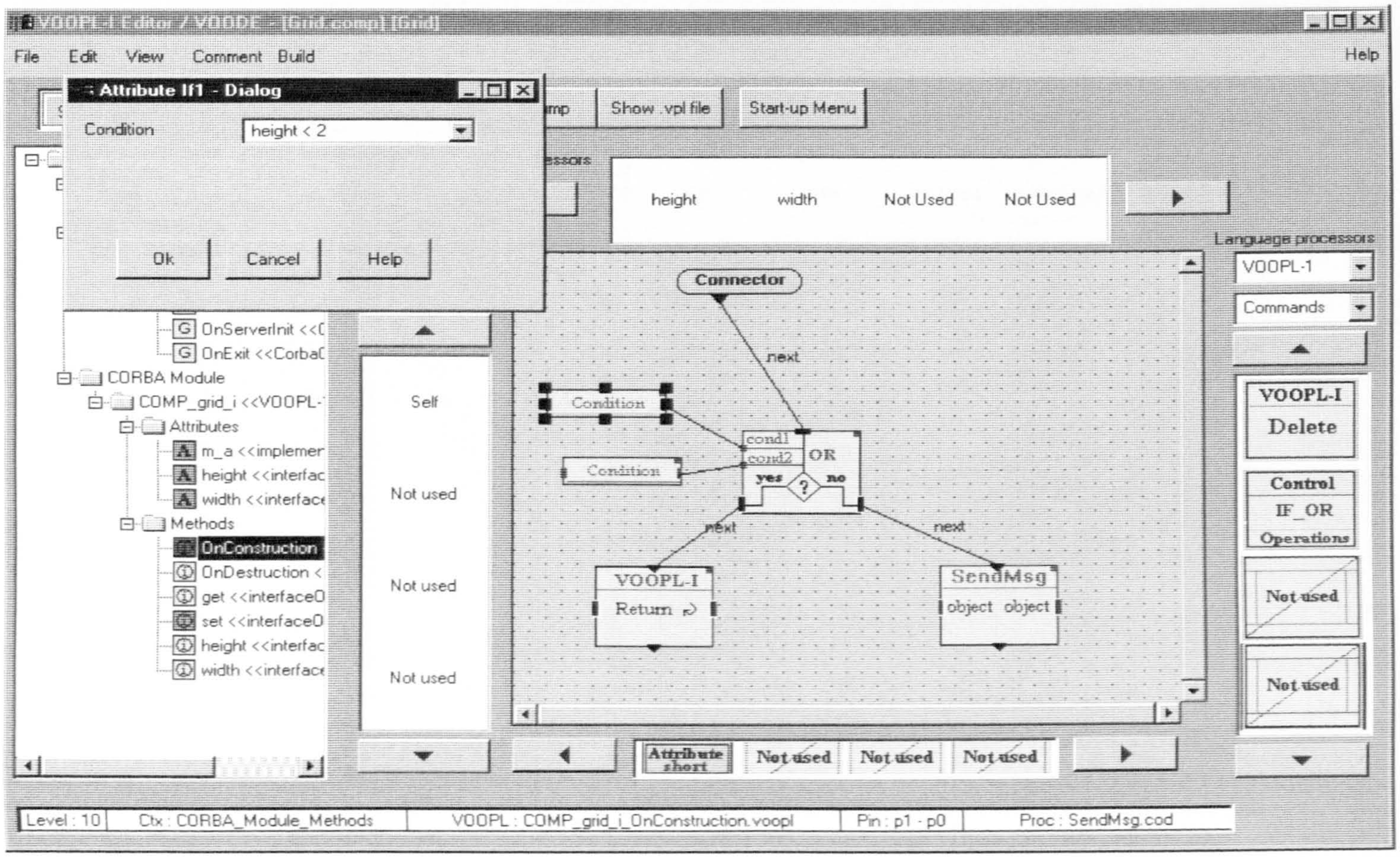
Overview of actions

1b) edit file **grid_i.cpp**
method grid_i::grid_i
- add if statement processor (VOOPL-1\ControlFlow\IF)

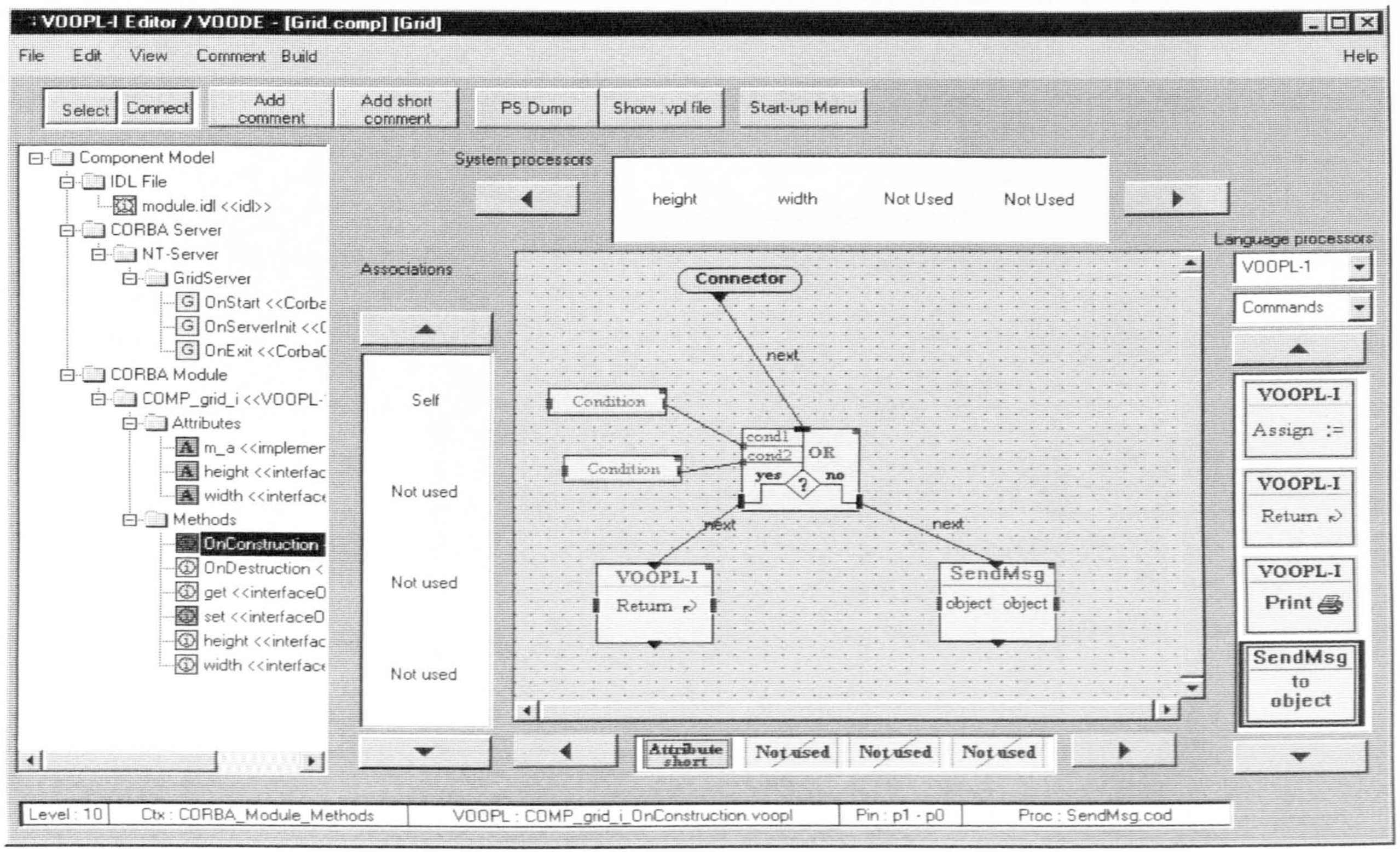
Solution Task 1b (alternative 1)



