Improving design processes through structured reflection : a prototype software tool

Improving Design Processes through Structured Reflection: A Prototype Software Tool

Isabelle M.M.J. Reymen
Elisabeth Melby

SAI Report 2001/2

October 2001
Eindhoven, The Netherlands

SAI Reports
is a series of research reports of the
Stan Ackermans Institute, Center for Technological Design at the
Technische Universiteit Eindhoven.

The reports present the ongoing research efforts and results of the research group of the Stan Ackermans Institute.

ISSN 1570-0143

Reports are available at: http://www.sai.tue.nl/research

Reports can be ordered from:
Technische Universiteit Eindhoven, Stan Ackermans Institute
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
Phone: +31(0)40 247 2452; Fax: +31(0)40 246 5342

Improving Design Processes through Structured Reflection: A Prototype Software Tool

Isabelle M.M.J. Reymen
Elisabeth Melby

Abstract

A prototype software tool facilitating the use of a design method supporting structured reflection on design processes is presented. The prototype, called Echo, has been developed to explore the benefits of using a software system to facilitate the use of the design method. Both the prototype software tool and the design method are developed as part of the Ph.D. project of Isabelle Reymen. The goal of the *design method* is supporting designers with reflection on design processes in a systematic way and regularly during the design process. The design method only supports the preparation step of a reflection process, a process that consists of the steps preparation, image forming, and conclusion drawing. The main concepts of the design method are the description and analysis of design situations and design activities and the idea of design sessions. The *prototype software tool* only supports the description and analysis of design situations.

The goal of this report is to offer design researchers insight into the concepts of the developed prototype, to offer designers some help when using the prototype, and to offer programmers information about the current implementation of the prototype for improving and/or extending the prototype. A description of the main concepts of the prototype, user documentation, and implementation documentation can be found in this report. The design method is explained in detail in (Chapter 5 of) the Ph.D. thesis of Isabelle Reymen, which is also published as SAI Report 2001/1.

Keywords

Design research ; tool support / design method ; domain independence / design process ; description

Contents

1 Introduction

In this report, a description of a prototype software tool facilitating the use of a design method supporting structured reflection on design processes is given. The prototype, called Echo, was developed to explore the benefits of using a software system to facilitate the use of the design method. Both the prototype software tool and the design method were developed as part of the Ph.D. project of Isabelle Reymen.

The goal of this report is to offer design researchers insight into the concepts of the developed prototype, to offer designers some help when using the prototype, and to offer programmers information on the current implementation of the prototype for improving and/or extending the prototype. The report starts in Chapter 1 with an introduction. In Chapter 2, it continues with a description of the general concepts of the prototype. Chapter 3 covers the user documentation of the prototype software tool. Chapter 4 documents the implementation of Echo. The report ends in Chapter 5 with conclusions.

The goal of the introduction is to give some background information to the reader. First, in Section 1.1, main concepts of the design method are explained. In Section 1.2, main concepts of the prototype software tool are discussed. Finally, in Section 1.3, the development process of the prototype is described.

1.1 Main concepts of the design method

The design method is explained in detail in Chapter 5 of the Ph.D. thesis of Isabelle Reymen [Reymen, 2001a], which is also published as SAI Report 2001/1 [Reymen, 2001b]; the main concepts of the design method are summarised below. The design method is a domain-independent aid that offers designers support for reflecting on design processes in a structured way. It is domain-independent because it is meant for supporting designers in several design disciplines. The domain-independent concepts on which the design method is based, described in a design philosophy and design frame, can be found in Chapters 3 and 4 of the Ph.D. thesis [Reymen, 2001a] and [Reymen, 2001b]. Together, the design philosophy and design frame offer concepts, a terminology, and a structure to describe design processes in a domain-independent way. In the design method, structured reflection on design processes is defined as the combination of systematic and regular reflection; the design method supports (partly) how to reflect and when to reflect. The design method is thus not a design method in a classical way: It does not explicitly guide a complete design process.

In [Reymen, 2001a] and [Reymen, 2001b], *reflection on a design process* is defined as an introspective contemplation on the designer's perception of the design situation and on the remembered design activities. A design situation is defined as the combination of the state of the product being designed, the design process, and the design context. A reflection process is described as a process that consists of three main activities that are called preparation, image forming, and conclusion drawing. For reflection on a design process, facts in the preparation step that must be collected and analysed critically are the properties, factors, and relations in the design situation and the design activities performed during the design process. The goal of the image-forming step is to get an image of the design process as a whole. During the conclusion-drawing step, the image of the design process and the goal of the design process are taken into account to determine the next design activities in the design process. The design method only supports the preparation step of a reflection process. Developing support for the image-forming and conclusion-drawing steps of a reflection process has been outside the scope of the Ph.D. project, but must be part of further research.

To support *structured reflection* on design processes, two main concepts have been developed. As a first concept of the design method, the *description and analysis of design situations and design activities* are supported. The descriptions are aimed to give an overview of the important facts; the

analysis of these descriptions forms the basis for reflection on the design process. Three checklists and three forms are developed to help designers with performing the preparation step in a systematic way. The second concept of the design method, namely the idea of *design sessions*, aims to support regular reflection. A design session is defined as a period of time during which one or more designers are working on a subtask of a certain design task. It is a period of time between two periods during which the designer(s) are not executing that subtask. A design session can take a period of time like one afternoon, a whole day, some days, or a week. Designers can themselves determine the duration of a design session. A design process can be seen as a sequence of design sessions. In the design method, Isabelle Reymen proposes reflection to take place at the beginning and end of a design session. By planning a design session, designers do thus automatically also plan moments for reflection. During a design session, designers can follow various other design methods.

Based on these two concepts, the design method proposes the following five steps: planning a design session, defining the (sub)task of the design session, reflecting at the beginning of a design session (using the developed forms and checklists), designing during the core of a design session, and reflecting at the end of a design session (again using the developed forms and checklists). Summarising, the design method stimulates designers to reflect on the current design situation and on performed design activities in a systematic way and on a regular basis, with the goal to derive possible design activities for the future. Figure 1.1 illustrates the kind of reflection the design method supports.

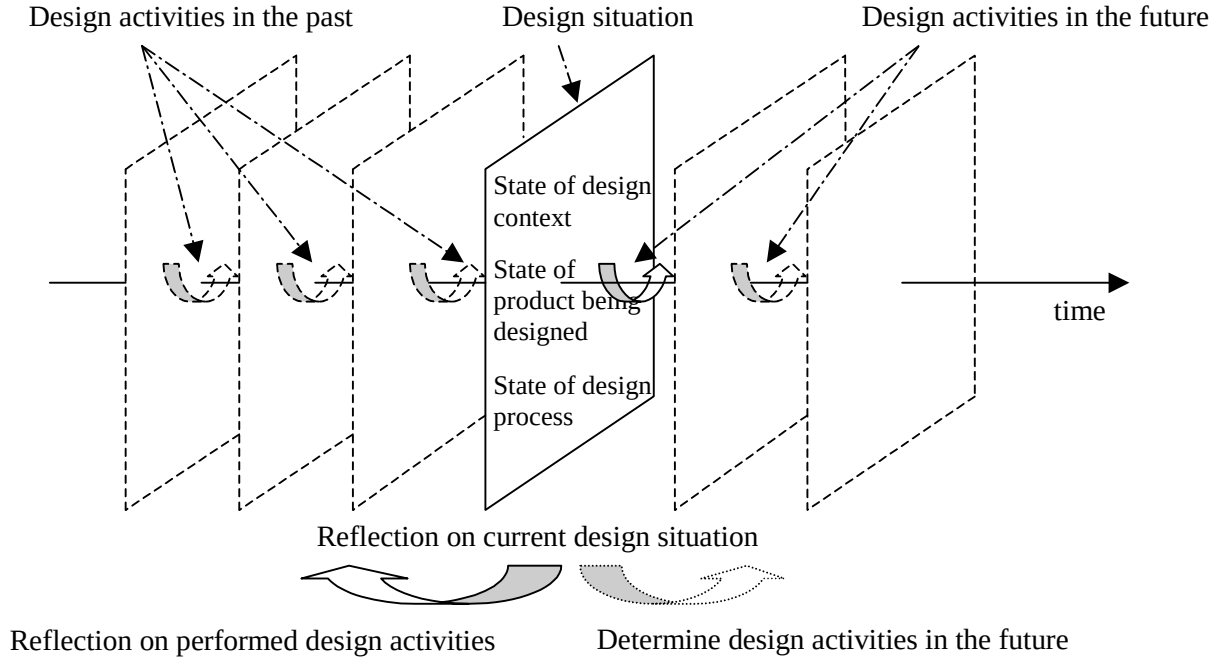


Figure 1.1: Reflection on a design process.

1.2 Main concepts of the prototype software tool

Echo has been developed to explore the benefits of using a software system to facilitate the use of the design method. A software system can be good for storing and managing information and in offering overview on complex data. A software system might thus be helpful for the preparation step of a reflection process, namely for describing and analysing design situations and design activities. In the current prototype, the description and analysis of design situations is supported.

In the design method, the description and analysis of design situations is supported by FORMS Properties&Factors and Relations and CHECKLISTS Description Design Situation and Analysis Design Situation. FORM Properties&Factors can be used to list properties and factors and to describe them more precisely; CHECKLIST Description Design Situation can help to inventory the most

important properties and factors; FORM Relations allows to describe relations; CHECKLIST Analysis Design Situation contains questions for a critical analysis of the design situation.

In the current version of Echo, (part of) FORM Properties&Factors and CHECKLIST Description Design Situation are combined in the concept of a '*tree*'. *Electronic forms* have been developed to make a precise description of properties, factors, and relations; they replace FORMS Properties&Factors and Relations. Several *queries* and some *checks* support the analysis of design situations (part of CHECKLIST Analysis Design Situation). The prototype is domain independent, but the concept of a *template* makes it possible to tailor the use of the design method to the terminology and concepts of a specific discipline. Templates makes it easy for a user to change the terminology (attributes) of the forms and (the topics of) the questions in CHECKLIST Description Design Situation. Each of these concepts is explained in more detail in Chapter 2 of this report.

Echo can be used during the third and fifth step of the design method for describing and analysing a design situation. To *describe a design situation*, a user can add properties and factors in the tree, describe properties, factors, and relations precisely via the electronic forms, and change values for the different attributes in the electronic forms. To *analyse a design situation*, a designer can query information and check design relations. The prototype consists of a database for storing properties, factors, and relations, a database-management system, and a user interface, as illustrated in Figure 1.2. The prototype is implemented in Java Swing. Because Java is platform-independent, the application can be used on PC/Windows and Sun/Unix platforms. The software is still a prototype; if it is decided that a new version will be built, it is going to be implemented from scratch.

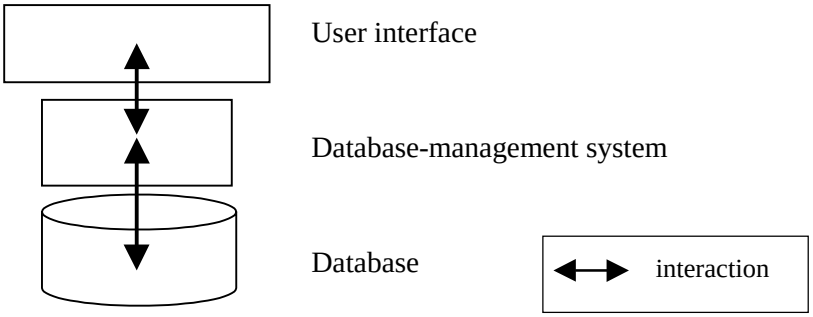


Figure 1.2: Basic structure of the prototype.

1.3 Development process of the prototype software tool

This section describes the design process and implementation process of the prototype. Echo has been developed in parallel with the design method. Although the design method evolved[1] after the development of the prototype stopped (the prototype is thus based on a previous version of the design method), the current concepts of the design method for reflection on a design situation can still be recognised in Echo.

The complete development process took one and a half year. The *design process* of the prototype software tool started at the beginning of the third year of the Ph.D. project of Isabelle Reymen (September 1998). At the beginning of the design process, the goal of the design process was not yet clear: "develop a tool using the developed domain–independent descriptive design knowledge and the concepts of the design method". The challenge of the design process was to develop something useful. The design team consisted of Dieter Hammer, Kees van Overveld, and Isabelle Reymen. Some months later, they defined that the prototype should reach two goals, namely, first, the development of a methodological aid to investigate the suitability of the design frame and the design method in practical situations and, second, the development of a design aid to support (student) designers.

Halfway the third year (February 1999), the implementation process of a prototype started. From that moment, the *design and implementation* of the prototype were performed in parallel. Isabelle Reymen made a description of the implementation task to offer enough documentation to let a programmer

[1] Based on the results of confrontations of the design method and the prototype with design practice (see below).

make the prototype. A new team consisting of Elisabeth Melby (programmer), Isabelle Reymen, and Jack van Wijk and Huub van de Wetering for supervision and technical advice worked together and had regular meetings in which important decisions were taken. The implementation process started with an exploration of the implementation task. The prototype should combine the benefits of the design frame and the design method with the possibilities of the medium (software); it would concentrate on storage of information (description of all properties and factors of a design situation) and on searching through information. The goal was to develop the prototype in several versions, based on the results of the design of the prototype. The implementation concentrated first on the data-structure and then on the user interface. In September 1999, a first version of the prototype was ready.

The current prototype was confronted with design practice to get *feedback* from design practice on the design method and prototype: in January and February 2000, expert designers gave comments on the prototype and in March 2000, junior designers used the prototype. A detailed description of the performed empirical research, can be found in Chapter 2 and Appendix D in [Reymen, 2001a] or [Reymen, 2001b]. The results of the feedback can also be found in Chapter 5 of this report. After the confrontation with design practice, no changes were made to the prototype, but a list of possible improvements and extensions was made (see Chapter 5).

2 General Concepts

This chapter describes the general concepts of the prototype software tool, already mentioned in Section 1.2, in more detail. The concept of templates is explained in Section 2.1. Sections 2.2 and 2.3 explain the concepts required to describe a design situation, namely a tree and electronic forms. Sections 2.4 and 2.5 describe the queries and checks introduced to analyse the data.

2.1 Templates

The basic Echo template is a structure that can be used to define specific attributes of properties, factors, and relations, predefined values for some of the attributes, and topics and structure of a checklist (CHECKLIST Description Design Situation). A template can be filled in for each type of design task, user, project, and/or discipline. When starting the prototype, a particular (filled in) template can be chosen to initialise the prototype. An advantage of the use of templates is that attributes, predefined values, and checklists do not have to be fixed in the prototype. This means that users can use their own terminology and structuring principles (if defined in templates). Templates can also easily be changed, which makes it (from a research point of view) easy to test different concepts.

