Separation of the User Interface and Application
Separation of the User Interface and Application

Proefschrift

ter verkrijging van de graad van doctor aan de Technische Universiteit Delft, op gezag van de Rector Magnificus, prof. drs. P.A. Schenck, in het openbaar te verdedigen ten overstaan van een commissie aangewezen door het College van Dekanen op dinsdag 1 oktober 1991 te 14.00 uur

door

Johannes Martinus Versendaal

geboren te Alblaserdam, informatica ingenieur
Dit proefschrift is goedgekeurd door de promotoren:


en door de leden van de promotiecommissie:

prof. dr. J.H.T.H. Andriessen
prof. dr. ir. R. den Buurman
prof. dr. J.D. Foley
prof. dr. D.J. McConalogue
Published and produced by
Johan Versendaal
Schielaan 46A
3043 HD Rotterdam
The Netherlands

Cover illustration designed by Gerber van der Graaf
Picture of Rosetta stone from the "British Museum Souvenir Guide",
© 1989 by the Trustees of the British Museum

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Versendaal, Johannes Martinus

Separation of the user interface and application
Johannes Martinus Versendaal. - [S.l. : s.n.]
Proefschrift Delft. - Met lit. opg.
ISBN 90-9004274-1
Trefw.: gebruikersinterfaces.

© 1991, by Johan Versendaal
No part of this book may be reproduced without written permission of the author.
## CONTENTS

Preface ix
Summary xi
Samenvatting xv

Chapter 1 Interactive Computer Systems 1
1.1 Introduction
1.2 User Interface
1.3 Methodological Aspects
1.4 Separation of the User Interface and Application
1.5 Research Questions and Outline of the Study

Chapter 2 Basic Concepts 11
2.1 Introduction
2.2 Direct Manipulation
2.3 Classes of ICSs
2.4 Research Questions Revisited

Chapter 3 An Alternative Model for ICSs 19
3.1 Introduction
3.2 The Delft Model for ICSs
3.3 Complicating Factors in the Delft Model

Chapter 4 An ICS Development Environment 41
4.1 Introduction
4.2 Input to "Design ICS"
4.3 Output of "Design ICS"
4.4 Design ICS
4.5 Output of "Implement ICS"
4.6 Implement ICS
4.7 Present User Interface and Execute Application Semantics
4.8 Conclusions
Chapter 5  Case 1: Graphics Manager  75
  5.1  Introduction
  5.2  Description
  5.3  Construction of Design Specifications
  5.4  Modifications
  5.5  Conclusions

Chapter 6  Case 2: Graphic Calculator  113
  6.1  Introduction
  6.2  Description
  6.3  Construction of Design Specifications
  6.4  Modifications
  6.5  Conclusions

Chapter 7  Conclusions  135

Appendix A  Graphics Manager’s Application Semantics Code  139

Appendix B  Graphic Calculator’s Application Semantics Code  147

References  159

List of Publications  165

Curriculum Vitae  167
My interest in user interfaces was aroused by ir. C.A.P.G. van der Mast of the Delft University of Technology; he suggested me to start an MSc-task under his guidance, with prof. ir. D.H. Wolbers as supervisor. I agreed to it and added a graphics facility in a system that helped in the design and maintenance of courseware. Having successfully finished my MSc-thesis, I was offered a PhD-place at the Delft University with prof. ir. D.H. Wolbers and prof. dr. H.G. Sol as supervisors. The subject was the search for a methodology for the design and maintenance of highly interactive systems. I gradually became aware of the extensiveness of the subject. I began to focus on User Interface Management Systems (UIMSs), which emphasize the separation of the user interface and application. I determined as main research question:

Is it possible to separate the user interface and the application in highly interactive systems?

Many people have contributed to the realization of this dissertation. The main contributors are undoubtedly prof. ir. D.H. Wolbers, prof. dr. H.G. Sol and ir. C.A.P.G. van der Mast.

Prof. Wolbers, Henk, thank you for the time you spent on reading the manuscript and discussing key issues, like "masters and slaves" with relation to UIMSs.

Prof. Sol, Henk, thank you for assisting me with research planning and for reading and discussing the manuscript again and again and again.

Ir. Van der Mast, Charles, thank you for guiding me in my research, reading and discussing the manuscript and supporting me even in hard times. May you soon finish your own dissertation too!

I want to thank the students Marco Kruijt and Willem Beekman for their fruitful ideas and discussions about the UIMS D2M2 and for their implementation of parts of D2M2. Further, I want to thank the students Hans Dinger, Jan Henriksen, Joseph Tweed, Anton Spaans and Eric Magielse for their contributions.

I am indebted to Theodoor van Donge for the Graphics Manager case; I am indebted to Joost Klep for the Graphic Calculator case.

I want to thank my English friends Jane and Steve Dearlove -nomen est omen- who improved my English writing.

A number of people have indirectly contributed to this research. I want to thank my wife, Ariëtte, and my parents. I want to thank Robbert Fokkink, Hans Hellendoorn, Roelof Koekock, Afke van Rijn, Nico Roos, René Swarttouw and Paul van der Weerd for the many pleasant hours singing and discussing many subjects.

Above all, I thank God for giving me the abilities and perseverance for this research.
SUMMARY

Interactive Computer Systems (ICSs) are designed to help people in performing tasks. The user interface of an ICS is being defined as consisting of an input language for the user, an output language for the ICS, and the definition of the causal relation between input and output. The user interface of an ICS influences the efficiency and effectiveness of performing tasks with the ICS.

The user interface of an ICS determines how human-computer interaction may be performed. We assume that human-computer interaction always consists of user input task execution, computer task execution and output task execution. A computer task (a task to be executed by the ICS) is always triggered by a user input task (a task to be initiated by the user); during or after computer task execution output tasks redirect results of computer task execution to the user.

It is to be recommended to separate the user interface and application semantics of an ICS. In the case of separation the application semantics are relieved from all details concerning input and output; the user interface part can be altered without having to change the application semantics; dialogue specialists can compose a user interface most suitable for a given environment and/or application; interaction techniques may become available as dialogue libraries; a dialogue run-time support system may provide facilities for user help, error recovery and audit-trails. Logical and physical separation can be distinguished. As for logical separation, the user interface and the application semantics do not share data and have only a limited ability to communicate with each other during run-time. As for physical separation, the specification of the user interface is in a different file (or files) than the specification of the application semantics.

In literature it is assumed that separation of the user interface and application semantics is not always possible. It is decided to aim at the following research question.

Is it possible to achieve logical and physical separation of the user interface and application semantics? Are there constraints?

We propose a model for ICSs, which enforces logical separation during run-time. Broadly, there are three types of models for ICSs: linguistic, spatial and agent models. In linguistic models human-computer interaction is treated as human-human communication: a user communicates with an ICS with a kind of language through which the user wants the ICS to perform tasks. After task execution the ICS generates output to the user. Spatial models put emphasis on data models in ICSs. Instances of the data types defined in a data model may be presented on screen. These instances may be manipulated by the user. Agent models structure an ICS into a collection of specialized agents which, as independent units, produce and react to events.

Several instances of linguistic, spatial and agent models have been proposed, but neither of these enforces logical separation during run-time. We propose a model, the Delft model, that does support logical separation. The model inherits properties from both a linguistic and spatial model type. In the Delft model the user interface data of an ICS can be distinguished and the application semantics data of an ICS can be distinguished. Both types of data are fully separated and each type is controlled by a separate monitor. In the Delft model, it is only through the
monitors that information exchange takes place. The user input tasks and output tasks of an ICS must be specified for the monitor which controls the user interface data. The computer tasks of an ICS must be specified for the monitor which controls the application semantics data.

In order to make it plausible that also physical separation can be supported by the Delft model, we create an environment, based on the Delft model, consisting of methods, techniques and tools. We call the environment D2M2environment, because we create it around the Delft Direct Manipulation Manager (D2M2), a User Interface Development System allowing for the construction, maintenance and presentation of user interfaces of ICSs.

D2M2environment consists of D2M2design, D2M2edit and D2M2run. D2M2design consists of methods and techniques to design ICSs; the methods and techniques are not worked out in this dissertation. D2M2edit is that part of the tool D2M2 allowing for the construction and maintenance of specifications of ICSs. D2M2run is that part of the tool D2M2 presenting user interfaces of ICSs.

We apply two cases to test whether the user interface and application semantics of an ICS can be physically separated in D2M2environment; moreover, the two cases are applied to test the usefulness of D2M2environment. We implement and alter a Graphics Manager and a Graphic Calculator.

**Graphics Manager**
The Graphics Manager is an ICS with which data can be visualized by graphs; the graphs can be manipulated by the user. A functional description of the Graphics Manager is made in cooperation with Baan Info Systems, a software house. We are able to implement specifications of the Graphics Manager in D2M2environment according to the functional description.

In order to make it plausible that the user interface and application semantics are separated, a number of changes are suggested to the specification of the Graphics Manager in D2M2environment. We learn that there are two types of imperfections in the separation of the user interface and application semantics.

1. **Semantic feedback** (i.e. feedback generated during user input task execution for which application semantics data must be consulted) in the Graphics Manager limits physical separation in D2M2environment.

2. **Intermediate output** (i.e. feedback generated during computer task execution) in the Graphics Manager limits physical separation in D2M2environment.

**Graphic Calculator**
The Graphic Calculator is an ICS with which children can learn to add and subtract via manipulations of either cubes or coins. A functional description of the Graphic Calculator is made in cooperation with SLO (Institute for Curriculum Development). It proves to be possible to implement specifications of the Graphic Calculator in D2M2environment according to the functional description.

In order to make it plausible that the user interface and application semantics are separated, we suggest a major alteration to the specifications of the Graphic Calculator: via manipulations of either cubes, coins or an abacus, children may learn to add and subtract. It proves that all

xii
changes needed in the user interface, do not affect the application semantics at all. The difference with the Graphics Manager case is that the Graphic Calculator does not contain semantic feedback or intermediate output.

We conclude that D2M2environment supports logical separation of the user interface and application semantics: during run-time the user interface (managed by D2M2run) and the application semantics do not share data and have only a limited ability to communicate with each other. Further, we conclude that physical separation is not fully achieved: at least semantic feedback and intermediate output violate physical separation in D2M2environment.

Because D2M2environment allows dialogue specialists to focus on user interfaces of ICSs, there is reason to assume that ICSs developed in D2M2environment allow for high quality user interfaces. Moreover, there is reason to assume that ICSs developed in D2M2environment can be maintained more efficiently and effectively than ICSs developed in an environment that does not support separation that much. In case of separation the user interface can be altered without having to change the application semantics.
SAMENVATTING

Interaktieve Computer Systemen (ICSen) worden ontworpen om mensen in de uitvoering van taken te helpen. De gebruikersinterface van een ICS wordt gedefinieerd als: de invoertaal van de gebruiker naar de computer, de uitvoertaal van de computer naar de gebruiker en de definitie van het oorzakelijk verband tussen invoer en uitvoer. De gebruikersinterface van een ICS beïnvloedt de doelmatigheid en de doeltreffendheid van de uitvoering van taken met behulp van het ICS.

De gebruikersinterface van een ICS bepaalt hoe mens-computer interaktie kan worden uitgevoerd. Wij veronderstellen dat mens-computer interaktie altijd bestaat uit de uitvoering van een gebruikersinvoer-taak, de uitvoering van een computer-taak en de uitvoering van een uitvoertaak. Een computer-taak (een taak uit te voeren door het ICS) wordt altijd aangeroepen door een gebruikersinvoer-taak (een taak geïnitieerd door de gebruiker); tijdens of direct na uitvoering van een computer-taak geven uitvoer-taken de resultaten van de uitvoering van de computer-taak weer.

Het is aan te bevelen de gebruikersinterface en de applicatiesemantiek (van een ICS) te scheiden. In het geval van scheiding is de applicatie verloopt van alle details die betrekking hebben op de invoer en uitvoer; de gebruikersinterface kan worden veranderd zonder de applicatiesemantiek te moeten veranderen; dialoogspecialisten kunnen een gebruikersinterface samenstellen die past in de gegeven omgeving en/of applicatie; interaktietechnieken kunnen beschikbaar komen in dialoogbibliotheken; een dialoog-interpretator kan faciliteiten bieden op het terrein van gebruikershulp, foutverstel en prototyping. Er wordt onderscheid gemaakt tussen logische en fysieke scheiding. Bij logische scheiding hebben de gebruikersinterface en de applicatiesemantiek geen data gemeenschappelijk; verder is er tijdens run-tijd slechts beperkte kommunikatie mogelijk tussen de gebruikersinterface en de applicatiesemantiek. Bij fysieke scheiding is de specifiek van de gebruikersinterface opgeslagen in een andere file (of files) dan de specificatie van de applicatiesemantiek.

In de literatuur wordt verondersteld dat scheiding van de gebruikersinterface en de applicatiesemantiek niet altijd mogelijk is. We besluiten om ons te richten op de volgende onderzoeks vraag.

Is het mogelijk om logische en fysieke scheiding van de gebruikersinterface en de applicatiesemantiek te verwezenlijken? Zijn er randvoorwaarden?

Wij stellen een model voor ICSen voor, dat logische scheiding tijdens run-tijd afdwingt. Er zijn ruwweg drie typen modellen voor ICSen: 'linguistic', 'spatial' en 'agent' modellen. Bij linguistic modellen wordt mens-computer interaktie behandeld als mens-mens interaktie: een gebruiker communiceert met een ICS door middel van een taal, waarmee de gebruiker wil bereiken dat het ICS een bepaalde taak uitvoert. Na uitvoering van de taak geneereert het ICS uitvoer naar de gebruiker. 'Spatial' modellen leggen de nadruk op gegevensmodellen in ICSen. Exemplaren van gegevenstypen die in een gegevensmodel zijn gedefinieerd kunnen op het beeldscherm worden afgebeeld. Deze exemplaren kunnen worden gemanipuleerd door de gebruiker. 'Agent' modellen structureren een ICS in een verzameling van gespecialiseerde 'agents' die,
als onafhankelijke eenheden, reageren op inkomende berichten.


Om aannemelijk te maken dat ook fysieke scheiding kan worden ondersteund door het Delftse model creëren we een omgeving, gebaseerd op het Delftse model, bestaande uit methoden, technieken en gereedschappen. We noemen de omgeving D2M2environment, omdat we de omgeving rond de Delft Direct Manipulation Manager (D2M2) creëren; D2M2 is een User Interface Development System waarmee gebruikersinterfaces van ICSen kunnen worden gekonstrueerd, onderhouden en gepresenteerd.

D2M2environment bestaat uit D2M2design, D2M2edit en D2M2run. D2M2design bestaat uit methoden en technieken om ICSen te ontwerpen; deze methoden en technieken worden in dit proefschrift niet uitgewerkt. D2M2edit is dat deel van het gereedschap D2M2 dat de constructie en het onderhoud van ICSen ondersteund. D2M2run is dat deel van het gereedschap D2M2 dat de presentatie van ICSen ondersteund.

We voeren twee case-studies uit om te testen of de gebruikersinterface en de applicatiesemantiek van een ICS inderdaad fysiek gescheiden kunnen worden in D2M2environment; bovendien worden de twee case-studies uitgevoerd om te testen of D2M2environment een bruikbare omgeving is. We implementeren en wijzigen een ‘Graphics Manager’ en een ‘Graphic Calculator’.

**Graphics Manager**

De Graphics Manager is een ICS waarmee data kan worden afgebeeld in grafieken; de grafieken kunnen worden gemanipuleerd door de gebruiker. Een functionele beschrijving van de Graphics Manager is gemaakt in samenwerking met Baan Info Systems, een software huis. We blijven in staat om specifikaties van de Graphics Manager in D2M2environment te implementeren uitgaande van de functionele beschrijving.

Om aannemelijk te maken dat de gebruikersinterface en de applicatiesemantiek gescheiden zijn, worden een aantal wijzigingen in de specifikaties van de Graphics Manager voorgesteld onder D2M2environment. We concluderen dat er twee typen onvolkomenheden zijn in de scheiding van de gebruikersinterface en de applicatiesemantiek.

1. Semantische feedback (dat is feedback gegenereerd tijdens uitvoering van een gebruikersinvoer-taak waarvoor de applicatiesemantiek moet worden geraadpleegd) in de Graphics Manager beperkt fysieke scheiding in D2M2environment.
2. Tussentijdse uitvoer (dat is uitvoer gegenereerd tijdens uitvoering van een computer- taak) in de Graphics Manager beperkt fysieke scheiding in D2M2environment.

Graphic Calculator
De Graphic Calculator is een ICS waarmee kinderen kunnen leren optellen en aftrekken via manipulaties van kubussen of munten. Een functionele beschrijving van de Graphic Calculator is gemaakt in samenwerking met SLO (Instituut voor Leerplan Ontwikkeling). Het blijkt mogelijk om de specificaties van de Graphic Calculator in D2M2environment te implementeren uitgaande van de functionele beschrijving.

Om aannemelijk te maken dat de gebruikersinterface en de applicatiesemantiek gescheiden zijn, stellen we een uitbreiding voor in de specificaties van de Graphic Calculator: niet alleen via manipulaties van kubussen en munten kunnen kinderen leren optellen en aftrekken, maar ook via manipulaties met een abacus. Het blijkt dat alle wijzigingen die nodig zijn in de gebruikersinterface geen gevolgen hebben voor de applicatiesemantiek. Het verschil met de Graphics Manager is dat de Graphic Calculator geen semantische feedback of tussentijdse uitvoer bevat.

We concluderen dat D2M2environment logische scheiding van de gebruikersinterface en de applicatiesemantiek ondersteunt: tijdens run-tijd hebben de gebruikersinterface en de applicatiesemantiek geen gegevens gemeenschappelijk. Verder is er tijdens run-tijd slechts beperkte communicatie mogelijk tussen de gebruikersinterface en de applicatiesemantiek. Bovendien concluderen we dat fysieke scheiding niet volledig is bereikt: semantische feedback en tussentijdse uitvoer schenden fysieke scheiding in D2M2environment.

Omdat D2M2environment toelaat dat dialoogspecialisten zich kunnen richten op de gebruikersinterface van ICSen, is er reden om aan te nemen dat ICSen ontwikkeld in D2M2environment kwalitatief hoge gebruikersinterfaces hebben. Bovendien is er reden om aan te nemen dat ICSen ontwikkeld in D2M2environment doelmatiger en doelstrevender kunnen worden onderhouden dan ICSen ontwikkeld in een omgeving waarin geen scheiding is afgedwongen tussen gebruikersinterface en applicatiesemantiek. In geval van Scheiding kan de gebruikersinterface immers worden gewijzigd zonder de applicatiesemantiek te wijzigen.
1. INTERACTIVE COMPUTER SYSTEMS

1.1. INTRODUCTION
Interactive Computer Systems (ICSs) are designed to help people in performing tasks. A user communicates with an ICS in order to achieve goals; the communication is called human-computer interaction or human-computer dialogue. The user interface of an ICS highly influences the efficiency and effectiveness of performing tasks (Potosnak, 1987; Saja, 1985; Shneiderman, 1987).

Today, much research is being done in the field of user interfaces. The results of that research are increasingly being applied to commercial software products. Shneiderman states that the enormous interest in user interfaces of ICSs "arises from the complementary recognition of how poorly designed many current systems are and the genuine desire to create elegant systems that effectively serve the users. This increased concern emanates from four primary sources: life-critical systems; industrial/commercial uses; office, home and entertainment applications; and exploratory, creative, and expert systems." (Shneiderman, 1987, p. 15-16).

After identifying the broad field of user interfaces, and the current research on design aspects of user interfaces, we formulate the research questions in the last section of this chapter.

1.2. USER INTERFACE
Different definitions of a user interface have been offered. Foley defines a user interface as consisting of an input language for the user, an output language for the machine and the sequencing of input and output (Foley, 1984). This view of the user interface deals with the physical dimension (i.e. the actual input elements sent by the user, like key clicks, mouse movement, etc. and the output elements sent to the user, like characters, cursor shape, etc.) as well as the structure of the dialog (i.e. the definition of how a computer system reacts to user inputs and the syntactical structure of user inputs and computer outputs).

Ziegler and Fähnrich (1988) suggest that the user interface should also include the semantic level that describes the tasks which can be carried out with the ICS.

In figure 1.1 the views of Foley and Ziegler et al. can be identified. The figure is abstracted from Ziegler and Fähnrich (1988, p. 125). Foley takes the dialog level and the i/o-level into account, Ziegler and Fähnrich take all three levels into account.

The semantic level deals with the tasks which can be carried out with the system. The dialog level deals with the dialog structure and sequences and the syntactical structure of inputs and outputs. The i/o-level deals with the atomic elements of input (like mouse movements) and output (like character fonts).
Besides the levels depicted in figure 1.1, Potosnak (1987) and Van der Mast (1990) also regard explicitly tutorials, reference manuals, on-line help and other training materials as part of the user interface.

The user interface of an ICS defines how human-computer interaction will be performed. Coats and Vlaeminke (1987, p. 22) define human-computer interaction as: "... an exchange of information governed by agreed conventions which takes place between a computer based system and its users via an interactive terminal".

Generally, human-computer interaction and associated task execution is performed through the following steps (see figure 1.1 and Coats and Vlaeminke (1987)).

1. The user has a goal: a task to be performed.
2. The user plans actions to initiate the task to be performed.
3. The user enters inputs, like key clicks, mouse clicks, etc.
4. The ICS recognizes the user inputs.
5. The ICS verifies the syntactical structure of the inputs.
6. The ICS determines what operation should be performed.
7. The operation is executed.
8. The ICS defines the syntactical structure of the computer outputs.
9. The ICS sends elementary output elements to the user.
10. The user notices the computer outputs.
11. The user interprets the outputs.
12. The user evaluates the outputs.
After step 12, step 1 can be revisited.

An important issue within the user interface is feedback. Feedback can be generated by an ICS to e.g. indicate that the user has correctly or incorrectly entered input. Three types of feedback can be identified (Foley, et al., 1990, p. 406): lexical, syntactic and semantic feedback.

**Lexical feedback.** During human-computer interaction the ICS may give lexical feedback on elementary input, like key clicks, mouse movements etc. In that case the lexical feedback is character echoing or cursor moving on the screen.

**Syntactic feedback.** Syntactic feedback e.g. indicates that a complete command with parameters is (not) correctly entered into the ICS.

**Semantic feedback.** Two forms of semantic feedback are distinguished. One form of semantic feedback is generated by the ICS on completion of or during execution of a routine carried out by the ICS. It indicates e.g. whether or not routine execution was successful. The other form of semantic feedback is generated by the ICS during human-computer interaction (Hudson, 1987): the ICS generates such feedback to let the user know that input is semantically (not) correct. To accomplish this semantic feedback, semantic information is needed during human-computer interaction. We abstract an example of semantic feedback during human-computer interaction from Hartson (1989, p. 64): "To delete a file on the Macintosh, for example, you drag it to the trash-can icon. If in the process the file icon passes over another file icon, it doesn’t highlight that icon. But if you drag the file icon over a folder icon, it does highlight the folder icon, reminding you that there is a semantic relationship between files and folders. If you release the button at that point, the file is deposited in the folder. If you want to echo input actions in the dialogue component the dialogue must have semantic information about the relationship between, in this case the folder and the file icon."

For indicating the broad field of the user interface we gather from Helander (1988) the following aspects of user interfaces. The aspects arise from various disciplines.

1. Psychological aspects.
2. Methodological aspects.
3. Hardware oriented aspects.
4. Supportive aspects.
5. Applications.

As for the psychological aspects e.g. user mental models (Carroll and Olson, 1988) (the way the user thinks the system behaves) and user interface metaphors are taken into account. As for the methodological aspects e.g. methods and techniques for ICS-design, guidelines for user interface design, tools supporting methods and techniques are taken into account. As for the hardware oriented aspects ergonomics of keyboards, screens, etc. are taken into account. As for the supportive aspects tutorials, reference manuals, on-line help and training materials are taken
into account. Finally, as for applications, those systems that emphasize the user interface are taken into account; examples are: Computer Aided Design-systems, Computer Assisted Instruction-systems and Computer Graphics-systems.

We will mainly emphasize the methodological aspects of user interface development. We note that the discussed different aspects are not independent.

1.3. METHODOLOGICAL ASPECTS
In this section models for ICSs will be discussed. Following on, methods and techniques for ICS design will be considered. Finally, tools will be mentioned, distinguishing between design and run-time tools.

MODELS
Broadly, there are three types of models for ICSs: linguistic models, spatial models and agent models.

The basic principle of a linguistic model is that human-computer interaction is treated like human-human communication. This means that a user communicates with an ICS in a kind of language through which the user wants the ICS to perform tasks. After task execution the ICS generates output to the user.

A well known example of the linguistic model is the Seeheim model, discussed by Green (1985a). In figure 1.2 the Seeheim model is depicted.

![Figure 1.2: the Seeheim model](image)

The Seeheim model describes a single user, single tasking environment, i.e. concurrency and multitasking as described by Lantz, et al. (1987) cannot be simply supported by the Seeheim model. The model depicted expresses a run-time view. The application contains the implementation of the tasks that the user can perform with the ICS. The application interface component contains the definition of tasks together with their pre-conditions (representing the conditions that must hold before a task can be executed) and post-conditions (representing actions that may change the outcome of pre-conditions). The dialog control component contains the structure of the dialog, which defines how a computer system behaves on certain user actions. The presentation component contains code that compiles tokens to the actual i/o-devices and vice versa; the tokens are the information units the dialog control component works with. In the presentation component the layout on screen is defined as well as the way of physical interaction with the ICS.

The components of the Seeheim model reflect the levels of figure 1.1. The presentation component of the Seeheim model deals with the i/o-level of the ICS. The dialog control component deals with the dialog level of the ICS. The application interface component deals with the con-
ceptual level of the ICS.

Spatial models put emphasis on data models in ICSs. Instances of the data types defined in a data model may be presented graphically on screen. An interpretation of a spatial model is presented by Hudson and King (1986); it is depicted in figure 1.3. In contrast with the Seeheim model this model changes the focus from dialog management to modelling objects that can be manipulated by the user.

![Diagram](image)

**Figure 1.3: Hudson’s model**

Again concurrency and multitasking cannot be described and visualized simply by this model. During run-time the application data model component contains application data that is used by the application. The view data model component contains view data that is necessary to give a view of the application on the screen. View data is derived from application data. The presentation component contains so called picture plans that describe how an image may be drawn and how input devices may be used to manipulate images presented on the screen. Images are derived from view data.

Agent models structure an interactive system into a collection of specialized agents which produce and react to events. "An agent is a complete information processing system: it includes event receivers and event transmitters, a memory to maintain a state, and a processor which cyclically processes input events, updates its state, and may produce events or change its interest in input event classes." (Coutaz, 1989). Users may communicate with certain agents; thereby sending user events to particular agents. In contrast to the Seeheim model and Hudson’s model, multithreaded dialogues and multitasking can easily be described by agent models (Coutaz, 1989).

It is not possible to talk about modelling and models without discussing methods, techniques and tools. In the next two parts of this section methods, techniques and tools related to models are discussed.

**METHODS AND TECHNIQUES**

Methodologies for user interface design consist of a combination of methods and techniques. Based on the linguistic model such a methodology should at least address methods and techniques for the design of the dialogue structure and the design of the presentation part (input and output). The methodology of Foley (Foley, et al., 1990; Foley, 1984) indeed addresses these issues. Another methodology for ICS design is the User Software Engineering (USE) methodology (Wasserman, 1982). This methodology is very complete and is supported by a number of tools. The Adaptive User Interface AUI methodology (Kuo and Karimi, 1987) is another example of a methodology supported by a number of tools.
The far most addressed part in user interface design with respect to the Seeheim model is dialog structure design (Green, 1985a). The sequence of inputs and outputs, i.e. the behaviour of the system, must be defined by such a method. The notation techniques that help designing the dialog structure mainly fall into three categories: transition networks, context free grammars and event models Green (1986). "Transition network notations view the user interface as a collection of states. The grammar group views the dialogue between the user and the computer as a language, and uses grammar based techniques to describe this interaction. The event group views the user interface as a collection of events and event handlers. When the user interacts with the computer, one or more events are generated, which are processed by the event handlers, possibly generating more events." (Green, 1985a, p 14). In addition, e.g. dialog charts (Ariav and Calloway, 1987) or special transition networks for Direct Manipulation (Jacob, 1986) may be used to design the dialog structure. Traditionally, methods for the presentation component specification were not so often mentioned (Green, 1985a); lately, however, more and more techniques and toolkits are supplied to specify the presentation of objects on the screen e.g. AutoCODE (1988) for SunView applications and ExoCODE (1989) for XView applications. The application interface model is, however, still hardly addressed (Green, 1985a).

Methods and techniques for the design of the specification of the different components of the spatial model are badly addressed. Of course, methods and techniques for conceptual data modelling have a long-standing existence, but their application in user interface design has not.

Because agent models are rather new, methods and techniques for designing systems based on an agent model are hardly addressed.

It is summarized that methods and techniques based on the Seeheim model are supported rather well; methods and techniques based on spatial models and agent models are not well supported.

**USER INTERFACE DEVELOPMENT ENVIRONMENTS**

Design tools for an ICS support methods and techniques. Design tools are used to automate the process of ICS-design. They should result in better user interfaces and easier creating and maintaining of user interfaces (Myers, 1989).

Design tools may form a User Interface Design Environment (UIDE), supporting e.g. design of the dialog structure, design of the screen layout, design of the input event handling, error handling design, etc. An example of such a design tool environment is the IDE of the George Washington University (Foley, 1987; Foley, et al., 1988; Foley, et al., 1989). Another example is the commercially available MacApp toolkit for the Apple Macintosh (Schmucker, 1989).

A User Interface Management System (UIMS) is a tool (set) for user interface management during run-time. Such a system interprets specifications for ICSs which may have been generated by design tools. In the literature a UIMS is often defined as both a run-time tool and a set of design tools of a IDE: "A User Interface Management System (UIMS) is a tool (or tool set) designed to encourage interdisciplinary cooperation in the rapid development, tailoring and management (control of the interaction) in an application domain across varying devices, interaction techniques and user interface styles. A UIMS tailors and manages (controls) user interaction in an application domain to allow for rapid and consistent development." (Betts, et al., 1987, p. 73). Because UIMSs often encapsulate both run-time and development, we follow Myers in his suggestion to call UIMSs User Interface Development Systems (UIDSs) (Myers, 1989).
Most UIDSs are based on a user interface model. The University of Alberta UIMS (Green, 1985b) is a UIDS based on the Seeheim model. The three parts of the model can be filled with specifications through tools of this UIDS. During run-time the UIDS manages the three components of the Seeheim model. Another UIDS based on the Seeheim model is MIKE (Olsen, 1986). Unique to MIKE is that only the application interface model has to be specified for prototyping purposes; specifications for the dialog control component and the presentation component are generated automatically. These default generated specifications can of course be altered.

HIGGENS (Hudson and King, 1986; Hudson and King, 1988) and the UIDS part of Shelley (Wielemaker and Anjewierden, 1989) are examples of UIDSs, which are based on Hudson’s model. In these UIDSs application data must be specified in order to be interpreted by the UIMS during run-time; view data must be specified in order to be interpreted by the UIMS during run-time; presentation issues must be defined in order to be interpreted by the UIMS during run-time.

IRENE (Coutaz, 1989) is an example of a system that is based on an interpretation of the agent model.

1.4. SEPARATION OF THE USER INTERFACE AND APPLICATION

An issue often discussed is whether the user interface can be separated from the rest of the ICS. E.g. Ten Hagen and Derksen (1985, p. 112) state that a program is specified by defining algorithms, which we call the application semantics part, and a dialogue language, which we call the user interface part. So, an ICS can be said to consist of two parts: the application semantics part and the user interface part. The application semantics part contains an implementation of the actual tasks to be performed by the ICS. The user interface part contains the specification of the dialogue structure, the input elements that can be sent to the ICS by the user, and the output elements that can be sent to the user by the ICS.

What is meant by separation of the user interface and application semantics? Hill (1987) defines logical separation and physical separation. According to Hill logical separation means that during run-time the user interface and the application semantics do not share data, and have only a limited ability to communicate with each other. Physical separation means that specification of the user interface is in a different file (or files) than the specification of the application routines (p. 118).