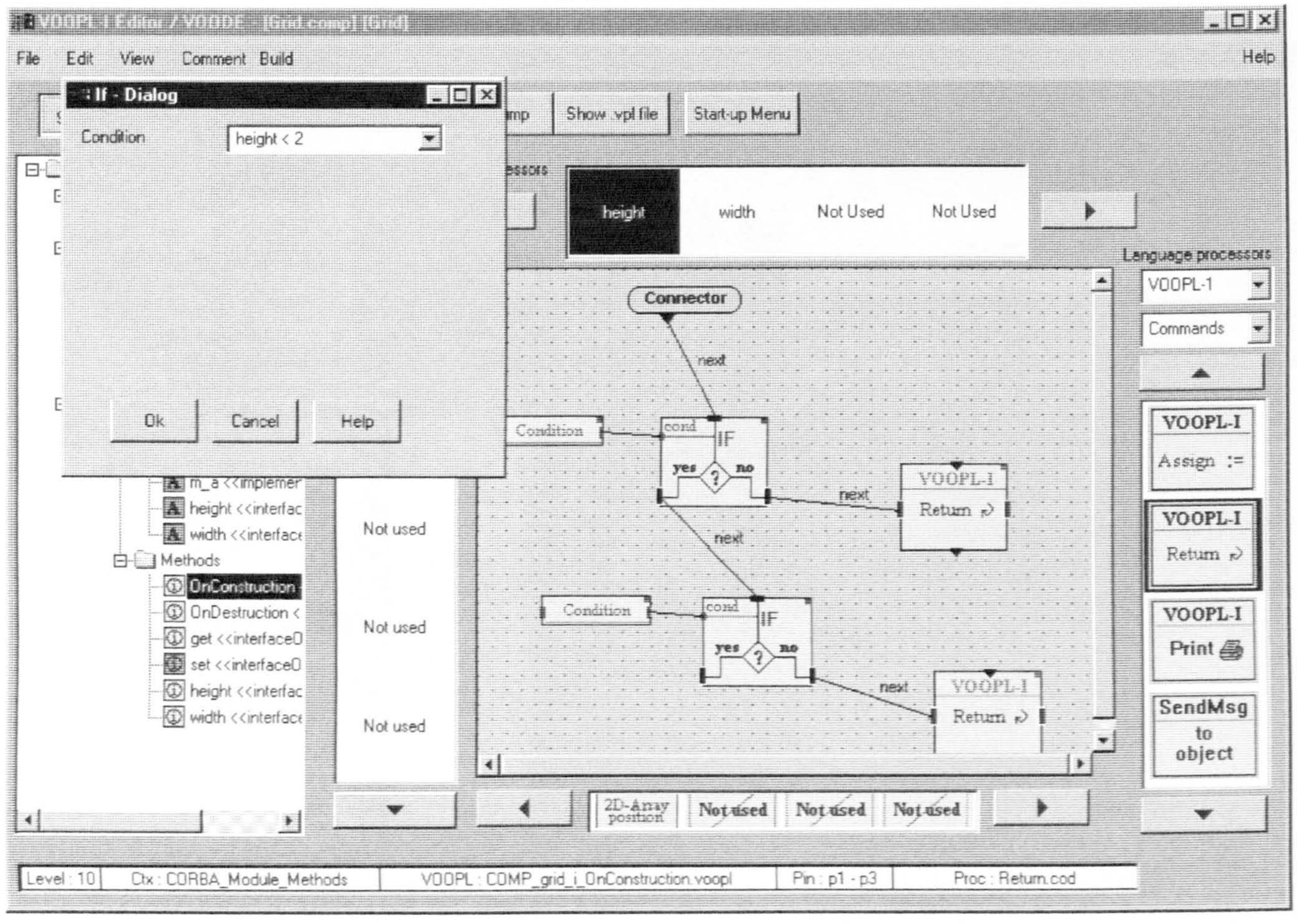
Solution Task 1b (alternative 1)