2.2 A tree

To understand the concept of a tree, a file system can be taken as an example. The tree concept is used in Echo to implement CHECKLIST Description Design Situation and (part of) FORM Properties&Factors. As mentioned in the previous section, topics and structure can be defined in a template. The checklist defined in the template is made visible as a tree in the basic screen of the prototype, using the style of Windows Explorer. A user can enter properties and factors at different levels in the tree; when information is entered, the tree shows (part of) the current design situation (this function corresponds to that of FORM Properties&Factors). An example of a tree is given in Figure 2.1. A ‘+’ in the tree indicates that information exists at a lower level in the tree. A ‘-’ means that all information at lower levels is visible. Properties and factors are distinguished from the topics and structure of the tree by a ‘>’ sign and a different colour. In the example, three factors are visible.

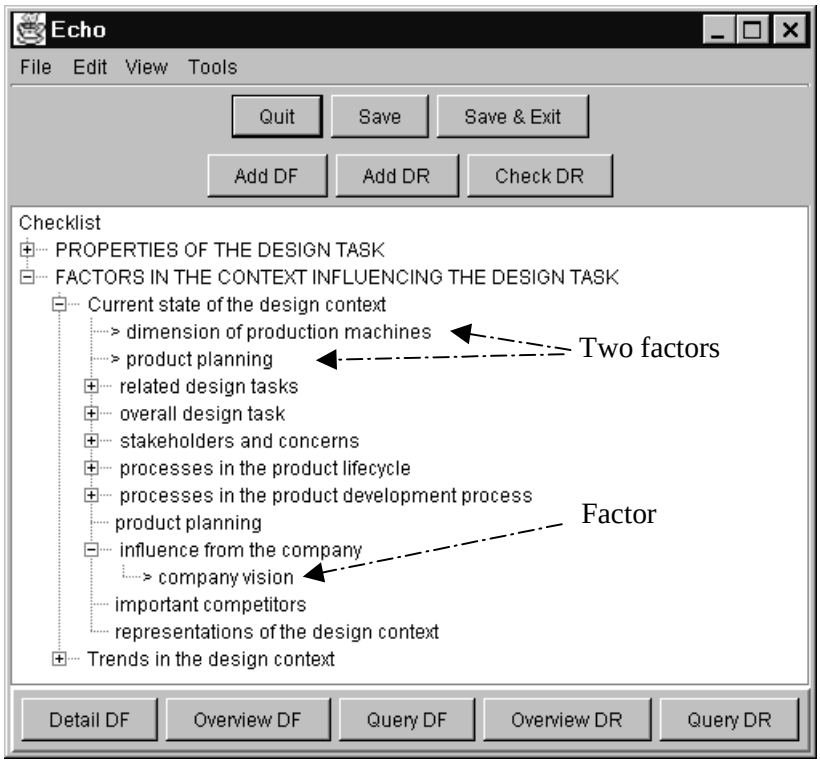


Figure 2.1: Basic screen of Echo, including the tree.

2.3 Electronic forms

The design method contains forms for making a precise description of a design situation. In Echo, electronic forms replace FORMS Properties&Factors and Relations. An example of electronic FORM Relations is given in Figure 2.2. The names (attributes) in the left column of the electronic form correspond to the names of the columns in FORMS Properties&Factors and Relations. In the right column, values for these attributes can be filled in or they can be chosen from a pull-down menu; the values in this menu are predefined by a user in a template or are generated by the prototype. In the example of Figure 2.2, values for the attributes 'From DF', 'To DF', and 'Type' can be chosen in a pull-down menu; values for the first two of these three attributes are generated by the prototype (they are the properties and factors in the database); values for the type attribute are predefined by the user.

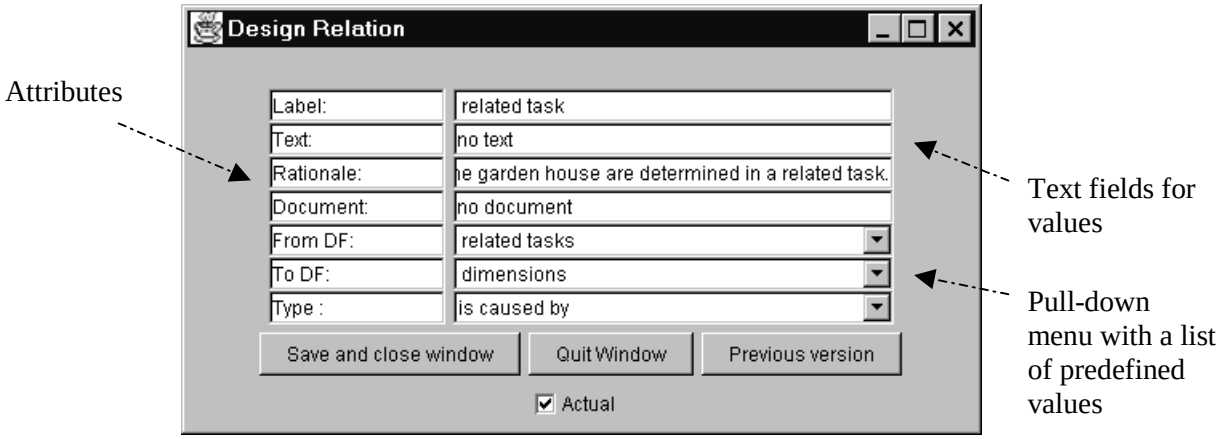


Figure 2.2: Electronic form to describe a relation precisely.

Three types of attributes are distinguished in the prototype, namely attributes that always occur in an electronic form (called basic attributes), attributes that can be defined by a user in a template (called additional attributes), and extra attributes (explained below). I defined the following *basic attributes* for properties and factors: label, value, source, reference, and rationale and for relations: label, from,

to, and rationale. Examples of *additional attributes* for properties and factors are stadium and fixation. Values for the stadium attribute are, for example, for the field of architecture ‘sketch design’, ‘preliminary design’, and ‘detail design’; for software engineering, the values can be ‘requirements analysis’, ‘design’, and ‘implementation’. Values for the fixation attribute are, for example, negotiable and unnegotiable. An example of an additional attribute for relations is the attribute type, which can have, for example, the values ‘is part of’, ‘is caused by’, ‘depends on’, ‘is a variant of’, ‘is an answer to’, ‘is a consequence of’, ‘is important because of’, ‘is output (or input) for’, ‘is a refinement of’, ‘is a’, ‘has a’, ‘transforms’, ‘fulfils’, ‘is influenced by’, and ‘limits’. *Extra attributes* are defined to perform some kind of configuration and version management. These attributes do not correspond to attributes defined in the design method. The extra attributes are not visible for the user and values for these attributes are added automatically by the software. The extra attributes are the following (In the remainder, I use the term ‘item’ to refer to ‘a property, a factor, or a relation’.):

- *Unique identifier*: A value for this attribute is a unique code given to each item that is added to the database.
- *Time-stamp*: The value for this attribute is the date and time when the item is added to the database.
- *Topic*: A value of this attribute gives the topic (question) under which the property or factor is added to the checklist. This attribute is only defined for properties and factors, not for relations.
- *Status*: Values for the status attribute are ‘actual’ and ‘obsolete’. When an item is added, it automatically gets the value ‘actual’. When a user changes this value, the old value becomes obsolete. Making an item obsolete replaces the action of deleting the item. An obsolete item can also be changed back again into an actual one by a user. The advantage of using a status-attribute is that no items are removed permanently (they can all be retraced) and that the data in the database is not static.
- *Version*: The value of this attribute is the version-number of an item. A new version is made of an item when any value of an attribute of the item is changed. The software detects this change automatically and increments the version-number by one. A user can look up earlier versions of an item (see also the next section).

2.4 Queries

With the current prototype, only some simple analysis of the data entered in the database can be performed; CHECKLIST Analysis Design Situation is only partly supported. Echo includes three kinds of query mechanisms: a mechanism to look at the detailed data in the database; a mechanism to look from different points of view to the data; and a mechanism to search for specific information.

With the *first type of query*, all the details of one property, factor, or relation can be viewed. The tool works similar for properties and factors. The screen ‘detail of a design factor’ in Figure 2.3 shows the result of a query of the first type applied to a selected factor. The screen shows all values of the selected factor and all properties and factors that are related to this factor. The screen consists of three columns. The middle column (B) gives the attributes of the selected factor and values for these attributes. Previous versions of the factor can be requested. The left (A) and right (C) columns represent properties and/or factors related to the factor in column B (from A to B and from B to C). By selecting a relation and subsequently select ‘Relation A->B’ from the ‘View’ menu, values for attributes of the selected relation can be viewed.

The *second type of query* offers an overview of all properties and factors that are entered in the tree under a certain topic; for example, all properties about the product being designed or all properties about the desired state of the product being designed. Also, an overview of all relations concerning a property or factor that is entered in the tree under a certain topic can be asked. As an example, the screen ‘overview design factors’ is shown in Figure 2.4; in this case, all factors in the design context that have the value ‘no’ for the document attribute are shown.

The *third type of query* offers the possibility to search for specific values of specific attributes defining certain items. Examples are queries giving

- an overview of all properties of the product being designed, with the value ‘negotiable’ for the fixation attribute;
- all factors with status ‘actual’, with stadium ‘desired’, and with some value including the word ‘coating’;
- all relations of a certain type.

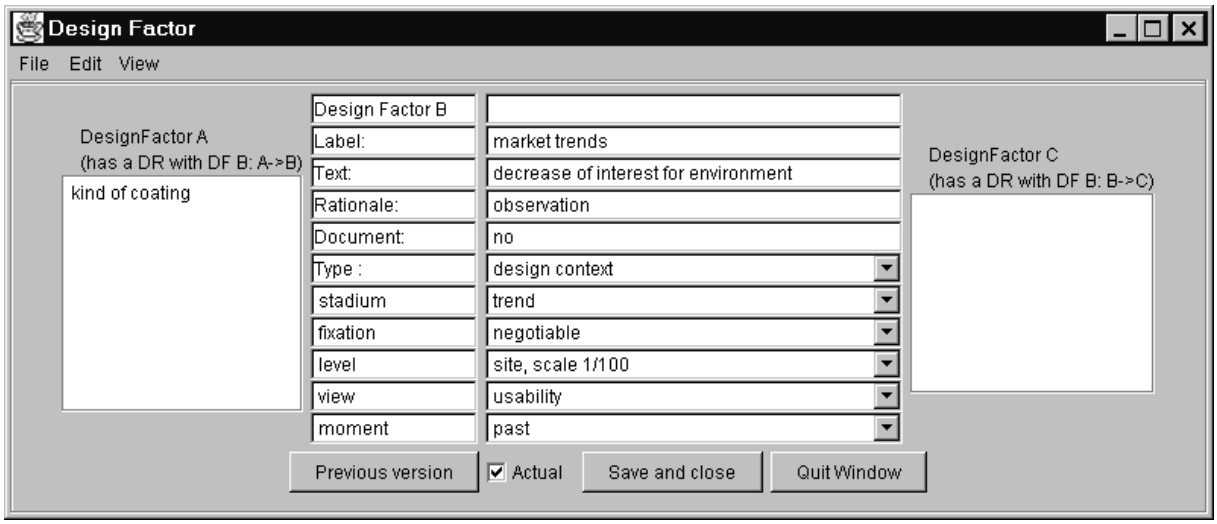


Figure 2.3: Screen ‘Detail of a design factor’.

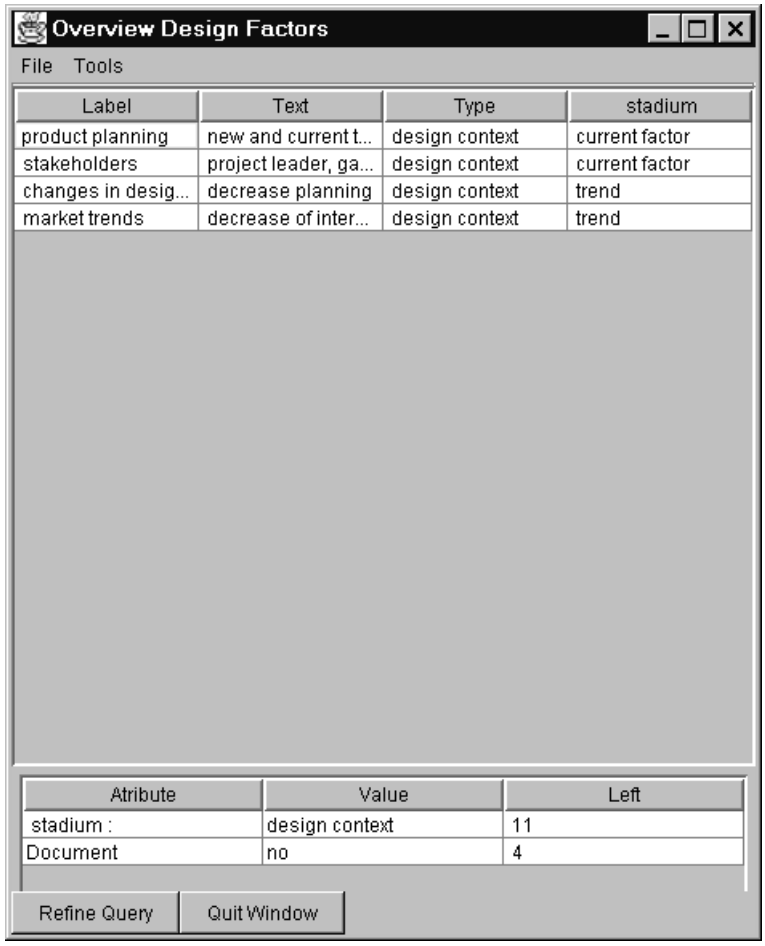


Figure 2.4: Screen ‘Overview design factors’.

The third type of query can be performed on properties, factors, and relations. The screen ‘Query Design Factors’ is shown in Figure 2.5. In the left column, specific values for attributes can be entered; the middle column shows the number of items left in the selection (updated each time a

refinement of the query is made); in the third column, a user can mark attributes and values to be shown in the overview. The result of the specific query in Figure 2.5 is the overview shown in Figure 2.4.