Separation of the user interface and application semantics is recommended (e.g. Ten Hagen and Derksen, 1985; Tanner and Buxton, 1985; Hartson, 1989). Ten Hagen and Derksen summarize the advantages of separation (1985, p. 113):

"- The algorithmic part is relieved from all details concerning in- and output such as: test whether input is correct, check whether output is possible right now.
- One can alter the dialogue part (e.g. adapt to new hardware), without having to change the algorithms.
- Dialogue specialists can write a dialogue most suitable for a given environment and/or application."
- Interactive techniques may become available as (sub)dialogue libraries.
- A dialogue run time support system may provide facilities for user help, error recovery and audit-trails."

Separation has been the foundation for the Seeheim model. However, it is not always possible to separate the user interface and the application semantics in the Seeheim model. Feedback seems to be a key issue for this separation problem. Specifically, semantic feedback during human-computer interaction disturbs a clean separation of the user interface and application semantics of a certain program (Dance, et al., 1987; Edmonds, 1990; Hartson, 1989; Hudson, 1987; Löwgren 1988; Sibert, et al., 1986).

In Hudson's model separation of the user interface and application semantics is not completely achieved: during run-time view data (user interface) are derived from application data (application semantics) (Hudson and King, 1988). If the user interface of an ICS which is based on Hudson's model is altered, the new user interface must fit onto the application data.

1.5. RESEARCH QUESTIONS AND OUTLINE OF THE STUDY

In the previous section a problem was identified concerning the separation of the user interface and application semantics: it does not always seem possible to achieve this separation. We want to contribute to the separation of the user interface and application semantics. Is, indeed, e.g. semantic feedback a constraint for such separation? The research question related to this is:

1. Is it possible to achieve a separation of the user interface and application semantics? Are there constraints?

Roughly, there are two ways of handling the issue of both supporting semantic feedback during human-computer interaction, and separation. Firstly, it is possible to encapsulate all information necessary for semantic feedback into the user interface; this means that part of the application semantics is brought to the user interface. This solution is discussed by Hartson (1989). Its main disadvantage is the affection of the logical separation of the user interface and application semantics.

Secondly, it is possible to intensificate communication between the user interface and application semantics: whenever semantic feedback occurs and thus information is needed during human-computer interaction, communication between user interface and application semantics is initiated. The main disadvantage is the communication overhead one may get.

In tackling the research question an alternative model is constructed (different from the Seeheim model and Hudson's model), based on logical separation and allowing for physical separation. We now pose an additional research question:

2. Is it possible to create an environment for an ICS developer that supports both physical and logical separation? What kind of methods, techniques and tools should such an environment provide?

To test the environment we will design and implement two cases.
In chapter 2 basic concepts for this research are presented. In chapter 3 an alternative model is offered. In chapter 4 an environment based upon this alternative model is presented. In chapter 5 and 6 cases are discussed that use and test the described environment. In chapter 7 the stated research questions are reconsidered taking into account the constructed environment and the implemented cases.
2. BASIC CONCEPTS

2.1. INTRODUCTION
More and more human-computer interaction is performed by directly manipulating (graphical) objects on screen (Shneiderman, 1983; 1987). By directly manipulating objects on screen often semantic feedback occurs. It is intended to include such user interfaces in this research. In this chapter definitions and attractiveness of this 'style' of human-computer interaction are discussed.

In some ICSs the integration of the user interface and the application semantics seems to be higher than in others. We structure the measure of integration: based on a proposed sequencing of tasks in human-computer interaction presented in section 2.3 we define a number of classes of ICSs. The ICSs of different classes differ in integration of the user interface and application semantics. This classification allows us to determine whether a certain ICS will be easy or difficult to separate. Further, if it can be demonstrated that ICSs of a class with the highest integration of the user interface and application semantics can be separated into a user interface part and an application semantics part, it will be possible to separate the two parts in ICSs of all other classes.

In section 2.4 we will refine the research questions, based on the issues discussed in section 2.2 and 2.3.

2.2. DIRECT MANIPULATION
In this section we discuss Direct Manipulation as a 'style' of human-computer interaction. We discuss characteristics of Direct Manipulation and we present our definition. Moreover, the attractiveness of Direct Manipulation will be discussed and compared with the conventional dialogue styles, which are often not graphical.

DEFINITIONS
Direct Manipulation is a way of communicating with an ICS. The term is first coined by Shneiderman (1983, 1987). He states that Direct Manipulation exhibits the following characteristics (see also Ziegler and Fähnrich (1988)).

1. Continuous representation of the object of interest.

2. Physical actions or labelled button presses instead of complex syntax and command names.

3. Rapid incremental reversible operations whose impacts on the object of interest are immediately visible.
So, an ICS with Direct Manipulation characteristics allows its users to communicate with it through (direct) manipulation of objects displayed on the screen; manipulation should be directly visible and reversible. Examples of ICSs with Direct Manipulation characteristics are display editors, spread-sheets and video games.

Hutchins, et al. (1986; Hudson, 1987) characterize Direct Manipulation more thoroughly and abstractly in terms of "feeling of directness" on the part of the user. "Engagement", "articulatory distance" and "semantic distance" are the keywords for their characterization of Direct Manipulation.

Engagement is the feeling of communicating with the objects of interest directly (Hutchins, et al., 1986): "... not with programs, the computer, but with the semantic objects of our goals and intentions. Articulatory distance is the degree to which the form of communication with the system reflects the application objects and tasks. "Articulatory distance" concerns the actual form that communication takes; for example, the choice of type and mode of use of input devices, or the kinds of images drawn for the user. A small articulatory distance results when the input techniques and feedback representations used are well suited to conveying the required information." (Hudson, 1987, p. 121). Semantic distance is the degree to which the semantic concepts used by the system are 1) compatible with those of the user and 2) can be used to easily accomplish the user’s goals. "Semantic distance involves the ease with which the user can express desired actions with the concepts of the system [...] Semantic distance also involves a measure of how closely the user’s conception of the task domain matches that of the system." (Hudson, 1987, p. 121). To summarize: the feeling of directness on the part of the user is characterized by three aspects: engagement, articulatory distance and semantic distance. The higher the engagement and the smaller the articulatory and semantic distances are, the more direct Direct Manipulation is.

Siochi, et al. (1989), however, agitate against "feeling of directness" as characterization of Direct Manipulation, because of its subjectiveness. They propose the following working definition of Direct Manipulation: a Direct Manipulation interface is one in which the user deals with interface objects - entities of the model world - in a manner analogous to corresponding real world action. Siochi et al. put emphasis on 'manipulation' instead of 'directness' with respect to the definition of Direct Manipulation.

Question & Answer, Menu Selection, Form Fill In and Command Language are conventional dialogue styles e.g. described by Coats and Vlaeminck (1987) and Shneiderman (1987). We think that Direct Manipulation is not another dialogue style. Although (complex) graphical interaction is often identified as Direct Manipulation, we state that conventional user interface styles have Direct Manipulation characteristics too.

Menu Selection deals with menus. Menus have options; the options can be manipulated (selected, probably inverted), further often a menu can be manipulated (put in another position, widened, etc.). In figure 2.1 a menu is depicted; the option 'Front' is inverted; the option 'Props' is greyed out, which means that that option cannot be selected. In this figure the objects of interest are the menu itself and the options.
With respect to Form Fill In, fields can be manipulated (change contents) by e.g. entering or deleting character objects: the objects of interest are the form itself, the form labels (which indicate where the user can e.g. type in and delete characters; 'Width' is such a label in figure 2.2), the form fields (i.e. the actual places the user can e.g. type in and delete characters; the space just after 'Width' in figure 2.2 is such a field), and the characters that can be entered or deleted in the fields. In figure 2.2 a form is depicted with additional objects of interests: radio-buttons and toggles. From a set of radio-button objects only one can be chosen. A toggle has one of two values; manipulating a toggle is choosing the opposite value of the toggle by clicking the toggle.

With respect to Question & Answer, characters can be manipulated (entered or deleted) to form the answer to a posed question: objects of interest are the question, the answer and characters that form an answer.

With respect to Command Language, characters can be manipulated (entered or deleted) to form a command: objects of interest are the command and the characters that form a command; they can be entered or deleted.

Of course, the "feeling of directness" (see the definition of Hutchins, et al.) in the conventional dialogue styles is hardly achieved, but nevertheless these traditional dialogue styles can be considered Direct Manipulation interfaces, because objects that can be manipulated can be identified. We conclude that Direct Manipulation is not a user interface style itself, but a way of looking at a certain user interface. We now give our working definition of Direct Manipulation.
Direct Manipulation: a viewpoint, which considers human-computer interaction as manipulation of objects (characters, icons, windows, etc.) on the screen.

We will use this definition in the first statement of section 2.4.

ATTRACTIVENESS OF DIRECT MANIPULATION
Users of ICSs state that Direct Manipulation in graphical environments is attractive because of the following (Shneiderman, 1983, p. 57).

1. Mastery of the system; the user has the initiative in the dialogue.
3. Ease in learning the system originally and in assimilating advanced features.
4. Confidence in their capacity to retain mastery over time.
5. Enjoyment in using the system.
6. Eagerness to show it off to novices.
7. Desire to explore more powerful aspects of the system.

In Direct Manipulation interfaces in graphical environments often user interface metaphors are used: on screen a real world situation is simulated. Because the user is familiar with the metaphor model on screen, the ICS is not hard to interact with. Examples of metaphors are e.g. (Carroll, et al., 1988, p. 68): ledger sheet, desktop, business form, etc.

The success of Direct Manipulation in graphical environments is formally explained by Shneiderman using the syntactic/semantic model (Shneiderman, et al., 1979; 1983; 1987). The idea is that with Direct Manipulation the object of interest is displayed so that actions can be performed in the high-level task domain. "There is little need for the mental decompositions of tasks into multiple commands with a complex syntactic form. On the contrary, each action produces a comprehensible result in the task domain that is immediately visible. The closeness of the task to the action syntax reduces operator problem-solving load and stress". (Shneiderman, 1987, p. 202).

2.3. CLASSES OF ICSs
We mentioned earlier that users of ICSs try to perform tasks with the help of ICSs. Following Bots (1989) in his description of tasks a task is a collection of coordinating decisions and subtasks. Coordinating decisions affect the sequence in which subtasks are performed. Tasks without subtasks are called leaf tasks. Leaf tasks are the elementary tasks that are performed by the ICS; user intervention is not possible in leaf tasks. To define:

Computer Task: a leaf task, which the user wants to be performed by the ICS.

User Input Task: a series of user actions (like keyboard input, mouse movement, etc.) probably associated with ICS responses (feedback during human-computer interaction) through which a user can enable a computer task.

Output Task: an action by which output is sent to the user during or just after computer task execution.
It is now possible to describe the sequencing in human-computer interaction in terms of user input tasks, output tasks and computer tasks: a user performs a user input task; the user input task will trigger a computer task; during and/or just after computer task execution output tasks will be executed.

As an example: suppose we have an ICS that gives us the opportunity to delete files. In typing:

```
delete file1
```

which is a user input task, the ICS indeed deletes the file, which is done by a computer task. The ICS now displays:

```
file1 deleted
```

which is performed by an output task.

Figure 2.3 illustrates the sequencing in human-computer interaction.

![Figure 2.3: the sequencing of tasks](image)

The sequencing of tasks can be applied to a large number of ICSs. Every ICS that can be described by the Seeheim model can be represented by the sequencing of tasks: the specifications of the presentation component and the dialog control component can be represented by user input tasks and output tasks; the specifications of the application interface component and the application can be described by computer tasks and coordinating decisions for the computer tasks. Further, every ICS that can be described by Hudson’s model can be described by the sequencing of tasks: the specifications of the presentation component and view data model component can be represented by user input tasks and output tasks; the specifications of the application data model component and the application itself can be represented by computer tasks and coordinating decisions for the computer tasks. Note that the sequencing of tasks only represents single threaded, single user dialogs, like the Seeheim and Hudson’s model.

Using the sequencing of tasks, separation of the user interface and application semantics is a matter of separating user input tasks and output tasks from computer tasks, because user input tasks and output tasks are concerned with the user interface and computer tasks are concerned with the application semantics. We then notice three complicating issues in the separation of the user interface and application semantics.
1. Semantic feedback is presented during user input task execution (complication I).

2. Output task execution is performed during computer task execution (intermediate output task execution, complication II).

3. Computer task execution is only dealing with the user interface and not with the application semantics (complication III).

ad 1) This is mentioned by Dance, et al. (1987), Hartson (1989), Hudson (1987), Löwgren (1988) and Sibert, et al. (1986). The issue can be generalized to cases in which information of the application semantics must be consulted during user input task execution: e.g. generation of semantic error messages during user input task execution is also taken into account (Sibert, et al., 1986).

ad 2) This form of feedback informs the user about the progress of computer task execution: it is "some indication that the computer is at least working on the problem" (Foley, et al., 1990, p. 406). It is generated during execution of a computer task. After this feedback, the computer task is further being executed. It is obvious that this may complicate the separation of the user interface and application semantics.

ad 3) Johnson and Roberts (Johnson, et al., 1989) indicated this problem which often arises in complex graphical systems. Images on screen (user interface objects) are what the computer task (application semantics) is intended to manipulate.

Ideally, user interfaces of ICSs are consistent. Consistency within the user interface includes several dimensions, among them (Rhyne, et al., 1987, p. 80):

"Role consistency:
Interactions involving objects which play similar roles should be similar in structure so the end user may generalize over roles.

Categorial consistency:
End users should be able to perform similar manipulations on objects of the same type.

Presentational consistency:
The same object should not appear in several representations that are visually unrelated.

Interaction consistency:
Components of interaction such as dragging and selection should always be performed in the same way.

Metaphorical consistency:
Where a metaphor is employed (e.g. the physical office metaphor) the behavior of objects should be consistent with their real counterparts (e.g. files dragged to the filing cabinet or the wastebasket will disappear from view, while files dragged to the printer will reappear)."

16
Consistency implies that if a user input task with semantic feedback (complication I) is defined in the user interface, it is likely that other user input tasks, which are similar in structure, should provide semantic feedback too; further if a computer task is accompanied by complication II, the complication should likely be consistently generalized to other computer tasks; if complication III occurs, it is likely that it will occur more that once. Therefore we classify ICSs instead of individual tasks.

Taking complications I, II and III into account we define eight classes of ICSs.

- Class 0 ICS: an ICS without complications I, II and III.
- Class 1 ICS: an ICS with complication I, but without complications II and III.
- Class 2 ICS: an ICS with complication II, but without complications I and III.
- Class 3 ICS: an ICS with complication III, but without complications I and II.
- Class 4 ICS: an ICS with complications I and II, but without complication III.
- Class 5 ICS: an ICS with complications I and III, but without complication II.
- Class 6 ICS: an ICS with complications II and III, but without complication I.
- Class 7 ICS: an ICS with complications I, II and III.

We call ICSs of the classes 1 to 7 Highly Interactive Computer Systems (HICSs).

Because the three complicating factors are independent the following assertions with respect to the integration of the user interface and the application semantics can be concluded.

- Assertion I: HICSs are more interwoven than ICSs.
- Assertion II: HICSs of the classes 4, 5 and 7 are more interwoven than HICSs of class 1.
- Assertion III: HICSs of the classes 4, 6 and 7 are more interwoven than HICSs of class 2.
- Assertion IV: HICSs of the classes 5, 6 and 7 are more interwoven than HICSs of class 3.
- Assertion V: HICSs of class 7 are more interwoven than HICSs of the classes 4, 5 and 6.

2.4. RESEARCH QUESTIONS REVISITED

User interfaces are considered in a Direct Manipulation way. This means that we view human-computer interaction as manipulation of objects on screen; here, objects are instances of data types. Related to the first research question of section 1.5 we present the following statement.

1. It is possible to construct an alternative model (different from the Sechtem and Hudson’s model) which supports logical separation, considering the user interface in a Direct Manipulation way.

Consequently, traditional dialog styles like Command Language, Menu Selection, Form Fill In and Question & Answer as well as Direct Manipulation in graphical environments are taken into account.

As argued in section 1.5, in an alternative model semantic feedback may be dealt with by intensifying communication between the user interface and application semantics. So we present the following hypothesis.

2. In such an alternative model semantic feedback during user input task execution does not limit logical or physical separation.
Output tasks, whether executed during or after computer task execution, are a matter of the user interface. Therefore we can present the following hypothesis.

3. In such an alternative model output task execution during computer task execution does not limit logical or physical separation.

If we consider "computer tasks that only update the screen" to be output tasks, we are able to present the following hypothesis.

4. In such an alternative model computer tasks that only update the screen do not limit logical or physical separation.

The assertions of section 2.3 imply that if separation of the user interface and application semantics of a HICS of class 7 succeeds, all other (H)ICSs of the classes 0 to 6 will be separable too. Moreover, because the three identified types of complications are independent, we can decide about the rejection of one of the hypotheses 2 to 4, if separation does not succeed completely.

With respect to physical separation of the user interface and application semantics an attempt is made to create an environment which supports such separation (see research question 2 of section 1.5). The environment must reflect the alternative model, which we construct, and should support the design of (H)ICSs of all eight classes. Further this environment should include a run-time tool, with which design specifications of (H)ICSs can be interpreted. We present the following statements related to the second research question of section 1.5.

5. It is possible to create an environment for an ICS-developer, which supports physical separation.

6. It is possible to construct a run-time tool, based on the alternative model. The run-time tool should interpret design specifications that have been generated by the design tools of the environment.

In chapter 3 we construct the alternative model for (H)ICSs. In chapter 4 we present a development environment for ICS development. Also a run-time tool will be presented; this tool will interpret the specifications of the ICS designed with the help of the methods, techniques and design tools.

To test the environment we will design and implement two HICSs incorporating as many complicating factors as reasonable. To test to what extend separation is supported by the alternative model and the environment we will examine alterations in the user interfaces of the HICSs.
3. AN ALTERNATIVE MODEL FOR ICSs

3.1. INTRODUCTION

In chapter 2 we stated that three complications may occur when separating the user interface and the application semantics of an ICS. Hudson's model does not support strict separation and the Seeheim model abandons strict separation as soon as all three complications occur.

In this chapter a model is presented with which an attempt is made to prove that, although the three complications occur, separation of the user interface and application semantics can be maintained. Throughout this dissertation we call this model the Delft model. In the Delft model the three complications are being dealt with by basically intensifying communication between the user interface and the application semantics as optioned in section 1.5.

In section 3.2 we introduce the Delft model; in section 3.3 we discuss the three complications in this model.

3.2. THE DELFT MODEL FOR ICSs

Hartson (1989) states that a possible way to deal with semantic feedback and other difficulties that obscure a clear separation of the user interface and application semantics is by putting more of the application semantics in the user interface. Hudson's model is an example of this approach: both the user interface as well as the application semantics can access the (shared) application data (Hudson, 1987); moreover, user interface data is derived from application data by "a semantic restructuring process. This process is a translation from the underlying data model, which is meaningful to the system designer, into a form that is more meaningful to the user." (Hudson and King, 1986, p. 139).

However, we believe that such a semantic restructuring process is not necessarily needed: intensifying communication between the user interface and the application semantics is also a suitable approach. E.g. semantic feedback can then be dealt with by sending messages to and from the application semantics to obtain information from the application semantics during human-computer interaction.

In figure 3.1 we propose the Delft model through which human-computer interaction can be described. Like the Seeheim model and Hudson's model, the Delft model does not support concurrency and multitasking.

During run-time of a certain ICS, the user interface data component contains all the data appropriate for the user interface; the application semantics data component contains all the data appropriate for the application semantics. The two different types of data are not intermingled; neither knows of the other's existence. It is only through monitors that user interface data or application semantics data can be updated, created or deleted.
In order to update, create or delete data each of the monitors has to be programmed. For the monitor that manages the user interface data this is done by specifying user input tasks and output tasks; for the monitor that manages application semantics data this is done by specifying computer tasks.

![Diagram](image)

**Figure 3.1: the Delft model**

Note that the Delft model both resembles a linguistic and spatial model. Linguistic, because it supports the "sequencing of tasks" as specified in chapter 2, figure 2.3; spatial, because it emphasizes user interface data (types) and application semantics data (types).

In the Delft model either internal or external control may be supported. As for external control the user has full control of the sequence of interaction; as for internal control the application dictates the sequence of interaction (Strubbe, 1985; Enderle, 1985; Takala, 1985). Applied to the Delft model, internal control means that the application semantics monitor manages the sequence of interaction, i.e. the actions the user may perform. External control means that the user interface monitor manages, through actions the user performs, the sequence of interaction. The key difference, thus, between the two types of control seems to be whether or not coordinating decisions are integrated in the application semantics. As for internal control coordinating decisions can be specified for the application semantics monitor; as for external control coordinating decisions never have to be specified, because the user is fully and exclusively in control of the sequence of interaction.

Many applications, however, are not exclusively based on either internal or external control (Strubbe, 1985). Take, for example, a graphics editor, with which lines and dots can be drawn and with which pictures (of lines and dots) can be saved to or loaded from secondary memory. The graphics editor is a typical external control application when the user is allowed to choose at any time one of the four possible functions: "draw line", "draw dot", "save picture" and "load picture". However, when, for example, the user chooses "save picture" and the ICS forces the user to type in the name of the file, the ICS takes the control the user had over the system.

On the contrary, a flight reservation system can be an internal control application. The ICS forces the user to type in the name of the passenger; after typing the name, the address of the passenger must be typed in, and so on. However, if the ICS allows its users to type in the name, address, etc. in an arbitrary order, the internal control of the flight reservation system is extended with external control features.
As soon as external control is extended with application initiated control and internal control is extended with user initiated control the difference between the two types of control becomes diffuse.

To decide on what kind of control the Delft model must be based we considered the design process of ICSs. Designing ICSs is a problem solving process (e.g. Sol, 1982). The main results of this problem solving process are specifications of the application semantics part; this is independent of the starting point of the methodology. Whether the starting point is the definition of data types (e.g. NIAM (Wintræecken, 1985)) or the definition of functions and subfunctions (Yourdon and Constantine, 1979), specifications of computer tasks, coordinating decisions and application semantics data types will be the result. Because methodologies for ICS-design mainly result in specifications of the application semantics part in the first place, we decide to base the Delft model on internal control. However, because Direct Manipulation ICSs generally support great freedom of interaction on part of the user, we extend the internal control with user initiated control.

Internal control extended with user initiated control has the following impact on ICSs modelled according to the Delft model.

1. The application semantics part contains, in addition to computer tasks, coordinating decisions.

2. The application semantics part can enable user input tasks, which the user can initiate.

3. The user interface monitor may take control, because the user may choose between all enabled interactions tasks in order to permit execution of a certain computer task.

Figure 3.2 depicts the necessary specifications by which an ICS must be modelled according to the Delft model. Note that we do not identify semantic feedback and other complications yet; in section 3.3 the complications will be discussed in relation to the Delft model.

![Diagram](image)

\textit{Figure 3.2: specifications of applications for the Delft model}

The communication between the user interface monitor and the application semantics monitor is bidirectional. From application semantics monitor to user interface monitor the activities associated with the communication are as follows.
1. Enable user input tasks.
2. Trigger output task.

From the user interface monitor to the application semantics monitor the activities are as follows.

1. Send user information after user input task execution.
2. Acknowledge that output task execution is finished.

It is now possible to describe the run-time flow of an arbitrary ICS modelled according to the Delft model. The description of a meta-model for arbitrary ICSs is presented by task hierarchy and task flows.

To illustrate task hierarchy and task flows we will discuss the extremely simple ICS "filemanager" as arbitrary ICS. The user can perform the following actions with the filemanager.

1. Delete a file.
2. Copy a file to a directory.
3. Change the working directory.
4. Leave the filemanager.

A possible layout for the filemanager is depicted in figure 3.3.

![Filemanager Diagram](image)

*Figure 3.3: the filemanager*
Let one assume that a user can "1) delete a file" by bringing the cursor over a file_icon, pressing the left mouse button, dragging the file_icon over the trash_can and releasing the left mouse button; further, a user can "2) copy a file to a directory" by bringing the cursor over a file_icon, pressing the left mouse button, dragging the file_icon over a directory_icon and releasing the left mouse button; further, a user can "3) change the working directory" by bringing the cursor over a directory_icon and pressing and immediately releasing the left mouse button; finally, a user can "4) leave the filemanager" by bringing the cursor over the trash_can and pressing and immediately releasing the left mouse button.

The task hierarchy consists of tasks, which can be subdivided into subtasks. Tasks in the task hierarchy are represented by rectangles; a description in the rectangle indicates the meaning of the task. The task hierarchy for running an ICS under the Delft model is modelled in figure 3.4.

Figure 3.4: task hierarchy of the task "run ICS"

The task "run ICS" consists of subtasks "initialize" (initialization of the ICS) and "perform human-computer interaction" (the run-time human-computer interaction between user and ICS). There is a certain flow between these two subtasks: "initialize" should be executed before "perform human-computer interaction". A possible way to represent task flows is by flow charts.
Usually, flow charts are used as a way to describe the detailed structure of programs (Martin and McClure, 1985). Since the run-time flow of an arbitrary ICS can be described in a "meta-program", flow charts will be used.

In flow charts, tasks are represented by rectangles. If we subdivide a certain task into subtasks with their own subtask flow, we represent that certain task as a rectangle with a dashed comment bracket on the right side for indicating the identification of the figure, which holds the subtask flow. Decisions whether or not a task will be executed are represented by diamonds with descriptions of the decisions. The start and the end of the flow of tasks in a flow chart is indicated by an oval. If a leaf task (i.e. a task which is not divided in subtasks) must be activated by the user interface monitor, it contains a 'U'; otherwise, if a leaf task must be activated by the application semantics monitor, it contains an 'A'. A flow chart which contains leaf tasks will be illustrated by the filemanager example.

Figure 3.5 depicts the flow between the tasks "initialize" and "perform human-computer interaction".

```
run ICS:

start

initialize

perform human-computer interaction

stop

[subtasks in figure 3.6]

[subtasks in figure 3.7]
```

Figure 3.5: task flow of "run ICS"

In figure 3.5 it is seen that as soon as an ICS is started it is initialized, after which the user can communicate with it by performing human-computer interaction. The task "initialize" is subdivided in four subtasks: "initialize application semantics data", "trigger initial output task", "execute initial output task" and "acknowledge end of initial output task execution". Figure 3.6 depicts the flow between these tasks.
In figure 3.6 the task "initialize application semantics data" creates instances of data types, which are defined in the application semantics data model. The decision "output necessary?" determines whether or not output should be sent to the screen after "initialize application semantics data". The task "trigger initial output task" sends a message from the application semantics monitor to the user interface monitor in order to start output task execution. The task "execute initial output task" executes the triggered output task by creating instances of data types, which are defined in the user interface data model. The task "acknowledge end of initial output task execution" sends a message from the user interface monitor to the application semantics monitor to acknowledge that the user interface monitor has ended initial output task execution.
For example, in the file manager "initialize application semantics data" (figure 3.6) represents the creation of instances of files and directories in the application semantics\(^1\); further, "trigger initial output task" represents preparing the displaying of a frame, trash_can and file_icons and subdirectory_icons of the current directory; "execute initial output task" represents the displaying of a frame, etc.; "acknowledge end of initial output task execution" represents sending the acknowledgement of successfully displaying the frame, etc.

The task "perform human-computer interaction" of figure 3.5 is subdivided in the subtasks: "enable user input tasks", "execute user input task", "send user information", "execute computer task", "trigger output task", "execute output task" and "acknowledge end of output task execution". Figure 3.7 depicts the flow between these tasks.

By means of the task "enable user input tasks" of figure 3.7 the possible user input tasks are enabled; this means that the user can start such a user input task and the user is prevented from starting any not-enabled user input task. As soon as the user starts a user input task the task "execute user input task" is active. After finishing "execute user input task" user information (constructed during execution of a user input task) is forwarded from the user interface monitor to the application semantics monitor by means of the task "send user information". Given the forwarded user information, the application semantics monitor is able to execute the task "execute computer task", which updates and consults the application semantics data. If an output task should follow the execution of a computer task the task "trigger output task" is executed. If an output task should not follow "execute computer task" it is determined whether or not the user has executed a final user input task, e.g. pressing the trash_can in the file manager; if not, "enable user input tasks" is again executed. The task "trigger output task" sends a message from the application semantics monitor to the user interface monitor to indicate that a certain output task must be executed. The task "execute output task", then, updates user interface data in order to update the screen. Finally, the task "acknowledge end of output task execution" sends a message from the user interface monitor to the application semantics monitor to indicate that output task execution is finished. Now, the task "enable user input tasks" can again be executed if the user has not finished human-computer interaction while e.g. previously pressing, as final user input task, the quit command of the ICS.

As for the file manager "execute user input task" (figure 3.7) represents e.g. bringing the cursor over a file_icon, pressing the left mouse button, dragging this icon over the trash_can and releasing the left mouse button. The user information sent by the task "send user information" consists of the name of the file_icon; "execute computer task" deletes the file with the name that corresponds to the name of the file_icon. Using the same example, "trigger output task" represents preparing the deletion of the file_icon, which represents the just deleted file; "execute output task" represents deletion of the file_icon; "acknowledge end of output task execution" represents the acknowledgement that the deletion of the file_icon has succeeded.

In figure 3.8 the task flow of "enable user input tasks" of figure 3.7 is depicted.

---

1. The instances of files and directories in the application semantics refer to the actual files and directories in the computer system. When these instances are created, really references to the actual files and directories are created. In case an actual file in the computer system must be deleted, both the instance of the file in the application semantics as well as the actual file in the computer system must be deleted. In case the file manager must be left, only the instances of the files in the application semantics must be deleted.
perform human-computer interaction:

start

enable user input tasks

execute user input task U

send user information U

execute computer task A

output necessary?

user finished human-computer interaction?

no

no

no

toggle output task A

execute output task U

acknowledge end of output task execution U

user finished human-computer interaction?

yes

yes

stop

Figure 3.7: task flow of "perform human-computer interaction"
enable user input tasks:

```
\[\text{start} \rightarrow \text{select possible computer tasks} \rightarrow \text{enable, per computer task, possible user input tasks} \rightarrow \text{wait for user interaction} \rightarrow \text{stop}\]
```

*Figure 3.8: task flow of "enable user input tasks"

The task "select possible computer tasks" of figure 3.8 selects the computer tasks the user can initiate through user input tasks. The task "enable, per computer task, possible user input tasks" enables the user input tasks through which the selected computer tasks can be initiated. The task "wait for user interaction" waits until the user starts interacting with the ICS.

As for the filemanager, "select possible computer tasks" (figure 3.8) represents selecting *all* possible computer tasks. So, delete a file, copy a file, change the current directory and quit the filemanager are selected; "enable, per computer task, possible user input tasks", then, enables exactly one user input task for each selected computer task. Bringing the cursor over a file_icon, pressing the left mouse button, dragging the file_icon over the trash_can and releasing the left mouse button corresponds to deleting a file; bringing the cursor over a file_icon, pressing the left mouse button, dragging it over a directory_icon and releasing the left mouse button corresponds to copying a file to a subdirectory; bringing the cursor over a directory_icon and pressing and releasing the left mouse button corresponds to changing the working directory; bringing the cursor over the trash_can and pressing and releasing the left mouse button corresponds to leaving the filemanager. Next, "wait for user interaction" represents waiting until the user starts one of the user input tasks to delete a file, copy a file, change the current directory or quit the filemanager.

If we add the three complications to the Delft model the task hierarchy and flow are to be extended. In the next section we present the different hierarchy and task flows.
3.3. COMPLICATING FACTORS IN THE DELFT MODEL

The three complicating factors extend and intensivate communication between the user interface monitor and the application semantics monitor. To consider subsequently: semantic feedback during user input task execution (complication I), output task execution during computer task execution (complication II) and computer task execution only dealing with the user interface (complication III).

Semantic feedback during user input task execution implies that application semantics data must be consulted during execution of a user input task; the control must switch from the user interface monitor to the application semantics monitor during human-computer interaction. We choose the following scenario for semantic feedback in the Delft model. The user interface monitor sends a message to the application semantics monitor to indicate that semantic feedback is needed. After this message sending, control switches from the user interface monitor to the application semantics monitor. Information from the application semantics data component is needed; this information is derived from the application semantics data by a so called semantic feedback function. After calling the semantic feedback function, the application semantics monitor sends a message to the user interface monitor with the result of the semantic feedback function; depending on the result semantic feedback is displayed on the screen.