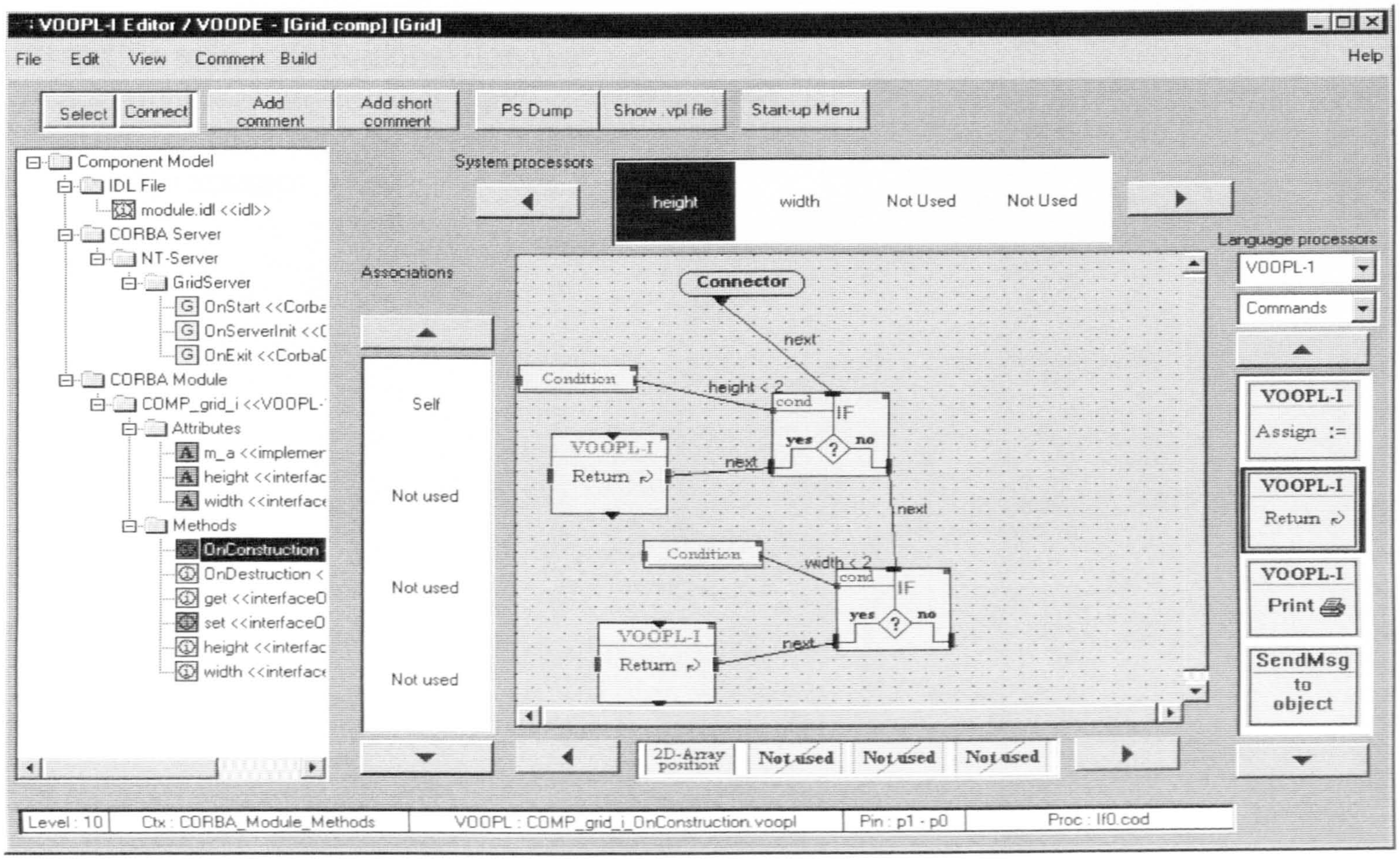
Solution Task 1b (alternative 1)



Solution Task 1b (alternative 2)