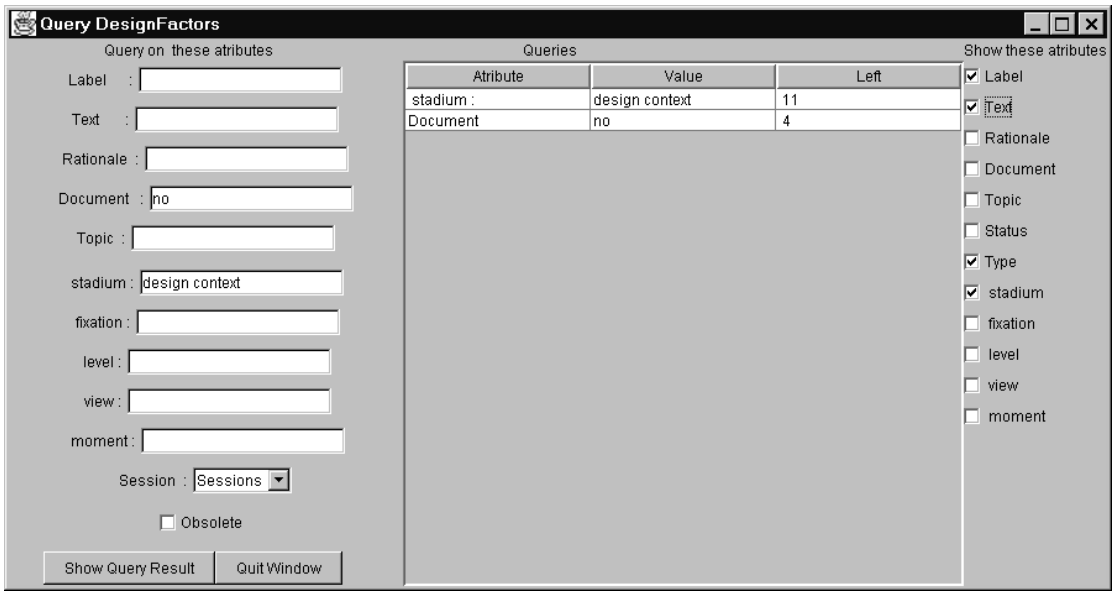


Figure 2.5: Screen ‘Query Design Factors’.

2.5 Checks

Echo can do some very simple consistency checking on the data entered in the database and can give warnings to the user. The prototype offers the possibility to check if both properties and/or factors of a relation exist and are actual (when a new relation is entered) and if the relation between two obsolete items is also obsolete. If a relation is not consistent, then, a warning message appears on the screen.

3 User documentation

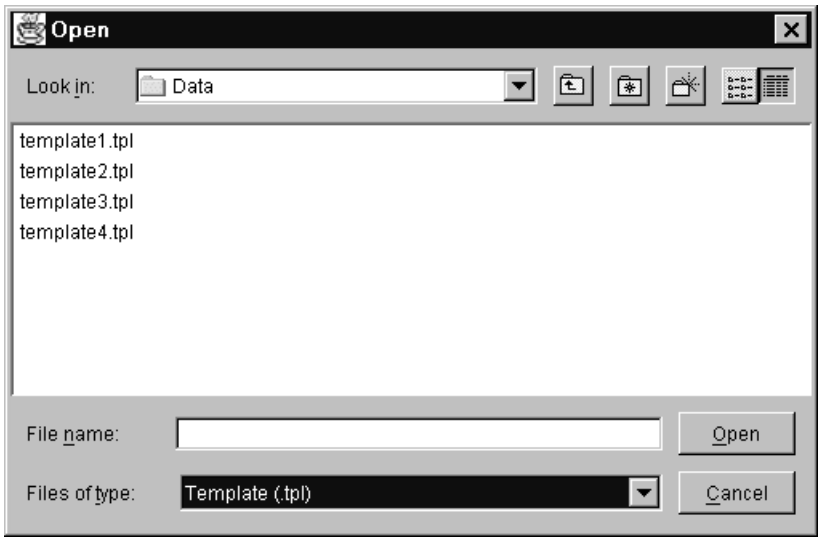
The goal of this chapter is to offer designers an introduction to using the prototype. The user documentation of Echo describes how to get started (in Section 3.1), the possibilities of Echo (the screens are displayed in Section 3.2, the explanation is given in Section 3.3), some remarks about viewing (obsolete) factors and relations (in Section 3.4), and an explanation of how to define templates (Section 3.5).

3.1 Getting started

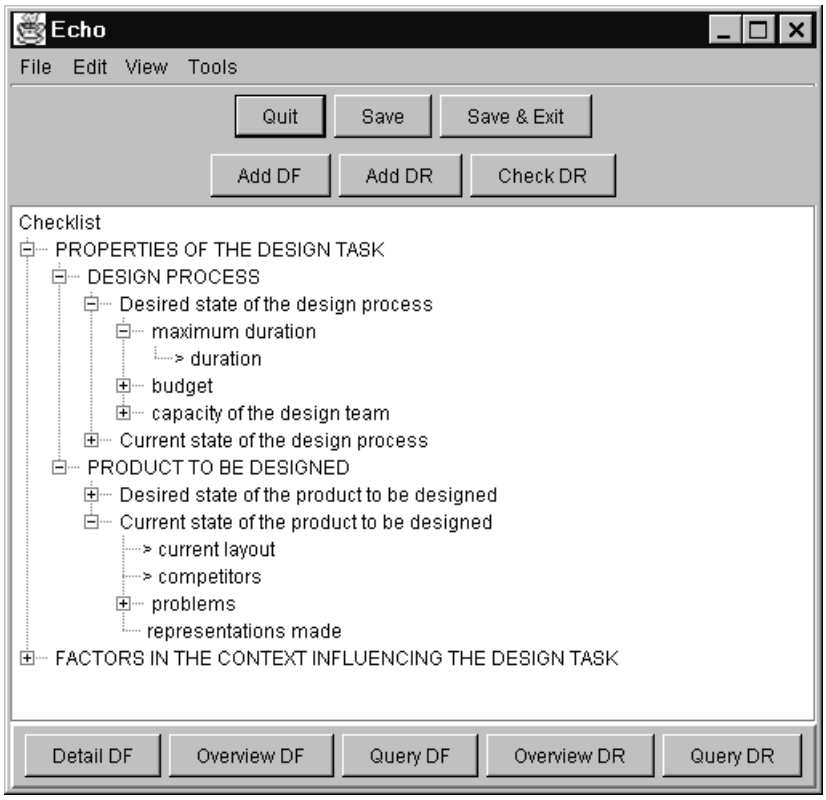
To run the prototype in a Java Runtime Environment (see also Section 4.7.1):
Download a Java micro edition from the Java site, i.e. http://java.sun.com/products/jdk/1.2/jre Java 2 Runtime Environment v 1.2.2-001 Windows 95/98/NT Production Release (6MB) to install Java. Run **Echo**.bat (A batch file starting Echo.) If necessary, correct the pathname of Echo in the **Echo**.bat file, using a text editor.

When starting the tool, a 'file-browser' is shown to open a file. (SCREEN 1, depicted in Section 3.2)
When the tool is used for a first time in a project, choose a template (*.tpl) (and save it to a new created data-file (*.dit) when finishing the session). For the next uses, clicking the Open button can open the data-file of the project. Each period of using the prototype is called a session. When a data-file is opened, a new session is started.