Complication II implies that during computer task execution control must go to the user interface monitor in order to send output to the screen. This may be done by the same signals as already exist for the execution of an output task, i.e.: "trigger output task" and "acknowledge output task execution is finished".

Complication III implies that computer task execution only updates the screen. This implies that computer task execution does not update or consult the application semantics data. This complication is included in the Delft model by assuming that the computer task is dummy; the update of the screen is performed by an output task.

The necessary specifications by which an application must be modelled according to the Delft model, taking into account the three complicating factors, are depicted in figure 3.9.

![Figure 3.9: specifications of ICSs for the Delft model (revised)]
Taking into account the three complicating factors we add two types of communication between user interface monitor and application semantics monitor of the Delft model. Now, the three types of communication from application semantics monitor to user interface monitor are as follows.

1. Enable user input tasks.
2. Trigger output task.
3. Send result of semantic feedback function.

Further, the three types of communication from user interface monitor to application semantics monitor are as follows.

1. Send user information after user input task execution.
2. Acknowledge output task execution is finished.
3. Request for semantic feedback.

It is now possible to construct a new task hierarchy for the run-time flow of ICSs, modelled according to the Delft model. Figure 3.10 depicts the task hierarchy; figures 3.11 to 3.17 depict the task flows.

Compared to the task hierarchy of figure 3.4 the revised task hierarchy of "run ICS" of figure 3.10 is extended with some subtasks; furthermore, some existing tasks of figure 3.4 are altered. In figure 3.10 existing tasks with alterations are marked with an asterix (*). The task "initialize application semantics data" is subdivided in the subtasks "create instances of application semantics data", "trigger intermediate output task", "execute intermediate output task" and "acknowledge end of intermediate output task execution" to allow for intermediate output during initialization of the application semantics data. Note that "trigger intermediate output task" does not imply another kind of output task. Only to distinguish it from an output task which is called after "initialize applications semantics data" we add the word 'intermediate'. The "execute user input task" is subdivided in the subtasks "update/consult user interface data", "request for semantic feedback", "execute semantic feedback function", "send result of semantic feedback function" and "update user interface data for semantic feedback" to allow for semantic feedback during user input task execution. Finally, the task "execute computer task" is subdivided in the subtasks "update/consult application semantics data", "trigger intermediate output task", "execute intermediate output task" and "acknowledge end of intermediate output task execution" to allow for intermediate output during computer task execution.

The task flows of figures 3.11 to 3.17 illustrate the justification of the new tasks in the task hierarchy.

The task "run ICS" is subdivided into two subtasks: "initialize" and "perform human-computer interaction". The task flow on this level remains the same as the task flow in figure 3.5; note, however, that the comment texts in the comment brackets have changed. Figure 3.11 depicts the task flow of "run ICS".
Figure 3.10: revised task hierarchy of "run ICS"
run ICS:

![Diagram](image)

*Figure 3.11: (revised) task flow of "run ICS"

The task "initialize" of figure 3.11 is subdivided into four subtasks: "initialize application semantics data", "trigger initial output task", "execute initial output task" and "acknowledge end of initial output task execution". Figure 3.12 depicts the task flow of "initialize".

For example, in the filemanager "trigger initial output task" (figure 3.12) represents a trigger to display a frame, an icon_manager, a trash_can and file_icons and directory_icons of the current directory; "execute initial output task" represents the actual displaying of a frame, trash_can, etc.; "acknowledge end of initial output task execution" represents sending the acknowledgement of successfully displaying the frame, trash_can, etc.

Compared to the task flow of figure 3.6 the task "initialize application semantics data" of figure 3.12 can be expanded to allow for intermediate output during initialization. Figure 3.13 depicts the expanded task "initialize application semantics data". It contains the subtasks "create instances of application semantics data", "trigger intermediate output task", "execute intermediate output task" and "acknowledge end of intermediate output task execution".

In figure 3.13 the task "create instances of application semantics data" creates instances of data types, which are defined in the application semantics data model. Until either instantiation of application semantics data is finished or intermediate output must be generated, the task "create instances of application semantics data" is executed. If intermediate output must be generated the subsequent tasks "trigger intermediate output task" (sends message from the application semantics monitor to the user interface monitor), "execute intermediate output task" (creates instances of user interface data types, which are defined in the user interface data model) and "acknowledge end of intermediate output task execution" (sends a message from the user interface monitor to the application semantics monitor to acknowledge that the user interface monitor has ended intermediate output task execution). After generating intermediate output the task "create instances of application semantics data" can proceed.
initialize:

Figure 3.12: (revised) task flow of "initialize"

Suppose that the filemanager sends a message to the screen during the initial creation of file and directory application semantics data, to indicate that all file data instances have been successfully created. The message could look like:

ALL FILE INSTANCES CREATED, TRY TO CREATE DIRECTORY INSTANCES.

Displaying such a message could be identified as execution of an intermediate output task. If so, "trigger intermediate output task" (figure 3.13) represents preparing the displaying of the message; "execute intermediate output task" represents actually displaying the message; "acknowledge end of intermediate output task execution" represents sending the acknowledgement of successfully displaying the message.
The revised task flow of "perform human-computer interaction" of figure 3.11 greatly resembles the task flow depicted in figure 3.7. Only the tasks "execute user input task" and "execute computer task" must be expanded because of semantic feedback and intermediate output respectively. Figure 3.14 depicts the revised task flow of "perform human-computer interaction".

initialize application semantics data:

![Flowchart](image)

Figure 3.13: (revised) task flow of "initialize application semantics data"
perform human-computer interaction:

Figure 3.14: (revised) task flow of "perform human-computer interaction"
For example, in the filemanager, if the user input task executed was the one to trigger the deletion of a file by dragging its file_icon to the trash_can, "send user information" (figure 3.14) represents sending the name by which a file_icon is identified. Using the same example, "trigger output task" represents preparing the deletion of the file_icon, which represents the just deleted file; "execute output task", then, represents the deletion of the file_icon; "acknowledge end of output task execution" represents the acknowledgment that the deletion of the file_icon has succeeded.

The task flow of "enable user input tasks" of figure 3.14 is depicted in figure 3.15. Note that the task flow is identical to the flow depicted in figure 3.7.

enable user input tasks:

![Task Flow Diagram](image)

**Figure 3.15: (revised) task flow of "enable user input tasks"

As for the filemanager "select possible computer tasks" (figure 3.15) represents selecting all possible computer tasks; i.e. delete a file, copy a file, change the current directory and quit the filemanager are computer tasks the user can initiate; "enable, per (selected) computer task, possible user input tasks", then, enables exactly one user input task for each selected computer task: one for delete a file, one for copy a file, one for change the current directory and one for quit the filemanager. Next, "wait for user interaction" represents waiting until the user starts one of the user input tasks.
The task "execute user input task" of figure 3.14 is subdivided in the subtasks "update/consult user interface data", "request for semantic feedback", "execute semantic feedback function", "send result of semantic feedback function" and "update user interface data for semantic feedback". Figure 3.16 depicts the task flow between these subtasks.

Figure 3.16: (revised) task flow of "execute user input task"
The flow of task "execute user input task" of figure 3.16 starts with consulting and updating user interface data. Until either execution of a user input task is finished or a semantic feedback function must be called, the task "update/consult user interface data" is executed. If a request for semantic feedback is needed, the user interface monitor sends such a request to the application semantics monitor (task "request for semantic feedback"). The application semantics monitor executes an inquiry function to retrieve information from the application semantics data (task "execute semantic feedback function"). After "execute semantic feedback function" the result is sent to the user interface monitor. Depending on the result of the semantic feedback function, semantic feedback is displayed by the task "update user interface data for semantic feedback". Whether or not semantic feedback is displayed on screen, the user input task execution proceeds.

Suppose we add semantic feedback in the filemanager by extending the user input task that initiates the deletion of a file. A new scenario for this user input task could be as follows. Bring the cursor over a file_icon, press the left mouse button, drag the file_icon over the trash_can; if the file is not protected (in other words, is allowed to be deleted), invert the trash_can; release the left mouse button. This user input task will only trigger a computer task if the trash_can was inverted; in other words, a file can be deleted only if it is not protected. Using this scenario, on a certain moment during execution of the user input task for deleting a file, the file_icon will enter the trash_can region. If so, "request for semantic feedback" represents requesting the check whether the file, which corresponds to the file_icon over the trash_can, is protected; "execute semantic feedback function" executes this check; "send result of semantic feedback function", then, sends a 'yes', or 'no' back; if the file was not protected "update user interface data for semantic feedback" will invert the trash_can; back to "update/consult user interface data" the x- and y-coordinates of the file_icon will be updated, so it will be displayed inside the trash_can.

The task "execute computer task" of figure 3.14 deals with intermediate output during computer task execution. The subtasks are: "update/consult application semantics data", "trigger intermediate output task", "execute intermediate output task" and "acknowledge end of intermediate output task execution". Figure 3.17 shows the task flow of "execute computer task".

The task "update/consult application semantics data" of figure 3.17 updates/consults application semantics data. Until either computer task execution is finished or intermediate output must be generated, the task "update/consult application semantics data" is executed. If intermediate output must be generated the subsequent tasks "trigger intermediate output task" (sends message from the application semantics monitor to the user interface monitor), "execute intermediate output task" (creates/updates instances of user interface data types, which are defined in the user interface data model) and "acknowledge end of intermediate output task execution" (sends a message from the user interface monitor to the application semantics monitor to acknowledge that the user interface monitor has ended intermediate output task execution) are executed. After generating intermediate output the task "update/consult application semantics data" is activated.
execute computer task:

![Diagram](image)

**Figure 3.17: task flow of "execute computer task"**

In the filemanager all file and directory data instances are deleted from the application semantics, as soon as the user clicks the trash can. Suppose that the filemanager triggers an intermediate output task to indicate that all files have been deleted successfully from the application semantics data, after which all directories will be deleted from the application semantics data. In this scenario "update/consult application semantics data" represents deleting all file instances from the application semantics data; after deleting the final file "trigger intermediate output task" becomes active. It stands for triggering the message that all file instances have been deleted from the application semantics data. Next, "execute intermediate output task" represents actually displaying this message; the message could look like:
ALL FILE INSTANCES DELETED; TRY TO DELETE DIRECTORY INSTANCES.

"acknowledge end of intermediate output task execution" represents sending the acknowledgment of successfully displaying this message.

So far the discussion of the task hierarchy and task flows of the "run ICS"-activity, which includes the three complications.

By constructing a task hierarchy and task flows for arbitrary ICSs, we have a framework for specification of ICSs. In the next chapter an environment around this framework will be presented.
4. AN ICS DEVELOPMENT ENVIRONMENT

4.1. INTRODUCTION

We recall from chapters 1 and 2 that a problem with HICSs is that the user interface is often difficult to separate from the application semantics. In this chapter we present an environment that supports logical separation, because it is based on the Delft model. This chapter contributes especially to the following statements stated in chapter 2.

5. It is possible to create an environment for an ICS-developer, which supports physical separation.

6. It is possible to construct a run-time tool, based on the alternative model. The run-time tool should interpret design specifications that have been generated by the design tools of the environment.

Sage and Palmer (1990, p. 296) define environment as follows: a system design and development environment is the set of methods, design methodologies, and systems management processes that [...] is used to produce a trustworthy system. So an environment comprises the following.

1) Methods.
2) Methodology.
3) Systems management.

ad 1) Methods are sets of prescriptions supported by techniques and tools in order to design ICSs.

ad 2) A Methodology is a set of procedures to design ICSs involving methods. A methodology is a kind of meta-method, which prescribes when to use a certain method.

ad 3) Systems management is concerned with the technical direction and management of ICS design, production and maintenance.

The vision of Bots and Sol (1990) of a modelling environment elaborates this from a problem solving point of view. They state that an inquiry system as a modelling environment, which aids information workers when trying to solve a particular problem, "can reduce the cognitive strain on information workers in each of these activities:

1. Conceptualization [...], an inquiry system can aid theory formulation and global problem and goal definition [...].

2. Specification [...], an inquiry system can help to make problem situations visual ('tangible') [...].
3. Validation [...] an inquiry system can provide the information worker with a simulation and analysis environment that makes it possible to assess the validity of the model(s) of the problem situation [...].

4. Solution finding [...] an inquiry system can reduce cognitive strain on an information worker while he or she is experimenting with and evaluating different task structures as solutions to perceived (coordination) problems [...].

5. Verification [...] checking the syntactic and semantic consistency of the task structure [...].

6. Implementation [...] an inquiry system can provide facilities to extract an agreed-upon solution model its task structure [...], but also its 'support component' [...].

Our environment encapsulates the Delft Direct Manipulation Manager UIDS (D2M2). Therefore, throughout this dissertation, the environment will be called the D2M2environment.

The D2M2environment consists of the following.

- Methods and techniques to analyse input and design (H)ICSs.
- A tool to implement, maintain and document (H)ICSs.
- A tool to present the user interface of (H)ICSs.

Currently, our D2M2environment does not specifically focus on the activities of conceptualization, specification, validation and verification (Bots and Sol, 1990); solution finding and implementation, however, are activities supported by the D2M2environment. D2M2environment should, eventually, support an ICS design team with ICS prototyping, user interface consistency check, user interface evaluation, etc. For now, we conform ourselves to a restricted D2M2environment.

The development of (H)ICSs with the help of D2M2 is depicted in figure 4.1.

We call the "design ICS"-process D2M2design; the "implement ICS"-process is supported by a tool called D2M2edit; the "present user interface"-process is supported by a tool called D2M2run; the "execute application semantics"-process is a process separately running during run-time.

D2M2run is the actual UIMS of the D2M2environment: it takes care of the user interface of an application and it takes care of the communication with the underlying application semantics process. Currently, D2M2run is designed for user interfaces that resemble the OPEN LOOK user interface style definition (OPEN LOOK, 1989). D2M2edit is the tool in the D2M2environment to implement design specifications of ICSs; it also allows for maintaining and documenting ICSs. Because D2M2run is based on OPEN LOOK, design specifications to be entered in D2M2edit will contain OPEN LOOK entities like windows, buttons, etc. In D2M2environment we use the XView implementation of OPEN LOOK (Heller, 1990) associated with the C++ programming language. Both D2M2edit and D2M2run run under Sun's OpenWindows (OpenWindows, 1990).
The D2M2 environment contains methods, techniques and a methodology in D2M2 design; further, the D2M2 environment contains the D2M2 edit and D2M2 run tools. D2M2 edit and D2M2 run both constitute the actual D2M2 UIDS.

In the following sections we discuss the data flows and the processes depicted in figure 4.1. We will discuss them in a sequence so that first the input data flow of a process and the output data flow of a process will be dealt with, before we start to discuss the process itself. Concretely, section 4.2 describes the input data flow of the "design ICS"-process; section 4.3 describes the output data flow of the "design ICS"-process (i.e. design specifications); section 4.4 discusses the "design ICS"-process itself; section 4.5 describes the output data flows of the "implement ICS"-process (i.e. ui-run-time specifications and as-run-time specifications); section 4.6 discusses the "implement ICS"-process itself; section 4.7 discusses the "present user interface"-process and "execute application semantics"-process, together. We summarize and conclude in section 4.8.

4.2. INPUT TO "DESIGN ICS"

In this section we propose a general structure of the input to D2M2 design. The choice of the structure influences the determination of activities in D2M2 design. E.g. JSD of Jackson (1983) is a methodology starting with a descriptive model of the real world and arriving at a prescriptive model of the information system to be developed. Although Jackson and also Sol (1984) state that starting with a descriptive model is a good working method in order to determine the real world problem, we do not discuss it. We assume that the input to D2M2 design is always a functional description of the prescriptive model of the HICS to be developed. This means that, in terms e.g. used by Sol (1982, p. 9) D2M2 design is mainly concerned with the datalogical problem in the prescriptive domain. That is, in D2M2 design the determination of data entities and the determination of processes of the HICS to be developed are the main activities.
So, the input to D2M2design is a description of what must be supported by the HICS to be developed.

We suggest that the functional description (i.e. the input to "design ICS") of the filemanager e.g. looks like:

SECTION start/stop filemanager

UNIT
The filemanager can be started from operating system level.

UNIT
The filemanager can be stopped, implying that control returns to operating system level.

END SECTION start/stop filemanager

SECTION file and directory management

UNIT
A file can be deleted from a directory.

UNIT
A file can be copied to a directory.

UNIT
The working directory can be changed.

END SECTION file and directory management

Here, a section is considered to be related to a group of tasks of the same domain. In the filemanager two sections are identified. The units in the sections correspond to tasks eventually to be performed by the (H)ICS. Additionally, the input to D2M2design could also contain data concerning certain objects of the HICS to be developed: object properties and object relations. As for the filemanager D2M2design's input could contain the following section:

OBJECTS
RELATION1
A file belongs to a directory.

We recall from chapter 2 that coordinating decisions affect the sequence in which computer tasks are performed. If there is a mutual dependency between computer tasks this can be expressed in the input to D2M2design. We choose to include such dependencies inside the SECTION of the UNITS they are related to. The filemanager does not contain any dependencies between computer tasks: every computer task can be initiated any time by the user.

So far we discuss the input to D2M2design. In the next section we look at the output of D2M2design (i.e. design specifications).

4.3. OUTPUT OF "DESIGN ICS"

Design specifications are output of the D2M2design process and input to the "implement ICS"-process. We closely relate the design specifications with the sequencing of tasks (see figure 2.3) and the Delft model (see figure 3.9). Therefore the design specifications consists, among others, of the following components, which directly correspond to the concepts depicted in figure 3.9.
- User interface data model
- Applications semantics data model
- Computer tasks
- Semantic feedback functions

Especially with graphical user interfaces the user interface data model can become unmanageable, because the application is highly interactive. Therefore we structure the user interface data model in the design specifications by identifying sets. A set consists of a number of user interface data types from the user interface data model. A set instance corresponds to a rectangular area on screen in which the system can present output and in which the user probably can enter input. If a user can enter input in a set, at a certain moment during run-time, the set is called an interaction set; if not, the set is called a feedback set. Now, the design specifications also contain the following components.

- Interaction sets
- Feedback sets

As for the user input tasks in the Delft model we introduce the following component in the design specifications.

- Interaction tasks

An interaction task is identical to a user input task if this user input task is concerned with exactly one interaction set instance. If a user input task is concerned with more than one interaction set instance, the user input task contains more than one interaction task. An interaction task is that part of a user input task which concerns a specific interaction set instance. So, in order to initiate a certain computer task, one or more interaction tasks have to be performed by the user.

As for the output tasks in the Delft model, we introduce the following component in the design specifications.

- Feedback tasks

A feedback task is identical to an output task if this output task is concerned with exactly one set instance. If an output task is concerned with more than one set instance, the output task contains more than one feedback task. A feedback task is that part of an output task which is related to a specific set instance.

Sets may be interrelated. There are at least three reasons for this.

1. While the user is entering input in a certain set instance, feedback may appear in another set instance. A set relation may help in accessing data of a set other than the set in which the interaction task is performed.
2. A certain interaction task performed in a certain set instance may initiate a computer task. During or after computer task execution a feedback task is triggered in another set instance. A set relation in this situation releases the application semantics from having knowledge of the user interface: the application semantics directs the feedback to the related set.

3. A user input task consists of more interaction tasks concerning different set instances. A set relation in this case can enable the user to perform a succeeding interaction task in a related set.

So, we introduce the following component in the design specifications.

- Relations between sets.

As for the coordinating decisions of figure 3.9, which stand for the "enable user input tasks"-task of figure 3.10, we introduce the following component.

- Interaction task enablings

If a user input task consists of more than one interaction task, the first interaction task is enabled. In that case, after the first one is performed, the user will be forced to perform the second one in a different set instance.

In summary, the design specifications of the D2M2environment consists of the following components.

1. User interface data types.
2. Interaction sets.
3. Feedback sets.
4. Interaction tasks.
5. Feedback tasks.
6. Relations between sets.
7. Application semantics data types.
9. Interaction task enablings.
10. Semantic feedback functions.

Components 1 to 6 deal with the user interface; components 7 to 10 deal with the application semantics. We discuss each of the components in detail.

At this point a diagram notation would be useful for the components and their mutual relationships. So a notation technique reflecting entities and relationships is needed. A well known technique to depict entities and relationships between entities is the technique of entity-relationship diagrams (see e.g. Martin and McClure (1985)). However, an entity-relationship diagram is not unique for a given problem. The solution depicted in such a diagram is dependent on the chosen development trajectory as identified by e.g. Ter Bekke (1991). That is why we use generalization and aggregation abstractions. Generalization and aggregation abstractions have been used successfully in semantic data modelling (Smith and Smith, 1977).
It is decided that our diagrams should support entity classification, generalization and aggregation. We use these types of diagrams also later in this chapter, in the discussion of user interface data types, interaction sets, feedback sets, set relations and application semantics data types. The notation technique introduced by Ter Bekke (1991) is the only technique that, as far as we know, allows the depiction of these concepts into one single diagram. These diagrams are therefore used throughout this dissertation.

Ter Bekke suggests that classification of an entity be depicted as a rectangle with a label inside, describing the entity classification. It is assumed that an entity classification has a unique list of properties, called attributes. In the filemanager e.g. we identify the entity classification 'file_icon' with attributes: x_position, y_position, file_label. An instance of 'file_icon' then has, during run-time of the filemanager, a unique combination of values for its attributes. Entity classifications can have relations among each other; i.e. generalization and aggregation relations (see also (Smith and Smith, 1977)).

In case of a generalization relation an entity classification is said to inherit all attributes of the other entity classification, and besides that to have some extra attributes. As an example, in the filemanager we may identify the entity classification 'icon' as well as 'file_icon' (a special icon representing a file on screen). A generalization relation can be composed between 'icon' and 'file_icon'. The 'icon' has e.g. the attributes: x_position, y_position. The 'file_icon' has moreover the extra attribute: file_label. Now, the entity classification 'file_icon' inherits the attribute values x_position and y_position from 'icon'. So, as soon as, during run-time of the filemanager, an instance of 'file_icon' is created, it can be identified by the attribute values of x_position, y_position and file_label, although the former two attributes actually belong to the entity classification 'icon'. In this case the entity classification 'icon' is called the generalized entity classification; the entity classification 'file_icon' is called the specialized entity classification. A generalization relation is depicted by connecting the lower left corner of the specialized entity classification with the upper right corner of the generalized entity classification. In figure 4.2 the described example of a generalization relation is depicted.

![Figure 4.2: example of a generalization relation](image)

The little bubble on the generalization line has no meaning; it is introduced as a manipulation point to delete the generalization relation in D2M2edit.

In case of an aggregation relation an entity classification is said to always have the other entity classification associated with it. As an example from the real world: suppose we have cars and number plates. We then identify the entity classifications 'car' and 'number_plate'. An aggregation relation between 'car' and 'number_plate' can now be composed: a car always has a number plate associated with it. In this case, the entity classification 'car' is called the aggregated entity classification; the entity classification 'number_plate' is called the decomposed entity classification. An aggregation relation is depicted by connecting the lower line of the ag-
ggregated entity classification with the upper line of the decomposed entity classification. In figure 4.3 the described example of an aggregation relation is depicted.

```
  car
     ↓
  number_pla
     ↑
   te
```

Figure 4.3: example of an aggregation relation

Again, the little bubble on the aggregation line has no meaning; it is introduced as a manipulation point to delete the aggregation relation in D2M2edit.

Attributes of entity classifications are not visualized in the same diagram as the entity classifications, generalization relations and aggregation relations; they must be specified separately, textually.

So far, we discuss the notation technique proposed by Ter Bekke. For a more extensive overview we refer to Ter Bekke (1991).

Referring back to the components in the output of D2M2design, we identify each of the ten components as separate entity classifications. We further identify the additional concepts of the Delft model (i.e. user input task, output task and coordinating decision) as entity classifications. In this case attributes of the entity classifications are irrelevant; only the entity classifications and relations among them are identified.

So, we identify the entity classifications 'ui_data_type', 'interaction_set', 'feedback_set', 'interaction_task', 'feedback_task', 'set_relation', which correspond with the first six summarized components of the output of D2M2design. These entity classifications all deal with the user interface and are therefore all specialized entity classifications of a generalized entity classification 'ui_component'; this is depicted in figure 4.4.

We also identify the entity classifications 'as_data_type', 'computer_task', 'interact_task_enabl' and 'sem_fb_function', which correspond with the components 7 to 10 of the output of D2M2design. These entity classifications all deal with the application semantics and are therefore all specialized entity classifications of a generalized entity classification 'as_component'; this is depicted in figure 4.5.

We recall that an interaction task is always part of a user input task. Further, the same interaction task can be part of several user input tasks. We introduce an extra entity classification by which the described relations can be depicted (see figure 4.6).

An output task generally consists of several feedback tasks. The same feedback task can be part of several output tasks. We introduce an extra entity classification by which the described relations can be depicted (see figure 4.7).

Further, a relation holds between the entity classifications 'coordinating_dec' and 'interact_task_enabl', for each interaction task enabling is part of a coordinating decision.
Further, the same interaction task enabling may be part of several coordinating decisions. In figure 4.8 this is depicted by an extra entity classification.

*Figure 4.4: specialized entity classifications of 'ui_component'*

*Figure 4.5: specialized entity classifications of 'as_component'*

*Figure 4.6: aggregation relations concerning 'interaction_task' and 'user_input_task'*
Between a user interface data type, interaction set and feedback set the following relations hold: a user interface data type can be used in an interaction set as well as in a feedback set. We add two entity classification to the three relevant entity classifications (being `ui_data_type', 'interaction_set' and 'feedback_set') in order to express the relations which hold. The two new entity classifications are: `ui_data_type_in_IS' and `ui_data_type_in_FS'. They are used to indicate whether a user interface data type belongs to an interaction set or feedback set or probably both. In figure 4.9 the discussed relations are depicted with the two new entity classifications.

An interaction task has always an interaction set associated with it. In figure 4.10 this is depicted via an aggregation relation.

A feedback task can be associated with an interaction set as well as a feedback set. We add two new entity classification ('feedback_task_in_IS' and 'feedback_task_in_FS' respectively) to express this. In figure 4.11 the relations concerning entity classification 'feedback_task' are depicted.
Figure 4.11: aggregation relations concerning feedback tasks and sets

As for relations between sets: an interaction set can be related to another interaction set, and an interaction set can be related to a feedback set. Set relations between feedback sets never occur as can be determined from the reasons for introducing set relations (see pages 45 and 46). We introduce two new entity classifications in order to express the above described relations: 'IS_relation_IS' and 'IS_relation_FS' (for an interaction set - interaction set relation and an interaction set - feedback set relation respectively). In figure 4.12 this is depicted.

Figure 4.12: aggregation relations concerning set relations and sets

Note that the two lines between the entity classifications "IS_relation_IS" and "interaction_set" depict that a set relation between interaction sets is always related to two interaction sets.

The figures 4.4 to 4.12 can be joined into one diagram containing all generalization relations, aggregation relations and all entity classifications. We do not depict this diagram for complexity reasons.

We now discuss the several components of the design specifications subsequently. To illustrate the components we use the filemanager ICS of chapter 3 (see figure 3.3).

user interface data types (1):

User interface data types are classifications of objects which are often visual on screen. In the filemanager 'file_icon' is a user interface data type; further, 'directory_icon', 'trash_can', 'frame' and 'icon_manager' are user interface data types. For the description of the user interface data types we use the same concepts as those that are the basis of the notation technique of Ter Bekke. So, user interface data types are identified as entity classifications which have attributes. Further, user interface data types have generalization relations among each other. However we do not take aggregation relations into account. This is done because we are convinced that little semantics are added to user interface data type definitions by identifying logical relations (i.e. aggregation relations) between user interface data types, if instances of these types are physically presented on screen; it makes more sense if we take physical relations into account as far as user interface data types are concerned. This is, then, the course we take.
There is another difference in concepts used for describing the user interface data types: we add operations to the user interface data types specification. An operation can be compared with a method in for example Smalltalk (Goldberg and Robson, 1983) or C++ (Stroustrup, 1986); only through operation activation the attribute values of a certain instance of a user interface data type can be updated. A certain operation is linked to a certain user interface data type, like an attribute is linked to a user interface data type. So, a user interface data type contains a number of attributes as well as a number of operations. An implication of adding operations to user interface data types is that two or more specialized user interface data types can now be generalized if they have, not only attributes, but also operations in common.

The reason for adding operations to user interface data types is that the specification of the user interface data types can now easily be translated into C++, being the object language of D2M2.

E.g. the user interface data types 'file_icon', 'directory_icon' and 'trash_can' of the filemanager can be generalized to the user interface data type 'icon', which contains e.g. the attributes x_position, y_position, width, height and a label. The user interface data types 'file_icon', 'directory_icon' and 'trash_can' all have unique ways of being displayed; they each contain their own unique display operation, e.g. display_file_icon(), display_directory_icon() and display_trash_can(), respectively. Note that an icon itself is never visualized on screen; nevertheless, it is a user interface data type.

We are now able to present a possible description of the user interface data types of the filemanager.

<table>
<thead>
<tr>
<th>USER INTERFACE DATA TYPES</th>
<th>ATTRIBUTES</th>
<th>OPERATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame</td>
<td>x-coordinate</td>
<td>display_frame</td>
</tr>
<tr>
<td></td>
<td>y-coordinate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>width</td>
<td></td>
</tr>
<tr>
<td></td>
<td>height</td>
<td></td>
</tr>
<tr>
<td>icon_manager</td>
<td>x-coordinate</td>
<td>display_icon_manager</td>
</tr>
<tr>
<td></td>
<td>y-coordinate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>width</td>
<td></td>
</tr>
<tr>
<td></td>
<td>height</td>
<td></td>
</tr>
<tr>
<td>icon</td>
<td>x-coordinate</td>
<td>return_x_coordinate</td>
</tr>
<tr>
<td></td>
<td>y-coordinate</td>
<td>return_y_coordinate</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>return_label</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td></td>
</tr>
<tr>
<td></td>
<td>label</td>
<td>display_icon</td>
</tr>
<tr>
<td>file_icon (inherits from</td>
<td>-</td>
<td>move_file_icon</td>
</tr>
<tr>
<td>icon)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>directory_icon (inherits</td>
<td>-</td>
<td>display_directory_icon</td>
</tr>
<tr>
<td>from icon)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trash_can (inherits from</td>
<td>-</td>
<td>display_trash_can</td>
</tr>
<tr>
<td>icon)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Note that probably not all attributes and operations are taken into account. Further, during runtime, the file manager has only one instance of a frame, icon_manager and trash_can. The instance of frame (Frame in XView) contains the functional part of the file manager. The instance of icon_manager (Canvas in XView) is the area in which icons can be manipulated.

Using the notation technique of Ter Bekke we visualize the user interface data types and generalization relations of the file manager as depicted in figure 4.13.

![Diagram of data types and relations of the file manager]

**Figure 4.13: data types and relations of the file manager**

Note that the data types 'native' and 'own' are introduced to every ICS; all specializations of 'native' inherit attributes and operations of the data types that belong to XView; all specializations of 'own' inherit attributes and operations of data types not directly related to XView. Both 'native' and 'own' contain a number of attributes and operations, common to all standard data types of XView and non-standard data types of XView respectively.

Figure 4.14 depicts the attribute 'label' of user interface data type icon of the file manager.

**Type name:** icon  
**Description:** square on screen with visual information  
**Simple attributes**  
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>label (text,25)</td>
<td>text</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 4.14: single attribute of user interface data type icon in file manager**
Note that all other attributes, like x-coordinate and y-coordinate are inherited from 'own'.

Figure 4.15 depicts the methods of user interface data type file_icon.

![Diagram](image)

**Figure 4.15: methods of user interface data type file_icon**

Generalized operations are those operations that are inherited from generalized data types (in this case the data types 'icon' and 'own'). Specialized operations are those operations that are directly associated with file_icon. With the help of the depicted buttons, new operations can be constructed.