Solution Task 1b (alternative 2)
VOODE/VOOPL-1 for CORBA

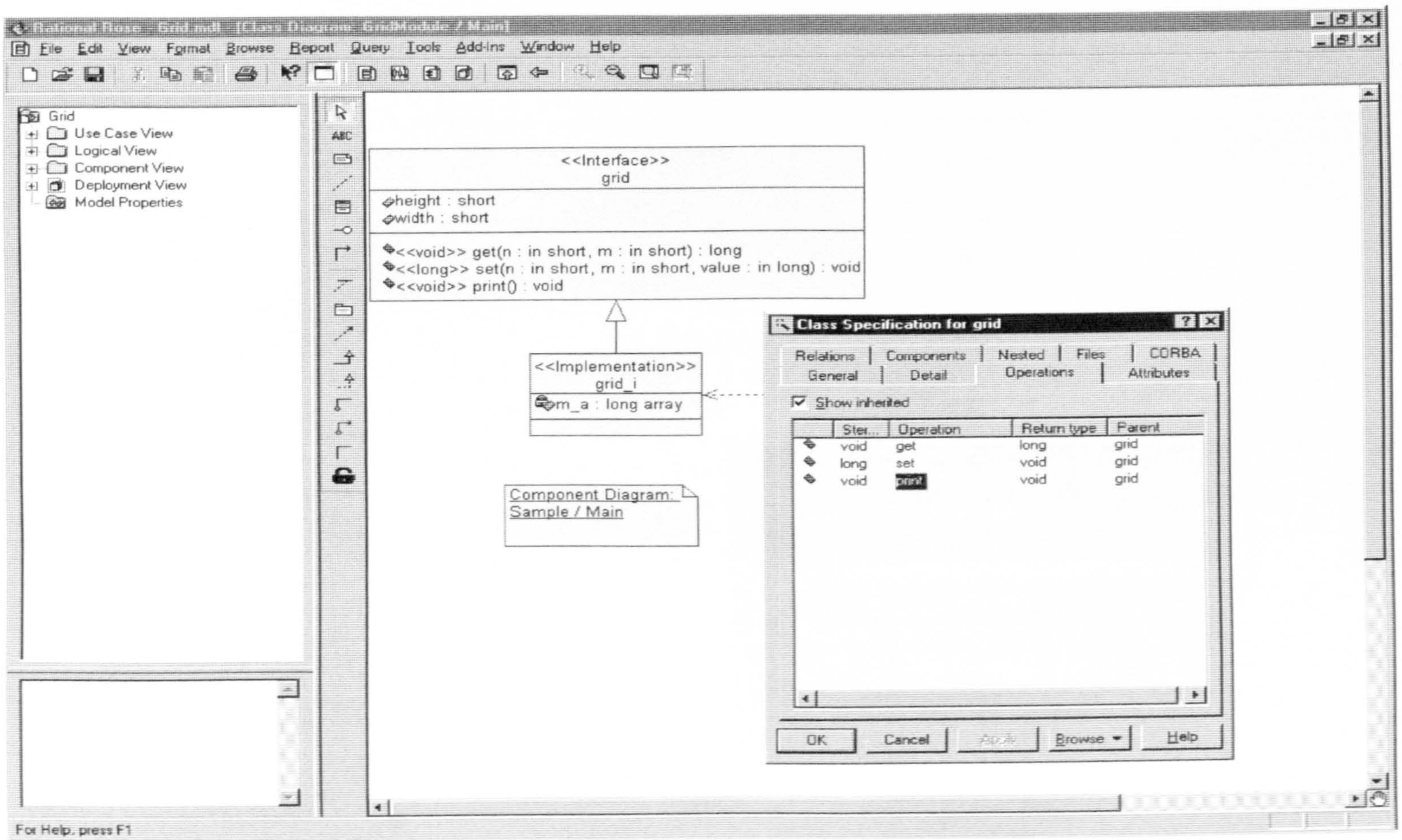


Solution Task 2a

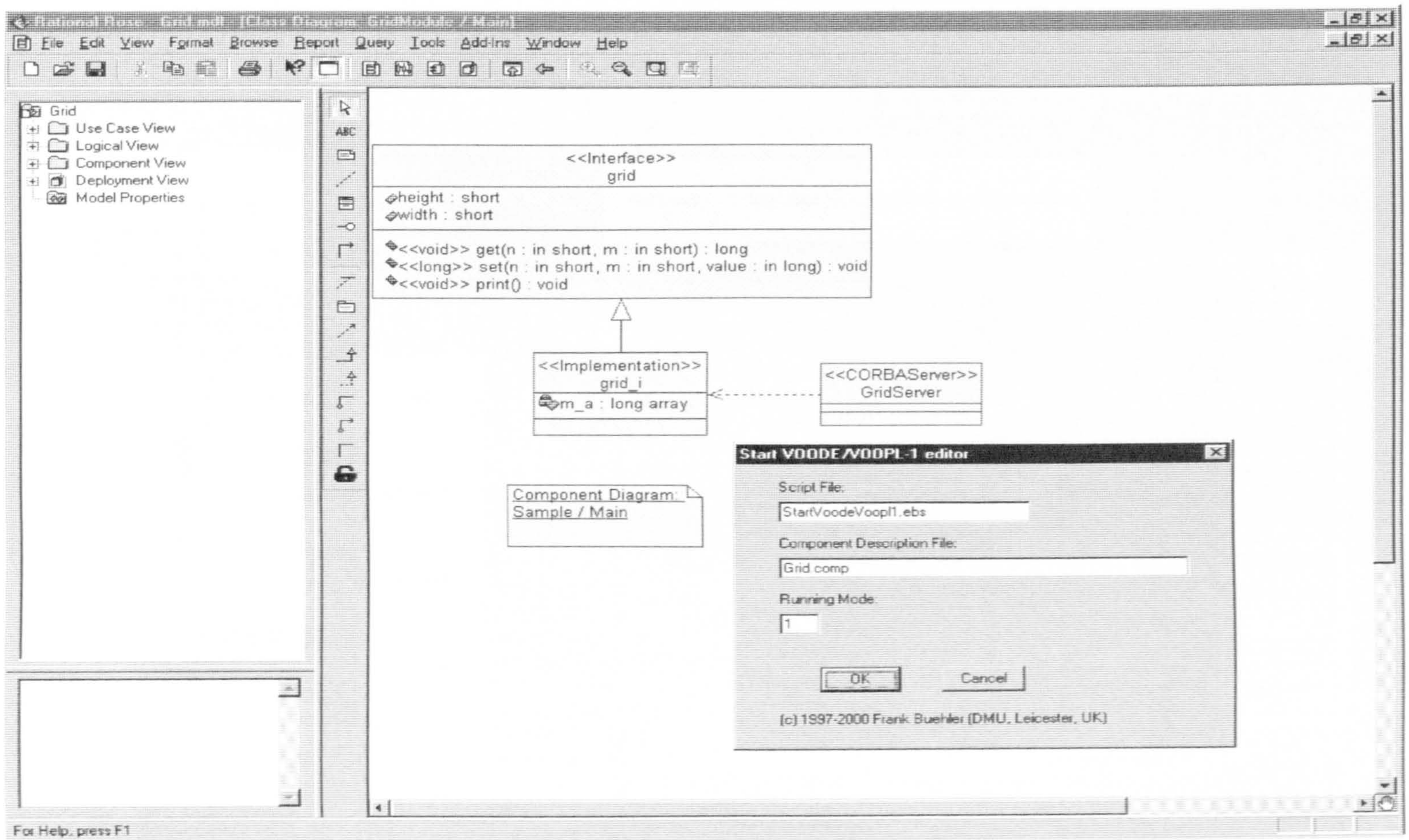
Overview of actions

2a) edit file **grid.mdl** (Rational Rose)
add print method to class "grid" (<<Interface>>)
execute "Tools\Voode Voopl-1 Editor"
add if processor