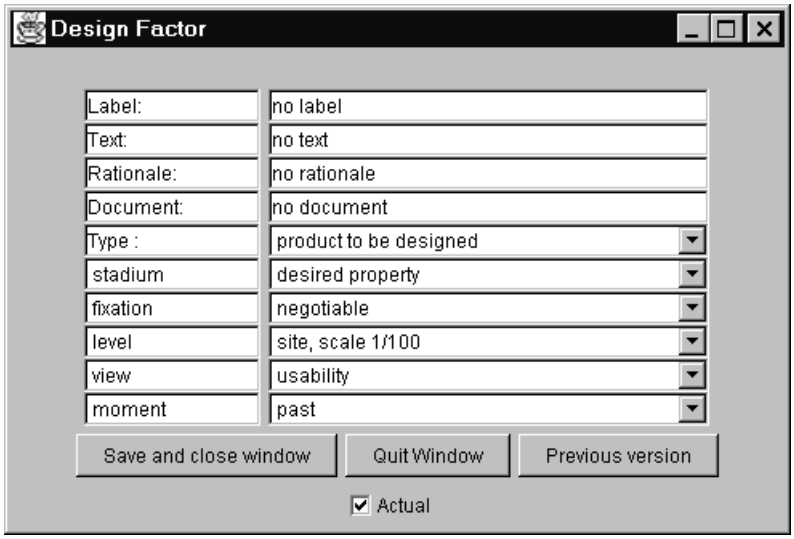
3.2 Screen dumps



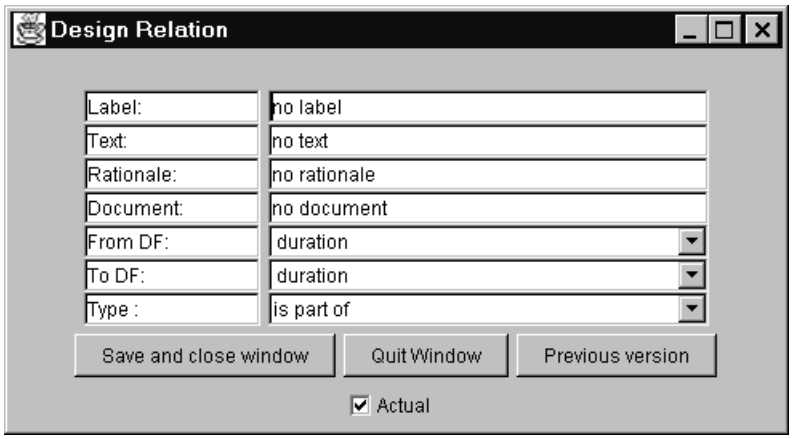
SCREEN 1



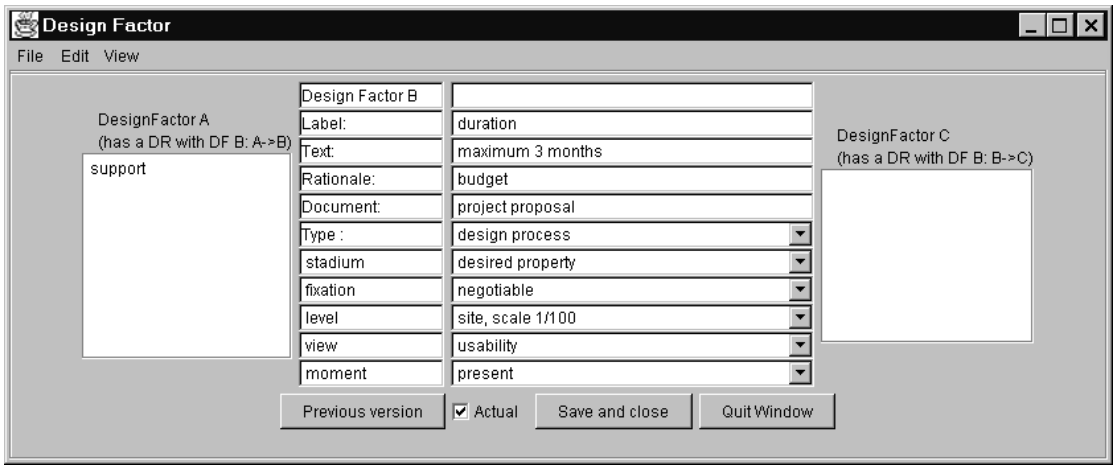
SCREEN 2



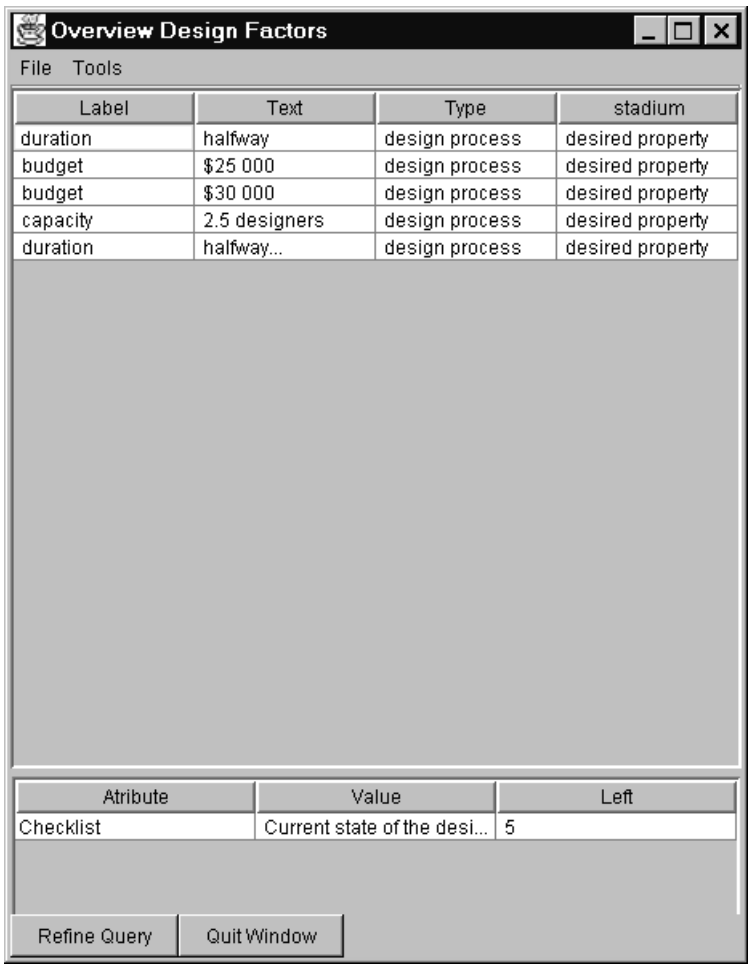
SCREEN 3



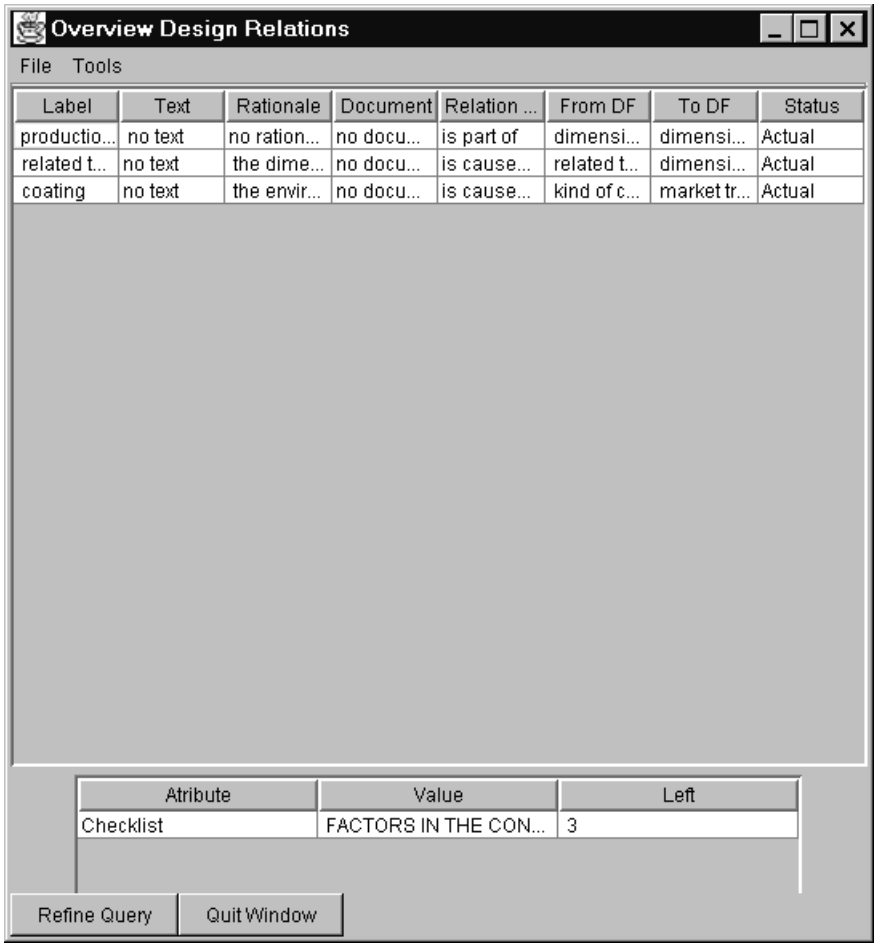
SCREEN 4



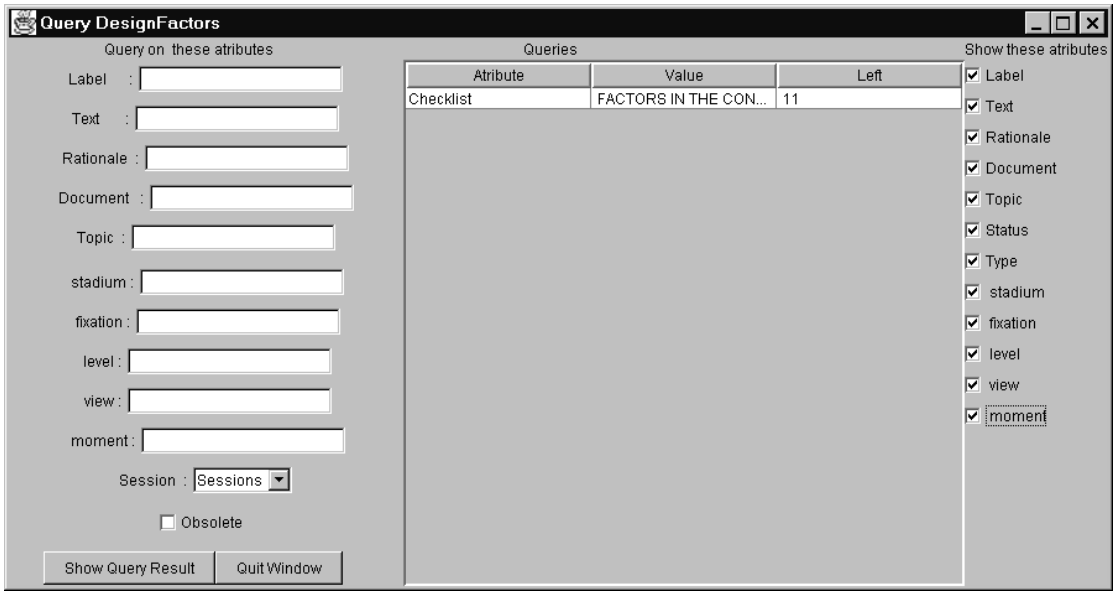
SCREEN 5


Overview Design Factors
File
Tools
Label
Text
Type
stadium
duration
halfway
design process
desired property
budget
$25 000
design process
desired property
budget
$30 000
design process
desired property
capacity
2.5 designers
design process
desired property
duration
halfway...
design process
desired property
Atribute
Value
Left
Checklist
Current state of the desi...
5
Refine Query
Quit Window

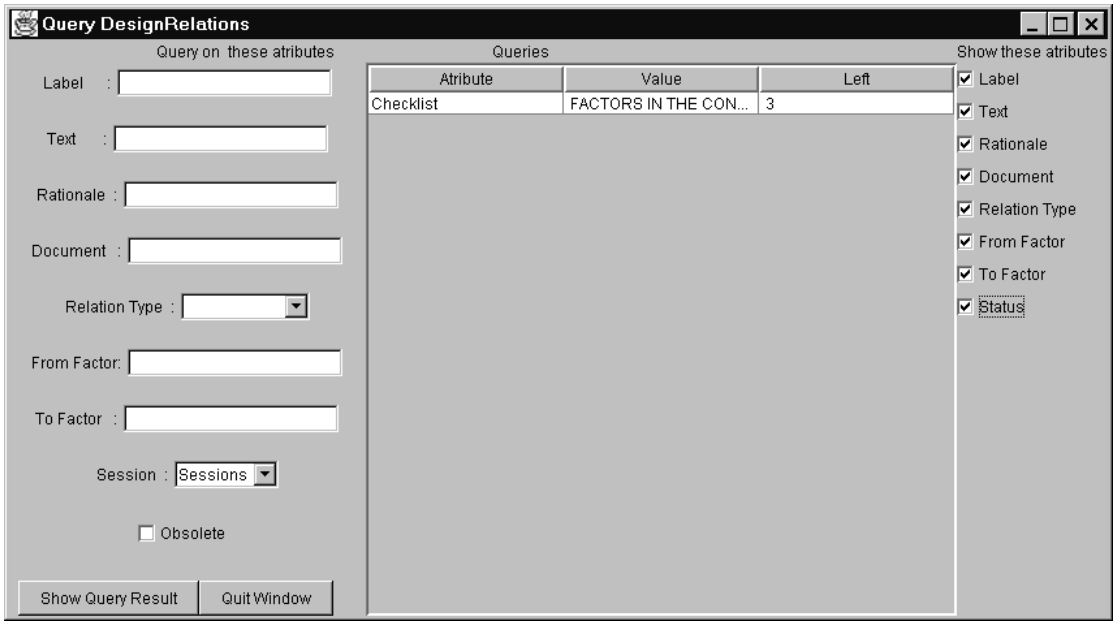
SCREEN 6



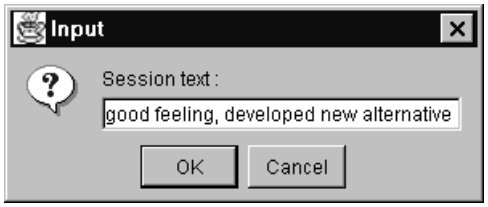
SCREEN 7



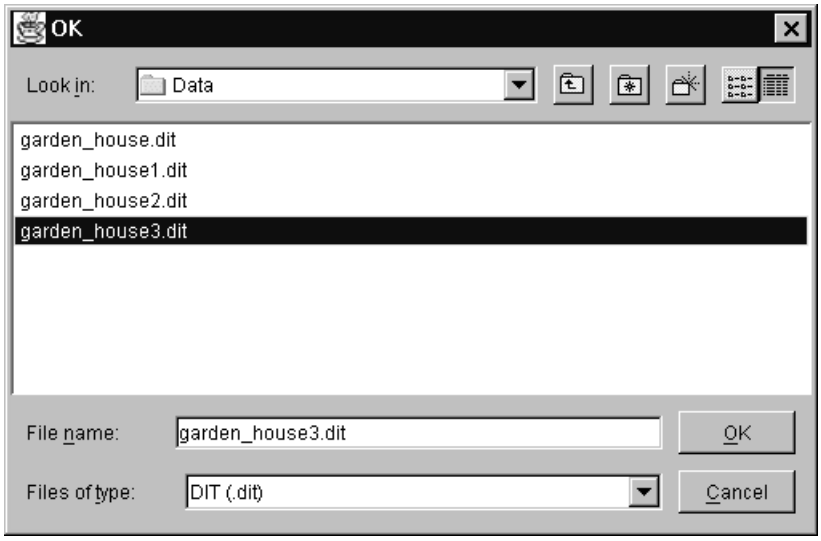
SCREEN 8



SCREEN 9



SCREEN 10



SCREEN 11

3.3 Explanation of the screens

Basic screen of ECHO: Checklist (SCREEN 2)
A tree with nodes is shown; it represents a checklist. The structure and topics of the checklist depend on the chosen template. A checklist includes several topics, under which factors can be positioned. The distinction between topics and factors is visualised as follows: before a factor, a '>'is placed, and when selecting a factor, it becomes yellow; when selecting topics in the checklist, they become green (the colours may be different). A '+' means that there are still more levels below the topic in the checklist; a '-' before a topic means that this is the lowest level. The basic screen has a menu bar with

pull-down menus and buttons, both having the same functionality; an explanation is given for the pull-down menus. In the explanation, the names of pull-down menus and topics in these pull-down menus are represented in bold. All explanation is given for factors, but also holds for properties. The screen referred to, is displayed in the previous section.

File: Quit ECHO
Quits the tool

File: Save
New window: (SCREEN 10)
Information about the current design session can be entered. The session will be closed and saved afterwards.
New window: (SCREEN 11)
A filename for the data can be entered in a 'file-browser'. The file must receive the extension '.dit'.

File: Save and Exit
New window: (SCREEN 10)
Information about the current design session can be entered. The session will be closed and saved afterwards.
New window: (SCREEN 11)
A filename for the data can be entered in a 'file-browser'. The file must receive the extension '.dit'. The tool will be closed.

Edit: Add DF: adds new factor one level below the selection in the tree
Start position: node in the tree, a new factor will be put one level below the selection. Factors can be added everywhere in the tree: under topics, under other factors, and even under multiple topics (doubles are permitted).
New window: (SCREEN 3)
Attributes and text fields for values
Attributes and pull-down menus with predefined values (predefined values depend on chosen template; new values can be defined)
Previous version
Actual
Save and close window
Quit window

After every textual entry, a return must be given. The 'label' attribute must always be filled in. When entering a value that is not predefined, it must be longer than one character. For the 'type' attribute, only predefined relations can be chosen, because of the checks performed on relations. Not all values have to be filled in at once. When a value of an attribute of a factor is changed and the window is saved (and closed), a new version is made.

Edit: Add DR: adds new design relation one level below the selection
Start position: does not matter, the design relation is not visible on the screen and is stored with the factors of the relation.
New window: (SCREEN 4)
Attributes and text fields for values
Attributes and pull-down menus (for 'from' and 'to' attributes: all existing design factors can be chosen; for the 'type' attribute: predefined values can be chosen)
Previous version
Actual
Save and close window
Quit window
After every textual entry, a return must be given. The label attribute must always be filled in. When entering a value that is not predefined, it must be longer than one character. Not all values have to be filled in at once. When a value of an attribute of a factor is changed and the window is saved (and closed), a new version is made.

Tools: Check DR: checks design relations
Checks if both factors exist and are ‘actual’.

View: Detail DF: shows detailed information about selected factor, including related factors
Start position: on the desired factor; this factor will be shown in detail
New window: (adapt window to your screen!) (SCREEN 5)

Overview with three columns:

- in the middle: all information of the selected factor (Design factor B), this information can be edited, as described in Add Design Factor,
- at the left-hand side: a list with all factors (Design factors A) having a relation with factor B (Design relation: DF A -> DF B),
- at the right-hand side: a list with all factors (Design factors C) having a relation with factor B (Design relation: DF B -> DF C).

The screen has a menu bar and buttons, both having the same functionality.

File: Save and Close window: saves the changes and closes the window

File: Quit window

Edit: Edit Design relation DF A -> DF B
Start position: the desired factor selected
New Window: as described in Add Design Relation; the relation from this factor to the factor in the middle can be edited

Edit: Edit Design relation DF B -> DF C
Start position: the desired factor selected
New Window: as described in Add Design Relation; the relation from the factor in the middle to this factor can be edited

View: Detail Design factor A
Start position: the desired factor selected (in the first column)
New Window: as described Detail DF, with the selected factor A in the middle (becomes Design factor B)

View: Detail Design factor C
Start position: the desired factor selected (in the third column)
New Window: as described Detail DF, with the selected factor C in the middle (becomes Design factor B)

View: Previous version Design factor B
New window: see Add DF

Two factors or relations cannot be edited at the same time.

View: Overview DF: shows overview of all factors one level below the selection
Start position: node in tree, all factors one level below the selection will be shown
New window: (SCREEN 6)

Overview with:

- one table listing all factors of the selection (including values for the chosen attributes),
- one table listing the following:
  - attribute: the kind of start position of this kind of query (checklist or attribute of query),
  - value: the name of the selected node in the start position or the chosen value,
  - numbers left: the number of factors left in the selection.

The screen has a menu bar and buttons, both having the same functionality.

File: Quit window

Tools: Refine Query Design Factors
New window: see Query DF
When clicking on one row in the first table, this factor can be edited, as described in Add DF.
The order of the columns in the table of Overview DF can be changed.

View: Overview DR: shows overview of all design relations linked to all factors one level below the selection
Start position: node in tree, all relations linked to all factors one level below the selection will be shown
New window: (SCREEN 7)

Overview with:

- one table listing all design relations of the selection (including values for the chosen attributes),
- one table listing the following:
  - attribute: the kind of start position of this kind of query (checklist or attribute of query),
  - value: the name of the selected node in the start position or the chosen value,
  - numbers left: the number of design relations left in the selection.

The screen has a menu bar and buttons, both having the same functionality.

File: Quit window

Tools: Refine Query Design Relations

New window: see Query DR
The selection can be refined infinitely.

When clicking on one row in the first table, this relation can be edited, as described in Add DR. The order of the columns in the table of Overview DR can be changed.

Tools: Query DF: offers the possibility to make queries on factors
Start position: node in tree, all factors one level below the selection can be shown (which factors are shown depends on the query)
New window: (SCREEN 8)

Window with three columns:

- column one: attributes and text fields: these are the search criteria. The search can be executed with part of a word that must match. Typing text must be followed by a return. Adding a search criterion results in a new row in the second column;
- column two: the selection made, listing the following:
  - attribute: the kind of start position of this kind of query (checklist or attributes),
  - value: the name of the selected node in the start position or the value of the chosen attribute,
  - numbers left: the number of factors in the selection;
- column three: the attributes that must be shown in the result of the query.

When searching on session, all new factors added in that session are shown.

Show Query Result

A window as described under Overview DF appears, showing the factors of the query.

Quit window

Tools: Query DR: offers the possibility to make queries on design relations
Start position: node in tree, all design relations one level below the selection can be shown (which relations are shown depends on the query)
New window: (SCREEN 9)

Window with three columns:

- column one: attributes and text fields: this are the search criteria. When typing in a search criteria, use return to enter it into Echo. Each new search criterion results in a new row in the second column;
- column two: the selection made, listing the following:
  - attribute: the kind of start position of this kind of query (checklist or attributes),
  - value: the name of the selected node in the start position or the value of the chosen attribute,
  - numbers left: the number of design relations in the selection;
- column three: the attributes that must be shown in the result of the query.

When searching on session, all new relations added in that session are shown. When starting from the root of the tree (start-point) with a query, the attribute in the second column is suppressed.

Show Query Result
A window as described under Overview DR appears, showing the relations of the query.
Quit window

3.4 Some remarks about viewing (obsolete) factors and relations

In the previous section, the meaning of each button/pull-down menu is explained. This section gives an overview of the different possibilities for viewing factors and relations and explains how to view obsolete factors and relations.