*interaction sets (2):*

An interaction set is a collection of user interface data types, of which instances are perceived as a physical unit by the user. An interaction set on screen is a physical region in which the user can perform manipulations and to which feedback tasks can be sent. Together with the identification of user interface data types in an interaction set, parent-child relations have to be specified between the identified user interface data types. These are the physical relations we mentioned when discussing the user interface data types component of the output of D2M2design. A parent-child relation between two user interface data types means that an instance of the child data type is always displayed inside an instance of its parent data type on screen.

It is quite natural to identify only one interaction set in this filemanager. Deleting a file, copying a file, changing the working directory and leaving the filemanager are all operations which can be initiated in the same rectangular region. In the following the user interface data types and the parent-child relations of the only interaction set of the filemanager are identified.

```
PARENT: frame          CHILD: icon_manager
PARENT: icon_manager   CHILD: directory_icon
                     CHILD: file_icon
                     CHILD: trash_can
```

So, icon_manager is the parent of directory_icon, file_icon, trash_can; and frame is the parent of icon_manager.
Again we use the notation technique of ter Bekke. Because aggregation and generalization relations are not relevant in interaction sets, they do not occur in the diagram that describe interaction sets. We choose arrows to depict parent-child relations between user interface data types. The tail of the arrow is connected to the base line of the rectangle of the parent data type; the head of the arrow is connected to the top line of the rectangle of the child data type.

Note that the entity classifications in the diagram of an interaction set must be existing data types, already defined as user interface data types.

In figure 4.16 the only interaction set of the file manager is visualized.

![Diagram](image)

*Figure 4.16: parent-child relations between data types in file manager*

**feedback sets (3):**

A feedback set is an interaction set in which the user cannot manipulate objects; only through feedback tasks the layout of such a feedback set can be updated. The file manager does not contain a feedback set. A feedback set e.g. could be used as a rectangular region which contains the last produced error message or help message of an ICS.

Analogous to specifying interaction sets, feedback sets must be specified through user interface data types and parent-child relations.

**interaction tasks (4):**

A user input task is a task a user performs in order to initiate a computer task (see figure 2.3), while an interaction task is part of that user input task. Different user input tasks may initiate the same computer task. In the file manager, however, every computer task can be initiated by only one user input task. Moreover, in the file manager, every user input task corresponds to one instance of an interaction set, i.e. a user input task is identical to an interaction task. We show every interaction task and associated computer task of the file manager in the following.
INTERACTION TASK

1) Bring cursor over file_icon, press left mouse button, drag file_icon over trash_can, release left mouse button.

2) Bring cursor over file_icon, press left mouse button, drag file_icon over directory_icon, release left mouse button.

3) Bring cursor over directory_icon, press and release left mouse button.

4) Bring cursor over trash_can, press and release left mouse button.

COMPUTER TASK

1) Delete file.

2) Copy file.

3) Change directory.

4) Leave filemanager.

Several techniques exist for specifying the structure of the dialogue. Green (1986) gives an overview of three major techniques: transition networks, grammars and event models.

A transition network is a set of transition diagrams. "A transition diagram consists of a set of states (represented by circles) and a set of arcs (represented by arrows leading from one state to another). The states represent the states in the dialogue between the user and the computer system. In the simplest form of transition networks, each arc is labeled by an action (an input token) that the user can perform. The arcs in the diagram determine how the dialogue moves from one state to another. The dialogue will move from state A to state B if there is an arc between these two states labeled by the action the user performed. A path through a transition diagram is a sequence of arcs that lead from the start state of the diagram to one of its final states. A sequence of user actions is accepted if they label the arcs on a path through the diagram." (Green, 1986, p. 247). To reduce diagram complexity, a transition diagram may call subdiagrams. In order to support subdiagram calls in certain transition diagrams, any arc can be labeled by a call to a subdiagram.

A grammar can also be used to describe the language employed by the user to communicate with the computer. The terminals in the grammar represent user's actions or system actions. Terminals are combined by the productions in the grammar to form higher level structures called non-terminals.

In the event model there is an arbitrary number of event types. Generally, most of them originate from the user. When an event is generated, it is sent to one or more event handlers, which process events. When an event handler receives an input it can process, it executes a procedure (method). This procedure may perform some computation, generate new events, call application procedures, etc. The behaviour of an event handler is programmed in a programming language.

We have chosen to specify interaction tasks by transition diagrams, because of their natural graphical representation (see also Green (1986)). For specifying an interaction task we only have one transition diagram and no subdiagrams: because our interaction task is always associated with one interaction set, the interaction task (and thus the transition diagram) will gen-
erally be simple. If a user input task deals with more than one set instance, more than one interaction task has to be executed in order to allow for computer task execution.

Figure 4.17 depicts the interaction task to delete a file in the file manager.

Note that s0 is the initial state. The final state (s2) is indicated by a double circle. Further, a "T" consists of a premise part and a conclusion part. If the premise part holds, the conclusion part is executed and the arc is traversed. The premise part consists of a user action, probably a consultation of user interface data and probably a consultation of application semantics data; the conclusion part probably consists of an update of user interface data. If, in a premise, a consultation of application semantics data is needed a semantic feedback function must be executed before traversing the arc; then, the conclusion part is said to present semantic feedback. Premises and conclusions must be programmed in C++, being the object language of D2M2.

We recall that as soon as the interaction task reaches the final state, control switches either to the application or to another interaction set instance from which a following transition diagram can be traversed. In the first case the user input task only deals with one interaction set instance; in the second case the user input task deals with more than one interaction set instance.

Further, in the first case user information must be directed from the user interface to the application semantics. The user information should be abstracted from the user interface data and sent to the application semantics. This is done through C++ (being the object language of D2M2) code that must be specified in the, so called, "data transfer".

Different user input tasks may initiate the same computer tasks. The corresponding interaction tasks, by which interaction starts, all have the same "entry name". When the application semantics enable interaction tasks (see also figure 3.15) entry names are supplied by the application semantics; this enforces that all interaction tasks, which belong to the supplied entry names, are enabled, i.e. the user can start executing the interaction tasks. It also has the advantage that the application semantics process does not have to have knowledge of the names of interaction tasks, but only of the entry names.

User input tasks which deal with more than one interaction set instance will only be finished when all corresponding transition diagrams have been traversed. If a final state of the transition diagram of the interaction set, in which user interaction was started, is reached, the data transfer determines in what interaction set and through what entries the user interaction may proceed. If the user interacts in the last interaction set, in which user interaction is allowed in order to satisfy the total user input task, and reaches the final state, the user information is directed from the user interface to the application semantics together with the entry name of the interaction task in which the user input task execution started.
feedback tasks (5):

If a certain output task is linked only with one single set instance, the output task is identical to the feedback task; however, if a certain output task is linked with more than one set instance, the output task contains more than one feedback task. So, an output task execution is the overall output on screen, after computer task execution or during computer task execution (then called 'intermediate’ output task execution), while feedback task execution is part of the output task execution. In the filemanager, however, every output task deals with one instance of a set, so, an output task is identical to a feedback task. By means of a feedback task an instance of an interaction or feedback set can be created, or deleted. But also, by means of a feedback task, instances of data types in an interaction or feedback set can be created, updated or deleted. As for the filemanager we have the following feedback tasks.

**FEEDBACK TASK CONCERNING SET**

1) Create frame and icon_manager with directory_icons and file_icons of current directory and with trash_can.

2) Delete frame and icon_manager with directory_icons and file_icons of current directory and with trash_can.

**FEEDBACK TASK CONCERNING DATA TYPE(S)**

3) Delete all directory_icons and file_icons of old current directory and display all directory_icons and file_icons of new current directory.

4) Delete a particular file_icon.

Feedback task 1 is associated with the initial presentation of the filemanager on screen. Feedback task 2 is associated with the computer task "leave filemanager". Feedback task 3 is associated with the computer task "change directory". Feedback task 4 is associated with the computer task "delete file".

Because C++ is the object language of D2M2 we decided to choose to construct feedback tasks in C++ for the time being. We identify the following types of feedback tasks in the design specifications.

1. The creation code for an instance of an interaction or feedback set, specified in C++.

2. The creation code for an instance of a user interface data type in an interaction or feedback set, specified in C++.

3. The C++ code of a feedback task, belonging to an interaction or feedback set, by which updates or deletions of instances of data types can be performed; or, by which deletions of instances of interaction or feedback sets can be performed.

relations between sets (6):

A user input task may be concerned with different set instances; it then contains different interaction tasks. Performing these different interaction tasks most certainly implies that user interface data from different set instances must be forwarded to one another. Therefore, relations between sets may be needed. Through these relations a certain set instance can get information from the user interface data of another set instance.
Another application of relations between sets occurs with feedback tasks. If a certain interaction task always implies that a subsequent feedback task has to be directed to another set instance, a relation between the two sets can help: now, in the application it is not necessary to specify the set instance for the feedback task explicitly, because the interaction task provides the specification of the set for the feedback task indirectly, through the relation which exists between the two set instances.

A third application of relations between sets occurs when user input in a certain interaction set instance triggers feedback in another set instance.

The filemanager only contains one set instance, so relations between sets do not make sense. The HICSs discussed in chapter 5 and 6 do have relations between sets.

Again we use the notation technique of Ter Bekke to depict set relations. Sets are only related through aggregation relations; generalization relations do not occur. In figure 4.18 a relation between two sets is depicted, as it can be represented in D2M2edit.

![Figure 4.18: relation between sets](image)

Note that the relation of figure 4.18 expresses that an instance of set1 always has an instance of set2 associated with it.

**application semantics data types (7):**

Application semantics data types are classifications of objects which are identified in the application. In the filemanager 'file' and 'directory' are application semantics data types, because they are the data types of interest in the application. For the description of the application semantics data types we use the same concepts as those concepts that form the basis of the notation technique of Ter Bekke. So, application semantics data types are identified as entity classifications which have attributes and operations. Further, application semantics data types have generalization and aggregation relations. In contrast with the description of user interface data types, the description of application semantics data types can contain aggregation relations; now physical relations add little semantics to the application semantics data types.

We show the two application semantics data types of the filemanager together with their aggregation relation in the following.
APPLICATION SEMANTICS DATA TYPES | ATTRIBUTES | OPERATIONS
---|---|---
directory | current | return_current
 | directory_name | make_current
 | directory_path | make_not_current
file (belongs to directory) | file_pointer | return_directory_name
 | file_name | return_directory_path
 | | return_file_pointer
 | | return_file_name

So, every file is associated with a directory by an aggregation relation. Note that the attributes and methods presented are not completely enumerated.

Using the notation technique of Ter Bekke we can visualize the application semantics data types and relation of the file manager as follows in figure 4.19.

![Diagram](image)

Figure 4.19: the application semantics data types in the file manager

computer tasks (8) and interaction interaction task enablings (9):

Computer tasks constitute the functionality of an ICS. They update, create and/or delete instances of application semantics data types. In the file manager the following computer tasks can be identified.

Initialize file manager: create instances of application semantics data types for every physical directory and file in the computer system.

Delete file: delete an instance of the application semantics data type file and its physical counterpart in the computer system.

Copy file: create an instance of the application semantics data type file and its physical counterpart in the computer system in the given directory.

Change current directory: update old current application semantics data type directory as not current; update new current application semantics data type directory as current.

Leave file manager: delete all instances of file data types and directory data types; keep the physical directories and files of the computer system untouched.
Coordinating decisions define the flow of control of the ICS: they prescribe what interaction tasks the user can start in order to let execute a certain computer task. A coordinating decision consists of one or more 'enablings' of interaction tasks. In the filemanager the user can start any time any interaction task. Figure 4.20 depicts the flow.

Besides computer tasks (i.e. Initialize filemanager, Delete file, Copy file, Change current directory and Leave filemanager) also feedback tasks (i.e. feedback task 1, feedback task 2, feedback task 3 and feedback task 4) are triggered.

![Flowchart](image-url)

*Figure 4.20: flow of application semantics part in filemanager*
For the moment, the specifications of computer tasks and interaction task enablings are combined in the design specifications. The application semantics code (in C++) contains the computer tasks and the interaction task enablings of the ICS. As an example the C++ code for the file manager is depicted in the following. Note that a computer task is visualized boldly, a trigger of a feedback task is underlined, and an interaction task enabling is visualized in italics.

```
initialize FileManager();
feedback task 1();
// feedback_task_1 projects the initial layout of the file manager
// on screen
enable delete file entry();
enable copy file entry();
enable change current directory entry();
enable leave FileManager_entry();

interaction = wait_for_interaction_task_execution_by_user();
while (interaction != leave filemanager_entry) {
    switch (interaction) {
        case delete file entry:  delete file();
            feedback_task 4();
// feedback_task 4 removes a particular file_icon from screen
            break;

        case copy file entry:     copy file();
            break;

        case change current directory entry:  change current dir();
            feedback_task 3();
// feedback_task_3 deletes all the file_icons and directory_icons
// of the old current directory from screen and projects the file_icons
// and directory_icons of the new current directory on screen
            break;
    }

    enable delete file entry();
    enable copy file entry();
    enable change current directory entry();
    enable leave FileManager_entry();

    interaction = wait_for_interaction_task_execution_by_user();
}

leave FileManager();
feedback task 2();
// feedback_task_2 removes the frame, and icon manager with file_icons,
// directory_icons and trash_can from screen
```
**semantic feedback functions (10):**

Semantic feedback means that during execution of an interaction task, application semantics data must be consulted. We recall from section 3.3 the situation in which semantic feedback was added in the file manager in the interaction task which initiated the "delete file" computer task. The extended scenario for this interaction task is: bring the cursor over a file_icon, press the left mouse button, drag the file_icon over the trash_can; if the file is not protected, invert the trash_can; release the left mouse button. For this scenario the application semantics data type 'file' can be extended as follows.

<table>
<thead>
<tr>
<th>APPLICATION SEMANTICS DATA TYPES</th>
<th>ATTRIBUTES</th>
<th>METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>file_pointer</td>
<td>return_file_name</td>
</tr>
<tr>
<td></td>
<td>file_name</td>
<td>return_filepointer</td>
</tr>
<tr>
<td></td>
<td>protected</td>
<td>return_protected</td>
</tr>
</tbody>
</table>

We describe the semantic feedback function in C++, for C++ is the object language of D2M2.

**SEMANTIC FEEDBACK FUNCTION**

```cpp
BOOLEAN file_not_protected() {
  RETURN NOT(return_protected());
}
```

This semantic feedback function must be called when dragging the file_icon over the trash_can.

So far the discussion has dealt with the design specifications. We now discuss the actual D2M2 design process.

### 4.4. DESIGN ICS

It is not our intention to make a complete summary of methods and techniques that can be used in D2M2 design. We describe what steps are needed in order to translate the input specifications to design specifications, and we do not discuss in detail how these steps must be performed.

D2M2 design consists of two main steps: an analysis step and a design step.

**STEP 1: analysis**

Every UNIT from the functional description, which is input to D2M2 design, should correspond with one cycle of the sequencing in human-computer interaction as depicted in figure 2.3; such a cycle contains a user input task, a computer task and probably more than one output tasks. From every UNIT the user input task, the computer task and (probably more than one) output tasks must be identified.

**STEP 2: design**

Firstly, the computer tasks and the application semantics data types must be designed. Computer tasks and application semantics data types must be designed cooperatively, because they are mutually dependent: a computer task activates operations of application semantics data types. This activity uses the computer task identification of STEP 1. As for the application se-
mantics data types the entity classifications, generalization relations, aggregation relations, attributes and operations must be designed. Out of the computer tasks the code (which contains calls to application semantics data types operations) must be designed.

Secondly, the interaction tasks, the feedback tasks, the interaction and feedback sets, the user interface data types and set relations must be designed and it must be determined which interaction tasks contain calls to semantic feedback functions. All these activities must be performed cooperatively, because they are mutually dependent. From the identification of user input tasks of STEP 1 interaction tasks can be identified if we determine the interaction sets. From the identification of output tasks of STEP 1 feedback tasks can be identified if we determine the interaction and feedback sets. Interaction tasks and feedback tasks activate operations of user interface data types. As for the user interface data types the entity classifications, generalization relations, attributes and operations must be designed. As for the interaction and feedback sets the layout and the parent-child relations must be designed. For the set relations aggregation relations must be identified between sets. After identifying interaction tasks the transition diagrams (containing premise parts and conclusion parts, which contain calls to user interface data type operations) and data transfer must be designed and entry names must be specified. After identifying feedback tasks, the code for feedback tasks (which contains calls to user interface data type operations) must be designed. In determining the user interface data model and the application semantics data model the OBJECTS parts of the functional description can be consulted.

Thirdly, the semantic feedback functions must be coded. The code contains calls to operations of application semantics data types.

Fourthly and finally, interaction task enablings must be determined and together with the computer tasks and triggers to feedback tasks coded in a way as presented on page 62. In determining the interaction task enablings the functional description, which is the input to D2M2design, can be consulted. Note that not the interaction tasks itself are enabled, but the entries to which the interaction tasks belong.

4.5. OUTPUT OF "IMPLEMENT ICS"

Run-time specifications are generated by the "implement ICS"-process. They can be divided in run-time specifications for the user interface and run-time specifications for the application semantics. We start with a discussion of the run-time specifications for the user interface.

USER INTERFACE RUN-TIME SPECIFICATIONS

As for the user interface design specifications we sequentially discussed user interface data types, interaction sets, feedback sets, interaction tasks, feedback tasks and set relations in section 4.3. We do the same with user interface run-time specifications. Because C++ is the object language of D2M2, user interface run-time specifications and application semantics run-time specifications are specified in C++-code.

user interface data types:

The user interface data types in the design specifications are translated into C++ classes by the "implement ICS"-process. As for the file manager, we identify the following C++ classes in the run-time specifications (see also figure 4.13):
class icon {
    private:
    char label[80];
    public:
    char *return_label();
}

class file_icon : public icon {
    private:
    public:
    display_file_icon(int x, y);
    move_file_icon(int x, y);
}

class directory_icon : public icon {
    private:
    public:
    display_directory_icon(int x, int y);
}

class icon_manager {
    private:
    public:
    display_icon_manager();
}

class frame {
    private:
    public:
    display_frame();
}

class trash_can : public icon {
    private:
    public:
    display_trash_can();
}

interaction sets:

For each interaction set in the ICS, the "implement ICS"-process generates a C++ function call 'AddInteractionset()' in the run-time specifications; this function has the following parameters: the name of the interaction set (as a string), the name of the interaction set (in capitals), the name of the creation function for instantiating an interaction set and (always) an 0, respectively. So, specifying the filemanager, the following function call will be generated:

AddInteractionset("FileManager", (unsigned int) FILEMANAGER,
(fie_p) crf_FileManager, 0);

Further, the "implement ICS"-process translates every parent-child relation between user interface data types into a C++ function call 'AddParentChildRelation()'. This function contains three parameters: the name of the interaction set, the parent type and the child type. As for the filemanager the following calls to the function exist (see also figure 4.16):

AddParentChildRelation(FileManager, NULL, UI_FRAME);
AddParentChildRelation(FileManager, UI_ICONMANAGER, UI_DIRECTORY_ICON);
AddParentChildRelation(FileManager, UI_ICONMANAGER, UI_TRASH_CAN);
AddParentChildRelation(FileManager, UI_ICONMANAGER, UI_FILE_ICON);
AddParentChildRelation(FileManager, UI_FRAME, UI_ICONMANAGER);

feedback sets:

For feedback sets the same procedure holds, except that instead of the function AddInteractionset(), the function AddFeedbackset() must be generated for every feedback set in the ICS.
interaction tasks:

All interaction tasks of a certain interaction set, must be combined into one large interaction diagram, represented as a data structure in D2M2run, which is evaluated during run-time by D2M2run. In each transition between states in the interaction diagram it is remembered to which interaction task it belongs. In the run-time specifications the "implement ICS"-process generates C++-code, including calls to AddState(), in order to allow for creation of a state in the interaction diagram; via generated function calls to SetEndState(), end states of interaction tasks are added in the interaction diagram; via generated function calls to AddTransition(), transitions are added in the interaction diagram; via generated function calls to AddTransitionTask(), a transition will be linked to a certain interaction task in the interaction diagram. To summarize, the functions AddState(), SetEndState(), AddTransition() and AddTransitionTask() build up the interaction diagram.

AddState() has parameters: the name of the interaction set, the name of the state (a string), and a boolean indicating whether the state is an initial state, respectively. SetEndState() has parameters: the name of the state, the name of the entry (in capitals, preceded by the name of the interaction set), the name of the interaction task the state belongs to (in capitals, preceded by the name of the interaction set), the name of the "data transfer"-function, and the name of the 'aftermath'-function ('aftermath' is not further worked out), respectively. AddTransition() has parameters: the 'from'-state, the 'to'-state, the name of the transition (a string, followed by the name of the interaction task), the name of the premise function, and the name of the conclusion function, respectively. AddTransitionTask() has parameters: the transition and the name of the interaction task.

For the filemanager the "implement ICS"-process generates for the "delete a file"-interaction the following code as part of the user interface run-time specifications (see also figure 4.17):

```c++
...
...
...
...
s0 = AddState(FileManager, "s0", 1);
s1 = AddState(FileManager, "s1", 0);
s2 = AddState(FileManager, "s2", 0);
SetEndState(s2, FILEMANAGER_ENTRY_DELETE, FILEMANAGER_TASK_DELETE_FILE,
            (tie_p) dtf_delete_file, (tie_p) afm_delete_file);
t0 = AddTransition(s0, s1, "T0_delete_file", (tie_p) T0_delete_file_prms,
                   (tie_p) T0_delete_file_act);
AddTransitionTask(t0, FILEMANAGER_TASK_DELETE_FILE);
t1 = AddTransition(s1, s1, "T1_delete_file", (tie_p) T1_delete_file_prms,
                   (tie_p) T1_delete_file_act);
AddTransitionTask(t1, FILEMANAGER_TASK_DELETE_FILE);
t2 = AddTransition(s1, s2, "T2_delete_file", (tie_p) T2_delete_file_prms,
                   (tie_p) T2_delete_file_act);
AddTransitionTask(t2, FILEMANAGER_TASK_DELETE_FILE);
...
...
...
...
```

Other interaction tasks add their states and transitions to the same diagram by the discussed functions. Note that a state is only added in the interaction diagram, if no path from the initial state of the interaction diagram to that state already exists.
feedback tasks:

We recall from the discussion of the interaction sets that the "implement ICS"-process generates a function AddInteractionSet (or AddFeedbackset), in which one of the parameters is the creation code of an instance of the set. Such a creation is one type of feedback task. Creation of an instance of a user interface data type in an existing interaction or feedback set can be accomplished because the "implement ICS"-process generates the function AddCreation() for every instance that can be created in an interaction or feedback set. The parameters of this function are: the name of the set, the identification of the task (in capitals, preceded by the name of the set), and the name of the creation function, respectively. The third kind of feedback task (i.e. that feedback task which updates or deletes instances of data types of an interaction or feedback set) can be accomplished, because the "implement ICS"-process generates the function AddOutput() for every feedback task of the third kind. The parameters of this function are: the name of the set, the name of the feedback task (in capitals, preceded by the name of the set) and the name of the feedback task function, respectively.

In the example of the file manager e.g. the following C++-code will be generated:

```cpp
AddCreation(FileManager, FILEMANAGER_CREATE_FILE, 
(file_p) crf_file_icon);
AddOutput(FileManager, FILEMANAGER_OUTPUT_DELETE_FILE, 
(file_p) ophtf_delete_file_icon);
```

relations between sets:

As for relations between sets the "implement ICS"-process generates a function AddSetRelation() for every relation there exists between sets. The parameters of this function are: the name of the parent set and the name of the child set. Since we do not have set relations in the file manager, there are no calls to the AddSetRelation() function in the user interface run-time specifications.

So far the discussion of the user interface run-time specifications, which, as we have presented, mainly consists of generated C++-classes representing user interface data types and calls to specific functions.

APPLICATION SEMANTICS RUN-TIME SPECIFICATIONS

For each component in the application semantics design specifications we discuss the generated run-time specifications.

application semantics data types:

The application semantics data types in the design specifications are translated into C++ classes by the "implement ICS"-process. As for the file manager we identify the following classes in the application semantics run-time specifications. Note that the attributes and operations of both types are not described exhaustively.
class directory {
    private:
    short current;
    char directory_name[80];
    char directory_path[80];
    public:
    short return_current();
    void make_current();
    void make_not_current();
    char *return_directory_name();
    char *return_directory_path();
}

class file {
    private:
    directory *dir;
    FILE *file_pointer;
    char file_name[80];
    public:
    char *return_filename();
    FILE *return_file_pointer();
    directory *return_dir();
}

computer tasks and interaction task enablings:

Actually, the "implement ICS"-process does not change any design specifications concerning computer tasks or interaction task enablings, but just forwards the specifications to the application semantics run-time specifications. So, the application semantics run-time specifications for interaction task enablings and computer tasks resembles what is presented earlier in section 4.3. on page 62.

semantic feedback functions:

As with computer tasks and interaction task enablings, design specifications (in C++) of semantic feedback functions are not translated by the "implement ICS"-process, but forwarded to the application semantics run-time specifications.

So far the discussion has considered the application semantics run-time specifications: application semantics data types are translated to C++-classes; other specifications are not translated, but forwarded.

4.6. IMPLEMENT ICS

The "implement ICS"-process consists of two major subprocesses: the "input design specifications"-process and the "generate code"-process. Input of design specifications is the major part of the "implement ICS"-process. The "generate code"-process can be activated, when the input of design specifications is completed. Figure 4.21 depicts the Data Flow Diagram of the "implement ICS"-process.

It is also possible, as can be seen in figure 4.21, to save and load intermediate files. With this option, already entered design specifications can be maintained.
Figure 4.21: Data Flow Diagram of the "implement ICS"-process

D2M2edit is the tool that fully supports the "implement ICS"-process. The main functions D2M2edit provides can be summed up as follows.

1. **Implementation of ICSs**
   This includes implementation of the user interface components, the application semantics components and the communication between these two in the run-time environment.

2. **Management of reusable interface components**
   Through extensive reuse of user interface components from previous projects, substantial savings in designing, building, testing and training can be made. D2M2edit offers a user interface library for storage and retrieval of user interface components.

3. **Support for documentation and administrative tasks**
   By providing simple and efficient documentation facilities, concise and faultless reports can be generated, while support for project management, like authorization, can ease administration.

4. **Conversion of design specifications to run-time specifications**
   To be able to run an application, specified with D2M2edit, these design specifications may be transformed to specifications which can be interpreted by the D2M2run run-time environment.

Since D2M2edit must be used by non-programming designers and administrators, it offers a graphical interface, based on Sun's OPEN LOOK standard.

The main functions of D2M2edit are shown in the editor window after starting up. Figure 4.22 depicts the editor window with the pull down menus to activate the main functions of D2M2edit.
The relation between the design specifications and the submenu of D2M2edit is obvious. Note that "Application Code" represents computer tasks and interaction task enablings.

One can choose between library maintenance (not implemented yet) and application (project) management. Only the latter will be presented, for this is the prime function of the development environment. Selecting application management brings up its menu, offering the following functions.

1. Application file management, which comprises retrieval, storage, cataloguing, creation and deletion of applications under implementation.

2. Application editing facilities, the creation and manipulation of the user interface and application components probably through the use of 'views' (see below).

3. Printing of documentation of the application being edited.


Of these, the application editing e.g. shows the graphical interactive character of D2M2edit. As explained before, design specifications under D2M2environment consist of user interface components (user interface types, interaction sets, feedback sets, interaction tasks, feedback tasks and relations between sets) and application semantics components (application semantics data types, application code and semantic feedback functions), together with the parts required for communication (e.g. entries to group interaction tasks, data transfer code to return results of interactions and creation code for instantiation of an interaction or feedback set). All these can be edited, to which end sometimes views are available. A view displays a small portion of the specifications, of e.g. user interface data types, and allows easy and manageable creation and manipulation of the user interface and application semantics, giving all information required and retaining consistency without bothering the designer with irrelevant details or distracting relationships.
An example of such a view is the view on user interface data types (see submenu option "UI-Data Type" of figure 4.22). Through this view, the designer can specify the hierarchy of data types comprising the interface, and their behaviour. The basic user interface data types, offered by the relevant window manager and available through the built-in library, are always present and called native types. A special user interface data type 'native' is ever present in an application: every user interface data type directly related to the window manager is directly or indirectly derived from 'native' by specialization, inheriting its basic properties and operations (methods), but adding some of its own. User interface data types not related to the window manager are specializations of the ever present user interface data type 'own'.

Graphically, in this view two types of objects can be distinguished: user interface data types, represented by labelled rectangles, and generalization relationships, shown as labeled lines, connecting the user interface data types, according to the notation technique of Ter Bekke (1991). In figures 4.2 of section 4.3 the way user interface data types and their relationships are represented in D2M2edit is depicted.

Buttons in the view window are used to Create new types, Show types created earlier, but not visible in this view, and Set subtypes (i.e. create generalization relations between types). These are all functions that place graphical objects in the view. All other functions to be performed on these graphical objects are invoked through manipulation of the objects themselves, as dictated by the concept of direct manipulation interaction. They fall into two categories: functions which merely change the appearance of the view and functions changing the user interface data type specifications.

In the first category are e.g. actions to move objects around the view's graphics canvas by dragging (conform the OPEN LOOK standard), automatically redrawing all connected objects. Another feature of D2M2edit to keep the views as clear as possible is the 'expanding' of data types. When an existing user interface data type is added to a view, it probably has several relationships with other types. To show these as well would result in a waterfall of inclusions, cluttering up the display. Only showing the single type would misinform the designer. D2M2edit displays such a type as a double-edged rectangle. When the designer requires more information he expands the type, causing D2M2edit to arrange all directly related types in the view as well. In figure 4.23 two views of one user interface data model (fully depicted in figure 4.13) are depicted.

![Diagram of user interface data model](image)

*Figure 4.23: two views on the user interface data model of figure 4.13*

The second category of functions actually edits the specifications of the user interface data types. Examples are: deleting types (with their connected relationships) and editing the prop-
erties of data types and specializations. A special property window is placed on screen when
the designer clicks on the object, displaying all properties and allowing them to be edited. E.g.
in the case of user interface data types there are the name, description, attributes (each with a
name, data type and size) and methods (each with a return type, parameter list and body). In
figures 4.14 and 4.15 snapshots of D2M2edit of the UI-data property window are depicted.

The above represents a brief impression of part of the facilities offered by D2M2edit. Consis-
tent use of OPEN LOOK interaction mechanisms and graphical displays makes it likely that
D2M2edit is a suitable tool, in particular for user interface designers, not used to heavily text-
based programming environments.

4.7. PRESENT USER INTERFACE AND EXECUTE APPLICATION SEMANTICS

The "present user interface"-process takes care of the presentation of the user interface of a cer-
tain application. It does so by interpreting the user interface run-time specifications and com-
municating with the "execute application semantics"-process. Figure 4.24 depicts the commu-
nication between the two processes during run-time.

![Diagram](image)

**Figure 4.24: communication between D2M2run and application**

The "present user interface"-process and "execute application semantics"-process are two dif-
f erent processes, which communicate using a certain protocol. The "present user interface"-
process consists of two major parts: an "initialize"-process and a "present and communicate"-
process. Figure 4.25 depicts the Data Flow Diagram of D2M2run.

![Diagram](image)

**Figure 4.25: Data Flow Diagram of the processes in D2M2run**

The process "present and communicate" communicates with the "execute application process".

The "initialize"-process translates the raw user interface run-time specifications in specifi-
cations, which we call "fine user interface run-time specifications". During run-time the fine user
interface run-time specifications are interpreted by the "present and communicate"-process.
In order to show in what way the task flows of chapter 3 are implemented by the processes "present user interface" and "execute application semantics" (see figure 4.24) we discuss the functions of both mentioned processes referring to the task flows of chapter 3.

After performing the "initialize"-subprocess, D2M2run (supporting the "present and communicate"-process) is idle, waiting for the application (i.e. the "execute application semantics"-process) to seek contact. Referring to figure 3.12, the application starts with initializing its application semantics data (see task "initialize application semantics data"). Hereafter, the application triggers initial output (see task "trigger initial output task"); it does so by sending requests to D2M2run in order to trigger feedback tasks to be executed. These requests should also indicate whether instances of sets (interaction or feedback sets) must be created.

Now the D2M2run process has control to execute the initial feedback tasks (see task "execute initial output task"). As acknowledgement of successfully executing the feedback tasks (see task "acknowledge end of initial output task execution" of figure 3.12) the identifications of the created frames, instances of sets and instances of user interface data types are sent back.