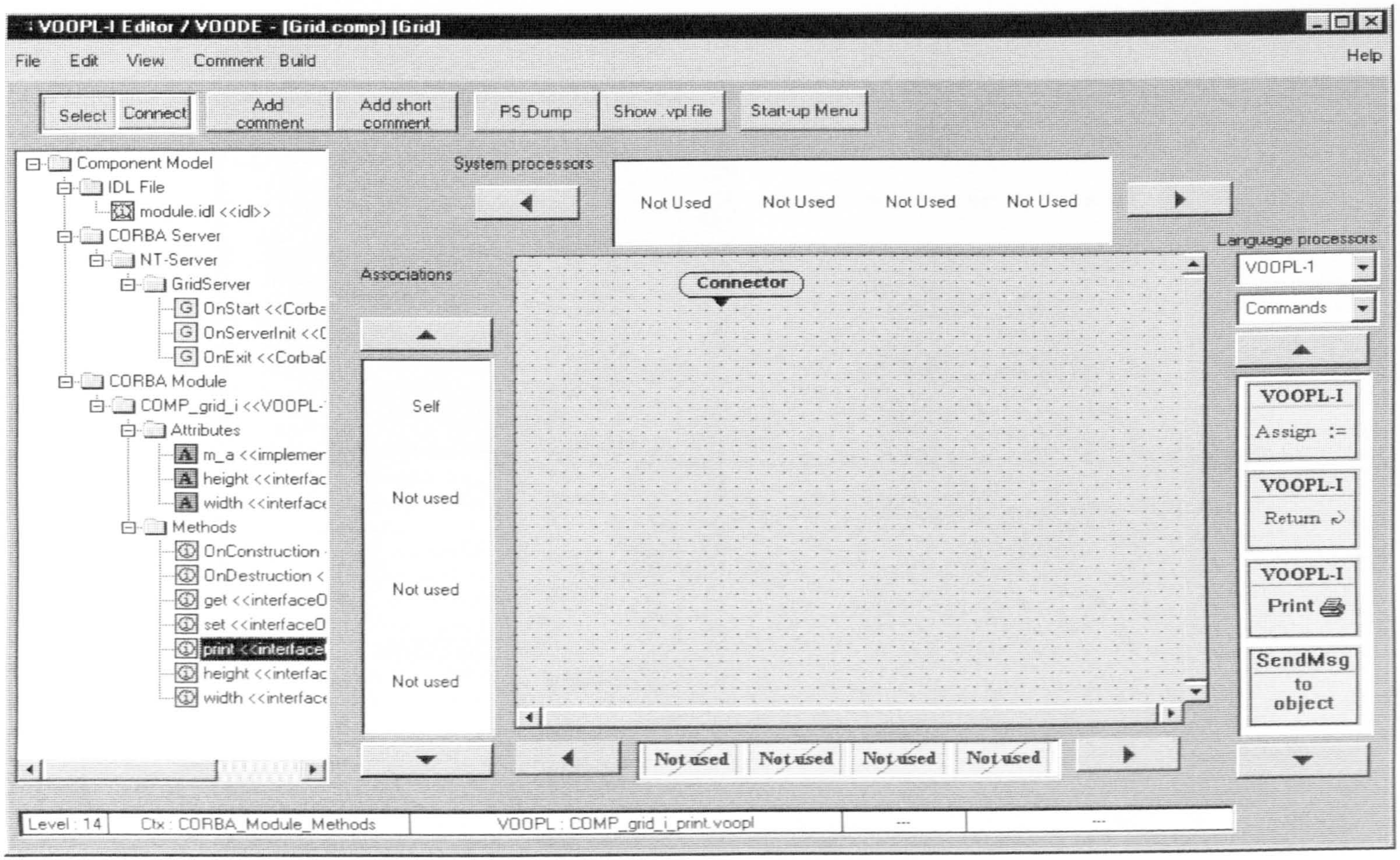
Solution Task 2a



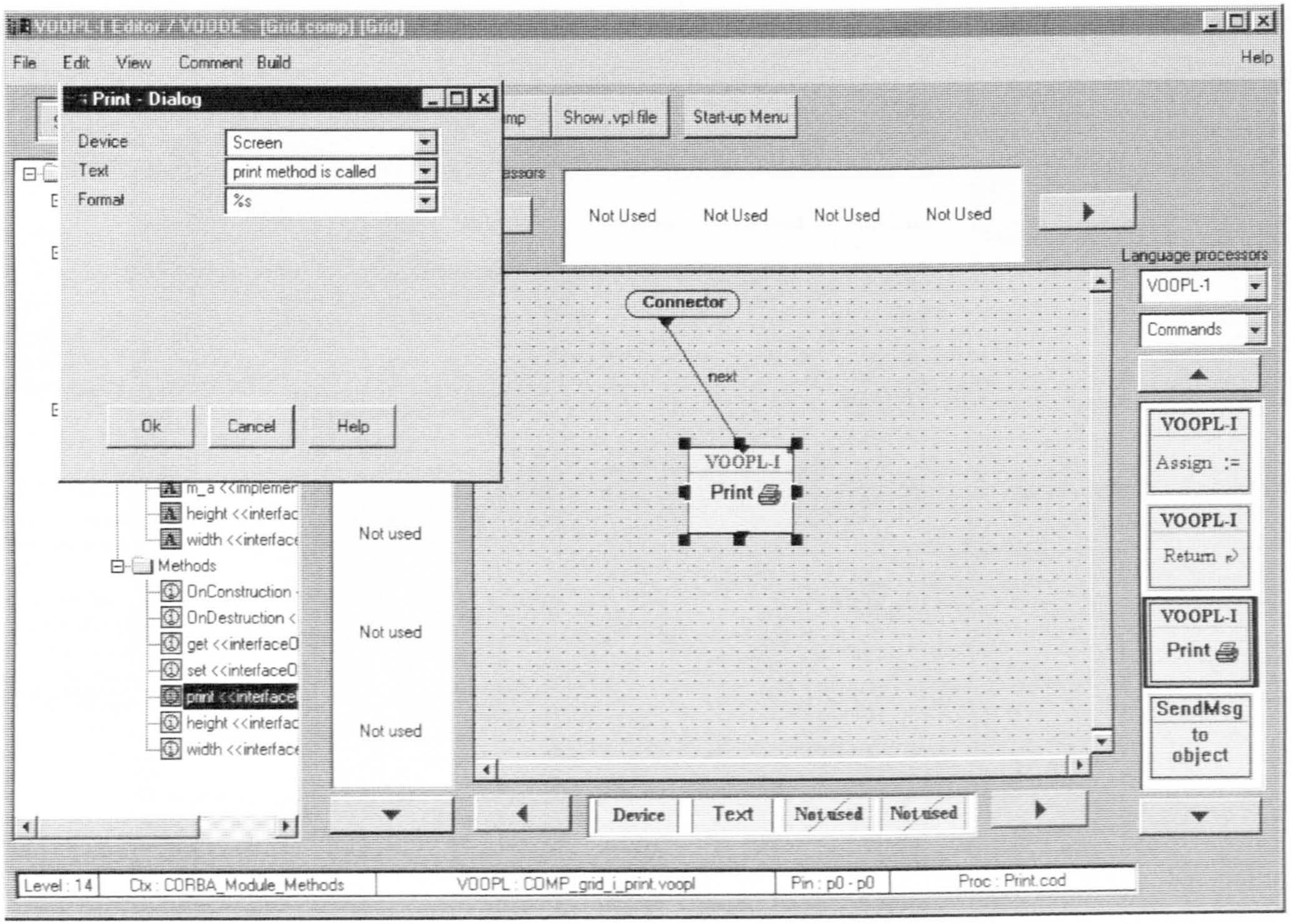
Solution Task 2a