3.4.1 Viewing factors

- To see one factor in detail (including all relations of the factor): click on **Detail DF**,
- to see all factors on the same level in the tree: click on **Overview DF**,
- to make a specific selection of factors, possibly starting from a level in the tree: click on **Query DF**,
- to edit a factor: click on **Detail DF**, or in **Overview DF**, click on a row.

3.4.2 Viewing relations

- To see one design relation in detail: click on **Detail DF**, click on **Edit From** or **Edit To relation**,
- to see all relations of factors on the same level in the tree: click on **Overview DR**,
- to make a specific selection of relations, possibly starting from a level in the tree: click on **Query DR**,
- to edit a design relation: in **Detail DF**, or in **Overview DR**, click on a row.

3.4.3 Viewing obsolete factors and relations

The tree only shows actual factors. Obsolete factors and relations are not visible in regular **Overview DF** or **Overview DR** (see below). Obsolete factors are also suppressed when viewing Design factor A or C in **Detail DF**. In queries, the attribute status is default ‘actual’; only actual factors and relations are then shown. When changing in queries the value of the status attribute from actual to obsolete, obsolete factors and relations are shown. Obsolete factors and relations can also be seen when selecting the upper left position of the tree (checklist) and then clicking **Overview DF** or **Overview DR**. Obsolete factors and relations can be made actual in the **Overview DF** or **Overview DR** of a query or in **Detail DF**.

3.5 Defining a template

A user can define one or more specific templates according to his needs and preferences (for example, different predefined values for the attributes of factors and different topics in the checklist can be useful in different disciplines). There is no user interface to define a template yet; a text editor can best be used. The template chosen at the beginning of a design project, however, must be used during the complete design project (a template cannot be changed while using the same data-file; this is because after the first use of a specific template, the chosen template is stored in the data-file to avoid inconsistencies between data and template). Many data-files, however, can include the same template. A definition of a template may consist of the definition of attributes and predefined values for factors and of topics in the checklist.

3.5.1 Definition of attributes and predefined values

In the prototype, some attributes for factors and relations are fixed and always appear on the screen (these are called basic attributes in Section 2.3). The user cannot change these attributes in the template. The attributes are label, text, rationale, document, and type. The values for these attributes are not fixed. The attributes a user can define can be defined on two levels: categories and subcategories. The categories are values for the basic attribute ‘type’; these correspond with the additional attributes in Section 2.3. Depending on the value for the ‘type’ attribute, different subcategories can be defined by the user. For example, in Screen 3 in Section 3.2, basic attributes and their values can be recognised. The value for the type attribute is ‘product to be designed’ (a category). For this category, the subcategories stadium, fixation, level, view, and moment are defined, together with some predefined values that can be chosen in the pull-down menu.

For each attribute, a label for the user interface must be defined (the name of the category or subcategory) and values can be predefined. It is possible to define as many categories as desired; the number of subcategories is limited to ten, due to some technical problems in the programming language. There is no limit to the number of values a subcategory can take, though it is nice if they all fit on the screen at the same time; this saves the user some scrolling. Since the options are shown in a pull-down menu, at least one option should be given. If blanks are kept around the delimiters, everything should work fine. An example of a template including the definition of these attributes (categories and subcategories) and values is given in Subsection 4.6.2. The definition of attributes and values for factors in general is given in Section 4.6.1.

3.5.2 Definition of topics in the checklist

In the template, topics can be defined on different levels. Topics and their level indicator must be listed in the desired order of appearance on screen. The topics of the checklist on Screen 2 in Section 3.2, are defined in the example of a template in Subsection 4.6.2. The definition of topics in a checklist looks in general as given in Section 4.6.1.

4 Implementation documentation

The goal of this chapter is to offer programmers information about the current implementation of Echo for improving and/or extending the prototype. Echo is a prototype software tool facilitating the use of a design method supporting structured reflection on design processes. The prototype has been developed to explore the benefits of using a software system to facilitate the use of the design method. A short description of the concepts of the design method can be found in Section 1.1. The prototype software tool is a domain-independent tool that only supports the description and analysis of design situations, part of the preparation step of a reflection process. The main concepts of the prototype are explained in Section 1.2 and Chapter 2. The prototype has mainly been developed by Elisabeth Melby (programmer) and Isabelle Reymen. The development process of the prototype is described in Section 1.3.

The implementation process started from the following requirements. Because the prototype must be useful for *several design disciplines*, the data structure must be flexible, the terminology used in the prototype must be adaptable to the terminology of different design disciplines, and the software should work on several platforms. For the *description of design situations*, a user must be able to enter important properties, factors, and relations in the tool, possibly supported by a checklist. For the *analysis of design situations*, a user must be able to query the data via different types of queries. To support the *dynamics of a design process*, the changes made during design sessions must be recorded and the software must support consistency in the data storage. Besides fulfilling these requirements, we decided that a first version of the prototype software tool would only support a single user, would not be linked with other tools, would include no help function, and would only include a two dimensional visualisation.

In the current prototype, all the above mentioned requirements are met. Echo is implemented in Java using Swing. Java has proven to be portable: Echo has been tested on various PC's/laptops under Windows 95 and 98, on a SGI Windows NT machine, and on a Sun workstation under Solaris. No specific environment has been used to build the prototype (only an editor). For the implementation, no real database has been used, to keep the prototype portable and because the expected amount of data was small. Main concepts of the prototype are classes and a data-structure.

In Section 4.1, a model of the current class structure is given. In Sections 4.2 to 4.5, an explanation of the important classes is given. More information about the implementation of each of the classes can be found in Javadoc (which is coupled to the code); in Section 4.7, more about Java and Javadoc can be found. Section 4.6 gives a description of the structure of a data-file.

4.1 Model of the class structure

Figure 4.1 gives a model of the class structure.

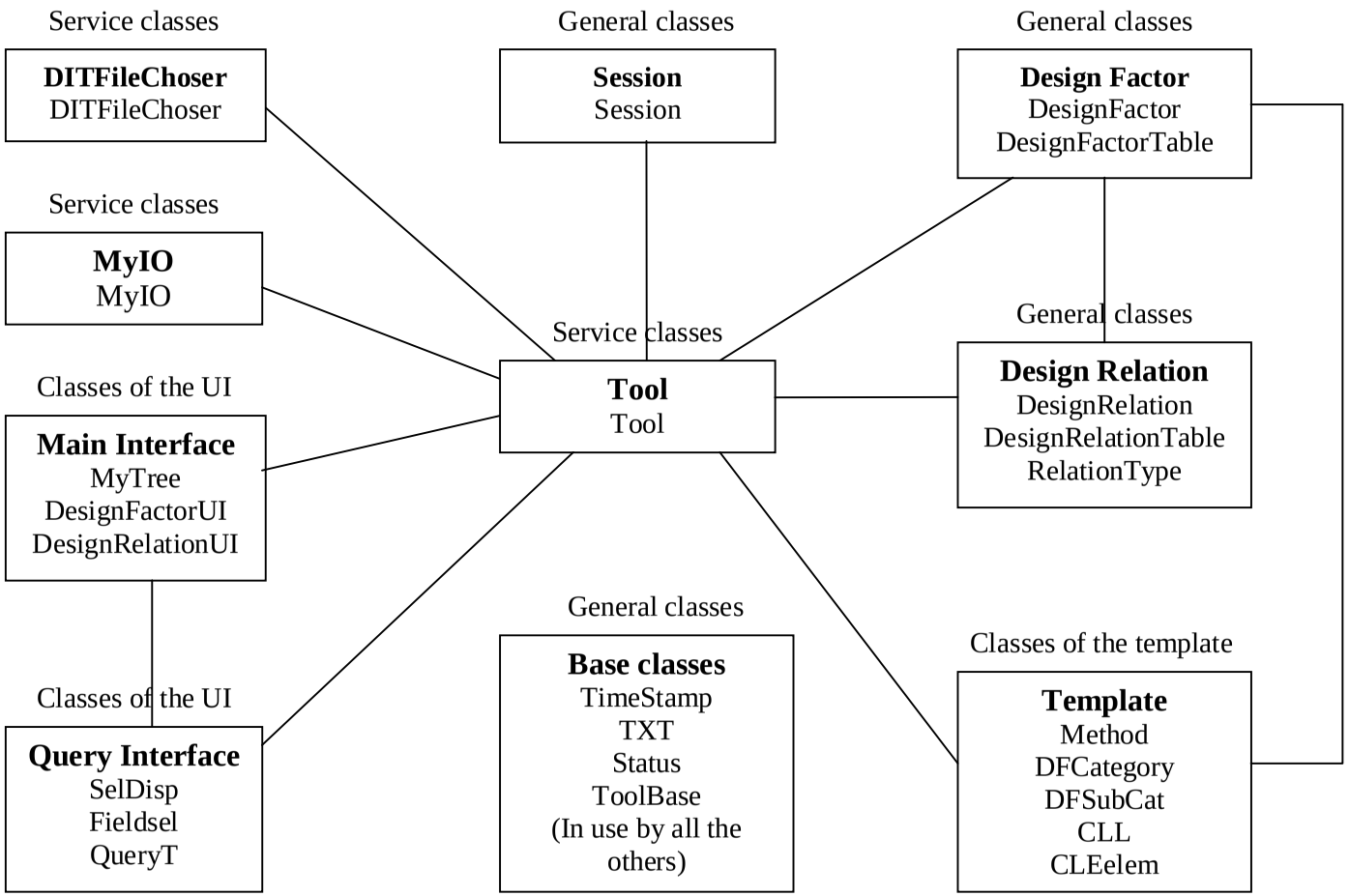


Figure 4.1: Model of the class structure.

4.2 General classes

For each class, a short general explanation, the important data, and the important methods are mentioned.

4.2.1 Class Session

Each time Echo is run, a new session is generated. The session runs until the user exits Echo. The most important information stored in a Session, is begin time and end time. Since all information in Echo is stored with a timestamp, the begin time and the end time of a session are enough to find the information entered during one session.

Data:

```
private static Vector sessions;
private int number;
private TimeStamp beginTime, endTime;
private String text;
private int status;
```

Methods:

Basically a number of constructors, getX and setX with various formats (including add).

`Public void finnish(DataOutputStream out, myIOmy, String sesText)`: save the template, the data and the session information as it is now and leave Echo.

`Public void writeSave(DataOutputStream out, myIO my, String sesText)`: save the template, the data and the session information as it is now and continue the session.

`Public Enumeration enumSes()`: returns an enumeration of the sessions.

`Public void analyse(StringTokenizer st, myIO my, Session s)`: builds up the sessions based on an input file.

4.2.2 Base Classes

Some basic classes were formed because the prototype was written in Java with inheritance possibilities. These classes were made to make life easier for the programmer.

4.2.2.1 Class TimeStamp

As all information in Echo was to be stored with a time stamp, it seemed sensible to have a timestamp class handling that side of things. Time/Date proved to be less standard than one would have thought, making this class very useful indeed.

Data:

```
date remember;
format fmt;
```

Methods:

`TimeStamp()` without parameters, creates a TimeStamp with time = now.

`TimeStamp(String time)` creates a timestamp with the time given in the string.

`TimeStamp(TimeStamp ts)` creates a copy of the TimeStamp given.

`Date GetDate()` returns the Time in Date format.

`Void t_print()` prints the time of the TimeStamp to standard out (the screen)

`Void t_print(DataOutputStream out)` prints the time of the TimeStamp to the output stream out (output file).

`String getDateString()` returns the Date in the correct format with the time of the day.

`Boolean t_before(Date t)` returns true if the time is before the parameter time.

4.2.2.2 Class TXT

As every item has a timestamp, a class functioning as a new datatype, was created. This simplifies the handling of strings with their creation/modification time.

Data:

```
private String txt;
private TimeStamp time;
```

Methods:

Constructors, getX, some print methods.

`Boolean t_before(Date d)` returns true if this TXT is created before the time(Java type Date) given.

4.2.3 Class ToolBase

Looking at the information Echo was to support, it seemed sensible to factor out the common entities to a ToolBase (which corresponds to additional attributes as mentioned in Section 2.3). The DesignFactors and DesignRelations can be modified, causing the creation of previous versions. In the current version, modification of the Method class (see Section 4.3.1) is not supported. Though if the Method class should become modifiable, the present setup is still logical.

Data:

```
TXT label;          // label for screen
TXT text;           // full describing text
TXT rationale;      // the why of this object
TXT doc;            // path to a document giving more information
TXT cle;            // Checklist element. In a new version, cle could be moved to Design Factor
                    // as the Design Relations are no longer a part of the tree (main User Interface)
int status;         // one of status values
vector hist;        // an array containing the history = old versions of this object
```

Methods:

Constructors enough for most situations.

`GetX` for all data – value and TXT/Object and String/Text if useful – a string with for instance the status in full text in stead of a number.

`SetX` for all data – also sets like `setActual()` for the status.

`PrintX` prints X on screen.

`Public boolean show()`: true if the status is actual.

`Public boolean isFirst()`: true if this is the first of it's kind.

`Public boolean hasPrev()`: true if previous versions exist.

`Public boolean isInTable()`: flag for UI.

`Public boolean isInTree()`: flag for UI.

`Public void addIn()`: flag for UI.

`Public void setPrev(Object x)`: store previous version.

`Public int getNumPrev()`: return number of previous versions.

`Public Object getFPrev()`: return the last previous added.

`Public Object getPrev(int i)`: return a specified previous version.

`Public int toggleStatus()`: set the status to the opposite value of the actual – obsolete pair.

`Public boolean t_before(Date tst)`: true if the timestamp associated with the label is earlier than the one given as parameter.

`Public void print2File(DataOutputStream out)`: print all data of this ToolBase object to file. The format is suitable for the analyse methods.

`Public static void main(String[] args)`: to test some of the methods.

4.2.4 Class Status

The legal status values are defined here along with minimum and maximum value.
The values are min = 0, max = 4,
ACTUAL = 1, OBSOLETE = 2 and FUTURE = 3.

The status class has no methods.

4.2.5 Class DesignFactor

The DesignFactor is one of the most important data items in the method. Apart from the basic elements from ToolBase, there are elements depending on the template. In the prototype, they are stored in an array. For implementation purposes, track is kept of the checklist element under which a design factor falls.

Data:
Descendant of ToolBase
`Vector mKeuz` // containing data as specified by the template(Method)
`Vector history` // not in use any more
`Static int relKey` // used for generating a numeric identification of the object
`Int relId` // a numerical identification
`Int version` // how many version exist (can do without)

Methods:
Enough constructors to satisfy most needs.
Get and set methods including:
`void setKeuz(TXT t, int index)`
`TXT getKeuz(int I)`
`Int numKeuz()` returns the number of items stored according to the template specification.
`DesignFactor copyDF(DesignFactor df)` returns a copy of the parameter.
`void arc()` put this Design Factor in the archive (history).
`Void print_Factor()`
`Void print2File(DataOutputStream out)`
`Void printPrev2File(Designfactor dh, dataOutputStream out)`
`Static void main(String[] args)` tests some of the methods, used during development.
The rest of the relevant methods are in the DesignFactorTable.