On ending the acknowledgement control switches again to the application, which enables certain interaction tasks (see tasks "select possible computer tasks" and "enable, per computer task, possible user input tasks" of figure 3.15). Doing so the application sends identifications of every entry in which allowed interaction tasks are grouped. Control switches to D2M2run and D2M2run waits for user interaction (see task "wait for user interaction" of figure 3.15).

When the user starts interaction, user interface data is consulted and updated (see task "update/consult user interface data" of figure 3.16). If information from the applications semantics data is needed during execution of an interaction task, a request for semantic feedback is sent from D2M2run to the application semantics (see task "request for semantic feedback" of figure 3.16). In this request the identification of the entry and the instance of the set, in which semantic feedback may be visualized, are included. The application decides, consulting the identifications sent, which semantic feedback function to execute (see task "execute semantic feedback function" of figure 3.16). After execution the application sends a 'true' or 'false' to D2M2run, depending on whether a certain condition in the application semantics data holds (see task "send result of semantic feedback function" of figure 3.16). If the result of the semantic feedback function is 'true', D2M2run will perform semantic feedback through updating user interface data (see task "update user interface data for semantic feedback" of figure 3.16). If the result of the semantic feedback function is 'false', D2M2run will not perform semantic feedback. Whether 'true' or 'false', interaction task execution will proceed.

When the user finishes an interaction task in a certain set and the user input task is not completely finished yet, control is switched to a different set in which another interaction task must proceed.

When the user finishes user input task execution, by means of execution of the last interaction task of the user input task, the user information will be sent to the application (see task "send user information" of figure 3.14). Included in the user information is the identification of a unique entry. The application now deduces, from the identification of the entry, which computer task to execute (see task "execute computer task" of figure 3.14). During the actual execution of a computer task application semantics data is consulted and updated (see task "update/consult application semantics data" of figure 3.17).
If intermediate output must be generated, the application sends requests to D2M2run (see task "trigger intermediate output task" of figure 3.17) in order to allow execution of feedback tasks. The requests can contain identifications of instances of sets, to which feedback must be sent. All requests contain the identifications of the functions which perform the actual output on screen. After execution of the feedback tasks (see task "execute intermediate output task" of figure 3.17) acknowledgements are sent (see task "acknowledge end of intermediate output task execution" of figure 3.17). The acknowledgements contain identifications of possibly created frames, instances of sets and instances of user interface data types.

On completion of computer task execution probably output must be generated. For this, the application sends requests to D2M2run (see task "trigger output task" of figure 3.14). The requests can contain identifications of instances of sets, to which output must be sent. All requests contain the identifications of the functions which perform the actual output on screen. After execution of the feedback tasks (see task "execute output task" of figure 3.14) acknowledgements are sent (see task "acknowledge end of output task execution" of figure 3.14). The acknowledgements contain identifications of possibly created frames, instances of sets and instances of user interface data types. After receiving these acknowledgements, the application can again enable interaction tasks.

4.8. CONCLUSIONS

The D2M2environment consists of the processes D2M2design, D2M2edit and D2M2run. As for D2M2design, ICS design starts from specification of requirements. The requirements are translated into processes (and data) only concerning the user interface and processes (and data) only concerning the application semantics of the ICS. If the user interface processes (and data) are separated from the application semantics processes (and data), user interface specialists can concentrate on user interface design and system engineers can concentrate on application semantics design.

Next in ICS design is the translation of user interface processes and data into user interface data types, interaction sets, feedback sets, interaction tasks, feedback tasks and relations between sets; and the translation of application semantics processes and data in application semantics data types, computer tasks, interaction task enablings and semantic feedback functions.

D2M2edit allows the ICS-developer to input both user interface design specifications and application semantics design specifications. Both types of specifications will be converted; they serve as input to D2M2run and the "execute application semantics"-process (the application, for short). During run-time the application semantics data of the ICS is monitored by the application; the user interface data is monitored by D2M2run.

In the following chapters two HICSs will be designed and implemented; further, changes will be made to the ICSs in order to verify the degree of separation of user interface and application semantics.
5. CASE 1: GRAPHICS MANAGER

5.1. INTRODUCTION
The reason for presenting cases in our research is twofold.

1. It can be empirically made plausible that a user interface of a HICS can be separated from the application semantics.
2. The cases can show that D2M2environment is a feasible environment.

Obviously, these reasons are related to the hypotheses and statements stated in chapter 2.

It is necessary that the HICSs to be developed in the cases show enough complexity, are representative and are already being described by a functional description in the prescriptive domain. The Graphics Manager, the first HICS to be developed, satisfies these needs. The Graphics Manager is a HICS with which data can be visualized by graphs; the graphs can be manipulated by the user.

A functional description of the Graphics Manager was made in cooperation with Baan Info Systems. The functional description is extensively specified by Versendaal (1989). Baan Info Systems has the intention to distribute the Graphics Manager to their clients as soon as it is fully implemented.

We decided to transform a subset of the functional description from Versendaal (1989) into SECTIONS and OBJECTS, so it can be treated by D2M2design; section 5.2 describes the functional description in terms of SECTIONS and OBJECTS. In section 5.3 the functional description is analysed; further the design specifications of the Graphics Manager are determined and described as they can be entered in D2M2edit. In section 5.4 modifications to the Graphics Manager are suggested. Finally, in section 5.5, conclusions are drawn with relation to the development and modification of the Graphics Manager in D2M2environment.

5.2. DESCRIPTION
The description of the Graphics Manager is deduced from Versendaal (1989). Before presenting it in terms of SECTIONS and OBJECTS we define some concepts, specific to the Graphics Manager.

<p>| Screen: | part of a workstation on which the Graphics Manager is visualised. |
| Coordinate System: | system with X-axis and Y-axis in which one or more Graphs can be visualised. |
| Graph: | bijective projection with an arbitrary domain and co-domain. |
| Table: | representation of Table Values in a survey (for an example, see figure 5.1). |</p>
<table>
<thead>
<tr>
<th>x</th>
<th>graph1</th>
<th>graph2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8.3</td>
<td>5.4</td>
</tr>
<tr>
<td>1</td>
<td>9.0</td>
<td>5.5</td>
</tr>
<tr>
<td>2</td>
<td>9.3</td>
<td>5.9</td>
</tr>
<tr>
<td>3</td>
<td>9.9</td>
<td>7.0</td>
</tr>
<tr>
<td>4</td>
<td>9.7</td>
<td>6.9</td>
</tr>
<tr>
<td>5</td>
<td>8.9</td>
<td>5.9</td>
</tr>
</tbody>
</table>

*Figure 5.1: example of a Table*

Scrolling of a Coordinate System: horizontal or vertical movement, so that other Domain Values or Co-domain Values are visualised.

Scrolling of a Table: vertical movement, so that other Table Values are visualised.

Mean Survey: representation of mean values of all Data Values of the Graphs in a certain Coordinate System (for an example, see figure 5.2).

```
mean graph1: 9.2
mean graph2: 6.0
```

*Figure 5.2: example of a Mean Survey*

Domain Values: x-values of data in a Graph.

Co-domain Values: y-values of data in a Graph.

Bar Diagram: type of Graph in a Coordinate System; Co-domain Values are represented by bars (for an example, see figure 5.3).

```
10
9
8
7
6
5
4
3
2
1
0

1 2 3 4 5
```

*Figure 5.3: example of a Bar Diagram*

Dot Diagram: type of Graph in a Coordinate System; Co-domain Values are represented by dots (for an example, see figure 5.4).
Figure 5.4: example of a Dot Diagram

Line Diagram: type of Graph in a Coordinate System; Co-domain Values are represented by dots; adjacent dots are connected to each other by lines (for an example, see figure 5.5).

Figure 5.5: example of a Line Diagram

Area Diagram: type of Graph in a Coordinate System; Co-domain Values are represented by dots; adjacent dots are connected to each other by lines; the areas between these lines and the X-axis are shadowed (for an example, see figure 5.6).

Figure 5.6: example of an Area Diagram
Data Values: input data for the Graphics Manager, consisting of pairs of *Domain Values* and *Co-domain Values*.

Table Values: alpha-numerical representation of *Domain Values* and *Co-domain Values* in a *Table*.

Domain Interval: united part of *Domain Values* in a *Coordinate System*.

Input File: file containing input for the Graphics Manager; in the Input File the Coordinate Systems to be presented are recorded; for each Coordinate System the name of the Coordinate System, the descriptions of the X-axis and Y-axis and the Graphs and their types are recorded; for each Graph the name of the Graph and the Domain and Co-domain Values are recorded.

Having defined the concepts, we present the list of functional requirements of the Graphics Manager including information about identified objects.

**SECTION1 start/stop Graphics Manager**

**UNIT1**
The Graphics Manager can be started. The Graphics Manager should initially visualise those Coordinate Systems on Screen that are specified in the Input File of the Graphics Manager. The Graphics Manager should initially visualise those Graphs in a Coordinate System that are specified in the Input File of the Graphics Manager.

**UNIT2**
The Graphics Manager can be stopped implying that control returns to the operating system.

**END SECTION1 start/stop Graphics Manager**

**SECTION2 Coordinate System**

**UNIT1**
A Coordinate System can be printed.

**UNIT2**
Out of an existing Coordinate System a Domain Interval can be selected and copied to a new Coordinate System.

**UNIT3**
The density of Domain Values along the X-axis can be changed.

**UNIT4**
The density of Co-domain Values along the Y-axis can be changed.

**UNIT5**
Domain Values can be visualised/hidden along the X-axis.

**UNIT6**
Co-domain Values can be visualised/hidden along the Y-axis.

**UNIT7**
The description along the X-axis can be visualised/hidden.

**UNIT8**
The description along the Y-axis can be visualised/hidden.

**UNIT9**
A Coordinate System can be rotated +90°.
UNIT10
   A Coordinate System can be scrolled horizontally.
UNIT11
   A Coordinate System can be scrolled vertically.
UNIT12
   A Coordinate System can be (un-)scaled logarithmically horizontally.
UNIT13
   A Coordinate System can be (un-)scaled logarithmically vertically.
UNIT14
   Horizontal help lines can be visualised/hidden in a Coordinate System, so that Co-
   domain Values can be read more easily.
UNIT15
   Vertical help lines can be visualised/hidden in a Coordinate System, so that Domain
   Values can be read more easily.
UNIT16
   The type of a Graph (not an Area Diagram) can be changed into an Area Diagram.
UNIT17
   The type of a Graph (not a Line Diagram) can be changed into a Line Diagram.
UNIT18
   The type of a Graph (not a Dot Diagram) can be changed into a Dot Diagram.
UNIT19
   The type of a Graph (not a Bar Diagram) can be changed into a Bar Diagram.
UNIT20
   A Line Diagram can be (un-)smoothed.
UNIT21
   The bar width of a Bar Diagram can be changed.
UNIT22
   The line width of a Line Diagram can be changed.
UNIT23
   The dot size of a Dot Diagram can be changed.
UNIT24
   The bar shadow of a Bar Diagram can be changed.
UNIT25
   The line type of a Line Diagram can be changed.
UNIT26
   The dot type of a Dot Diagram can be changed.
UNIT27
   The type of shadowing of areas in an Area Diagram can be changed.
UNIT28
   The Domain and Co-domain Values can be visualised/hidden in a Coordinate Sys-
   tem.
UNIT29
   A Coordinate System can be deleted from Screen.
END SECTION2 Coordinate System

SECTION3 Table
UNIT1
   A Table, associated with a Coordinate System, can be visualised/hidden on Screen.
UNIT2
    A Table can be scrolled vertically.
UNIT3
    A Table can be printed.
END SECTION3 Table

SECTION4 Mean Survey
UNIT1
    A Mean Survey, associated with a Coordinate System, can be visualised/hidden on
    Screen.
UNIT2
    A Mean Survey can be printed.
END SECTION4 Mean Survey

OBJECTS
RELATION1
    The Screen contains one to six Coordinate Systems.
RELATION2
    A Coordinate System contains one to six Graphs.
RELATION3
    A Graph is either a Bar Diagram, Line Diagram, Area Diagram or Dot Diagram.
RELATION4
    A Coordinate System contains a Legend.
RELATION5
    A Table is associated with a Coordinate System.
RELATION6
    A Mean Survey is associated with a Coordinate System.
END OBJECTS

So far the functional description for the development of the Graphics Manager has been given.

5.3. CONSTRUCTION OF DESIGN SPECIFICATIONS

In this section the functional description in terms of SECTIONS and OBJECTS is analysed and
design specifications are constructed; the steps of D2M2design are applied. A detailed descrip-
tion of the design specifications is presented by Dinger (1991).

On the next pages the results of the first step of D2M2design are presented. The first column
contains the SECTION and UNIT identifications to which the following three columns refer.
The second column contains the identifications of the user input tasks; the third column con-
tains the identifications of the computer tasks; the fourth column contains the identifications of
the output tasks.
<table>
<thead>
<tr>
<th>SECTION/UNIT</th>
<th>user input task</th>
<th>computer task</th>
<th>output tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>initiate initialise</td>
<td>initialise</td>
<td>display initial Screen with Coordinate Systems</td>
</tr>
<tr>
<td>1.2</td>
<td>initiate stop</td>
<td>stop</td>
<td>clear Screen</td>
</tr>
<tr>
<td>2.1</td>
<td>initiate print_COORDINATE_SYSTEM</td>
<td>print_COORDINATE_System</td>
<td>display intermediate message &quot;printing&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>display message &quot;System printed&quot;</td>
</tr>
<tr>
<td>2.2</td>
<td>initiate create_new_SYS</td>
<td>create_new_SYS</td>
<td>display new Coordinate System</td>
</tr>
<tr>
<td>2.3</td>
<td>initiate change_X_density</td>
<td>change_X_density</td>
<td>change X-density</td>
</tr>
<tr>
<td>2.4</td>
<td>initiate change_Y_density</td>
<td>change_Y_density</td>
<td>change Y-density</td>
</tr>
<tr>
<td>2.5</td>
<td>initiate display_or_hide_X_axis_values</td>
<td>display_or_hide_X_axis_values</td>
<td>display/hide values along the X-axis</td>
</tr>
<tr>
<td>2.6</td>
<td>initiate display_or_hide_Y_axis_values</td>
<td>display_or_hide_Y_axis_values</td>
<td>display/hide values along the Y-axis</td>
</tr>
<tr>
<td>2.7</td>
<td>initiate display_or_hide_X_axis_description</td>
<td>display_or_hide_X_axis_description</td>
<td>display/hide description of the X-axis</td>
</tr>
<tr>
<td>2.8</td>
<td>initiate display_or_hide_Y_axis_description</td>
<td>display_or_hide_Y_axis_description</td>
<td>display/hide description of the Y-axis</td>
</tr>
<tr>
<td>2.9</td>
<td>initiate rotate</td>
<td>rotate</td>
<td>rotate Coordinate System</td>
</tr>
<tr>
<td>2.10</td>
<td>initiate X_COORDINATE_SYSTEM_scrolling</td>
<td>X_COORDINATE_SYSTEM_scrolling</td>
<td>scroll Coordinate System horizontally</td>
</tr>
<tr>
<td>2.11</td>
<td>initiate Y_COORDINATE_SYSTEM_scrolling</td>
<td>Y_COORDINATE_SYSTEM_scrolling</td>
<td>scroll Coordinate System vertically</td>
</tr>
<tr>
<td>2.12</td>
<td>initiate X_COORDINATE_SYSTEM_log_(un)scaling</td>
<td>X_COORDINATE_SYSTEM_log_(un)scaling</td>
<td>log (un)scale Coordinate System horizontally</td>
</tr>
<tr>
<td>2.13</td>
<td>initiate Y_COORDINATE_SYSTEM_log_(un)scaling</td>
<td>Y_COORDINATE_SYSTEM_log_(un)scaling</td>
<td>log (un)scale Coordinate System vertically</td>
</tr>
<tr>
<td>2.14</td>
<td>initiate display_or_hide_X_grid</td>
<td>display_or_hide_X_grid</td>
<td>display/hide a horizontal grid</td>
</tr>
<tr>
<td>SECTION/UNIT</td>
<td>user input task</td>
<td>computer task</td>
<td>output tasks</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>2.15</td>
<td>initiate display_or_hide_Y_grid</td>
<td>display_or_hide_Y_grid</td>
<td>display/hide a vertical grid</td>
</tr>
<tr>
<td>2.16</td>
<td>initiate change_to_Area_Diagram</td>
<td>change_to_Area_Diagram</td>
<td>change Graph into an Area Diagram</td>
</tr>
<tr>
<td>2.17</td>
<td>initiate change_to_Line_Diagram</td>
<td>change_to_Line_Diagram</td>
<td>change Graph into a Line Diagram</td>
</tr>
<tr>
<td>2.18</td>
<td>initiate change_to_Dot_Diagram</td>
<td>change_to_Dot_Diagram</td>
<td>change Graph into a Dot Diagram</td>
</tr>
<tr>
<td>2.19</td>
<td>initiate change_to_Bar_Diagram</td>
<td>change_to_Bar_Diagram</td>
<td>change Graph into a Bar Diagram</td>
</tr>
<tr>
<td>2.20</td>
<td>initiate smooth_Line_Diagram</td>
<td>smooth_Line_Diagram</td>
<td>smooth Line Diagram</td>
</tr>
<tr>
<td>2.20</td>
<td>initiate un-smooth_Line_Diagram</td>
<td>un-smooth_Line_Diagram</td>
<td>un-smooth Line Diagram</td>
</tr>
<tr>
<td>2.21</td>
<td>initiate change_bar_width</td>
<td>change_bar_width</td>
<td>change the width of bars of a Bar Diagram</td>
</tr>
<tr>
<td>2.22</td>
<td>initiate change_line_width</td>
<td>change_line_width</td>
<td>change the width of lines of a Line Diagram</td>
</tr>
<tr>
<td>2.23</td>
<td>initiate change_dot_size</td>
<td>change_dot_size</td>
<td>change the size of dots of a Dot Diagram</td>
</tr>
<tr>
<td>2.24</td>
<td>initiate change_bar_shadow_1</td>
<td>change_bar_shadow_1</td>
<td>change the shadows of bars of a Bar Diagram into type 1</td>
</tr>
<tr>
<td>2.24</td>
<td>initiate change_bar_shadow_2</td>
<td>change_bar_shadow_2</td>
<td>change the shadows of bars of a Bar Diagram into type 2</td>
</tr>
<tr>
<td>2.25</td>
<td>initiate change_line_type_1</td>
<td>change_line_type_1</td>
<td>change the lines of a Line Diagram into type 1</td>
</tr>
<tr>
<td>2.25</td>
<td>initiate change_line_type_2</td>
<td>change_line_type_2</td>
<td>change the lines of a Line Diagram into type 2</td>
</tr>
<tr>
<td>2.25</td>
<td>initiate change_line_type_3</td>
<td>change_line_type_3</td>
<td>change the lines of a Line Diagram into type 3</td>
</tr>
<tr>
<td>SECTION/UNIT</td>
<td>user input task</td>
<td>computer task</td>
<td>output tasks</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------</td>
<td>-----------------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>2.26</td>
<td>initiate change_dot_type_1</td>
<td>change_dot_type_1</td>
<td>change the dots of a Dot Diagram into type 1</td>
</tr>
<tr>
<td>2.26</td>
<td>initiate change_dot_type_2</td>
<td>change_dot_type_2</td>
<td>change the dots of a Dot Diagram into type 2</td>
</tr>
<tr>
<td>2.26</td>
<td>initiate change_dot_type_3</td>
<td>change_dot_type_3</td>
<td>change the dots of a Dot Diagram into type 3</td>
</tr>
<tr>
<td>2.27</td>
<td>initiate change_area_shadow_1</td>
<td>change_area_shadow_1</td>
<td>change the shadows of an Area Diagram to type 1</td>
</tr>
<tr>
<td>2.27</td>
<td>initiate change_area_shadow_2</td>
<td>change_area_shadow_2</td>
<td>change the shadows of an Area Diagram to type 2</td>
</tr>
<tr>
<td>2.28</td>
<td>initiate display (Co-)Domain.Values</td>
<td>display_ (Co-)Domain_ Values</td>
<td>display the Domain and Co-domain Values</td>
</tr>
<tr>
<td>2.28</td>
<td>initiate hide (Co-)Domain.Values</td>
<td>hide_ (Co-)Domain_ Values</td>
<td>hide the Domain and Co-domain Values</td>
</tr>
<tr>
<td>2.29</td>
<td>initiate delete Coordinate_System</td>
<td>delete Coordinate_System</td>
<td>remove a Coordinate System from Screen</td>
</tr>
<tr>
<td>3.1</td>
<td>initiate display_or_hide_Table</td>
<td>display_or_hide_Table</td>
<td>display/hide a Table</td>
</tr>
<tr>
<td>3.2</td>
<td>initiate Y_Table.Scrolling</td>
<td>Y_Table.Scrolling</td>
<td>scroll Table vertically</td>
</tr>
<tr>
<td>3.3</td>
<td>initiate print_Table</td>
<td>print_Table</td>
<td>display intermediate message &quot;printing&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>display message &quot;Table printed&quot;</td>
</tr>
<tr>
<td>4.1</td>
<td>initiate display_or_hide_Mean_Survey</td>
<td>display_or_hide_Mean_Survey</td>
<td>display/hide a Mean Survey</td>
</tr>
<tr>
<td>4.2</td>
<td>initiate print_Mean_Survey</td>
<td>print_Mean_Survey</td>
<td>display intermediate message &quot;printing&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>display message &quot;Mean Survey printed&quot;</td>
</tr>
</tbody>
</table>
Note that we split SECTION/UNIT 2.20 in a 'smooth' and 'non-smooth' part. Note further that we limit the shadowing of bars in a Bar Diagram to two types of shadowing (see SECTION/UNIT 2.24). We limit the line style types of Line Diagrams to three types (see SECTION/UNIT 2.25). We limit the dot style types of Dot Diagrams to three types (see SECTION/UNIT 2.26). We limit the shadowing of Area Diagrams to two types of shadowing (see SECTION/UNIT 2.27). We further split section 2.28 in a 'display' and 'hide' part.

Having analysed the UNITS we are now able to construct the design specifications. We first present the application semantics data types and the computer tasks (see section 4.4 for the D2M2design process). We present the names, generalization relations, aggregation relations and attributes of the application semantics data types; we present the names and a description of the computer tasks.

APPLICATION SEMANTICS DATA TYPES AND COMPUTER TASKS
The Input File of the Graphics Manager contains Coordinate Systems, and associated Graphs; each Graph is in the Input File specified by Domain and Co-domain Values. Therefore we can identify the application semantics data types (in Dutch): 'datawaarde' (Domain and Co-domain Value), 'grafiek' (Graph) and 'stelsel' (Coordinate System). Figure 5.7 depicts the names and relations of the application semantics data types of the Graphics Manager.

![Diagram of application semantics data types and relations]

Figure 5.7: application semantics data types and relations

The application semantics data type 'datawaarde' contains the attributes Domain Value and Co-domain Value. Via the user interface data types 'staaf', 'lijnpunt', 'gebiedpunt' and 'punt' (which we define later), instances of the application semantics data type 'datawaarde' are visualised on Screen.

The application semantics data type 'grafiek' contains the attribute: name of the Graph. Via the user interface data types 'staafdiagram', 'puntdiagram', 'gebieddiagram' and 'lijndiagram' (which we define later), instances of the application semantics data type 'grafiek' are visualised on Screen. A number of instances of the application semantics data type 'datawaarde' belongs to one instance of 'grafiek'.

The data type 'stelsel' contains the attribute: name of the Coordinate System. Via the user interface data type 'grafiekendiagram' (which we define later), instances of the application semantics data type 'stelsel' are visualised on Screen. A number of instances of the application semantics data type 'grafiek' belongs to one instance of 'stelsel'.

Related to the application semantics data types we define the following computer tasks (in Dutch), which consult and/or update the application semantics data.
All other mentioned computer tasks of the result of the analysis step are not to be considered real computer tasks because they do not consult or update application semantics data: complication III (computer task execution only dealing with the user interface and not with the application semantics) manifests itself.

<table>
<thead>
<tr>
<th>computer task</th>
<th>translation</th>
<th>related UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialisatie</td>
<td>initialise</td>
<td>1.1</td>
</tr>
<tr>
<td>nieuwgrafstelsel</td>
<td>create_new_System</td>
<td>2.2</td>
</tr>
<tr>
<td>pl_gemoverzicht</td>
<td>display_or_hide_Mean_Survey</td>
<td>4.1</td>
</tr>
<tr>
<td>pl_tabel</td>
<td>display_or_hide_Table</td>
<td>3.1</td>
</tr>
<tr>
<td>printgrafstelsel</td>
<td>print_Coordinate_System</td>
<td>2.1</td>
</tr>
<tr>
<td>verwgrafstelsel</td>
<td>delete_Coordinate_System</td>
<td>2.29</td>
</tr>
<tr>
<td>printtabel</td>
<td>print_Table</td>
<td>3.3</td>
</tr>
<tr>
<td>printgemstelsel</td>
<td>print_Mean_Survey</td>
<td>4.2</td>
</tr>
<tr>
<td>stoppen</td>
<td>stop</td>
<td>1.2</td>
</tr>
</tbody>
</table>

We now present the interaction tasks, feedback tasks, interaction sets, feedback sets, the user interface data types and set relations. We discuss subsequently: user interface data types, interaction sets, feedback sets, interaction tasks, feedback tasks and set relations.

USER INTERFACE DATA TYPES

We present the names (in Dutch, between brackets and quotes), descriptions and generalization relations of the user interface data types. We start with the specialized user interface data types of 'own'.

We consider a Coordinate System to consist of an X-axis ('xas'), Y-axis ('yas'), Domain Values along the X-axis ('asdoeimenwaarde'), Co-domain Values along the Y-axis ('asbereikwaarde') and a number of Graphs ('diagram'). We consider a Graph to consist of Data Values ('datawaarde'), a Graph definition ('grafieke definitie'), areas ('gebied') in case of an Area Diagram and lines ('lijn') in case of a Line Diagram. We consider a Mean Survey to consist of rows ('gemoverzichtrij') containing Graph names, as headers for the mean values, and mean values. So, we consider rows in a Mean Survey as consisting of Graph names as headers for the mean values ('gemgrafiekwaa') and mean values ('gemgrafiekwaa'). We consider a Table as consisting of Graph names as headers for the Co-domain Values of a Graph ('tablgrafiekwaa'), a header for the Domain Values of the Graphs in a Table ('tablwaardenaa') and rows ('tablrij') containing Domain and Co-domain Values in a Table. So, we consider rows in a Table as consisting of a Domain Value ('tablitemx') and Co-domain Values ('tablitemy').

Figure 5.8 depicts the names, and relations of the user interface data types 'own' as they are entered in D2M2edit. Beyond the figure descriptions of the user interface data types are presented.
The generalized user interface data type 'own' is a data type of which specialized data types are not XView data types. Data type 'own' contains attributes and methods common to all its specialized data types. The user interface data types 'diagram' and 'datawaarde' are themselves generalized data types. Their hierarchy will be visualized later in this section.

![Diagram of data types]

**Figure 5.8: specialized data types of 'own'**

The descriptions of the specialized user interface data types of figure 5.8 are as follows.

- **asbereikwaarde:** Co-domain Value along the Y-axis;
- **asdomeinwaarde:** Domain Value along the X-axis;
- **datawaarde:** Data Value;
- **diagram:** Graph;
- **gebied:** area in an Area Diagram;
- **gemgrafieknaam:** Graph name in a Mean Survey;
- **gemgrafiekwaarde:** mean value of a Graph in a Mean Survey;
- **gemoverzichtrij:** row containing 'gemgrafieknaam' and 'gemgrafiekwaarde' in a Mean Survey;
- **grafiekdefinitie:** description of a Graph in a legend;
- **lijn:** line connecting adjacent dots in a Line Diagram;
- **tabelgrafieknaam:** Graph name as header for Co-domain Values in a Table;
- **tabelitemx:** Domain Value in a Table;
- **tabelitemy:** Co-domain Value in a Table;
- **tabelrij:** row containing 'tabelitemx' and 'tabelitemy's' in Table;
- **tabelxwaardenaam:** header for Domain Values in a Table;
- **xas:** X-axis in a Coordinate System;
- **yas:** Y-axis in a Coordinate System.

86
We elect to use the following XView user interface data types in order to allow for the user input tasks and output tasks and semantic feedback in the Graphics Manager (see figure 5.9).

![Diagram showing the hierarchy of XView data types]

**Figure 5.9: specialized data types of 'native'**

The generalized user interface data type 'native' is a data type of which specialized data types are XView data types. Data type 'native' contains attributes and methods common to all its specialized data types. The user interface data type 'canvas' is itself a generalized data type. Its hierarchy will be visualised later in this section.

The specialized user interface data types of figure 5.9 are XView data types.

- **button:** Panel Button Item;
- **canvas:** Canvas;
- **melding:** Panel Message Item;
- **menu:** Menu;
- **menu_item:** Menu Item;
- **panel:** Panel;
- **scrollbar:** Scrollbar;
- **slider:** Panel Slider Item;
- **toggle:** Panel Toggle Item;
- **window:** Frame.

Because we are dealing with Bar Diagrams, Dot Diagrams, Line Diagrams and Area Diagrams in a Coordinate System, we specialize 'datawaarde' (from figure 5.8) into bar in a Bar Diagram ('staaf'), (line) dot in a Line Diagram ('lijnpunt'), (area) dot in an Area Diagram ('gebiedspunt') and dot in a Dot Diagram ('punt'). In figure 5.10 the specialized data types are depicted as they are entered in D2M2edit.

87
Figure 5.10: specialized data types of 'datawaarde'

The user interface data types 'staaf', 'lijnpunt', 'gebiedpunt' and 'punt' have e.g. their own operations to depict instances on Screen.

The descriptions of the specialized user interface data types are as follows.

- **gebiedpunt**: dot in an Area Diagram representing a Co-domain Value;
- **lijnpunt**: dot in a Line Diagram representing a Co-domain Value;
- **punt**: dot in a Dot Diagram representing a Co-domain Value;
- **staaf**: bar in a Bar Diagram representing a Co-domain Value.

Because we are dealing with Bar Diagrams, Dot Diagrams, Line Diagrams and Area Diagrams in a Coordinate System we specialize 'diagram' (from figure 5.8) into Bar Diagram ('staafdiagram'), Dot Diagram ('puntdiagram'), Area Diagram ('gebieddiagram') and Line Diagram ('lijndiagram'). In figure 5.11 the specialized data types are depicted.

Figure 5.11: specialized data types of 'diagram'

The user interface data types 'lijndiagram', 'gebieddiagram', 'puntdiagram' and 'staafdiagram' have their own unique attributes; 'staafdiagram' for instance has an attribute shadow type, which indicates the type of shadowing of bars. These four data types also contain a common attribute: a boolean attribute indicating whether or not to show the Data Values along with the Graph.
The descriptions of the specialized user interface data types are as follows.

- **gebieddiagram**: Area Diagram;
- **lijndiagram**: Line Diagram;
- **puntdiagram**: Dot Diagram;
- **staafdiagram**: Bar Diagram.

We identify five types of canvases in the Graphics Manager: one for depicting Graphs ('grafiekendiagram'), one for depicting the legend of a Coordinate System ('legenda'), one for depicting the header of a Table ('tabelkopregel'), one for depicting Table Values ('tabel') and one for depicting the mean values of a Mean Survey ('gemiddeldeoverzicht'). In figure 5.12 the specialized data types of 'canvas' are depicted as they are entered in D2M2edit.

![Diagram showing the relationship between data types]

*Figure 5.12: specialized data types of 'canvas'*

The descriptions of the specialized user interface data types are as follows.

- **gemiddeldeoverzicht**: area (Canvas) in which a Mean Survey is presented;
- **grafiekendiagram**: area (Canvas) in which a Coordinate System is presented;
- **legenda**: area (Canvas) in which a legend of a Coordinate System is presented;
- **tabel**: area (Canvas) in which a Table is presented;
- **tabelkopregel**: area (Canvas) in which the header of a Table is presented.

We now construct the interaction sets and feedback sets.