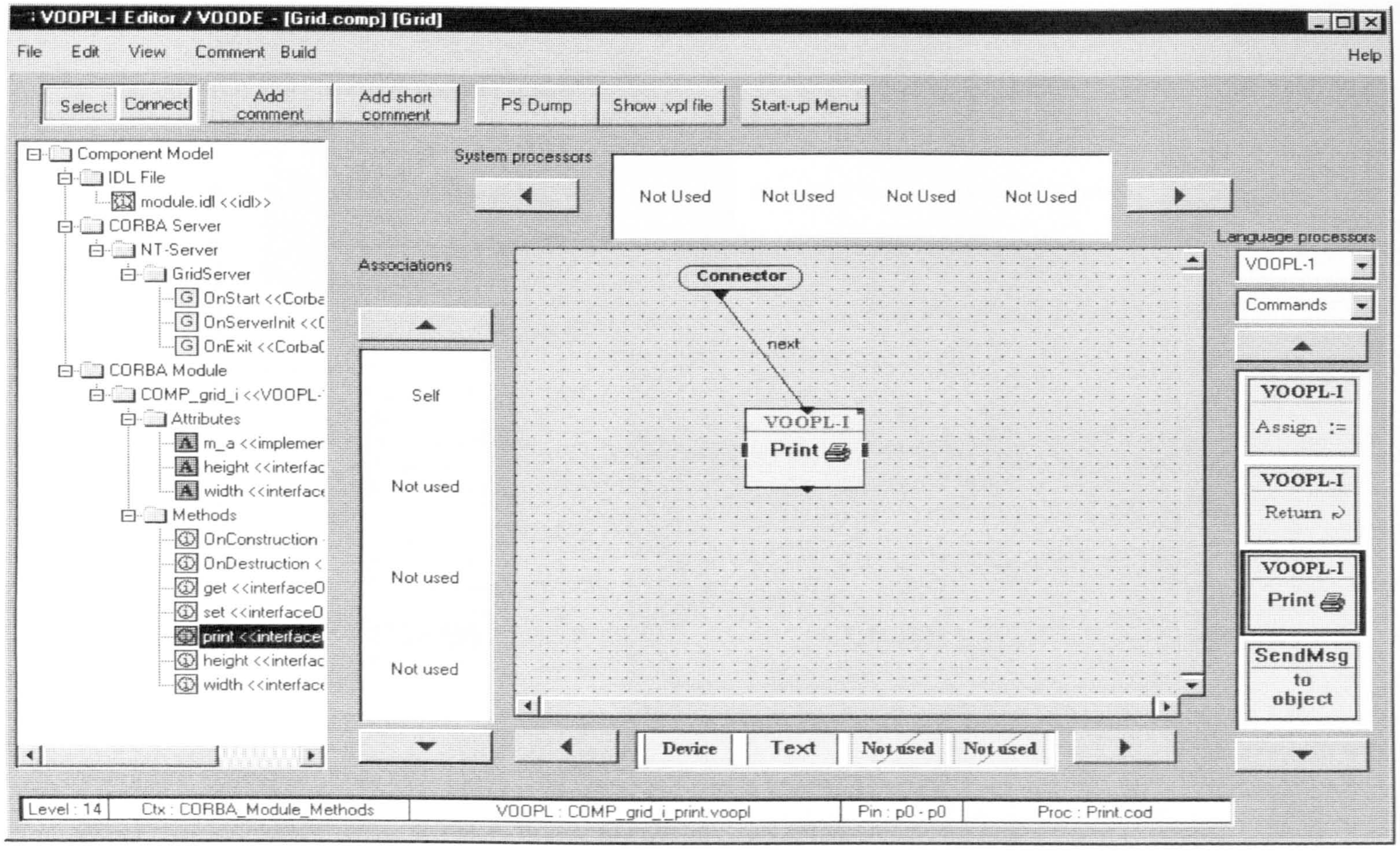
Solution Task 2a



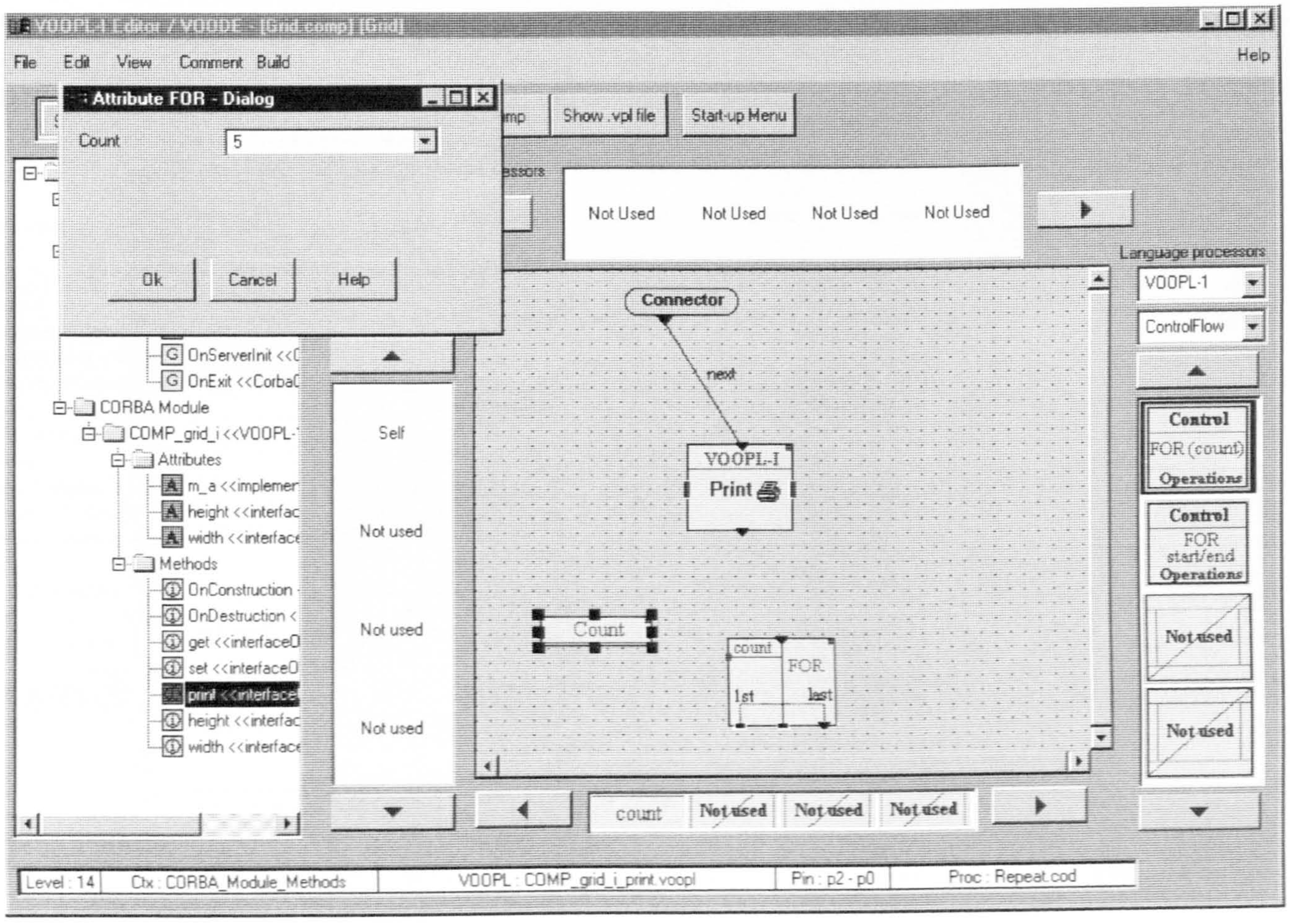
Solution Task 2a