4.2.6 Class DesignFactorTable

The DesignFactors are collected in the DesignFactorTable along with the methods needed to manipulate the collection.

Data:
`Static Vector Table` // a list of all Design Factors

Methods:
A constructor, gets to get number of actual Design Factors, the total number, get a specific DesignFactor indicated by a value or index, get the index when the factor is known, set (add) a Design Factor, and routines handling IO (read, write and analyse)

4.2.7 Class DesignRelation

A Design Relation describes the relationship between two Design Factors. It extends ToolBase.

Data:

```
DesignFactor DFFrom, DFTo      // Dubbel book-keeping as Echo was first developed
int  fromDFId, toDFId          // on a slow computer, not needed anymore
int relationType = 0           // default 0
```

Methods:
The methods consist of a number of constructors, gets and sets, and print options.

4.2.8 Class RelationType

This class is purely to define the legal values for the relation types. In a later version one could consider making this a part of the template to get more flexibility, since one of the questions of the checklist (which is stored in the template) concerns the Relation Type. The list below contains relation types of three kinds, namely NumTypes, Dependency and QA. NumTypes elements are used; dependency and QA are not in use in the current version. Each type has of a number of options. An option may have a constraint like the TIMEDEP where the From design factor must have been entered into the system before the To factor.

Data:

```
final static int NumTypes = 4;
final static int ISPARTOF = 1;
final static int ISVARIANTOF = 2;
final static int ISCAUSEDBY = 3;
final static int DEPENDSON = 4;
final static int SYMETRIC = 5;
final static int DEPEND   = 6;
final static int TIMEDEP  = 7;
final static int QQ       = 8;
final static int QA       = 9;
final static int AA       = 10;
final static int AQ       = 11;
final static String[] RTString  // with the headers
= { "is part of", "is a variant of", "is caused by", "depends on",
"Symetric", "Depend", "TimeDepend", "Q->Q", "Q->A", "A->A", "A->Q"};
```

Methods:
No methods in the present version of Echo.

4.2.9 Class DesignRelationTable

Data:

```
Static Vector rTable      // a list of all Design Relations
```

Methods:
A constructor, gets to get number of actual Design Relations, the total number, get a specific DesignRelation indicated by a value or index, get the index when the Relation is known, set (add) a Design Relation, check validity of the Relations in the Table, and routines handling IO (read, write and analyse).

4.3 Template classes

The Template contains the checklist and a description of the datastructure. Several classes together offer the Template functionality.

4.3.1 Class Method

The Method is the top class within the Template cluster. It extends the ToolBase with design factor categories (DFCategories). The classes combined make up a three dimensional datastructure. The three dimensions stem from the dependencies of the options shown on the screen. Depending on the category, a DesignFactor has items with possible values. This is partly realised in the prototype interface. If the category is "process", it will have certain subcategories like "view" which can have the value "past", "present" or "future". Another category could be "product" or "context" each with subcategories and value options. Some of the possibilities offered by the datastructure have not been fully utilised because of the time factor (implementation and runtime – the first machine used was too slow to change a frame midway).

Data:
`Vector DFCategories.`

Methods:
`Int numCol()` returning the number of data elements used for the DesignFactors.
`Void printEOM(DataoutputStream out)` prints *END_METHOD* to out.
`Void analyse(StringTokenizer st, myIO my, Method met)` – this method will analyse the input offered by the tokenizer. Basically this method builds up the template based on what is found in an input file. MyIO handles the inputfile side of it (and output).
`Void print2File(dataoutputStream out, myIO my)` the template is written to a file in a format suitable for reading back using fillFromFile – which calls analyse.
`Void fillFromFile(String filename, StringTokenizer st, myIO my, Method met)` Opens the necessary streams to read the file identified by fileName, makes a new tokeniser and calls on analyse(..).

4.3.2 Design Factor related classes

A Design Factor is a data item. How much information Echo needs to keep track of for the user, is specified in the template by the classes DFCategory and DFSubCat (description follows). Any given DesignFactor falls under one category. Depending on the category, there are a number of subcategories. So, the method must keep track of the categories, each with a labels/heading and it's own set of sub categories. The sub categories have, of course, their own labels. As Java has very nice vectors, vectors of objects are used that contained the next level of vectors. The number of subcategories defines the number of extra data elements in a Design Factor.

4.3.2.1 Class DFCategory

Contains a label for this DFCategory and a vector with the sub-categories (DFSubCat). A printData function is provided, mainly to write the information to file.

4.3.2.2 Class DFSubCat

Contains a label – the text that is used in the user interface, and a vector containing the possible options. A printData function is provided to write the information to file.

4.3.3 Check List related classes

In the prototype, the checklist is shown in a tree structure into which the designer can add design factors. The list itself is not changed. The classes related to the tree and its leaves are:

4.3.3.1 Class CLL

CLL contains the CheckList. It extends the DefaultTreeModel as supplied with Swing with the following:

Data:

```
Privet myDynamicTreeNode root  // the root of the data-tree
Private String rootName;
Tool tool                      // this is a pointer to the topclass in Echo. It is used to get to other
                               // parts of the datastructure.
MyTree mytree                  // a pointer to the extended tree, representing the CheckList
Private vector cle             // a vector with all the checklist elements
Int bInt
```

Methods:

```
GetSize(), getIndexOf(CLEelem cl), CLEelem getElemAt(int I)
Void addElement(String s, int l, int t, int I)
GetText()
GetLevel()
GetTable(int I)
GetIndex(int I)
Boolean isDF()
Void setIndexAt(int I, int val)
Void insert(CLEelem ny, CLEelem oe)
Void delete(CLEelem gone)
Void cleanup()
Void update(int nInd, int Oind, int tableP, String s)
```

If the index in a table changes, this function will update the tree data.

```
Void print2File(..)
Void buildTree(root)
Int myBuild(..)
Void Analyse(..)
```

reads the checklist from file.

4.3.3.2 Class CLEelem

This class contains an actual class list element.

Data:

```
static int ISDF = 1            // value to check leaf type
static int ISDR = 2            // against
private String text            // text shown on screen
private int level              // level in tree
private int tInd               // type of leave (-1 = checklist)
private int index              // index when more data in table
private myDynamicTreeNode tn   // pointer to java-treenode
```

Methods:

Two constructors, one for use during runtime (no parameter) and one for when all information is known, like when reading from file.

`GetX, setX` for all items.

`Boolean isDF()`: true when a leaf is a DesignFactor.

`Boolean isDR()`: not needed now as the relations are no longer in the tree.

4.4 Classes of the user interface

4.4.1 Main Interface

The main interface is in the form of a tree, using the Swing tree elements with some extensions. This structure shows the checklist and all the design factors.

4.4.1.1 Class MyTree

`Class MyTree` is the top level in this group of classes with the rest for special operations.

`Class MyTreeCellRenderer` displays the data, making a difference between checklist elements and the design factors.

`Class MyTreeData` for the information shown – and to avoid putting UI elements in the method data structure – colour, font and CLEelem is kept.

`Class MyTreeModel` to get the desired tree.

`Class MyDynamicTreeNode` to keep the index and know what is what. Boolean isDF() is one of the methods.

`Class MyComboBox`

`Class myDRListModel`
A class to simplify making pulldown menus with Design Factors. The pull-down menus are in use by the DesignRelationUI to choose DesignFactor From and To. The methods are all for getting the right information from an element.

`Class myDRFListModel`
For the From Factors. This is a possible place for extension. If there are many DesignFactors, it would be better if the selection showed the most likely towards the top or used another kind of selection interface.

`Class MyDR2ListModel`
For the To factors. Same comment as for From Factors applies.

4.4.1.2 Class DesignFactorUI

`DesignFactorUI`
This class contains methods for handling simple entering and changing of DesignFactors. The constructor is used to generate the interface, to add, or modify use fillDFFrame(..). The constructor and fillDFFrame are the only methods of importance to the outside world.

`ExtDFUI`
This class is used for showing one Designfactor with all relevant information. All DesignFactors having a relationship with the DesignFactor in question, are shown. The DesignFactor can be modified, all DesignRelationships can be inspected/changed and of course the other DesignFactors can be brought up for inspection or modification as well.

`DFVersion`
This class is used for showing a previous version of a DesignFactor. The past cannot be altered. It is, however, possible to get the “previous” of the previous back to the first version.

`DFFieldselUI`
This class is used to select which DesingFactors and which data items to show. The criteria are stored and showed as well.

`Class DFSelDisp`
Class DFSelDisp displays the selected values and the selection criteria.

4.4.1.3 Class DesignRelationUI

`Class DRFieldSelUI` interface for selecting which relations and which fields to show, and storage for the selected criteria.

`Class DRSelDisp` displays the selected values and the selection criteria.

4.4.2 Query interface

The queries are handled with one data class, QueryT, and two classes for the user interface per datatype/class queried. For the DesignFactors the classes are DFFieldSelUI for the actual querying and DFSelDisp for showing the result. The DesignRelations are handled with DRFieldSelUI for the querying and DRSelDisp for the showing.

4.4.2.1 Class QueryT

Data:

```
private  String atr,      // which attribute
private String  txt;      // value chosen
private int num;          // number left in selection
```

Methods:
A constructor and getX
`Public String getNumLeftS()` returning the number left as a string return new String(" " + num) ; }

4.4.2.2 Class DFFieldSelUI

See User Interface / DesignFactor

4.4.2.3 Class DFSelDisp

See User Interface / DesignFactor

4.4.2.4 Class DRFieldSelUI

See User Interface / DesignRelation

4.4.2.5 Class DRSelDisp

See User Interface / DesignRelation

4.5 Service classes

4.5.1 Class Tool

This one should qualify as a service class as its only purpose is to call everything else and provide a link where necessary. For instance, when checking the legality of the DesignRelationTypes, it might be necessary to retrieve the DesignFactors to actually do the checking.

Data:
`Tool declares everything Echo is using.`

Methods:
GetX and setX and a call for every functionality in Echo.
Important are a few update functions, to be used when something has been changed/added to ensure it turns up in other parts of Echo as well.

`Public boolean updateNDF(DesignFactor df, int indN)`: update tree with a new DesignFactor.
`Public boolean update(DesignFactor df, int indN)`: update tree when a DesignFactor label has been changed.
`Public void makeFrame()`: makes three frames and shows one of them – the main one. The others are for data entry (DesignFactor, DesignRelation).
`Public void readFromFile(String fileName, myIO my)`: open file, make tokenizer, analyse file with Method, DesignfactorTable, DesignRelationTable, Tree, Session.
`Public void DRAddType(String t)`: Does not do anything, but worth contemplating for a following release.
`Public void add2Met(int i, String t) )`: Does not do anything, but worth contemplating for a following release.

4.5.2 Class MyIO

This is a collection of useful IO routines:
```
Static DataInputStream openInFile(String filename)
Static void closeInFile(DataInputStream in)
Static DataOutputStream openOutFile(String filename)
Static DataOutputStream openOutFile(String filename)
```
`Static String getCompleteToken(StringTokenizer st,String begin, String end)`: returns the next string between the begin and end sequence. If a # is encountered before begin, everything up to and including the next # is not considered.
`Public int getNumber(String numString)` converts a number in String format to an int. It returns the trimmed version.

4.5.3 Class DITFile Choser

During a session at least two files must be opened, one for reading data at the beginning and one to save the work at the end. Saves in between are optional. If no file is written towards the end, nothing is saved. From this, the idea of the DIT-file cluster came. The user can either select an existing file without any danger of typing errors, or can give in a new name. It is possible to traverse the whole directory tree on the machine, all with one call: `DITFileChoser Choser = new DITFileChoser().` In the file, extensive documentation can be found. For example, public class DITFileView extends FileView - how the files are shown - has a number of methods in the file.

There are two constructors as the Swing classes behaved in a different way under Windows 98 and Windows NT. Both work in both cases, but the constructor with a string parameter will double-check if the user gives an illegal value (no filename). This can happen on the NT if the user modifies a name. That is ok under Windows 98; on the NT, the name must be given in its entirety.

The constructors do most of the work to check illegal input from the user:
```
Public DITFileChoser() filter options *.dit, *tpl, all
Public DITFileChoser(String s) filter options *.dit, all
```

To get information on the user's choice:

```
Public File getDITFile()
Public String getName(File f)
Public String getFullName(File f)
```

Supporting classes:

```
Public class DITFileFilter extends FileFilter
```

4.6 Data file

4.6.1 General explanation

The idea was to tailor the data structure to the needs; a screen could then be build-up, depending on the data and showing only what is relevant. This idea, however, has not been implemented, because the programmer could only use an old and slow computer. We have thus chosen to avoid unnecessary processing by building up as few screens as possible, only at the beginning of their use. Ease of implementation and reuse of elementary reading modules, were ruling factors for making implementation choices. The current format grew as the prototyping continued.

The data-file includes the following sets: attributes and predefined values, design factors, design relations, the checklist, and sessions. The data-file includes thus the sets of a template, namely the set 'attributes and predefined values' and the set 'the checklist'. This is done to avoid inconsistencies between data and template. After a template is chosen at the beginning of a design project, the chosen template is stored in the data-file. A template has been given the extension *.tpl; a data-file has been given the extension *.dit, though any other extension could be used, it is just a convention for the user. To change the extensions, look at the DITFileChoser (it sets the filter). In next versions, sets can be added to the data-file at the end or between sets; for example, syntax for checking design relations could be added. Below, each of the current sets is explained in more detail. In Section 4.6.2, an example of each of these sets is given.