**INTERACTION SETS**

Interaction and feedback sets are visualised inside windows (XView-frames). In the case of the Graphics Manager we propose six types of windows. We identify a separate window to allow for stopping the Graphics Manager HICS (FRAME 1); we identify a window to allow for manipulating and visualizing the Graphs in a Coordinate System (FRAME 2); we identify a (popup) window to allow for menu item selection (for e.g. changing a Graph into a Bar Diagram) (FRAME 3); we identify a (popup) window to allow for e.g. setting up the width of bars of a...
Bar Diagram (FRAME 4); we identify a (pop-up) window to depict Table Values (FRAME 5); we identify a (pop-up) window to depict mean values of a Mean Survey (FRAME 6).

We choose to associate every 'panel' and every 'canvas' user interface data type of XView with one set, either interaction set or feedback set. In FRAME 1 we identify the interaction set 'stoppencontroloset', which allows us to stop the Graphics Manager. In FRAME 2 we identify the interaction set 'grafiekendiaagramset', in which the Coordinate System with Graphs is depicted; in FRAME 2 we further identify the interaction set 'grafiekcontroloset', which allows us most of the interaction with the Coordinate System; in FRAME 2 we finally identify the interaction set 'legendas', in which the legend of the Graphs of the Coordinate System is depicted. In FRAME 3 we identify the interaction set 'menuset', from which a menu item can be selected. In FRAME 4 we identify the interaction set 'sliderpopupset', which allows slider interaction to e.g. set up the width of bars in a Bar Diagram. In FRAME 5 we identify the interaction set 'tabelcontroloset', which allows interaction on the associated Table. In FRAME 6 we identify the interaction set 'gemcontroloset', which allows interaction on the associated Mean Survey. Below a summary of interaction sets and FRAMEs is presented.

| FRAME 1:   | stoppencontroloset       |
| FRAME 2:   | grafiekendiaagramset     |
|           | grafiekcontroloset       |
|           | legendas                |
| FRAME 3:   | menuset                 |
| FRAME 4:   | sliderpopupset          |
| FRAME 5:   | tabelcontroloset         |
| FRAME 6:   | gemcontroloset           |

Having identified the interaction sets, we are now able to identify the user interface data types that are used within the interaction sets and we are able to construct the parent-child relations of the interaction sets. We present each interaction set by its parent-child relations. The layout of all sets (both interaction sets and feedback sets) is presented at the end of the discussion of feedback sets.

Figure 5.13 depicts the parent-child relations of the interaction set 'stoppencontroloset'.

```
window

panel

button
```

*Figure 5.13: parent-child relations of 'stoppencontroloset'*

Note that the user interface data type window reflects FRAME 1 in the case of the 'stoppencontroloset'.
Figure 5.14 depicts the parent-child relations of the 'grafiekendiagramset'.

![Diagram of parent-child relations of grafiekendiagramset]

*Figure 5.14: parent-child relations of 'grafiekendiagramset'*

Note that the user interface data type window reflects FRAME 2 in the case of the 'grafiekendiagramset'.

Figure 5.15 depicts the parent-child relations of the 'grafiekcontroloset'.

![Diagram of parent-child relations of grafiekcontroloset]

*Figure 5.15: parent-child relations of 'grafiekcontroloset'*

Note that the user interface data type window reflects FRAME 2 in the case of the 'grafiekcontroloset'.

Figure 5.16 depicts the parent-child relations of the 'legendaset'.

![Diagram of parent-child relations of legendaset]

*Figure 5.16: parent-child relations of 'legendaset'*
Note that the user interface data type window reflects FRAME 2 in the case of the 'legendaset'.

Figure 5.17 depicts the parent-child relations of the 'menuset'.

![Diagram](image)

*Figure 5.17: parent-child relations of 'menuset'*

Note that the user interface data type window reflects FRAME 3 in the case of the 'menuset'.

Figure 5.18 depicts the parent-child relations of the 'sliderpopupset'.

![Diagram](image)

*Figure 5.18: parent-child relations of 'sliderpopupset'*

Note that the user interface data type window reflects FRAME 4 in the case of the 'sliderpopupset'.

Figure 5.19 depicts the parent-child relations of the 'tabelcontroleset'.

![Diagram](image)

*Figure 5.19: parent-child relations of 'tabelcontroleset'*

Note that the user interface data type window reflects FRAME 5 in the case of the 'tabelcontroleset'.

Figure 5.20 depicts the parent-child relations of the 'gemcontroleset'.

Note that the user interface data type window reflects FRAME 6 in the case of the 'gemcontroleset'.
So far we have covered the parent-child relations of the interaction sets. We now discuss the feedback sets.

**FEEDBACK SETS**

In FRAME 2 we identify the feedback set 'grfstelselmeldingset', in which messages concerning the Coordinate System can be displayed. In FRAME 5 we identify the feedback set 'tabelkopregelset', in which the header of a Table can be displayed; in FRAME 5 we further identify the feedback set 'tabelset', in which the Table Values are visualized; in FRAME 5 we finally identify the feedback set 'tabelmeldingset', in which messages concerning a Table can be visualized. In FRAME 6 we identify the feedback set 'gemoverzichtset', which visualizes the mean values of a Mean Survey; in FRAME 6 we further identify the feedback set 'gemmeldingset', in which messages concerning a Mean Survey can be visualized. Below, the relations between the feedback sets and FRAMES are summarized.

- FRAME 2: grfstelselmeldingset
- FRAME 5: tabelkopregelset
  - tabelset
  - tabelmeldingset
- FRAME 6: gemoverzichtset
  - gemmeldingset

Having identified the feedback sets, we are now able to identify the user interface data types that are used within the feedback sets and we are able to construct the parent-child relations of the feedback sets. We present each feedback set by its parent-child relations.

Figure 5.21 depicts the parent-child relations of the feedback set 'grfstelselmeldingset'.

Note that the user interface data type window reflects FRAME 4 in the case of the
'grfstelselmeldingset'.

Figure 5.22 depicts the parent-child relations of the 'tabelkopregelset'.

```
window
  └── tabelkopregelset
      ├── tabelkwaardenaam
      └── tabelgrafienaam
```

*Figure 5.22: parent-child relations of 'tabelkopregelset'*

Note that the user interface data type window reflects FRAME 5 in the case of the 'tabelkopregelset'.

Figure 5.23 depicts the parent-child relations of the 'tabelset'.

```
window
  └── tabel
      ├── tabelrij
      └── scrollbar
          └── tabelitemx
          └── tabelitemy
```

*Figure 5.23: parent-child relations of 'tabelset'*

Note that the user interface data type window reflects FRAME 5 in the case of the 'tabelset'.

Figure 5.24 depicts the parent-child relations of the 'tabelmeldingset'.

```
window
  └── panel
      └── melding
```

*Figure 5.24: parent-child relations of 'tabelmeldingset'*

Note that the user interface data type window reflects FRAME 5 in the case of the 'tabelmeldingset'.

Figure 5.25 depicts the parent-child relations of the 'gemoverzichtset'.

94
Figure 5.25: parent-child relations of 'gemoverzichtset'

Note that the user interface data type window reflects FRAME 6 in the case of the 'gemoverzichtset'.

Figure 5.26 depicts the parent-child relations of the 'gemmeldingset'.

Figure 5.26: parent-child relations of 'gemmeldingset'

Note that the user interface data type window reflects FRAME 6 in the case of the 'gemmeldingset'.

In figure 5.27 the layout of all sets, as they appear on screen, is depicted.
INTERACTION TASKS

A user input task consists of one or more interaction tasks. With the identification of interaction sets, we are able to construct interaction tasks. Below, our construction of interaction tasks is presented; we describe the interaction tasks and show to which user input task they belong; user input tasks are identified by their description and UNIT number. We note that, of course, design decisions have been made in order to determine the interaction tasks being described.
User input task: initiate stop (1.2).
Interaction task(s): click the 'stop'-button in the instance of 'stoppencontroleset'.

User input task: initiate print_Coordinate_System (2.1).
Interaction task(s): click the 'print'-button in an instance of 'grafiekcontroleset'.

User input task: initiate create_new_System (2.2).
Interaction task(s): click the 'nieuw'-button in an instance of 'grafiekcontroleset';
click on two Domain Values on the X-axis in an instance of 'grafiekendiagramset'. These Domain Values constitute the borders of the new FRAME 2.

The second of the two interaction tasks associated with the user input task "initiate create_new_System" contains semantic feedback. While selecting the second Domain Value along the X-axis, application semantics data is consulted (through a semantic feedback function) in order to determine whether some Domain Values exist between the two selected Domain Values. If not, creating a new Coordinating System does not make sense (for it does not hold any data), and the semantic feedback message "Second Domain Value not correct" is presented in the involved instance of 'grfdstelselmeldingset'.

User input task: initiate change_X_density (2.3).
Interaction task(s): set the 'dichtheid xas'-slider in an instance of 'grafiekcontroleset'.

User input task: initiate change_Y_density (2.4).
Interaction task(s): set the 'dichtheid yas'-slider in an instance of 'grafiekcontroleset'.

User input task: initiate display_or_hide_X_axis_values (2.5).
Interaction task(s): click the 'plaats asdomeinwaarden'-toggle in an instance of 'grafiekcontroleset'.

User input task: initiate display_or_hide_Y_axis_values (2.6).
Interaction task(s): click the 'plaats asbereikwaarden'-toggle in an instance of 'grafiekcontroleset'.

User input task: initiate display_or_hide_X_axis_description (2.7).
Interaction task(s): click the 'plaats xasomschrijving'-toggle in an instance of 'grafiekcontroleset'.

User input task: initiate display_or_hide_Y_axis_description (2.8).
Interaction task(s): click the 'plaats yasomschrijving'-toggle in an instance of 'grafiekcontroleset'.

User input task: initiate rotate (2.9).
Interaction task(s): click the 'roteer'-button in an instance of 'grafiekcontroleset'.

User input task: initiate X_Coordinate_System_Scrolling (2.10).
Interaction task(s): manipulate the horizontal scrollbar in an instance of 'grafiekendiagramset'.

User input task: initiate Y_Coordinate_System_Scrolling (2.11).
Interaction task(s): manipulate the vertical scrollbar in an instance of 'grafiekendiaagramset'.

User input task: initiate X_Coordinate_System_log (un)scaling (2.12).
Interaction task(s): click the 'plaats horizontale logaritmische schaling'-toggle in an instance of 'grafiekcontroloset'.

User input task: initiate Y_Coordinate_System_log (un)scaling (2.13).
Interaction task(s): click the 'plaats verticale logaritmische schaling'-toggle in an instance of 'grafiekcontroloset'.

User input task: initiate display_or_hide X_grid (2.14).
Interaction task(s): click the 'plaats horizontale hulplijnen'-toggle in an instance of 'grafiekcontroloset'.

User input task: initiate display_or_hide Y_grid (2.15).
Interaction task(s): click the 'plaats verticale hulplijnen'-toggle in an instance of 'grafiekcontroloset'.

User input task: initiate change_to.Area_Diagram (2.16).
Interaction task(s): click on a Graph definition (not an Area Diagram definition) in an instance of 'legendaset'; select menu item 'gebied diagram' of an instance of 'menuset'.

User input task: initiate change_to.Line_Diagram (2.17).
Interaction task(s): click on a Graph definition (not a Line Diagram definition) in an instance of 'legendaset'; select menu item 'lijndiagram' of an instance of 'menuset'.

User input task: initiate change_to.Dot_Diagram (2.18).
Interaction task(s): click on a Graph definition (not a Dot Diagram definition) in an instance of 'legendaset'; select menu item 'puntdiagram' of an instance of 'menuset'.

User input task: initiate change_to.Bar_Diagram (2.19).
Interaction task(s): click on a Graph definition (not a Bar Diagram definition) in an instance of 'legendaset'; select menu item 'staaldiagram' of an instance of 'menuset'.

User input task: initiate smooth.Line_Diagram (2.20).
Interaction task(s): click on a Graph definition (a Line Diagram definition) in an instance of 'legendaset'; select menu item 'smooth' of an instance of 'menuset'.

User input task: initiate un-smooth.Line_Diagram (2.20).
Interaction task(s): click on a Graph definition (a Line Diagram definition) in an instance of 'legendaset'; select menu item 'unsmooth' of an instance of 'menuset'.

User input task: initiate change_bar_width (2.21).
Interaction task(s): click on a Graph definition (a Bar Diagram definition) in an instance of 'legendaset'; select menu item 'staalbreedte' of an instance of 'menuset';
set 'staafbreedte'-slider in an instance of 'sliderpopupsfset'.

User input task: initiate change_line_width (2.22).
Interaction task(s): click on a Graph definition in an instance of 'legendaset';
select menu item 'lijnbreedte' of an instance of 'menuuset';
set 'lijnbreedte'-slider in an instance of 'sliderpopupsfset'.

User input task: initiate change_dot_size (2.23).
Interaction task(s): click on a Graph definition in an instance of 'legendaset';
select menu item 'puntgrootte' of an instance of 'menuuset';
set 'puntgrootte'-slider in an instance of 'sliderpopupsfset'.

We limit the shadowing of bars in Bar Diagrams to two types of shadowing.

User input task: initiate change_bar_shadow_1 (2.24).
Interaction task(s): click on a Graph definition (a Bar Diagram definition) in an instance of 'legendaset';
select menu item 'staafarcering1' of an instance of 'menuuset'.

User input task: initiate change_bar_shadow_2 (2.24).
Interaction task(s): click on a Graph definition (a Bar Diagram definition) in an instance of 'legendaset';
select menu item 'staafarcering2' of an instance of 'menuuset'.

We limit the line style types in Line Diagrams to three types.

User input task: initiate change_line_type_1 (2.25).
Interaction task(s): click on a Graph definition (a Line Diagram definition) in an instance of 'legendaset';
select menu item 'lijnsoort1' of an instance of 'menuuset'.

User input task: initiate change_line_type_2 (2.25).
Interaction task(s): click on a Graph definition (a Line Diagram definition) in an instance of 'legendaset';
select menu item 'lijnsoort2' of an instance of 'menuuset'.

User input task: initiate change_line_type_3 (2.25).
Interaction task(s): click on a Graph definition (a Line Diagram definition) in an instance of 'legendaset';
select menu item 'lijnsoort3' of an instance of 'menuuset'.

We limit the dot style types in Dot Diagrams to three types.

User input task: initiate change_dot_type_1 (2.26).
We limit the shadowing in Area Diagrams to two types of shadowing.
User input task: initiate display_or_hide_Mean_Survey (4.1).
Interaction task(s): click the 'plaats gemiddeldeoverticht'-toggle in an instance of 'grafiekcontroleset'.

User input task: initiate print_Mean_Survey (4.2).
Interaction task(s): click the 'print'-button in an instance of 'gemcontroleset'.

The scrollbarring interactions on a certain 'tabelset' (UNIT 3.2) and 'grafiekendiaigramset' (UNITs 2.10 and 2.11) do not have to be constructed, because the associated computer tasks are fully performed in the action functions of the interaction tasks by the window manager implementation of OPEN LOOK (i.e. Sun's Openwindows). This means that no communication between user interface and application semantics is needed: all task execution takes place in the user interface as far as scrolling of a Table and scrolling of a Coordinate System are concerned. Since the scrollbar interaction is the only interaction which takes place in a 'tabelset', this set is to be considered a feedback set.

**FEEDBACK TASKS**

An output task consists of one or more feedback tasks. With the identification of interaction sets and feedback sets, we are able to construct feedback tasks. Below, our construction of feedback tasks is presented; we identify the feedback tasks, describe them and show to which output task they belong; output tasks are identified by their description and UNIT number. We note that, of course, design decisions have been made in order to determine the feedback tasks described.

Output task: display initial Screen with Coordinate Systems (1.1).
Feedback task(s): create_stappencontroleset (create an instance of 'stappencontroleset');
create_grafiekendiaigramset (create an instance of 'grafiekendiaigramset');
create_legendaset (create an instance of 'legendaset');
create_grafiekcontroleset (create an instance of 'grafiekcontroleset');
create_grfstelselmeldingset (create an instance of 'grfstelselmeldingset');
init_toon_grafieken (display the Graphs in a Coordinate System);
toon_legenda (display the legend associated with a Coordinate System).

Output task: clear Screen (1.2).
Feedback task(s): verwijder_grafiekcontroleset (remove a 'grafiekcontroleset' from Screen);
verwijder_grafiekendiaigramset (remove a 'grafiekendiaigramset' from Screen);
verwijder_grfstelselmeldingset (remove a 'grfstelselmeldingset' from Screen);
verwijder_legendaset (remove a 'legendaset' from Screen);
verwijder_tabelcontroleset (remove a 'tabelcontroleset' from Screen);
verwijder_tabelkopregelset (remove a 'tabelkopregelset' from Screen);
verwijder_tabelset (remove a 'tabelset' from Screen);
verwijder_tabelmeldingset (remove a 'tabelmeldingset' from Screen);
Screen);
verwijder_gemcontroleset (remove a 'gemcontroleset' from Screen);
verwijder_gemoverzichtset (remove a 'gemoverzichtset' from Screen);
verwijder_gemmeldingset (remove a 'gemmeldingset' from Screen).

Output task: display intermediate message "printing" (2.1).
Feedback task(s): toon_grfmelding (display message in an instance of 'grfstelselmeldingset').

Output task: display message "System printed" (2.1).
Feedback task(s): toon_grfmelding (display message in an instance of 'grfstelselmeldingset').

Output task: display new Coordinate System (2.2).
Feedback task(s): create_grafiekendiagramset (create an instance of 'grafiekendiagramset');
create_legendaset (create an instance of 'legendaset');
create_grafiekcwstroleset (create an instance of 'grafiekcwstroleset');
create_grfstelselmeldingset (create an instance of 'grfstelselmeldingset');
init_toon_grafieken (display the Graphs associated with the new Coordinate System);
toon_legenda (display the legend associated with the new Coordinate System).

Output task: change X-density (2.3).
Feedback task(s): set_xasDICheid (change the density of the Domain Values along the X-axis).

Output task: change Y-density (2.4).
Feedback task(s): set_yasDICheid (change the density of the Co-domain Values along the Y-axis).

Output task: display/hide values along the X-axis (2.5).
Feedback task(s): set_xaswaarden (display/remove the Domain Values along the X-axis in an instance of 'grafiekendiagramset').

Output task: display/hide values along the Y-axis (2.6).
Feedback task(s): set_yaswaarden (display/remove the Co-Domain Values along the Y-axis in an instance of 'grafiekendiagramset').

Output task: display/hide description of the X-axis (2.7).
Feedback task(s): set_xasomschrijving (display/remove the description of the X-axis in an instance of 'grafiekendiagramset').

Output task: display/hide description of the Y-axis (2.8).
Feedback task(s): set_yasomschrijving (display/remove the description of the Y-axis in an instance of 'grafiekendiagramset').

Output task: rotate Coordinate System (2.9).
Feedback task(s): rodeer (rotate the Coordinate System in an instance of 'grafiekendiagramset' +90 degrees).
There are no feedback tasks for the Scrolling of a Coordinate System, since it is completely dealt with by Sun’s Openwindows (see the previous discussion of interaction tasks).

Output task: log (un)scale Coordinate System horizontally (2.12).
Feedback task(s): set_horlog (apply/remove horizontal log scaling in an instance of 'grafiekendiagramset').

Output task: log (un)scale Coordinate System vertically (2.13).
Feedback task(s): set_vertlog (apply/remove vertical log scaling in an instance of 'grafiekendiagramset').

Output task: display/hide a horizontal grid (2.14).
Feedback task(s): set_horhulp (display a horizontal grid on Screen, or remove a horizontal grid from Screen in an instance of 'grafiekendiagramset').

Output task: display/hide a vertical grid (2.15).
Feedback task(s): setベルhulp (display a vertical grid on Screen, or remove a vertical grid from Screen in an instance of 'grafiekendiagramset').

Output task: change Graph into an Area Diagram (2.16).
Feedback task(s): verwijder_netsel (remove an instance of 'menuset' from Screen); changegebieddiagram (change a Graph in an instance of 'grafiekendiagramset' into an Area Diagram); change_grafiekdef (change definition of Graph in an instance of 'legendaset').

Output task: change Graph into a Line Diagram (2.17).
Feedback task(s): verwijder_netsel (remove an instance of 'menuset' from Screen); change_lijndiagram (change a Graph in an instance of 'grafiekendiagramset' into a Line Diagram); change_grafiekdef (change definition of Graph in an instance of 'legendaset').

Output task: change Graph into a Dot Diagram (2.18).
Feedback task(s): verwijder_netsel (remove an instance of 'menuset' from Screen); change_puntdiagram (change a Graph in an instance of 'grafiekendiagramset' into a Dot Diagram); change_grafiekdef (change definition of Graph in an instance of 'legendaset').

Output task: change Graph into a Bar Diagram (2.19).
Feedback task(s): verwijder_netsel (remove an instance of 'menuset' from Screen); change_staafdiagram (change a Graph in an instance of 'grafiekendiagramset' into a Bar Diagram); change_grafiekdef (change definition of Graph in an instance of 'legendaset').

Output task: smooth Line Diagram (2.20).
Feedback task(s): verwijder_netsel (remove an instance of 'menuset' from Screen); set_smooth (display the lines in a Line Diagram in a smooth
fashion in an instance of 'grafiekendiaagramset').

Output task: un-smooth Line Diagram (2.20).
Feedback task(s): verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_smooth (display the lines in a Line Diagram in a non-smooth fashion in an instance of 'grafiekendiaagramset').

Output task: change the width of bars of a Bar Diagram (2.21).
Feedback task(s): verwijder_sliderpopupset (remove an instance of 'sliderpopupset' from Screen);
set_staatbreedte (adjust width of bars in a Bar Diagram in an instance of 'grafiekendiaagramset').

Output task: change the width of lines of a Line Diagram (2.22).
Feedback task(s): verwijder_sliderpopupset (remove an instance of 'sliderpopupset' from Screen);
set_lijn breedte (adjust width of lines in a Line Diagram in an instance of 'grafiekendiaagramset').

Output task: change the size of dots of a Dot Diagram (2.23).
Feedback task(s): verwijder_sliderpopupset (remove an instance of 'sliderpopupset' from Screen);
set_puntgrootte (adjust size of dots in a Dot Diagram in an instance of 'grafiekendiaagramset').

Output task: change the shadows of bars of a Bar Diagram into type 1 (2.24).
Feedback task(s): verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_staatfarceering (change the shadowing of a Bar Diagram in an instance of 'grafiekendiaagramset');
change_grafieke_def (change definition of Graph in an instance of 'legendsaset').

Output task: change the shadows of bars of a Bar Diagram into type 2 (2.24).
Feedback task(s): verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_staatfarceering (change the shadowing of a Bar Diagram in an instance of 'grafiekendiaagramset');
change_grafieke_def (change definition of Graph in an instance of 'legendsaset').

Output task: change the lines of a Line Diagram into type 1 (2.25).
Feedback task(s): verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_lijnsoort (change the type of lines in a Line Diagram in an instance of 'grafiekendiaagramset');
change_grafieke_def (change definition of Graph in an instance of 'legendsaset').

Output task: change the lines of a Line Diagram into type 2 (2.25).
Feedback task(s): verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_lijnsoort (change the type of lines in a Line Diagram in an instance of 'grafiekendiaagramset');
change_grafieke_def (change definition of Graph in an in-
stance of 'legendaset').

Output task: change the lines of a Line Diagram into type 3 (2.25).
Feedback task(s):
verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_lijnsoort (change the type of lines in a Line Diagram in an instance of 'grafiekendiagramset');
change_grafiekefdef (change definition of Graph in an instance of 'legendaset').

Output task: change the dots of a Dot Diagram into type 1 (2.26).
Feedback task(s):
verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_puntsoort (change the type of dots in a Dot Diagram in an instance of 'grafiekendiagramset');
change_grafiekefdef (change definition of Graph in an instance of 'legendaset').

Output task: change the dots of a Dot Diagram into type 2 (2.26).
Feedback task(s):
verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_puntsoort (change the type of dots in a Dot Diagram in an instance of 'grafiekendiagramset');
change_grafiekefdef (change definition of Graph in an instance of 'legendaset').

Output task: change the dots of a Dot Diagram into type 3 (2.26).
Feedback task(s):
verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_puntsoort (change the type of dots in a Dot Diagram in an instance of 'grafiekendiagramset');
change_grafiekefdef (change definition of Graph in an instance of 'legendaset').

Output task: change the shadows of an Area Diagram into type 1 (2.27).
Feedback task(s):
verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_gebiedarcering (change the shadowing of an Area Diagram in an instance of 'grafiekendiagramset');
change_grafiekefdef (change definition of Graph in an instance of 'legendaset').

Output task: change the shadows of an Area Diagram into type 2 (2.27).
Feedback task(s):
verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_gebiedarcering (change the shadowing of an Area Diagram in an instance of 'grafiekendiagramset');
change_grafiekefdef (change definition of Graph in an instance of 'legendaset').

Output task: display the Domain and Co-domain Values (2.28).
Feedback task(s):
verwijder_menuaset (remove an instance of 'menuaset' from Screen);
set_toondatawaarden (display Domain and Co-domain Values associated with a Graph in an instance of 'grafiekendiagramset').

Output task: hide the Domain and Co-domain Values (2.28).
Feedback task(s): verwijder_menuzet (remove an instance of 'menuzet' from Screen); set_toondatawaarden (remove Domain and Co-domain Values associated with a Graph in an instance of 'grafiekendiagramset').

Output task: remove a Coordinate System from Screen (2.29).
Feedback task(s): verwijder_grafiekcontroleset (remove the 'grafiekcontroleset' from Screen); verwijder_grafiekendiagramset (remove the 'grafiekendiagramset' from Screen); verwijder_grafselselmeldingset (remove the 'grafselselmeldingset' from Screen); verwijder_legendaset (remove the 'legendaset' from Screen); verwijder_tabelcontroleset (if a Table corresponds with the Coordinate System: remove the associated 'tabelcontroleset' from Screen); verwijder_tabelkopregelset (if a Table corresponds with the Coordinate System: remove the associated 'tabelkopregelset' from Screen); verwijder_tabelset (if a Table corresponds with the Coordinate System: remove the associated 'tabelset' from Screen); verwijder_tabelmeldingset (if a Table corresponds with the Coordinate System: remove the associated 'tabelmeldingset' from Screen); verwijder_gemcontroleset (if a Menu Survey corresponds with the Coordinate System: remove the associated 'gemcontroleset' from Screen); verwijder_gemoverzichtset (if a Menu Survey corresponds with the Coordinate System: remove the associated 'gemoverzichtset' from Screen); verwijder_gemmeldingset (if a Menu Survey corresponds with the Coordinate System: remove the associated 'gemmeldingset' from Screen).

Output task: display/hide a Table (3.1).
Feedback task(s): create_tabelcontroleset(only if a Table is to be displayed; create an instance of 'tabelcontroleset'); create_tabelkopregelset(only if a Table is to be displayed; create an instance of 'tabelkopregelset'); create_tabelmeldingset(only if a Table is to be displayed; create an instance of 'tabelmeldingset'); create_tabelset(only if a Table is to be displayed; create an instance of 'tabelset'); toon_tabelkopregel (only if a Table is to be displayed; show the header of a Table in an instance of 'tabelkopregelset'); toon_tabel (only if a Table is to be displayed; show Table Values in an instance of 'tabelset'); verwijder_tabelcontroleset (only if a Table is to be hidden; remove an instance of 'tabelcontroleset' from Screen); verwijder_tabelkopregelset (only if a Table is to be hidden; remove an instance of 'tabelkopregelset' from Screen); verwijder_tabelmeldingset (only if a Table is to be hidden;
There are no feedback tasks for the Scrolling of a Table, since it is completely being dealt with by Sun's Openwindows (see the previous discussion of interaction tasks).

Output task: display intermediate message "printing" (3.3).
Feedback task(s): toon_tabmelding (show message in an instance of 'tabelmeldingset').

Output task: display message "Table printed" (3.3).
Feedback task(s): toon_tabmelding (show message in an instance of 'tabelmeldingset').

Output task: display/hide a Mean Survey (4.1).
Feedback task(s): create_gemcontroleset (only if a Mean Survey is to be displayed; create an instance of 'gemcontroleset');
create_gemmeldingsset (only if a Mean Survey is to be displayed; create an instance of 'gemmeldingsset');
create_gemoverzichtset (only if a Mean Survey is to be displayed; create an instance of 'gemoverzichtset');
toon_gemiddelden (only if a Mean Survey is to be displayed; show mean values in an instance of 'gemoverzichtset');
verwijder_gemcontroleset (only if a Mean Survey is to be hidden; destroy an instance of 'gemcontroleset');
verwijder_gemoverzichtset (only if a Mean Survey is to be hidden; destroy an instance of 'gemoverzichtset');
verwijder_gemmeldingsset (only if a Mean Survey is to be hidden; destroy an instance of 'gemmeldingsset').

SET RELATIONS
Set relations allow for updates of user interface data between sets. E.g., if an interaction task takes place in one set, user interface data can be updated in another set, by means of accessing the other set via the 'set relations'. Set relations in the Graphics Manager are needed in two occurrences:

1) An interaction task in one set eventually triggers feedback tasks in other sets. E.g. most interaction tasks to be triggered in a 'grafiekcontroleset' eventually trigger feedback tasks in the corresponding 'grafiekendiagramset'.

2) A user input task consists of more than one interaction task in different sets. E.g. the user input task associated with the computer task to create a new Coordinate System is concerned with a 'grafiekcontroleset' and a 'grafiekendiagramset'.

Figure 5.28 depicts the relations between the sets for the Graphics Manager.

107
We comment that this is not the only possible set relation schema for the Graphics Manager; other set relation constructions could suffice as well.

The third substep of the design step of D2M2design is the coding of the semantic feedback functions (see section 4.4).

**SEMANTIC FEEDBACK FUNCTIONS**

The Graphics Manager contains only one semantic feedback function which must be executed when the user is selecting two Domain Values for the computer task 'nieuwgrafstelsel' (create_new_System) (see the previous discussion of interaction tasks).

Semantic feedback task: geldig_Domain_Interval (Is the Domain Interval selected by the user valid? Is the last selected Domain Value, along the X-axis, on the right of the first selected Domain Value? Does the Domain Interval at least contain one Data Value?)

Parameters:
- the name of the involved instance of the user interface data type 'grafiekendiagram', which corresponds to the involved instance of the application semantics data type 'stelsel';
- the first Domain Value;
- the second Domain Value.

**INTERACTION TASK ENABLING**

The fourth substep of the design step of D2M2design is the determination of interaction task enablings and their integration with computer tasks and triggers to feedback tasks. We illustrate this substep by identifying the interaction task enablings; in appendix A the application semantics code is presented, as depicted on page 62.

The Graphics Manager is a HICS which allows its users to start an arbitrary user input task at an arbitrary point in run-time. Therefore the coordinating decisions constructed in the Graphics Manager enable all possible user input tasks; so, there is only one type of coordinating decision
which contains the following interaction task enablings. Note that the interaction tasks are not actually enabled, but the entries to which the interaction tasks belong. In the Graphics Manager, every entry corresponds to one interaction task and vice versa.

enable entry "initiate stop"
enable entry "initiate print_Coordinate_System"
enable entry "initiate create_new_System"
enable entry "initiate change_X_density"
enable entry "initiate change_Y_density"
enable entry "initiate display_or_hide_X_axis_values"
enable entry "initiate display_or_hide_Y_axis_values"
enable entry "initiate display_or_hide_X_axis_description"
enable entry "initiate display_or_hide_Y_axis_description"
enable entry "initiate rotate"
enable entry "initiate X_Coordinate_System_log_(un)scaling"
enable entry "initiate Y_Coordinate_System_log_(un)scaling"
enable entry "initiate display_or_hide_X_grid"
enable entry "initiate display_or_hide_Y_grid"
enable entry "initiate change_to_Area_Diagram"
enable entry "initiate change_to_Line_Diagram"
enable entry "initiate change_to_Dot_Diagram"
enable entry "initiate change_to_Bar_Diagram"
enable entry "initiate smooth_Line_Diagram"
enable entry "initiate un-smooth_Line_Diagram"
enable entry "initiate change_bar_width"
enable entry "initiate change_line_width"
enable entry "initiate change_dot_size"
enable entry "initiate change_bar_shadow_1"
enable entry "initiate change_bar_shadow_2"
enable entry "initiate change_line_type_1"
enable entry "initiate change_line_type_2"
enable entry "initiate change_line_type_3"
enable entry "initiate change_dot_type_1"
enable entry "initiate change_dot_type_2"
enable entry "initiate change_dot_type_3"
enable entry "initiate change_area_shadow_1"
enable entry "initiate change_area_shadow_2"
enable entry "initiate display_(Co-)Domain_Values"
enable entry "initiate hide_(Co-)Domain_Values"
enable entry "initiate delete_Coordinate_System"
enable entry "initiate display_or_hide_Table"
enable entry "initiate print_Table"
enable entry "initiate display_or_hide_Mean_Survey"

Note that if a user input task contains more than one interaction task, only the interaction task user input task execution can be started with, is enabled with the coordinating decision. Note further, that scrollbar interaction does not have to be enabled because Sun's Openwindows fully takes care of scrollbaring.