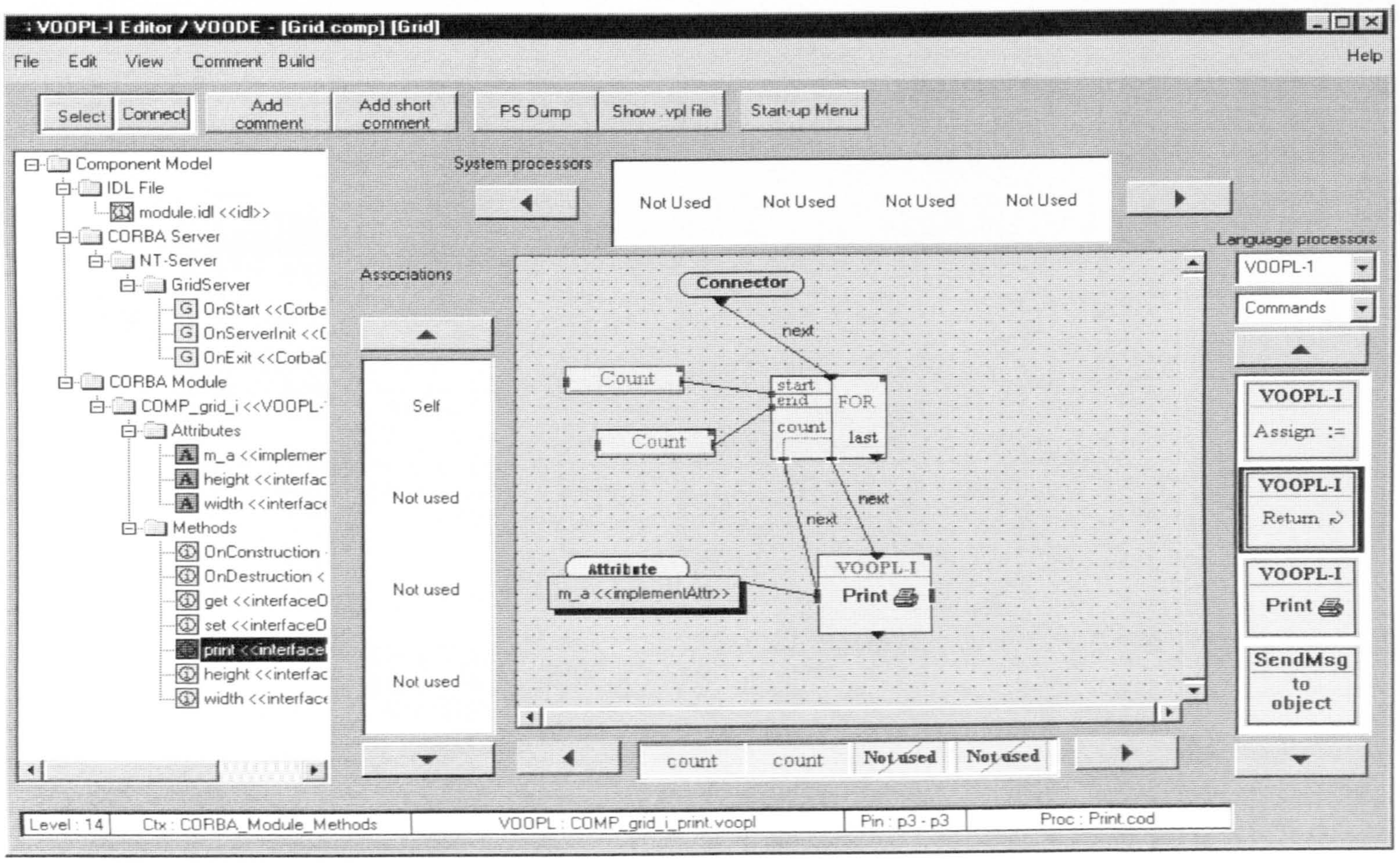
Solution Task 2a



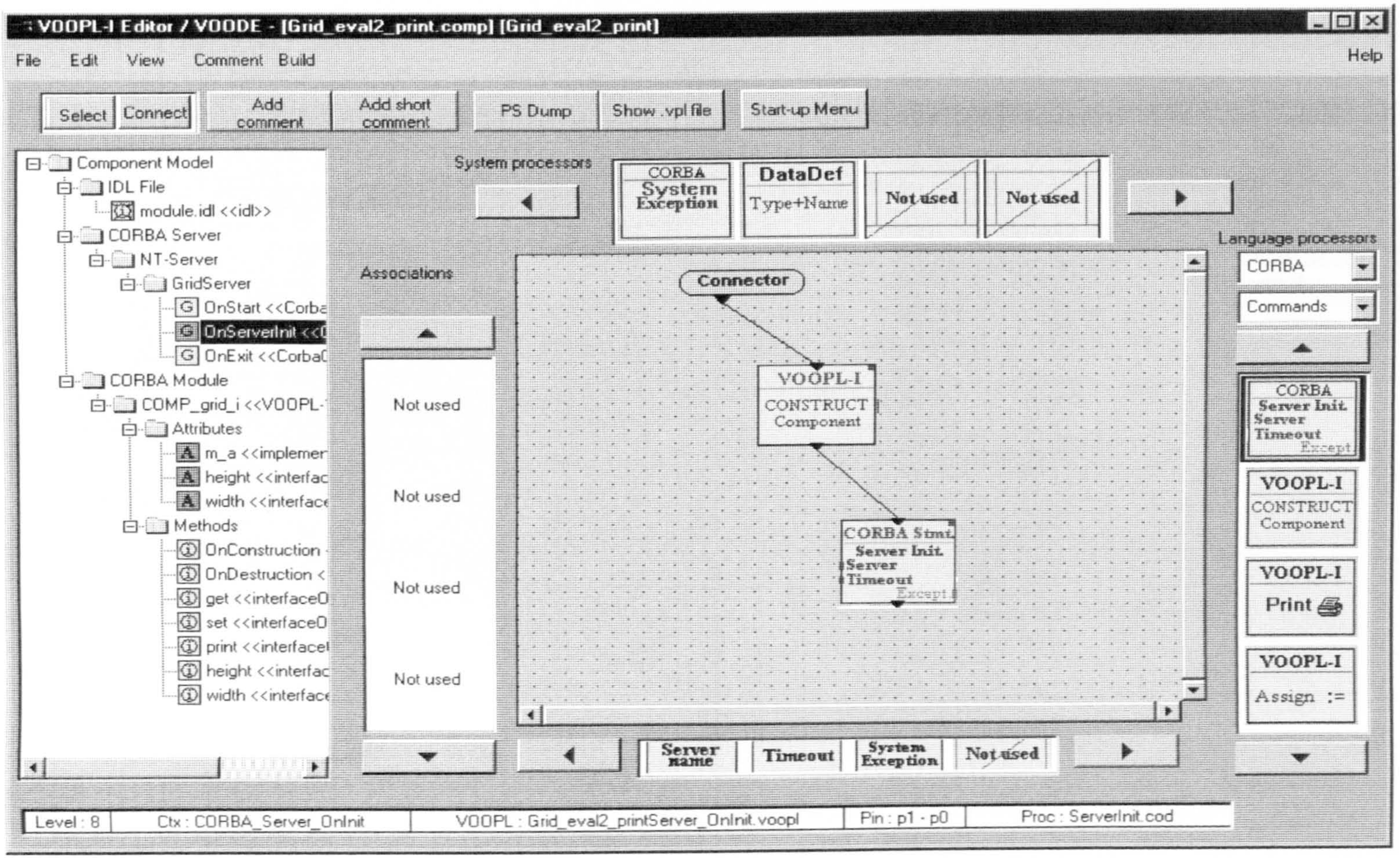
Solution Task 2b