Attributes and predefined values:

```
BeginCat [ Category label ]     // BeginCat indicates a new Category; the category text is
                                // enclosed by the square brackets.
BeginSub [ Sub-category label ]// BeginSub indicates the  beginning of a subcategory; the
                                // subcategory heading is enclosed by the square brackets,
< sub-category option label > // enclosed in <>, a sub category text.
                               // < this can be repeated for all possible texts for this sub cat >
< EndSub > [ Sub-category label ] // EndSub within <> is the end of one sub category.
                               // To help one patching the file, the sub category follows
                               // within square brackets. When a user interface comes for
                               // this part, the sub category heading becomes redundant.
                               // This can be repeated for all sub categories falling under the
                               // category at the beginning.
EndCat [Category label ]       // marks the end of one category. Again, the category text
                               // within [] is added to help the patcher.
                               // This should be repeated for all categories.
*END_METHODE*                   // Marks the end of the template
```

DesignFactors:

```
*BeginDF*
[ label ] < date: mm/dd/yy hh:mm:ss or –, if the information has not been entered yet >
 < text > < date, format see first line >
 < rationale > < date>
 < document reference/link > < date >
 < DESIGN PROCESS > < date >   // where in tree
< 1 >                          // status
```

```
< 63 >                          // numerical key/index
 ,  < category label > < date >
 ,  < subcategory option > < date >
 ,  < subcategory option > < date > // continues for the necessary number of
                                // sub-categories, empty ones are , < - > , < - >
*DF_PREV*                       // indicates that the following is a previous version of the DF
                                // – same format as above. The previous version of the
                                // previous version, is marked/stated in the same way.
*EndDF*                         // indicates the end of this DF
*END_DESIGNFACTORTABLE*         // marks the end of the DesignFactors
```

DesignRelations:

```
 *BeginDR*
 [ label ] < date format mm/dd/yy hh:mm:ss or - >
 < tekst > < date  >
 < rationale > < date  >
 < document reference/link > < - >
 < - > < - >                    // tree reference, not in use
 < 1 >                          // status
 < 1 >                          // from DF numerical key
 < 1 >                          // to DF numerical key
 < 1 >                          // relation type
*DR_PREV*                       // indicates that the following is a previous version of the DR
                                // – same format as above. The previous version of the
                                // previous version, is marked/stated in the same way.
*EndDR*                         // indicates the end of this DesignRelation.
*END_DESIGNRELATIONTABLE*       // marks the end of the DesignRelations
```

The CheckList:

```
*BeginCLL*                      // marks the beginning of the checklist
 < Check List topic >           // label in tree
 < 0 >                          // level in tree; a positive number
 < 0 >                          // table indicator
 < 0 >                          // index in table. The two last zeroes only take on other values
                                // for data added through the use of Echo.
< *EndCLL* >                    // indicates the end of the checklist
```

The Sessions:

```
 *BeginSESSION*
 <  Test Session 1 >            // name of session
 < mm/dd/yy hh:mm:ss >          // begin time
 < mm/dd/yy hh:mm:ss >          // end time
 < 0 >                          // session number
 < 1 >                          // status: actual/obsolete
                                // All sessions have the same format, so repeat until the last
                                // one has been written/read
< *EndSESSION* >                // end of session data
```

Note that there must be a blank on each side of any delimiter – like [,] , < and >. Which delimiter is used, is determined by the call for next sequence/string in the analyse function in the various classes.

4.6.2 Example

```
BeginCat [ product to be designed ]

BeginSub [ stadium ]
< desired property >
< current property >
< current factor >
< trend >
< EndSub > [ stadium ]
BeginSub [ fixation ]
< negotiable >
< not-negotiable >
< EndSub > [ fixation ]
BeginSub [ level ]
< site, scale 1/100 >
< garden house, scale 1/50 >
< components, scale 1/20 >
< details, scale 1/5 >
< EndSub > [ level ]
BeginSub [ view ]
< usability >
< make ability >
< durability >
< EndSub > [ view ]
BeginSub [ moment ]
< past >
< present >
< future >
< EndSub > [ moment ]
EndCat [ product to be designed ]

BeginCat [ design process ]
      analogous
EndCat [ design process ]

BeginCat [ design context ]
      analogous
EndCat [ design context ]

*END_METHODE*
 *BeginDF*
 [  duration ] < 3/15/00 11:16:0  >
 <  maximum 2 months > < 3/16/00 15:46:53  >
 <  budget > < 3/15/00 11:16:27  >
 <  project proposal > < 3/15/00 11:16:22  >
 <  maximum duration > < 3/15/00 11:17:3  >
 < 1 >
 < 4 >
 ,  <  design process > < 3/16/00 15:46:36  >
 ,  <  desired property > < 3/16/00 15:46:36  >
 ,  <  negotiable > < 3/16/00 15:46:36  >
 ,  <  future > < 3/15/00 11:16:58  >
 ,  <  present > < 3/15/00 11:17:0  >
 ,  <  present > < 3/16/00 15:46:36  >
 ,  <  design process > < 3/15/00 15:16:28  >
 ,  <  design process > < 3/15/00 15:16:28  >
 ,  <  means > < 3/15/00 15:16:28  >
 ,  <  future > < 3/15/00 15:16:28  >
 ,  <  present > < 3/15/00 15:16:28  >
```

```
 ,  <  present > < 3/15/00 15:16:28  >
 *DF_PREV*
 *BeginDF*
 [  duration ] < 3/15/00 11:16:0  >
 <  maximum 3 months > < 3/15/00 11:16:10  >
 <  budget > < 3/15/00 11:16:27  >
 <  project proposal > < 3/15/00 11:16:22  >
 <  maximum duration > < 3/15/00 11:17:3  >
 < 1 >
 < 4 >
 *EndDF*

*BeginDF*
 [  budget ] < 3/15/00 11:17:21  >
 <  $25 000 > < 3/16/00 15:45:15  >
 <  given by director > < 3/15/00 11:18:57  >
 <  project proposal > < 3/15/00 11:17:39  >
 <  budget > < 3/15/00 11:17:48  >
 < 1 >
 < 5 >
 ,  <  design process > < 3/16/00 15:45:30  >
 ,  <  desired property > < 3/16/00 15:45:30  >
 ,  <  negotiable > < 3/16/00 15:45:30  >
 ,  <  design process > < 3/16/00 15:45:30  >
 ,  <  means > < 3/16/00 15:45:30  >
 ,  <  present > < 3/16/00 15:45:30  >
 ,  <  design process > < 3/15/00 11:18:36  >
 ,  <  desired property > < 3/15/00 11:18:36  >
 ,  <  negotiable > < 3/15/00 11:18:36  >
 ,  <  design process > < 3/15/00 11:18:36  >
 ,  <  means > < 3/15/00 11:18:36  >
 ,  <  present > < 3/15/00 11:18:36  >
 *DF_PREV*
 *BeginDF*
 [  budget ] < 3/15/00 11:17:21  >
 <  $30 000 > < 3/15/00 11:17:30  >
 < - >  < - >
 <  project proposal > < 3/15/00 11:17:39  >
 <  budget > < 3/15/00 11:17:48  >
 < 1 >
 < 5 >
 *EndDF*
*END_DESIGNFACTORTABLE*

*BeginDR*
 [  production machines ] < 3/15/00 11:53:47  >
 <  no text > < 3/15/00 11:54:1  >
 < - >  < - >
 < - >  < - >
 < - >  < - >
 < 1 >
 < 15 >
 < 29 >
 < 1 >
 *EndDR*
 *BeginDR*
 [  support ] < 3/15/00 11:55:54  >
 < - >  < - >
 <  With more support, the duration of the design process can be shorter >
< 3/15/00 11:56:35  >
 < - >  < - >
```

```
 < - >  < - >
 < 1 >
 < 10 >
 < 4 >
 < 4 >
 *EndDR*
*END_DESIGNRELATIONTABLE*

 *BeginCLL*
 <  PROPERTIES OF THE DESIGN TASK >
 < 0 >
 < 0 >
 < 0 >
 <  DESIGN PROCESS >
 < 1 >
 < 0 >
 < 0 >
 <  Desired state of the design process >
 < 2 >
 < 0 >
 < 0 >
 <  maximum duration >
 < 3 >
 < 0 >
 < 0 >
 <  budget         >
 < 3 >
 < 0 >
 < 0 >
 <  capacity of the design team >
 < 3 >
 < 0 >
 < 0 >
 <  Current state of the design process >
 < 2 >
 < 0 >
 < 0 >
 <  duration         >
 < 3 >
 < 0 >
 < 0 >
 <  members of the design team >
 < 3 >
 < 0 >
 < 0 >
 <  budget         >
 < 3 >
 < 0 >
 < 0 >
 <  support         >
 < 3 >
 < 0 >
 < 0 >
 <  PRODUCT TO BE DESIGNED >
            analogous
 <  FACTORS IN THE CONTEXT INFLUENCING THE DESIGN TASK >
            analogous
< *EndCLL* >

*BeginSESSION*
 <  input >
```

```
 < 3/15/00 11:6:21  >
 < 3/15/00 12:1:41  >
 < 0 >
 < 1 >
 <  test 1 >
 < 3/15/00 12:3:41  >
 < 3/15/00 12:8:25  >
 < 1 >
 < 1 >
 <  test 2 >
 < 3/15/00 15:15:18  >
 < 3/15/00 15:49:15  >
 < 2 >
 < 1 >
< *EndSESSION* >
```

4.7 About Java

The Java Environment includes nice documentation and archive possibilities. In the next subsections, a quick overview for rank beginners is given.

4.7.1 Running Java

To run: `Java program`
If this does not work, try to set CLASSPATH and PATH.
Example PC:

```
set CLASSPATH=".;Tool.jar;c:\PROGRAM
FILES\JavaSoft\jre\1.2\lib\rt.jar"
set PATH="c:\PROGRAM FILES\JavaSoft\jre\1.2\bin"
```

Example Unix, adding to the previous values:

```
Setenv CLASSPATH .:/usr/local/bin/java:$CLASSPATH
```

4.7.2 Archive possibilities

To extract the contents of a JAR file: `Jar xf jar-file`
To view contents of a JAR file: `Jar tf jar-file`
To create a JAR file: `Jar cf jar-file files-to-be-archived`
To run an application packaged as a JAR file :

Option 1 : use a batch file. Example:

```
batchfile = Echo.bat :
set CLASSPATH=".;Tool.jar;c:\PROGRAM
FILES\JavaSoft\jre\1.2\lib\rt.jar"
set PATH="c:\PROGRAM FILES\JavaSoft\jre\1.2\bin"
:: set CLASSPATH="."
java Tool
```

to actual run: Echo

Option 2 : set CLASSPATH manually and use the java environment:

`Java –jar app.jar`

4.7.3 Documentation possibilities: Javadoc

To create the documentation: `javadoc -d ..\Doc\Html *.java`
The –d gives the directory where the output is placed, *.java means document everything. Javadoc will take any file it considers legal. That means it must be specified if a class is public or private. The default (not given) makes javadoc skip the whole file.

Javadoc creates
- an index,
- a file for every class,
- an overview of the class hierarchy (a tree),
- a deprecated list,
- an index of all functions, and
- a help-file.

They all have extension .html.

To see the documentation: use a browser like Netscape or Internet Explorer. In the next subsections, each type of html file created by Javadoc is discussed briefly.

4.7.3.1 Index.html

The resulting screen starts with some menu options: Class Tree Deprecated Index Help. These are the links to the different files created by Javadoc.

4.7.3.2 Class.html

The following menu options are shown:

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

With FRAMES, an index of the classes is shown in a frame on the left side of the screen: It is a cropped version of classes-frame.html. Below, some examples of classes, the summary of a constructor, the summary of a method, the detail of a constructor, and the detail of method are given.

All Classes
CLEelem
CLL
DesignFactor
…

The details of the first class are, for example:
Class CLEelem
java.lang.Object
|
+--**CLEelem**

---

public class **CLEelem**
extends java.lang.Object
Check list element - later called "topic" on screen. Contains the text shown in the tree, information on what kind of element it is (original Checklist topic or a DesignFactor), level in the tree, and a pointer to the corresponding tree node. The order in which the topics are stored in the CLL (CheckList) determines the order in the tree.

Constructor Summary	
CLEelem() default constructor - when nothing is known	
CLEelem(java.lang.String s, int l, int t, int i) constructor when all info is known, i.e. when reading data from file	

Method Summary	
Int	**getIndex**()
Int	**getLevel**()

Int	**getTable**()
Java.lang.String	**getText**()
MyDynamicTreeNode	**getTreeNode**()
Boolean	**isDF**()
Boolean	**isDR**()
Void	**setIndex**(int val)
Void	**setString**(java.lang.String newString)
Void	**setText**(java.lang.String s)
Void	**setTreeNode**(myDynamicTreeNode tnP)
Java.lang.String	**string**()
Java.lang.String	**toString**()

Methods inherited from class java.lang.Object
Clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

CLEelem
public **CLEelem**()
default constructor - when nothing is known

CLEelem
public **CLEelem**(java.lang.String s,
int l,
int t,
int i)
constructor when all info is known, i.e. when reading data from file

Method Detail

getText
public java.lang.String **getText**()

---

getLevel
public int **getLevel**()

---

getTable
public int **getTable**()

---

getIndex
public int **getIndex**()

---

isDF
public boolean **isDF**()

---

isDR
public boolean **isDR**()

---

setIndex
public void **setIndex**(int val)

---

setText
public void **setText**(java.lang.String s)

---

setTreeNode
public void **setTreeNode**(myDynamicTreeNode tnP)

---

getTreeNode
public myDynamicTreeNode **getTreeNode**()

---

setString
public void **setString**(java.lang.String newString)

---

string
public java.lang.String **string**()

---

toString

public java.lang.String **toString**()

Overrides:
toString in class java.lang.Object

4.7.3.3 Tree

Hierarchy For All Packages
Class Hierarchy

- class java.lang.Object
  - class javax.swing.AbstractListModel (implements javax.swing.ListModel, java.io.Serializable)
    - class javax.swing.DefaultListModel
      - class **myDRListModel**
        - class **myDR2ListModel**
        - class **myDRFListModel**
  - class **CLEelem**
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)

4.7.3.4 Deprecated

Empty at the moment – no warnings.

4.7.3.5 Index

A complete alphabetic index of all methods and classes

A

accept(File) - Method in class DITFileFilter

Returns true if this file should be shown in the directory pane, false if it shouldn't.

actionPerformed(ActionEvent) - Method in class DesignFactorUI.B1L

actionPerformed(ActionEvent) - Method in class DesignFactorUI.B2L

actionPerformed(ActionEvent) - Method in class DesignFactorUI.B3L

actionPerformed(ActionEvent) - Method in class DesignFactorUI.lT1L

4.7.3.6 Help

Help-doc.htm gives a general explanation on how the documentation is organised.

5 Conclusions

This chapter starts, in Section 5.1, with feedback on the software tool, given by practising designers and by the programmer of Echo. The chapter continues with a comparison of the software tool and the design method. Remember that the prototype software tool has been developed to explore the benefits of using a software system to facilitate the use of the design method. Section 5.3 describes possible improvements and extensions of the prototype. The chapter ends with some general conclusions.

5.1 Feedback on the software tool

Echo has been confronted with the design practice to get feedback on its concepts and its implementation. Expert designers have given comments on the concepts of the prototype. Junior designers have given feedback, after using the prototype, on the concepts and the implementation of the prototype. More about this empirical research can be found in Section 1.3 and Chapter 2 of this report and in [Reymen, 2001a] or [Reymen, 2001b]. Also the programmer of Echo has given some comments on the implementation of the prototype.