We are now able to construct the application semantics code for the Graphics Manager; this code contains the interaction task enablings, computer tasks and triggers to feedback tasks. In appendix A the code is listed.

In the pseudo code listed in appendix A a number of interesting issues can be noticed.
Firstly, after initialisation, all possible entries of the Graphics Manager are enabled: the coordinating decisions enable all possible entries. However, some enabled user input tasks just cannot be performed, because of the fact that the associate user interface objects are not present. This is e.g. the case with "enable entry 'initiate print_Table'"; initially, there is no Table present on screen, so the 'print'-button in a Table cannot be clicked. Obviously, enabling all possible entries eases the job to program the application code. Moreover, the phylosophy of Direct Manipulation in graphical environments, i.e. everything visual on screen can often be manipulated, is reflected by such coordinating decisions.

Secondly, most CASE-entries do not contain a computer task, i.e. application semantics data does not have to be consulted or updated after user interaction. Clearly, in the Graphics Manager type III complications (images on screen (user interface) are what the computer tasks (application semantics) are intended to manipulate) often occur.

Thirdly, e.g. computer task 'printgrafstelsel' contains a trigger to the 'toon_grfmelding' feedback tasks. Clearly, in the Graphics Manager type II complications (feedback task execution during computer task execution) occur.

So far we discuss the application semantics code and the presentation of design specification of the Graphics Manager as they can be entered in D2M2edit. In the following section we discuss some modifications to the design specifications of the Graphics Manager.

5.4. MODIFICATIONS

We recall from chapter 1, section 1.4, that when the user interface is separated from the application semantics the user interface can be easily changed, without having to change the application semantics part. Therefore, in this section, we modify the design specifications of the Graphics Manager (using D2M2edit) in order to verify whether the user interface is truly separated from the application semantics; if the modifications are not easily applied using D2M2edit, the separation of the user interface and the application semantics has clearly not succeeded for the Graphics Manager.

We distinguish the modifications: addition, deletion and change. Related to the sequencing of human-computer interaction of figure 2.3, user input tasks, output tasks, computer tasks and coordinating decisions are considered. So, we apply the following modifications to the Graphics Manager.

1. Addition of a User input task (AU).
2. Addition of an Output task (AO).
3. Addition of a Computer task (AC).
4. Addition of a coordinating Decision (AD).
5. Deletion of a User input task (DU).
6. Deletion of an Output task (DO).
7. Deletion of a Computer task (DC).
8. Deletion of a coordinating Decision (DD).
10. Change of an Output task (CO).
If a modification in the user interface does not affect the application semantics part of the design specifications or vice versa, the modification is said to be easy; otherwise, the modification is said to be difficult.

In the following a number of modifications are listed. The modifications are hypothetical, but may well be practised in case of the Graphics Manager. A more detailed discussion of the alterations of the Graphics Manager is presented by Versendaal (1991).

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Type</th>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Extend 'grafiekcontroleset' with a button, which, when clicked, visualises a Table.</td>
<td>AU</td>
<td>easy</td>
</tr>
<tr>
<td>2</td>
<td>Remove Table by clicking a new button (‘remove’) inside an instance of ‘tabellencontroleset’.</td>
<td>AU</td>
<td>easy</td>
</tr>
<tr>
<td>3</td>
<td>Add intermediate output task in computer task ‘pl_gemoverzicht’ to intermediately display the means of the Graphs in the ‘grastelselmeldingset’.</td>
<td>AO</td>
<td>difficult</td>
</tr>
<tr>
<td>4</td>
<td>Add a pop-up window with an ‘OK’-button to allow for displaying the message that a Coordinate System has been successfully printed.</td>
<td>AD</td>
<td>difficult</td>
</tr>
<tr>
<td>5</td>
<td>Remove the ‘table’-button from ‘grafiekcontroleset’ (see first AU).</td>
<td>DU</td>
<td>easy</td>
</tr>
<tr>
<td>6</td>
<td>Remove the ‘remove’-button (see second AU).</td>
<td>DU</td>
<td>easy</td>
</tr>
<tr>
<td>7</td>
<td>Delete the intermediate output task in a ‘grastelselmeldingset’, which indicates that a certain Coordinate System is being printed out.</td>
<td>DO</td>
<td>difficult</td>
</tr>
<tr>
<td>8</td>
<td>Delete the AD mentioned above.</td>
<td>DD</td>
<td>difficult</td>
</tr>
<tr>
<td>9</td>
<td>Click on a not-smooth Line Diagram definition to make it smooth, instead of selecting the ‘smooth-item’ in a menu.</td>
<td>CU</td>
<td>easy</td>
</tr>
<tr>
<td>10</td>
<td>Click on a non-smooth Line Diagram to make it smooth, instead of selecting the ‘smooth-item’ in a menu.</td>
<td>CU</td>
<td>easy</td>
</tr>
<tr>
<td>11</td>
<td>Delete the semantic feedback from the interaction task, through which two Domain Values can be selected in a Coordinate System.</td>
<td>CU</td>
<td>difficult</td>
</tr>
<tr>
<td>12</td>
<td>Change the language of the message, which indicates that a Coordinate System has been printed successfully.</td>
<td>CO</td>
<td>easy</td>
</tr>
<tr>
<td>13</td>
<td>Add an extra message in a ‘grastelselmeldingset’, which indicates that a Table is visualised.</td>
<td>CO</td>
<td>easy</td>
</tr>
<tr>
<td>No.</td>
<td>Description</td>
<td>Type</td>
<td>Effort</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------------------------------------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>14</td>
<td>Only allow printing on a laser printer.</td>
<td>CC</td>
<td>easy</td>
</tr>
<tr>
<td>15</td>
<td>Delete enabling of 'printtable' and 'printgemstelsel' in the first coordinating decision.</td>
<td>CD</td>
<td>easy</td>
</tr>
</tbody>
</table>

5.5. CONCLUSIONS
The design and modification of the Graphics Manager allows us to draw three conclusions.

1. D2M2environment is suitable for the development and maintenance of the Graphics Manager. We were able to construct and modify design specifications using D2M2edit.

2. In the Graphics Manager logical separation is fully achieved. Because D2M2environment is based on the Delft model, logical separation is fully achieved; application semantics data is managed by the application semantics process and user interface data is managed by D2M2run.

3. In the Graphics Manager physical separation is not fully achieved. As can be read from the listed modifications of the Graphics Manager, not all modifications were achieved easily. As for the modifications of the application semantics Hartson (1989, p 63) already stated: "Of course, if you add a new function to the application, you must add both dialogue and computational part".
6. CASE 2: GRAPHIC CALCULATOR

6.1. INTRODUCTION

The Graphic Calculator is a "Computer Assisted Instruction"-application which has the purpose to learn children to add and subtract. A functional description of the Graphic Calculator was determined after discussions with SLO (Institute for Curriculum Development). In section 6.2 the functional description is presented. In section 6.3 the functional description is analysed; further, the design specifications of the Graphic Calculator are presented as they can be entered in D2M2edit. In section 6.4 modifications to the Graphic Calculator are applied. Finally, in section 6.5, conclusions are drawn with relation to the development and modification of the Graphic Calculator in D2M2environment.

6.2. DESCRIPTION

Before we enumerate the requirements we define some concepts, specific to the Graphic Calculator.

M-set: a set of sets of M-elements.

M-set-label: alpha-numeric representation of an M-set, in which M-elements are grouped. E.g. an M-set containing 6 100ct pieces, 4 5ct pieces and 3 25ct pieces has M-set-label: "6G + 4ST + 3KW = 6.95".

M-element: either a Coin or a Cube.

Coin: either a 1ct piece (cent or CT), a 5ct piece (stuiver or ST), a 10ct piece (dubbeljetje or DB), a 25ct piece (kwartje or KW), a 100ct piece (gulden or G), a 250ct piece (rijksdaalder or RD) or a 500ct piece (vijf gulden or VG) (see figure 6.1).

![Coin types](image)

*Figure 6.1: the seven Coin types*

Cube: either a 1-block (1-b), 10-block (10-b), 100-block (100-b) or 1000-block (1000-b) (see figure 6.2).
Figure 6.2: the four Cube types

Instruction Problem: problem (addition or subtraction) worked on by the user.

Screen: area in which the user works on a certain Instruction Problem.

We now present the list of functional requirements of the Graphic Calculator. We note that this description is one of the possible interpretations of the discussions with SLO.

SECTION 1 start/stop Graphic Calculator
UNIT 1
   The Graphic Calculator can be started.
UNIT 2
   The Graphic Calculator can be stopped, implying that control returns to the operating system. Via a pop-up window the user is informed about leaving the Graphic Calculator; the user must remove the pop-up in order to leave the Graphic Calculator.
END SECTION 1 start/stop Graphic Calculator

SECTION 2 Instruction Problem solving
UNIT 1
   The user can input a new Instruction Problem, either an addition or a subtraction.
   The user is allowed to let generate default M-sets, just after entering a new Instruction Problem.
UNIT 2
   The user can change the representation of all M-sets on Screen.
UNIT 3
   The user can create an M-set.
UNIT 4
   The user can add a new M-elements in an M-set.
UNIT 5
   The user can move an M-element from one M-set to another.
UNIT 6
   The user can move all identical M-elements (e.g. all KWs) from one M-set to another.
UNIT7
The user can clear the Screen, in case the M-sets and the Instruction Problem will be removed.

UNIT8
The user can select an already worked on set of M-sets with a certain Instruction Problem associated with it.

UNIT9
The user can remove an M-set from Screen.

UNIT10
The user can remove a couple of M-elements from a set in order to perform a subtraction.

UNIT11
The user can exchange M-elements into identical M-elements (e.g. KWs).

UNIT12
The user can join M-sets in order to perform addition.

UNIT13
The user can select an M-set in order to identify the M-set giving the solution to the Instruction Problem.

END SECTION2 Instruction Problem solving

SECTION3 Help facility

UNIT1
The user can ask for context sensitive help information. The help information is displayed in a pop-up window, which must be removed as soon as the help information is read.

END SECTION3 Help facility

So far the functional description for the development of the Graphic Calculator has been given.

6.3. CONSTRUCTION OF DESIGN SPECIFICATIONS
In this section the functional description in terms of SECTIONS and OBJECTS is analysed and design specifications are constructed; the steps of D2M2design are applied. A more detailed description of the design specifications is presented by Versendaal (1991).

In the following the results of the analysis step of D2M2design are presented. The first column contains the SECTION and UNIT identifications to which the following three columns refer. The second column contains the identifications of the user input tasks; the third column contains the identifications of the computer tasks; the fourth column contains the identifications of the output tasks.

<table>
<thead>
<tr>
<th>SECTION/UNIT</th>
<th>user input task</th>
<th>computer task</th>
<th>output tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>initiate initialise</td>
<td>initialise</td>
<td>- display initial Screen</td>
</tr>
<tr>
<td>1.2</td>
<td>initiate stop</td>
<td>stop</td>
<td>- pop-up the window with stop information</td>
</tr>
<tr>
<td>SECTION/UNIT</td>
<td>user input task</td>
<td>computer task</td>
<td>output tasks</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>1.2</td>
<td>initiate stop_definitely</td>
<td>stop_definitely</td>
<td>- remove the pop-up window and clear Screen</td>
</tr>
<tr>
<td>1.2</td>
<td>initiate stop_not_definitely</td>
<td>stop_not_definitely</td>
<td>- remove the pop-up window</td>
</tr>
<tr>
<td>2.1</td>
<td>initiate construct_new_Instruction_Problem</td>
<td>construct_new_Instruction_Problem</td>
<td>- display new Instruction Problem</td>
</tr>
<tr>
<td>2.1</td>
<td>initiate generate_default_M_sets</td>
<td>generate_default_M_sets</td>
<td>- display default M_sets</td>
</tr>
<tr>
<td>2.2</td>
<td>initiate change_representation</td>
<td>change_representation</td>
<td>- display new representation</td>
</tr>
<tr>
<td>2.3</td>
<td>initiate create_M_set</td>
<td>create_M_set</td>
<td>- display new M-set</td>
</tr>
<tr>
<td>2.4</td>
<td>initiate add_new_M_element</td>
<td>add_new_M_element</td>
<td>- display new M-element in M-set</td>
</tr>
<tr>
<td>2.5</td>
<td>initiate move_M_element</td>
<td>move_M_element</td>
<td>- remove M-element from one M-set and display it in another M-set</td>
</tr>
<tr>
<td>2.6</td>
<td>initiate move_identical_M_elements</td>
<td>move_identical_M_elements</td>
<td>- remove M-elements from one M-set and display them in another M-set</td>
</tr>
<tr>
<td>2.7</td>
<td>initiate clear</td>
<td>clear</td>
<td>- remove all M-sets from Screen and clear Instruction Problem</td>
</tr>
<tr>
<td>2.8</td>
<td>initiate get_old_set_of_M_sets</td>
<td>get_old_set_of_M_sets</td>
<td>- remove current set of M-sets and display an old one</td>
</tr>
<tr>
<td>2.9</td>
<td>initiate remove_M_set</td>
<td>remove_M_set</td>
<td>- remove an M-set</td>
</tr>
<tr>
<td>2.10</td>
<td>initiate remove_M_elements</td>
<td>remove_M_elements</td>
<td>- remove M-elements from M-set</td>
</tr>
<tr>
<td>2.11</td>
<td>initiate exchange_M_elements</td>
<td>exchange_M_elements</td>
<td>- remove M-elements and display new ones in an M-set</td>
</tr>
<tr>
<td>2.12</td>
<td>initiate join_M_sets</td>
<td>join_M_sets</td>
<td>- replace M-sets by one M-set</td>
</tr>
<tr>
<td>2.13</td>
<td>initiate check_solution</td>
<td>check_solution</td>
<td>- show solution correctness</td>
</tr>
</tbody>
</table>

116
### Table:

<table>
<thead>
<tr>
<th>SECTION/UNIT</th>
<th>user input task</th>
<th>computer task</th>
<th>output tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>initiate give_help</td>
<td>give_help</td>
<td>- pop-up the window with help information</td>
</tr>
<tr>
<td>3.1</td>
<td>initiate remove_help</td>
<td>remove_help</td>
<td>- remove the pop-up window with the help</td>
</tr>
</tbody>
</table>

Having analysed the UNITS we are now able to construct the design specifications. We first present the application semantics data types and the computer tasks. We present the names, generalization relations, aggregation relations and attributes of the application semantics data types; we present the names and a description of the computer tasks.

**APPLICATION SEMANTICS DATA TYPES AND COMPUTER TASKS**

As mentioned the Graphic Calculator is a HICS allowing children to learn to add and subtract. They interact with the Graphic Calculator by entering an addition or subtraction on Screen and manipulating groups of Coins or Cubes. We therefore identify an application semantics data type which represents the addition or subtraction with its result. We call it 'set_of_M_sets' and give it attributes: first operand, second operand, operator (addition or subtraction) and result. Moreover we identify the application semantics data type 'M_set'. It classifies in the application semantics the groups of Coins or Cubes the user is interacting with. It has attributes: a unique M_set handle, the value of the M_set and a list of pairs representing the number of M-elements of all kinds included in the M_set. E.g. a group of 2DBs and 1ST on Screen has an associated M-set in the application semantics with value 15 and two pairs, (2, 10) and (1, 5), representing the 2DBs and 1ST. The relation between the two identifiable application semantics data types is depicted in figure 6.3.

![Figure 6.3: application semantics data types and relations](image)

So, a set_of_M_sets will be dealing with zero or more M_sets during run-time of the Graphic Calculator.

Related to the application semantics data types we define a number of computer tasks. All other mentioned computer tasks of the result of the analysis step are not to be considered real computer tasks because they do not consult or update application semantics data: complication III (computer task execution only dealing with the user interface and not with the application semantics) manifests itself.
<table>
<thead>
<tr>
<th>computer task</th>
<th>related UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop</td>
<td>1.2</td>
</tr>
<tr>
<td>construct_new_Instruction_Problem</td>
<td>2.1</td>
</tr>
<tr>
<td>generate_default_M_sets</td>
<td>2.1</td>
</tr>
<tr>
<td>create_M_set</td>
<td>2.3</td>
</tr>
<tr>
<td>add_new_M_element</td>
<td>2.4</td>
</tr>
<tr>
<td>move_M_element</td>
<td>2.5</td>
</tr>
<tr>
<td>move_Identical_M_elements</td>
<td>2.6</td>
</tr>
<tr>
<td>clear</td>
<td>2.7</td>
</tr>
<tr>
<td>get_old_set_of_M_sets</td>
<td>2.8</td>
</tr>
<tr>
<td>remove_M_set</td>
<td>2.9</td>
</tr>
<tr>
<td>remove_M_elements</td>
<td>2.10</td>
</tr>
<tr>
<td>exchange_M_elements</td>
<td>2.11</td>
</tr>
<tr>
<td>join_M_sets</td>
<td>2.12</td>
</tr>
<tr>
<td>check_solution</td>
<td>2.13</td>
</tr>
</tbody>
</table>

We now present the interaction tasks, feedback tasks, interaction sets, feedback sets, the user interface data types and set relations. Although they must be constructed cooperatively, in parallel, we discuss subsequently: user interface data types, interaction sets, feedback sets, interaction tasks, feedback tasks and set relations.

**USER INTERFACE DATA TYPES**

We present the names (in Dutch), descriptions and generalization relations of the user interface data types. We start with the specialized user interface data types of 'own'. These data types are the data types we consider as necessary in order to present the Graphics Manager on Screen.

We consider an M-set presented on Screen to consist of M-elements ('M_element' is identified as user interface data type), being either Coins or Cubes ('coin' and 'cube' are identified as user interface data types). We consider an information panel, which among others allows the presentation of help information, to consist of alpha-numeric text ('text' is identified as user interface data type).

Figures 6.4 depicts the names, and relations of the user interface data types 'own' as they are entered in D2M2edit. Below the figure, descriptions of the user interface data types are presented.
In contrast with the HICS of chapter 5, this HICS contains few specializations of 'own'. The generalized user interface data type 'own' contains attributes and methods common to all its specialized data types.

The descriptions of the specialized user interface data types of figure 6.4 are as follows.

- coin: Coin;
- cube: Cube;
- M_element: M-element, either a Cube or a Coin;
- text: text, displayed in a window to present help information and to present the text for stopping the Graphic Calculator.

We elect to use the following XView user interface data types in order to allow for the user input tasks and output tasks in the Graphic Calculator (see figure 6.5).

The generalized user interface data type 'native' contains attributes and methods common to all its specialized data types. The user interface data types 'choice' is itself a generalized data type. Its hierarchy will be visualised later in this section.

The specialized user interface data types of figure 6.5 are XView data types.

- button: Panel Button Item;
- canvas: Canvas;
- choice: Panel Choice Item;
- list: List;
- message: Panel Message Item;
- menu: Menu;
- menu_item: Menu Item;
- panel: Panel;
- scrollbar: Scrollbar;
- text_item: Panel Text Item
- window: Frame.
In the Graphic Calculator we identify two specialized types of choices: one for representing the M-set-labels and one for representing the choice between either Coins or Cubes to be used in the M-sets.

The description of the specialized user interface data types are as follows.

- **label_choice**: choice representing the M-set-label;
- **material_choice**: item representing the choice between either Coins or Cubes that can be used in the M-sets.

We now construct the interaction sets and feedback sets.
**INTERACTION SETS**

Interaction and feedback sets are visualised inside windows (XView-frames). In the case of the Graphic Calculator we propose five types of windows. We identify a window type for allowing displaying and manipulating either help or stop information (FRAME 1); we identify a window type for entering the Instruction Problem, displaying the material representation (either Coins or Cubes) and manipulating all M-sets (FRAME 2); we identify a window type for listing a number of already constructed sets of M-sets (FRAME 3); we identify a window type for displaying M-sets (FRAME 4); finally, we identify a window type for dragging M-elements over M-sets (FRAME 5).

We choose to associate every 'panel' and every 'canvas' user interface data type of XView with one set, either interaction set or feedback set. In FRAME 1 we identify the interaction set 'information_panel', which allows the user to e.g. stop the Graphic Calculator definitely. In FRAME 2 we identify the interaction set 'instruction_problem_panel', which allows the user to enter the Instruction Problem; in FRAME 2 we further identify the interaction set 'material_panel', which shows the M-elements the user can work with; in FRAME 2 we further identify the interaction set 'representation_panel', which allows the user to switch from type of M-element; finally, we identify the interaction set 'action_panel' in FRAME 2, which allows the user to e.g. initiate the presentation of help information. In FRAME 3 we identify the interaction set 'list_area', which allows the user to select an already constructed set of M-sets. In FRAME 4 we identify the interaction set 'M_element_canvas', in which the M-elements are displayed; in FRAME 4 we further identify the interaction set 'M_set_button_panel', in which some operations on the M-set can be accomplished; finally, we identify the interaction set 'M_set_label_panel' in FRAME 4, in which the M-set-label is depicted. In FRAME 5 we identify the interaction set 'ghost_canvas', in which to be dragged M-elements are depicted. Below a summary of interaction sets and FRAMEs is presented.

**FRAME 1:** information_panel  
**FRAME 2:** instruction_problem_panel  
material_panel  
representation_panel  
action_panel  
**FRAME 3:** list_area  
**FRAME 4:** M_element_canvas  
M_set_button_panel  
M_set_label_panel  
**FRAME 5:** ghost_canvas

Having identified the interaction sets, we are now able to identify the user interface data types that are used within the interaction sets and we are able to construct the parent-child relations of the interaction sets. We present each interaction set by its parent-child relations. The layout of all sets (both interaction sets and feedback sets) is presented at the end of the discussion of feedback sets.

Figure 6.7 depicts the parent-child relations of the interaction set 'information_panel'.

121
Figure 6.7: parent-child relations of 'information_panel'

Note that the user interface data type 'window' reflects FRAME 1 in the case of 'information_panel'.

Figure 6.8 depicts the parent-child relations of the interaction set 'instruction_problem_panel'.

Figure 6.8: parent-child relations of 'instruction_problem_panel'

Note that the user interface data type 'window' reflects FRAME 2 in the case of 'instruction_problem_panel'.

Figure 6.9 depicts the parent-child relations of the interaction set 'material_panel'.

Figure 6.9: parent-child relations of 'material_panel'

Note that the user interface data type 'window' reflects FRAME 2 in the case of 'material_panel'.

Figure 6.10 depicts the parent-child relations of the interaction set 'representation_panel'.
Figure 6.10: parent-child relations of 'representation_panel'

Note that the user interface data type 'window' reflects FRAME 2 in the case of 'representation_panel'.

Figure 6.11 depicts the parent-child relations of the interaction set 'action_panel'.

Figure 6.11: parent-child relations of 'action_panel'

Note that the user interface data type 'window' reflects FRAME 2 in the case of the 'action_panel'.

Figure 6.12 depicts the parent-child relations of the interaction set 'list_area'.

Figure 6.12: parent-child relations of 'list_area'

Note that the user interface data type 'window' reflects FRAME 3 in the case of the 'list_area'.

Figure 6.13 depicts the parent-child relations of the interaction set 'M_element_canvas'.

123
Note that the user interface data type 'window' reflects FRAME 4 in the case of the 'M_element_canvas'.

Figure 6.14 depicts the parent-child relations of the interaction set 'M_set_button_panel'.

Note that the user interface data type 'window' reflects FRAME 4 in the case of the 'M_set_label_panel'.

Figure 6.15 depicts the parent-child relations of the 'M_set_label_panel'.

Note that the user interface data type 'window' reflects FRAME 4 in the case of the 'M_set_label_panel'.

Figure 6.16 depicts the parent-child relations of the interaction set 'ghost_canvas'.
Note that the user interface data type 'window' reflects FRAME 5 in the case of the 'ghost_canvas'.

Feedback Sets
In FRAME 1 we identify the feedback set 'information_canvas', in which the help information and stop information can be depicted. In FRAME 2 we identify the feedback set 'message_panel', in which e.g. notification of correctness of a given solution to the Instruction Problem is presented. Below the relations between the feedback sets and FRAMEs are summarized.

FRAME 1:  information_canvas
FRAME 2:  message_panel

Having identified the feedback sets, we are now able to identify the user interface data types that are used within the feedback sets and we are able to construct the parent-child relations of the feedback sets. We present each feedback set by its parent-child relations.

Figure 6.17 depicts the parent-child relations of the feedback set 'information_canvas'.

Note that the user interface data type 'window' reflects FRAME 1 in the case of 'information_canvas'.

Figure 6.18 depicts the parent-child relations of the feedback set 'message_panel'. Note that the user interface data type 'window' reflects FRAME 2 in the case of the 'message_panel'.

In figure 6.19 the layout of all sets, as they appear on Screen, is depicted.
Figure 6.18: parent-child relations of 'message_panel'

Figure 6.19: layout of all sets of the Graphic Calculator

126
**INTERACTION TASKS**

A user input task consists of one or more interaction tasks. With the identification of interaction sets, we are able to construct interaction tasks. Below, our construction of interaction tasks is presented; we describe the interaction tasks and show to which user input task they belong; user input tasks are identified by their description and UNIT number. We note that, of course, design decisions have been made in order to determine the interaction tasks being described.

**User input task:** initiate stop (1.2).
**Interaction task(s):** click the 'stop'-button in the instance of 'action_panel'.

**User input task:** initiate stop_definitely (1.2).
**Interaction task(s):** click the 'ok'-button in an instance of 'information_panel'.

**User input task:** initiate stop_not_definitely (1.2).
**Interaction task(s):** click the 'herstel'-button in an instance of 'information_panel'.

**User input task:** initiate construct_new_instruction_Problem (2.1).
**Interaction task(s):** enter Instruction Problem in the text item of the instance of 'instruction_problem_panel'.

**User input task:** initiate generate_default_M_sets (2.1).
**Interaction task(s):** click the 'genereren'-button in the instance of 'action_panel'.

**User input task:** initiate change_representation (2.2).
**Interaction task(s):** click the not-selected choice item in the instance of 'representation_panel'.

**User input task:** initiate create_M_set (2.3).
**Interaction task(s):** click the 'creeren'-button in the instance of 'action_panel'.

**User input task:** initiate add_new_M_element (2.4).
**Interaction task(s):** click a material choice in the instance of 'material_panel';
drag popped-up instance of 'ghost_canvas' (holding the new M-element) to an instance of 'M_element_canvas'.

**User input task:** initiate move_M_element (2.5).
**Interaction task(s):** click an M-element in an instance of 'M_element_canvas';
drag popped-up instance of 'ghost_canvas' (holding the to be moved M-element) to another instance of 'M_element_canvas'.

**User input task:** initiate move_identical_M_elements (2.6).
**Interaction task(s):** click a label choice item in an instance of 'M_set_label_panel';
drag popped-up instance of 'ghost_canvas' (holding the to be moved M-elements) to another instance of 'M_element_canvas'.

**User input task:** initiate clear (2.7).
Interaction task(s): click the 'nieuw'-button in the instance of 'action_panel'.

User input task: initiate get_old_set_of_M_sets (2.8).
Interaction task(s): click the 'oud'-button in the instance of 'action_panel'; select item (representing a set of M-sets) in the popped-up instance of 'list_area'.

User input task: initiate remove_M_set (2.9).
Interaction task(s): click the 'verwijder'-button in the corresponding instance of 'M_set_button_panel'.

User input task: initiate remove_M_elements (2.10).
Interaction task(s): select M-elements in an instance of 'M_element_canvas'; click the 'trek af'-button in the associated instance of 'M_set_button_panel'.

User input task: initiate exchange_M_elements (2.11).
Interaction task(s): select M-elements in an instance of 'M_element_canvas'; click the 'wissel'-button in the associated instance of 'M_set_button_panel'; click a material choice item in an instance of 'material_panel'.

User input task: initiate join_M_sets (2.12).
Interaction task(s): click 'selekteer'-buttons in instances of 'M_set_button_panel'; click the 'tel op'-button in the instance of 'action_panel'.

User input task: initiate check_solution (2.13).
Interaction task(s): click the 'oplossing'-button in an instance of 'M_set_button_panel'.

User input task: initiate give_help (3.1).
Interaction task(s): click the 'help'-button in the instance of 'action_panel'.

User input task: initiate remove_help (3.1).
Interaction task(s): click the 'ok'-button in the popped-up instance of 'information_panel'.

FEEDBACK TASKS

An output task consists of one or more feedback tasks. With the identification of interaction sets and feedback sets, we are able to construct feedback tasks. Below, our construction of feedback tasks is presented; we identify the feedback tasks, describe them and show to which output task they belong; output tasks are identified by their description and UNIT number. We note that, of course, design decisions have been made in order to determine the feedback tasks being described.

Output task: display initial Screen (1.1).
Feedback task(s): create_instruction_problem_panel (create an instance of 'instruction_problem_panel'); create_material_panel (create an instance of 'material_
panel' with Coins as default representation);
create_representation_panel (create an instance of 'representation_panel' with Coins as default choice item);
create_action_panel (create an instance of 'action_panel');
create_message_panel (create an instance of 'message_panel').

Output task: pop-up the window with the stop information (1.2).
Feedback task(s): create_information_panel (create an instance of 'information_panel');
create_information_canvas (create an instance of 'information_canvas');
set_popup_panel (create button instance(s) in an instance of 'information_panel');
set_popup_text (display text in an instance of 'information_canvas').

Output task: remove the pop-up window and clear Screen (1.2).
Feedback task(s): delete_information_canvas (remove the instance of 'information_canvas' from Screen);
delete_information_panel (remove the instance of 'information_panel' from Screen);
delete_material_panel (remove the instance of 'material_panel' from Screen);
delete_representation_panel (remove the instance of 'representation_panel' from Screen);
delete_instruction_problem_panel (remove the instance of 'instruction_problem_panel' from Screen);
delete_action_panel (remove the instance of 'action_panel' from Screen);
delete_message_panel (remove the instance of 'message_panel' from Screen);
delete_M_element_canvas (remove an instance of 'M_element_canvas' from Screen);
delete_M_set_button_panel (remove an instance of 'M_set_button_panel' from Screen);
delete_M_set_label_panel (remove an instance of 'M_set_label_panel' from Screen).

Output task: remove the pop-up window (1.2).
Feedback task(s): delete_information_canvas (remove the instance of 'information_canvas' from Screen);
delete_information_panel (remove the instance of 'information_panel' from Screen).

Output task: display new Instruction Problem (2.1).
Feedback task(s): update_message (update message in the instance of 'message_panel').

Output task: display default M_sets (2.1).
Feedback task(s): create_M_element_canvas (create an instance of 'M_element_canvas');
create_M_set_button_panel (create an instance of 'M_set_button_panel');
create_M_set_label_panel (create an instance of 'M_set_label_panel');
add_M_elements (add new M-element(s) in an instance of
update_label (update the M-set-label in an instance of 'M_set_label_panel');

**Output task:** display new representation (2.2).

**Feedback task(s):** update_material_choice (update the material_choice in an instance of 'material_panel');
add_M_elements (add new M-element(s) in an instance of 'M_element_canvas');
update_label (update the M-set-label in an instance of 'M_set_label_panel');
delete_M_elements (remove M-elements from an instance of 'M_element_canvas').

**Output task:** display new M-set (2.3).

**Feedback task(s):** create_M_element_canvas (create an instance of 'M_element_canvas');
create_M_set_button_panel (create an instance of 'M_set_button_panel');
create_M_set_label_panel (create an instance of 'M_set_label_panel').