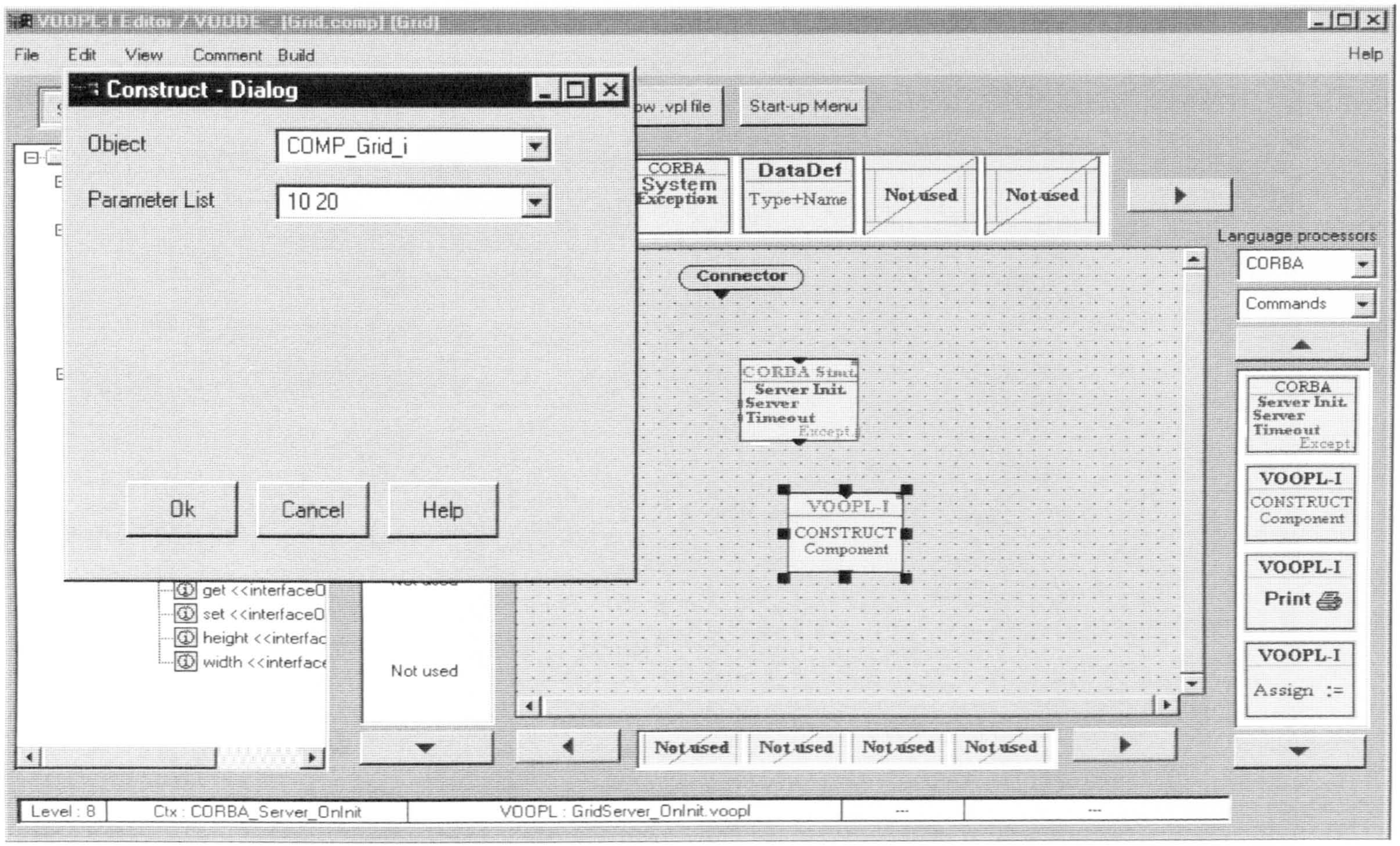
Solution Task 2b



Solution Task 3



Solution Task 3



Solution Task 3