5.1.1 Feedback of the expert designers

Suggestions from the expert designers to extend the prototype for use in practice are the following:

- The tool could be extended to support *learning*. Van der Sanden mentions that learning from earlier experiences is informal in practice: Existing support concentrates on development (how to design), modelling (how to represent reality), and management (how to control a process). Brouwer says that the biggest problem of a tool supporting learning and reflection is how to make people think about certain aspects and not only mention the aspects.
- The tool can be extended for *process support*. Döll qualifies process support as highly useful. For architecture, a link can be made to an online quality handbook. The idea of a checklist can be elaborated; with software, a checklist can easily be made specific for a project (copy and paste of a general checklist). The checklist could show what must be done.
- The tool could be extended for *concurrent engineering*. Koster sees possibilities for all parties to retrieve information on the project from the database of the tool. He adds that the tool must be tested in complex projects involving multiple disciplines.
- The tool could be extended for *documentation and control*. Van der Sanden mentions possible items for documentation: important decisions, ideas, user opinions, planned actions, risks (can be part of a checklist), and links to data of the product being designed. The tool could check factors influencing a goal and check that no factor is overlooked. He suggests starting with unstructured data in the tool; when the amount of data increases, some kind of structure can be added. He also mentions that a user must familiarise himself with the tool by means of a training; once familiar, the user can give input very fast and the ideas of the user can be checked very fast. A user also needs a training in choosing terminology. Döll, however, mentions that during a design process, millions of things are important. All decisions cannot be described and underpinned; many decisions are taken implicitly, only a few are taken explicitly. For Döll, an important requirement is that information can be entered very fast. Kruithof says that designers will not be motivated to use this kind of tool in practice if it is forced. Technicians like to make their own tool and they already think in an abstract way. The most important links between facts are already in their heads. In his perception, just entering data in a tool, does not offer a designer much added value; there is also a natural resistance against entering a lot of data. Only when a designer has experienced certain problems, will such a tool be appreciated; a designer sees the necessity to document relations only after overlooking an important relationship.

The designers made these suggestions based on the current prototype, which only supports the description and analysis of design situations. So far, it does not support a reflection process. Our suggestion is to extend and improve the prototype for reflection (and learning), as described in Section 5.3. Some kind of process support can be made part of the support for reflection. The suggestions about concurrent engineering and documentation and control seem useful to us and can also be taken into account when developing a new version of the prototype.

5.1.2 Feedback of the junior designers

5.1.2.1 Feedback on the general concepts

The junior designers found that the prototype had potential value; however, the tool has not reached a useful form yet. They mention that, for the further development of the tool, the following important questions must be answered:

- How can the goal of the tool, namely support reflection on a design process, be reached without developing a management tool (a management tool aiming to control and monitor the design process)?
- Will the tool become a group (reviews) or individual (reflection on own design process) tool? Will it be possible to see reflections of other designers? The answers are important because documentation made by a designer for himself and documentation made for others are fundamentally different (there is less argumentation and more detail in personal documentation).
- Is it necessary to couple administration of the design process and reflection?
- Is automating the design method a step forward? Computer support is not necessary for reflection.

Answers to these questions are (partly) given in Section 5.3, which discusses possible improvements and extensions of the prototype.

On several aspects of the current prototype, there was no consensus between the designers. Some designers mention that the tool makes it possible to get an overview, others describe that the dimensions of a computer screen are too small to make it possible to get a good overview. It was also said that, on the one hand, a good insight into the relations could be obtained with the prototype, whereas on the other hand, designers mention that the prototype does not contain a specific structure for relations and does not give a good overview of relations. The proportion of the time spent using the tool and the time spent designing, was judged both as reasonable and as too much.

The designers did agree on the following advantages and disadvantages of the tool. The tool simplifies administration and thereby prevents overlooking important aspects. Furthermore, the tree structure is useful, as are the predefined values. Disadvantages mentioned by the designers are that the tool requires too much work, the user interface is unclear, and the prototype is textual, whereas a designer prefers to work graphically.

The prototype is more flexible than the paper-based version of the design method. For some designers, the lack of overview makes the prototype less useful than the paper-based version. Compared to handwriting, the information is better readable on a computer, but there are limitations to the number of characters. Data can be edited more easily, but some designers judged navigating through the data as more difficult.

5.1.2.2 Feedback on the implementation

The designers suggested the following extensions and improvements for the implementation:

- an easy-to-use user interface (including more graphical visualisation (of relations) and functionality for reflection), for example:
  - add concrete questions in the tree;
  - add the possibility to copy, paste, drag, and drop properties, factors, and relations (as functions of the right mouse button) in the tree;

  - make it possible to expand the tree by pushing ‘enter’ and to close the tree by pushing ‘escape’;
  - when the user double clicks on a property or factor in the tree, the Detail DF window appears;
  - limit the possibilities of the predefined values in the electronic forms, based on the position of the properties and factors in the tree because the list of predefined values in the electronic forms will become too long after using the prototype for a longer time;
  - implement the actual/obsolete-attribute with a radio button;
  - make operation possible completely with keyboard and/or completely with the mouse.
- possible to change the template during a design process;
- import/export facilities for text documents, drawings, and animations;
- import/export facilities for existing tools;
- when using the prototype in current design practice, product and process data will be stored at least twice: in a domain-specific tool (compiler, CAD tool, etc.) and/or a project-management system and in the database of the prototype: It is desirable to separate the database from the prototype, thus allowing a common database for different tools.

5.1.3 Feedback of the programmer

Implementation oriented comments have been given by the programmer during the development of the prototype. She mentioned that a second version of the prototype, which has not been implemented yet, could include:

- software build for one hardware or machine type, for example, a PC. This could result is shorter development time (as a consequence, however, the software will be less general);
- interfaces with other software products, for instance to extract information entered elsewhere;
- multi-user support. This might indicate another language, probably, a database-based one. The latter could also make the storage of the data more efficient;
- attributes and predefined values of relations in the template to increase flexibility and reduce the number of options on the screen (in the relevant pull-down menu’s). The question is to find a good specification that will also enable the check specification of relation types;
- a possibility to define the template more than two levels deep;
- a better user interface for the definition of the factors of a relation;
- a user interface that supports the following: when an item is selected in a certain screen, it is selected in every screen;
- a user interface to define and edit a template;
- a user interface that supports the preferences of the user;
- a multi dimensional visualisation of the data;
- fixed dimensions of the windows or screens.

5.2 Comparing the software tool and the design method

The goal of the design method is to offer designers a domain-independent aid that supports reflection on design processes in a structured way. The design method supports the preparation step of a reflection process by providing designers some aid to describe and analyse design situations and design activities at the beginning and end of design situations. The concrete use of the design method can be realised in several ways: descriptions can be made on paper or a computer can assist the user. Echo has been developed to explore the benefits of using a software system to facilitate the use of the design method. In Echo, so far, only the description and analysis of design situations is supported. The prototype can be used during the third and fifth step of the design method for describing and analysing a design situation. To *describe a design situation*, a user can add properties and factors in the tree, describe properties, factors and relations more precisely by using the electronic forms, and change values for the different attributes in the electronic forms. To *analyse a design situation*, a designer can query information and check design relations. Echo is domain independent, but the concept of

templates makes it possible to tailor the use of the design method to the terminology and concepts of a specific discipline.

The current prototype was useful for exploring the advantages, the disadvantages, and the possibilities of using a software system to facilitate the use of the design method. Using the prototype for describing and analysing design situations has the following advantages compared to using pencil and paper:

- The administration of properties, factors, and relations is more flexible;
- electronic information can be edited more easily;
- relations between properties and factors can be visualised graphically (although not graphically in the current prototype);
- searching for information and comparing specific overviews of the information is easy;
- some simple consistency checking is possible;
- the prototype can easily be tailored to the needs of a specific discipline, project, or user because the attributes in the forms and the structure and questions of the checklist can be changed easily in the template.

A disadvantage of the software system is that information that is possibly already stored in other databases must be entered again; this is not time efficient and it does not motivate the users. Other disadvantages are specific for the current version of the prototype and can (easily) be eliminated in further versions, as explained in the next section.

5.3 Possible improvements and extensions

The current implementation of Echo can be improved in a technical way. The prototype can also be extended to support all concepts of the design method. In addition, the prototype can be extended with useful concepts that are not (yet) included in the design method.

Echo can be improved in a technical way as follows. The general *user interface* can be improved in the sense that the standard machine interaction can be extended with speech technology. A user interface for defining and editing a *template*, even during a design session, could be very useful. A user must also be supported when composing a new template from parts of other templates. The *tree structure* could be extended with more editing facilities. The changing of the position of properties and factors in the tree could be improved. The user interface of the tree can be improved further by using different colours for actual and obsolete items and by attaching questions to the topics in the checklist (these questions could, for example, also be refined automatically when more detail is needed). The use of the *electronic forms* could be made more user friendly; for example, for the ‘reference’ attribute in the electronic forms, a viewer could be included to open the documents referred to (images, drawings) in other applications. The user interface of the *query* output could be improved to stimulate the analysis of overviews, could be improved with a graphical user interface that shows relations between properties and factors, and could be extended with a document-generation and a printing facility.

Echo, currently only including concepts for describing and analysing a design situation, could be extended to support all concepts of the design method as follows. The prototype could be extended to support also the description and analysis of design activities and to include the concept of design sessions. The tool could, for example, support the comparison of planned and executed design activities; a user could then check if everything has been executed according to plan. The tool could also support the comparison of specific descriptions and analyses of different design-team members and designers in specific disciplines. This kind of comparison may result in differences between designers and can thus be a starting point to improve communication between designers having a different background. *Queries* could, for example, support qualitative and quantitative analyses; these are analyses of the values for the different attributes of properties and factors, namely numbers (quantitative) or, for example, aesthetic and ergonomic values (qualitative). The *checking mechanism* could be extended to check the consistency of the data by, for example, checking the uniqueness of

items (taking into account near doubles). Other syntactic and perhaps also some semantic consistency checks could be added, depending on the need and the required implementation effort. Some simple ‘requirements tracing’ could be added (for example, a check that verifies whether all desired properties are linked to a solution). Perhaps, also some progress checking could be done. The general *user interface* of the tool should stimulate serious and thorough reflection.

Echo could be extended with concepts beyond the design method as follows. The current concept of the database could be improved and extended in many possible ways. One possibility is to combine information from different sources (text documents, CAD) and databases (of other tools) in the database of the tool. It is also possible to develop a database (or use an existing one) that is not exclusively used by the tool, but that could be shared by many applications. Another extension of Echo would be to allow sharing of (part of) the information in the database by multiple users; for example, all participants of a project could share the information of the design context and of related design tasks. Important questions are then “Which data of a common database may be received, added, and changed by the users of the tool?” and “Who determines what is important for the different users of the common database (the creator of the data or the receiver of the data)?”. To support exchange of information all over the world, collaborative-engineering and groupware systems, which manage distributed and co-operative work, could be linked to the tool. The tool may also include a configuration and version management system; these systems are especially useful when large amounts of data must be stored. (The extra attributes introduced in the prototype are already placeholders for configuration and version management.)

Because the tool will offer only support for reflection and not for specific design activities, it must be used in co-operation with domain-specific tools. Currently used tools could be linked to Echo or Echo could be linked to one or more of the currently used tools (like a module that can be added to or is embedded in a CAD-system). The tool could also be linked with other general tools that support, for example, evaluation techniques, creativity techniques, or selection techniques. The tool could also be coupled to systems that have been developed for re-use of design information; this is especially useful for projects in almost the same design context. A design-history system stores the steps made in a design process, as well as the decisions and their rationale, and make these available for re-use. A case-based-reasoning system offers information on previous projects (cases) for use during the design process of a new product. More research is necessary to determine what kind of interaction with what kind of system is desired. It is also important to look at the latest techniques and possibilities in the field of software development. The tool must combine the benefits of the design method with the benefits of the medium (software).

For practical use, the tool must be tailored to a specific design environment. For example, to support communication in multidisciplinary design teams, the tool could be extended with a translation function. Individual designers could perform (part) of a reflection process with the tool, using their own terminology and the general concepts of the tool. The tool could then (1) translate the specific terminology of one designer to the general terminology and (2) translate the general terminology to the specific terminology of other designers, as illustrated in Figure 5.1. Comparison of the reflection processes of several designers could then be understood by each designer in his own language. For developing this translation function, specific terminology in several disciplines/domains should be studied.

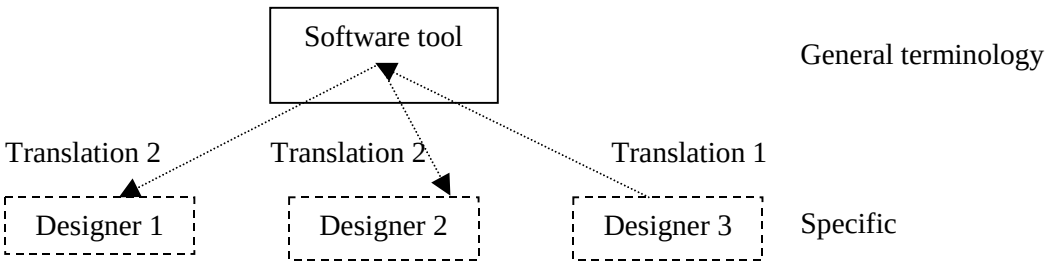


Figure 5.1: Translation function of a software tool.

5.4 General conclusions

Echo is one of many possible implementations of a software tool facilitating the use of the developed design method; it does not include all concepts of the design method yet. The current prototype is not useful in the world of designing yet, but was useful for exploring the advantages, the disadvantages, and the possibilities of using a software system to facilitate the use of the design method. We suggest that, before making a new version of the prototype, the design method should be improved and extended: first, the method should be extended to support the other two steps of a reflection process, namely image forming and conclusion drawing; second, the implementation of the method must be easy to tailor to a specific discipline; and finally, the multi user aspects should be addressed and method must be made useful for design teams. Based on the improved version of the method, the goal of software support must again be defined and related literature should be studied (to compare with existing tools). Finally, a new prototype could be developed. Because it will differ very much from the current version, we advise to start from scratch and decide on one hardware type. Current concepts of the prototype that proved to be useful could, however, be incorporated in a new version. For example, the current idea of templates seems suitable for making a tool domain and project specific.

Summarising, this report offers documentation for using the software and for improving and/or extending the prototype. We hope that the reader can find enough information to understand the concepts of Echo, to use the software, and/or to create a new version of the prototype.

References

Reymen, I.M.M.J. (2001a) *Improving Design Processes through Structured Reflection: A Domain-independent Approach*, Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands.

Reymen, I.M.M.J. (2001b) *Improving Design Processes through Structured Reflection*: A *Domain-independent Approach*, SAI Report 2001/1, Stan Ackermans Institute, Technische Universiteit Eindhoven, Eindhoven, The Netherlands.

SAI Reports

ISSN 1570-0143

In this series appeared:

2001/1 **Improving Design Processes through Structured Reflection: A Domain-independent Approach.** Ph.D. thesis
Isabelle M.M.J. Reymen

2001/2 **Improving Design Processes through Structured Reflection: A Prototype Software Tool.**
Isabelle M.M.J. Reymen, Elisabeth Melby