**Output task:** display new M-element in M-set (2.4).

**Feedback task(s):** add_M_elements (add new M-element(s) in an instance of 'M_element_canvas');
update_label (update the M-set-label in an instance of 'M_set_label_panel').

**Output task:** remove M-element from one M-set and display it in another M-set (2.5).

**Feedback task(s):** add_M_elements (add new M-element(s) in an instance of 'M_element_canvas');
update_label (update the M-set-label in an instance of 'M_set_label_panel');
delete_M_elements (remove M-elements from an instance of 'M_element_canvas').

**Output task:** remove M-elements from one M-set and display them in another M-set (2.6).

**Feedback task(s):** add_M_elements (add new M-element(s) in an instance of 'M_element_canvas');
update_label (update the M-set-label in an instance of 'M_set_label_panel');
delete_M_elements (remove M-elements from an instance of 'M_element_canvas').

**Output task:** remove all M-sets from Screen and clear Instruction Problem (2.7).

**Feedback task(s):** delete_M_element_canvas (remove an instance of 'M_element_canvas' from Screen);
delete_M_set_button_panel (remove an instance of 'M_set_button_panel' from Screen);
delete_M_set_label_panel (remove an instance of 'M_set_label_panel' from Screen);
update_instruction_problem (update the Instruction Problem in the instance of 'instruction_problem_panel');
update_message (update message in the instance of 'message_panel').
Output task: remove current set of M-sets and display an old one (2.8).
Feedback task(s):
delete_M_element_canvas (remove an instance of 'M_element_canvas' from Screen);
delete_M_set_button_panel (remove an instance of 'M_set_button_panel' from Screen);
delete_M_set_label_panel (remove an instance of 'M_set_label_panel' from Screen);
create_M_element_canvas (create an instance of 'M_element_canvas');
create_M_set_button_panel (create an instance of 'M_set_button_panel');
create_M_set_label_panel (create an instance of 'M_set_label_panel');
update_material_choice (update the material_choice in an instance of 'material_panel');
add_M_elements (add new M-element(s) in an instance of 'M_element_canvas');
update_label (update the M-set-label in an instance of 'M_set_label_panel');
update_representation (update the representation in the instance of 'representation_panel');
update_Instruction_Problem (update the Instruction Problem in the instance of 'instruction_problem_panel');
update_message (update message in the instance of 'message_panel').

Output task: remove an M-set (2.9).
Feedback task(s):
delete_M_element_canvas (remove an instance of 'M_element_canvas' from Screen);
delete_M_set_button_panel (remove an instance of 'M_set_button_panel' from Screen);
delete_M_set_label_panel (remove an instance of 'M_set_label_panel' from Screen).

Output task: remove M-elements from M-set (2.10).
Feedback task(s):
update_label (update the M-set-label in an instance of 'M_set_label_panel');
delete_M_elements (remove M-elements from an instance of 'M_element_canvas').

Output task: remove M-elements and display new ones in an M-set (2.11).
Feedback task(s):
add_M_elements (add new M-element(s) in an instance of 'M_element_canvas');
update_label (update the M-set-label in an instance of 'M_set_label_panel');
delete_M_elements (remove M-elements from an instance of 'M_element_canvas').

Output task: replace M-sets by one M-set (2.12).
Feedback task(s):
add_M_elements (add new M-element(s) in an instance of 'M_element_canvas');
update_label (update the M-set-label in an instance of 'M_set_label_panel');
delete_M_element_canvas (remove an instance of 'M_element_canvas' from Screen);
delete_M_set_button_panel (remove an instance of 'M_set_button_panel' from Screen);
delete_M_set_label_panel (remove an instance of 'M_set_label_panel' from Screen).

Output task: show solution correctness (2.13).
Feedback task(s): update_message (update message in the instance of 'message_panel').

Output task: pop-up the window with help information (3.1).
Feedback task(s): create_information_panel (create an instance of 'information_panel');
create_information_canvas (create an instance of 'information_canvas');
set_popup_panel (create button instance(s) in an instance of 'information_panel');
set_popup_text (display text in an instance of 'information_canvas').

Output task: remove the pop-up window with the help information (3.1).
Feedback task(s): delete_information_canvas (remove the instance of 'information_canvas' from Screen);
delete_information_panel (remove the instance of 'information_panel' from Screen).

SET RELATIONS

Relations between the three interaction sets of an M-set are needed. When, for example, an M-set must be removed from Screen because the user has clicked the 'verwijder'-button in an instance of 'M_set_button_panel' it must be determined which corresponding 'M_element_canvas' and 'M_set_label_panel' must be removed together with the instance of 'M_set_button_panel'.

In figure 6.20 our proposed set relations are depicted.

```
                M_set_button_panel
                /
             /   \   /
M_element_canvas  M_set_label_panel
```

*Figure 6.20: relations between sets of the Graphic Calculator*

The third substep of the design step of D2M2design is the coding of the semantic feedback functions (see section 4.4). However, the Graphic Calculator does not need semantic feedback functions: all information needed during execution of interaction tasks can be obtained from the user interface data. We therefore proceed with the fourth substep of the design step of D2M2design.

INTERACTION TASK ENABLING

It is decided that the flow in the application semantics code of the Graphic Calculator should be more complex than the flow in the Graphics Manager. As a consequence the Graphic Calculator contains relatively many types of coordinating decisions. It is therefore decided not to specify the types of coordinating decisions of the Graphic Calculator in this subsection. In appendix B the application semantics code of the Graphic Calculator is presented, including all
coordinating decisions.

A number of interesting issues can be noticed in the pseudocode presented in appendix B.

Firstly, like with the Graphics Manager the Graphic Calculator contains some interaction task enablings which can, for a certain coordinating decision, never be executed. An example is the enabling of "initiate stop_definitely", while the information window is not displayed on Screen.

Secondly, a number of CASE-entries does not contain computer tasks; complication III (images on screen (being part of the user interface) are what the computer tasks (being part of the application semantics) are intended to manipulate) manifests itself.

Thirdly, there is no occurrence of complication II (feedback task execution during computer task execution). In constructing the design specifications we did not find it necessary to include intermediate output tasks in the Graphic Calculator.

Fourthly, there is no occurrence of complication I (semantic feedback). In constructing the design specifications we did not find it necessary to include semantic feedback in the Graphic Calculator.

So far we discuss the design specifications of the Graphic Calculator. In the next section we discuss a major user interface extension of the Graphic Calculator.

6.4. MODIFICATIONS

The described version of the Graphic Calculator allows children to learn to add and subtract via manipulating either Coins or Cubes. The SLO felt that it should be possible to allow manipulations on an Abacus, too. In figure 6.21 an Abacus is depicted.

![Abacus](image)

*Figure 6.21: an Abacus*

This Abacus represents the value 3218.
In adding a third type of representation in the Graphic Calculator we assumed the following.

1. An M-set contains either Coins, Cubes or an Abacus.

2. An M-set label for the Abacus of figure 6.21 looks like: 3d 2h 1t 8e. 'd' stands for thousands, 'h' stands for hundreds, 't' stands for tens and 'e' stands for atomic units.

3. It is not possible to move an Abacus element from one M-set to another.


5. Subtracting can be initiated by selecting Abacus elements of one M-set followed by clicking the 'trek af'-button.

6. Exchanging Abacus elements in one M-set can be initiated by clicking the Abacus elements which must be exchanged followed by clicking the 'wissel'-button, followed by clicking the Abacus bar in which the selected Abacus elements must be exchanged.

The assumptions make it necessary to change the user interface design specifications. In contrast with the Graphics Manager, however, it does not prove necessary to change anything of the application semantics of the Graphic Calculator. The extension of the Graphic Calculator with Abacus manipulation can thus be described as easy (i.e. the changes of the user interface do not effect the application semantics). Details about the alterations of the Graphic Calculator are described by Versendaal (1991).

6.5. CONCLUSIONS

Clearly the Graphic Calculator is a HICS of class 3 (see chapter 2): semantic feedback and intermediate output task execution do not appear. The design and modification of the Graphic Calculator allow us to draw three conclusions.

1. D2M2environment is suitable for the development and maintenance of the Graphic Calculator. We were able to construct and modify design specifications using D2M2edit.

2. In the Graphic Calculator logical separation is fully achieved. Application semantics data is managed by the application semantics process and user interface data is managed by D2M2run.

3. In the described specification and modification of the Graphic Calculator physical separation is fully achieved. We were able to extend the user interface considerably without having to change the application semantics.

134
7. CONCLUSIONS

Considering the D2M2environment and the implemented HICSs we revisit the hypotheses and statements of chapter 2 and conclude the following.

1. It is possible to construct an alternative model which supports logical separation of the user interface and the application semantics, considering the user interface in a Direct Manipulation way.

The Delft model is based on a separation of user interface data and application semantics data. The user interface data, which are displayed as objects on screen, may be manipulated by the end user. The Direct Manipulation behaviour of user interface objects can be defined by user input task definitions. The Delft model contains a component which monitors the user interface data; the component is called the user interface monitor (or user interface process). Further, the Delft model contains a component which monitors the application semantics data; it is called the application semantics monitor (or application semantics process). Both processes communicate with one another during run-time of a HICS; neither process can consult or update each others data directly. Statement 1 is true.

We split hypothesis 2 into two subhypotheses: hypothesis 2a is dealing with logical separation, hypothesis 2b is dealing with physical separation.

2a. In such an alternative model semantic feedback during user input task execution does not limit logical separation.

2b. In such an alternative model semantic feedback during user input task execution does not limit physical separation.

The application semantics data that must be consulted during user input task execution remains in the application semantics zone. Semantic feedback functions, being part of the application semantics, consult the application semantics data. Semantic feedback function execution is triggered during run-time by the user interface process; the result of semantic feedback function execution is sent back to the user interface process. So, hypothesis 2a is upheld.

However, semantic feedback during user input task execution is a limiting factor of physical separation. Making changes concerning semantic feedback in the Graphics Manager HICS (see section 5.4) proved that both the user interface design specifications as well as the application semantics design specifications needed to be changed. So, D2M2environment does not support physical separation as for semantic feedback. Thus, hypothesis 2b cannot be upheld in D2M2environment.

We split hypothesis 3 into two subhypotheses: hypothesis 3a is dealing with logical separation, hypothesis 3b is dealing with physical separation.

3a. In such an alternative model output task execution during computer task execution does not limit logical separation.
3b. In such an alternative model output task execution during computer task execution does not limit physical separation.

Only the user interface data is by intermediate as well as not-intermediate output task execution consulted and updated. So, hypothesis 3a is upheld.

However, an intermediate output task is a limiting factor of physical separation. Making changes concerning intermediate output tasks in the Graphics Manager (see section 5.4) showed that both the user interface design specifications as well as the application semantics design specifications of the Graphics Manager needed to be changed. So, D2M2environment does not support physical separation as for intermediate output tasks. Thus, hypothesis 3b cannot be upheld in D2M2environment.

4. In such an alternative model computer tasks that only update the screen do not limit logical or physical separation.

Such computer tasks are treated as output tasks in D2M2environment. This enforces that execution of those computer tasks, really being execution of output tasks, can result in consulting and updating of user interface data.

Even physical separation seems to be fully achieved as for this complication III (see section 2.3). Making changes concerning occurrences of complication III in the Graphic Calculator (see section 6.4) did only affect changes in the user interface design specifications. So, D2M2environment seems to support logical as well as physical separation as for computer tasks which only update the screen. Therefore, hypothesis 4 is upheld.

5. It is possible to create an environment for an ICS-developer, which supports physical separation.

As long as semantic feedback and intermediate output tasks are not taken into account, statement 5 is true. However, if semantic feedback and/or intermediate output tasks are considered, statement 5 is false.

We decided that D2M2environment should consist of D2M2design, D2M2edit and D2M2run. D2M2design consists of methods and techniques to determine design specifications. Details of the methods and techniques that must be used are not discussed in this dissertation. The main steps of D2M2design are an analysis step to analyse the input to D2M2design and a design step to construct design specifications. D2M2edit is the tool to input and maintain specifications. Further, it can generate run-time specifications. Used notation techniques are data modeling notation techniques for e.g. the specification of user interface data and application semantics data, and transition diagrams for the specification of interaction tasks. D2M2run is the tool to interpret the user interface run-time specifications.

We now revisit statement 6.

6. It is possible to construct a run-time tool, based on the alternative model (i.e. Delft model).
D2M2run is the run-time tool that interprets user interface run-time specifications generated by D2M2edit. The application semantics process interprets the application semantics run-time specifications. Both processes, D2M2run and the application semantics process, communicate with each other during run-time. So, statement 6 is true.

We identify the following remaining questions.

1. What are, precisely, the methods and techniques of D2M2design?

Future research need to be performed in order to precisely identify methods and techniques. We only described globally what steps need to be taken in D2M2design in order to construct design specifications.

2. Is developing (H)ICSs in D2M2environment more efficient and effective than in other environments?

We only showed that it is possible to create HICSs in D2M2environment. In order to determine that using D2M2environment for HICS development is more effective and efficient, the methods and techniques of D2M2design should be precisely specified and experiments should be performed to compare development of HICSs with and without using D2M2environment. We note that the separation of the user interface and application semantics seems promising for the design of HICSs; it allows for the design of the user interface by dialogue specialists. E.g. Ten Hagen and Derksen (1985) considered this as recommendable (see also section 1.4).

3. Is maintaining (H)ICSs in D2M2environment more efficient and effective than in other environments?

We made it plausible that it is possible to 'easily' maintain HICSs of class 0 and class 3 (HICS with or without computer tasks that only concern the screen, see chapter 2). So, from class 0 and class 3 HICSs we can alter the user interface part without having to change the application semantics part; this gives reason to assume that maintaining those types of HICSs will be more efficient and effective in D2M2environment than in those environments in which separation of the user interface and application semantics is not or less supported. Experiments should support this assumption.
A. GRAPHICS MANAGER’S APPLICATION SEMANTICS CODE

A computer task is presented **boldly**, an "interaction task enabling" is presented in *italics* and a trigger of a feedback task is presented **underlined**. A pseudocode keyword is presented in **CAPITALS**.

```plaintext
initialisatie
create_stappencontroloset
FOR all instances of 'stelsel' DO
    create_grafiekbiedraamset
    create_legendaset
    create_grafiekcontroleset
    create_grafselsmeldingset
    init_toon_grafieken
    toon_legenda
END FOR

enable entry "initiate stop"
enable entry "initiate print_Coordinate_System"
enable entry "initiate create_new_System"
enable entry "initiate change_X_density"
enable entry "initiate change_Y_density"
enable entry "initiate display_or_hide_X_axis_values"
enable entry "initiate display_or_hide_Y_axis_values"
enable entry "initiate display_or_hide_X_axis_description"
enable entry "initiate display_or_hide_Y_axis_description"
enable entry "initiate rotate"
enable entry "initiate X_Coordinate_System_log_(un)scaling"
enable entry "initiate Y_Coordinate_System_log_(un)scaling"
enable entry "initiate display_or_hide_X_grid"
enable entry "initiate display_or_hide_Y_grid"
enable entry "initiate change_to_Area_Diagram"
enable entry "initiate change_to_Line_Diagram"
enable entry "initiate change_to_Dot_Diagram"
enable entry "initiate change_to_Bar_Diagram"
enable entry "initiate Smooth.Line_Diagram"
enable entry "initiate un-smooth.Line_Diagram"
// user input task "initiate (un)smooth.Line_Diagram" is split up
enable entry "initiate change_bar_width"
enable entry "initiate change_line_width"
enable entry "initiate change_dot_size"
enable entry "initiate change_bar_shadow_1"
// two types of bar shadowing are allowed
enable entry "initiate change_line_type_1"
enable entry "initiate change_line_type_2"
enable entry "initiate change_line_type_3"
// three line types are allowed
enable entry "initiate change_dot_type_1"
enable entry "initiate change_dot_type_2"
enable entry "initiate change_dot_type_3"
// three dot types are allowed
```

139
enable entry "initiate change_area_shadow_1"
enable entry "initiate change_area_shadow_2"
// two types of area shadowing are allowed
enable entry "initiate display_(Co-)Domain_Values"
enable entry "initiate hide_(Co-)Domain_Values"
// user input task "display_or_hide_(Co-)Domain_Values" is split up
enable entry "initiate delete_Coordinate_System"
enable entry "initiate display_or_hide_Table"
enable entry "initiate print_Table"
enable entry "initiate display_or_hide_Mean_Survey"
enable entry "initiate print_Mean_Survey"

WAIT FOR user interaction
WHILE (entry != "initiate stoppen") DO

CASE entry IS

"initiate create_new_System":
CALL semantic feedback function:
geldig_Domain_Interval
nieuwgrafstelsel (create a new 'stelsel')
create_grafiekendiagramset
create_legendset
create_grafiekcontroleset
create_grafstelselmeldingset
init_toon_grafieken (of the new FRAME 2)
toon_legenda (of the new FRAME 2)

"initiate display_or_hide_Mean_Survey":
IF the name of the grafiekendiagram was sent
THEN
pl_gemoverzicht (calculate Mean Values)
create_gemcontroleset
create_gemmeldingset
create_gemoverzichtset
toon_gemiddelden
END IF
ELSE
verwijder_gemcontroleset
verwijder_gemoverzichtset
verwijder_gemmeldingset
END ELSE

"initiate display_or_hide_X_grid":
set_horhulp

"initiate X_Coordinate_System_log_(un)scaling": set_horlog

"initiate display_or_hide_Table":
IF the name of the grafiekendiagram was sent
THEN
pl tabel (get Table Values)
create_tablecontroleset
create_tablekopregelsset
create_tablemeldingset
create_tableset
toon_tablekopregel

140
toon_tabel
END IF
ELSE
verwijder_tabelcontroleset
verwijder_tabelkopregelset
verwijder_tabelmeldingset
verwijder_tabelset
END ELSE

"initiate display_or_hide_Y_grid": set_verthulp

"initiate Y_Coordinate_System_log_(un)scaling": set_verlog

"initiate display_or_hide_X_axis_description": set_vasomschrijving

"initiate display_or_hide_X_axis_values": set_vaswaarden

"initiate display_or_hide_Y_axis_values": set_vasomschrijving

"initiate display_or_hide_Y_axis_values": set_vaswaarden

"initiate print Coordinate_System": printgrafstelsel (print a FRAME 2, contains toon_grfmelding, constituting intermediate feedback) toon_grfmelding

"initiate rotate": roteer

"initiate delete Coordinate_System": verwgrafstelsel (delete a ‘stelsel’) verwijder_grafiekcontroleset verwijder_grafiekendiagramset verwijder_grafiekstelselmeldingset verwijder_legendset
IF Table handle was sent THEN verwijder_tabelcontroleset verwijder_tabelkopregelset verwijder_tabelset verwijder_tabelmeldingset
END IF
IF Mean Survey handle was sent THEN verwijder_gemcontroleset verwijder_gemoverzichtset verwijder_gemmeldingset
END IF

"initiate change_X_density": set_kardichtheid

141
"initiate change_Y_density": set_vasdichtheid

"initiate change_area_shadow_1": verwijder_menu set_gebiedarcering change_graafiekef

"initiate change_area_shadow_2": verwijder_menu set_gebiedarcering change_graafiekef

"initiate change_to Area_Diagram": verwijder_menu set_gebieddiagram change_graafiekef

"initiate change_line_width": verwijder_sliderset set_linbreedte

"initiate change_to Line Diagram": verwijder_menu set_liindia diagram change_graafiekef

"initiate change_line_type_1": verwijder_menu set_liinoort change_graafiekef

"initiate change_line_type_2": verwijder_menu set_liinoort change_graafiekef

"initiate change_line_type_3": verwijder_menu set_liinoort change_graafiekef

"initiate change_to Dot_Diagram": verwijder_menu set_puntliindia diagram change_graafiekef

"initiate change_dot_size": verwijder_sliderset set_puntgrootte

"initiate change_dot_type_1": verwijder_menu set_puntsport change_graafiekef

"initiate change_dot_type_2": verwijder_menu
"initiate change_dot_type_3":
  set_puntaard
  change_grafiekdef

"initiate smooth_Line_Diagram":
  verwijder_menu
  set_smooth

"initiate change_bar_shadow_1":
  verwijder_menu
  set_stafarcering
  change_grafiekdef

"initiate change_bar_shadow_2":
  verwijder_menu
  set_stafarcering
  change_grafiekdef

"initiate change_bar_width":
  verwijder_slider_popup
  set_stafbreedte

"initiate change_to_Bar_Diagram":
  verwijder_menu
  change_stafdia
  change_grafiekdef

"initiate display_(Co-)Domain_Values":
  verwijder_menu
  set_toon_data

"initiate un-smooth_Line_Diagram":
  verwijder_menu
  set_smooth

"initiate hide_(Co-)Domain_Values":
  verwijder_menu
  set_toon_data

"initiate print_Table": print tabel (print a FRAME 5, contains
toon_tabmelding, constituting intermediate
feedback)
toon_tabmelding

"initiate print_Mean_Survey": printgemestelsel (print a FRAME 6, contains
toon_gemmelding, constituting intermediate
feedback)
toon_gemmelding

END CASE
enable entry "initiate stop"
enable entry "initiate print_Coordinate System"
enable entry "initiate create_new_System"
enable entry "initiate change_X_density"
enable entry "initiate change_Y_density"
enable entry "initiate display_or_hide_X_axis_values"
enable entry "initiate display_or_hide_Y_axis_values"
enable entry "initiate display_or_hide_X_axis_description"
enable entry "initiate display_or_hide_Y_axis_description"
enable entry "initiate rotate"
enable entry "initiate X_Coordinate_System_log_(un)scaling"
enable entry "initiate Y_Coordinate_System_log_(un)scaling"
enable entry "initiate display_or_hide_X_grid"
enable entry "initiate display_or_hide_Y_grid"
enable entry "initiate change_to_Area_Diagram"
enable entry "initiate change_to_Line_Diagram"
enable entry "initiate change_to_Dot_Diagram"
enable entry "initiate change_to_Bar_Diagram"
enable entry "initiate smooth_Line_Diagram"
enable entry "initiate un-smooth_Line_Diagram"
// user input task "initiate (un)smooth_Line_Diagram" is split up
enable entry "initiate change_bar_width"
enable entry "initiate change_line_width"
enable entry "initiate change_dot_size"
enable entry "initiate change_bar_shadow_1"
enable entry "initiate change_bar_shadow_2"
// two types of bar shadowing are allowed
enable entry "initiate change_line_type_1"
enable entry "initiate change_line_type_2"
enable entry "initiate change_line_type_3"
// three line types are allowed
enable entry "initiate change_dot_type_1"
enable entry "initiate change_dot_type_2"
enable entry "initiate change_dot_type_3"
// three dot types are allowed
enable entry "initiate change_area_shadow_1"
enable entry "initiate change_area_shadow_2"
// two types of area shadowing are allowed
enable entry "initiate display_(Co-)Domain_Values"
enable entry "initiate hide_(Co-)Domain_Values"
// user input task "display_or_hide_(Co-)Domain_Values" is split up
enable entry "initiate delete_Coordinate_System"
enable entry "initiate display_or_hide_Table"
enable entry "initiate print_Table"
enable entry "initiate display_or_hide_Mean_Survey"
enable entry "initiate print_Mean_Survey"

WAIT FOR user interaction

END WHILE

stoppen (delete all instances of all application semantics data types)
FOR all instances of 'grafiekcontroleset' DO
  verwijs grafiekcontroleset
  verwijs grafiekendiagramset
  verwijs grafatelgelmeldingset
  verwijs_legendaat
  IF an associated Table exists THEN

144
verwijder_tabelcontroleset
verwijder_tabellenregelset
verwijder_tabelset
verwijder_tabelmeldingset
ENDIF
IF an associated Mean Survey exists THEN
verwijder_gemcontroleset
verwijder_gemoverzichtset
verwijder_gemmeldingset
END IF
END FOR
B. GRAPHIC CALCULATOR’S APPLICATION SEMANTICS CODE

A computer task is presented **boldly**, an "interaction task enabling" is presented in *italics* and a trigger of a feedback task is presented **underlined**. A pseudocode keyword is presented in **CAPITALS**.

```
create_instruction_problem_panel
create_material_panel
create_representation_panel
create_action_panel
create_message_panel

enable entry "initiate stop"
enable entry "initiate construct_new_Instruction_Problem"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate give_help"

WAIT FOR user interaction

WHILE ((entry == "initiate give_help") OR
    (entry == "initiate remove_help") OR
    (entry == "initiate stop") OR
    (entry == "initiate stop_not_definitely")) DO

    CASE entry IS

    "initiate give_help":
      create_information_panel
      create_information_canvas
      set_popup_panel
      set_popup_text

      enable entry "initiate remove_help"

      WAIT FOR user interaction

    "initiate remove_help":
      delete_information_canvas
      delete_information_panel

      enable entry "initiate stop"
      enable entry "initiate construct_new_Instruction_Problem"
      enable entry "initiate get_old_M_set_of_M_sets"
      enable entry "initiate give_help"

      WAIT FOR user interaction

    "initiate stop":
      create_information_panel
      create_information_canvas
      set_popup_panel
      set_popup_text

```

147
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"

WAIT FOR user interaction

"initiate stop_not_definitely":
delete_information_canvas
delete_information_panel

enable entry "initiate stop"
enable entry "initiate construct_new Instruction_Problem"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate give_help"

WAIT FOR user interaction

END CASE

END WHILE

WHILE (entry != "stop_definitely") DO

CASE entry IS

"initiate stop":
create_information_panel
create_information_canvas
set_popup_panel
set_popup_text

enable entry "initiate stop_definitely"
enable entry "stop_not_definitely"

WAIT FOR user interaction

"initiate stop_not_definitely":
delete_information_canvas
delete_information_panel

enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"

148
enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate construct_new_instruction_Problem"

update_message

enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate generate_default_M_sets"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate generate_default_M_sets"

generate_default_M_sets

create_M_element_canvas
create_M_set_button_panel
create_M_set_label_panel
add_M_elements
update_label
create_M_element_canvas
create_M_set_button_panel
create_M_set_label_panel
add_M_elements
update_label

enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give help"
enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate change_representation":

update_material_choice
FOR all M-sets DO
delete M_elements
  add_M_elements
  update_label
END FOR

enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"

enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give help"

WAIT FOR user interaction

"initiate create_M_set":

create_M_set
create_M_element_canvas
create_M_set_button_panel
create_M_set_label_panel

enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate add_new_M_element":

  add_new_M_element
  add_M_elements
  update_label

  enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate move_M_element":

  move_M_element
  delete_M_elements
  update_label
  add_M_elements
  update_label

  enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate move_identical_M_elements":
move_identical_M_elements
delete_M_elements
update_label
add_M_elements
update_label

enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate clear":
clear
update_interaction_problem
update_message
FOR all M-sets DO
delete_M_element_canvas
delete_M_set_button_panel
delete_M_set_label_panel
END FOR

enable entry "initiate stop"
enable entry "initiate construct_new_instruction_problem"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate give_help"

WAIT FOR user interaction

WHILE ((entry == "initiate give_help") OR (entry == "initiate remove_help") OR (entry == "initiate stop") OR

152
(entry == "initiate stop_not_definitely") DO

CASE entry IS

"initiate give_help":
create_information_panel
create_information_canvas
set_popup_panel
set_popup_text

enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate remove_help":
delete_information_canvas
delete_information_panel

enable entry "initiate stop"
enable entry "initiate construct_new_Instruction_Problem"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate give_help"

WAIT FOR user interaction

"initiate stop":
create_information_panel
create_information_canvas
set_popup_panel
set_popup_text

enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"

WAIT FOR user interaction

"initiate stop_not_definitely":
delete_information_canvas
delete_information_panel

enable entry "initiate stop"
enable entry "initiate construct_new_Instruction_Problem"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate give_help"

WAIT FOR user interaction

END CASE

END WHILE

153
"initiate get_old_M_set_of_M_sets": get_old_M_set_of_M_sets
FOR all displayed M-sets DO
delete_M_element_canvas
delete_M_set_button_panel
delete_M_set_label_panel
END FOR
FOR all 'old' M-sets DO
create_M_element_canvas
create_M_set_button_panel
create_M_set_label_panel
add_M_elements
update_label
END FOR
update_material_choice
update_representation_panel
update_Instruction_Problem
update_message
enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"
WAIT FOR user interaction

"initiate remove_M_set": remove_M_set
delete_M_element_canvas
delete_M_set_button_panel
delete_M_set_label_panel
enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"

"initiate remove_M_elements":

remove_M_elements
delete_M_elements
update_label

enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate exchange_M_elements":

exchange_M_elements
delete_M_elements
add_M_elements
update_label

enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate join_M_sets":

join_M_sets
FOR all selected M-sets, except the last DO
delete_M_element_canvas
delete_M_set_button_panel
delete_M_set_label_panel
END FOR
add_M_elements
update_label

enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate check_solution":

check_solution
update_message

enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate give_help":
create_information_panel
create_information_canvas
set_popup_panel
set_popup_text

enable entry "initiate remove_help"

WAIT FOR user interaction

"initiate remove_help":
delete_information_canvas
delete_information_panel

enable entry "initiate stop"
enable entry "initiate stop_definitely"
enable entry "initiate stop_not_definitely"
enable entry "initiate change_representation"
enable entry "initiate create_M_set"
enable entry "initiate add_new_M_element"
enable entry "initiate move_M_element"
enable entry "initiate move_identical_M_elements"
enable entry "initiate clear"
enable entry "initiate get_old_M_set_of_M_sets"
enable entry "initiate remove_M_set"
enable entry "initiate remove_M_elements"
enable entry "initiate exchange_M_elements"
enable entry "initiate join_M_sets"
enable entry "initiate check_solution"
enable entry "initiate give_help"
enable entry "initiate remove_help"

WAIT FOR user interaction

END CASE

END WHILE

stop
delete_information_canvas
delete_information_panel
FOR all M-sets DO
delete_M_element_canvas
delete_M_set_button_panel
delete_M_set_label_panel
END FOR
delete_material_panel
delete_representation_panel
delete_instruction_problem_panel
delete_action_panel
delete_message_panel

157
REFERENCES


Goldberg, A. and Robson, D., Smalltalk-80, the Language and its Implementation, Addison-Wesley, 1983.


161

Potosnak, K., "Creating software that people can and will use", IEEE Software, pp 86-87, September 1987.


LIST OF PUBLICATIONS


Versendaal, J. and Van der Mast, C., Specifying applications for the Delft Direct Manipulation Manager (D2M2), Report 90-27, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, 1990.

165
CURRICULUM VITAE


Bij de Stichting EXIN is hij sinds maart 1990 lid van de examencommissie van de PDI-module "Digitale Systemen".
STELLINGEN

behorende bij het proefschrift

Separation of the User Interface and Application

Johan Versendaal
1 oktober 1991
De onduidelijkheid over de precieze definitie van de begrippen "lexicale feedback", "syntactische feedback" en "semantische feedback", zoals bijvoorbeeld aangegeven door Guedj (1980), kan worden weggenomen door deze begrippen binnen D2M2environment te beschouwen.


Ondanks het feit dat semantische feedback een wezenlijk onderdeel vormt van het Seeheim model, zoals beschreven door Green (1985), wordt deze feedback door anderen die dit model hanteren vaak niet in beschouwing genomen.


Ten onrechte beweren veel leveranciers van User Interface Management Syste- men dat scheiding tussen de gebruikersinterface en applikatiesemantiek volledig is bereikt.

Gebruikersvriendelijkheid is een gebruiksonvriendelijke term.
V

In plaats van de uitdrukking "What You See Is What You Get" (WYSIWYG) is in de meeste gevallen de uitdrukking "What You See Is What You Have Got" (WYSIWHG) te prefereren.

VI

Bij elke objekt-georiënteerde taal zouden reeds objectklassen moeten zijn gedefinieerd, die een programmeur ondersteunen bij het bouwen van gebruikersinterfaces.

VII

Bij objekt-georiënteerd ontwerpen is semantische datamodellering een uitstekend hulpmiddel.

VIII

Het scheiden van het privéleven en werk van een promovendus kan zeker zo lastig zijn als het scheiden van de gebruikersinterface en applicatiesemantiek van een hoog interactieve applicatie.