New Interface Primitives, extending the OSF/ Motif widget set.

Erik Borälv

CMD, Uppsala University
Lägerhyddvägen 18, S-752 37 Uppsala, Sweden
Telephone: +46 18 18 33 21
Email: Helios@cmd.uu.se, erikb@cmd.uu.se

Document number	47
Date	August 11, 1993
Product version:	1.0
Operating Systems and Versions:	Ultrix Version 4.3 and higher TeleUSE 2.0

This document describes interface primitives developed by CMD, Uppsala University.

AIM Project HELIOS-2 of the Commission of the EC (Project Number AIM 2015)

The HELIOS consortium is a consortium of seven University and Industrial partners created within the framework of the AIM (Advanced Informatics in Medicine) programme of the Commission of the European Communities. The consortium is involved in the development of medical software engineering tools and applications. Members of the consortium include:

Medical Informatics Department, Broussais University Hospital, Paris, France (Prime)
CAP Gemini Innovation, Grenoble, France
Deutsches Krebsforschungszentrum, Heidelberg, Germany
Geneva University Hospital, Geneva, Switzerland
Centre for Human Computer Studies, University of Uppsala, Sweden
Medical Informatics Department, University of Linköping, Sweden
Cooperative Engineering Centre, Digital Equipment B.V., Amsterdam, The Netherlands

The software described in this document is furnished under a licence and may be used or copied only in accordance with the term of such licence.

The information in this document is subject to change without notice and should not be construed as a commitment by the HELIOS Consortium. The HELIOS Consortium assumes no responsibility for any errors that may appear in this document.

DEC, DECstation, DECsystem, ULTRIX and VT are registered trademarks of Digital Equipment Corporation.

FrameMaker is a trademark of Frame Technology Corporation.

OSF, OSF/1, OSF/Motif and Motif are trademarks of the Open Software Foundation.

SUN is a registered trademark of SUN Microsystems, Inc.

TeleUSE is a registered trademark of TeleSoft, Inc.

Unix is a registered trademark of UNIX System Laboratories, Inc.

© HELIOS-2 Consortium 1993
All rights reserved

© HELIOS-2 Consortium 1993
All rights reserved

Contents

Preface

Objectives of the document

This document describes the work done in connection with Erik Borälv's work at CMD, Center for Human-Computer Studies at Uppsala University, in the year of 1993. The work described was made as a part of the studies in computing science for the degree of Master of Science

Audience

There is no intended audience.

Document structure

The *New Interface Primitives* document covers the following topics:

Part 1	Project background and general discussion. This part gives an introduction and background to the work within the Helios project.
Part 2	The part describing Helios in detail.

Acknowledgements

The Helios partners, Bengt Sandblad, Bengt Göransson, Eva Olsson, Mats Johnson, and the rest of the CMD staff.

Quote

"I think them wonderful you know, those golden moments when you get a chance to really sit back, (a little outside of It you follow), and watch the carrying on. I know you're not supposed to be cynical and that that is a killer and you should fight it, but really! I have to laugh, the shoutings an' a-feuding of the family, the frightening knowledge of utter hopelessness, the philistine knowledge mumblings of the pub crazed caveman, the greed-inspired treachery of the pound sign, the lifeless robots in the local social...

Oh! What a square circle to live in! But yes, so, unfortunately. You see they say let It flow and all will come right in the end, but I say depends which side of the golden handshake you're on. Clever right? So I have a mutual who says: 'Life is like a record. It has a start, a hookline, a bass solo and an end, proper ones do anyway.' And the boy wonder may have a point although I suppose you're thinking, 'what's to do with this record?' which is nothing and everything really as far as I'm concerned. Abstract in the extreme, I follow, but everyday I feel I have to tell my problems and type thoughts, mental or otherwise, to someone, though this record, I guess I should add swings! it really does and the reason is crystal to all those with heart(?) And anyway I've just thought that basically we're like cymbals in life's rich drumkit, sometimes we have to take the knocks to achieve effect. Well it was only a thought, a dawn thought." *The Cappuccino Kid*

1 Introduction

1.1 Software Development

1.1.1 Difficulties - in general

Software development is a complex process with very few simple paths towards a good solution. The cost and time consumption when building computer applications is in no way a linear process - with regard to the size and complexity of the target field - but rather an iterative process that evolves for several years, consuming much man-power and resources in an expensive way.

1.1.2 Difficulties - the medical domain

The difficulties above are particularly true for the medical domain, where the applications often are big, complex systems with special requirements - such as the end user’s (e.g. nurses, physicians) different and therefore possibly conflicting view of the problems to be solved, the need of hardware independent implementations and the integration of several standards and features in a distributed environment.

1.1.3 Solutions

Object orientated methods solve some of these problems as such techniques allow independent “plug-in’s” to be merged to form a complete system, rather than having to build a whole integrated system top-down.

2 HELIOS

2.1 Background

The Helios project is a research-project within the framework of the AIM (Advanced Informatics in Medicine) program of the CEC (Commission of the European Communities). [1][2]

2.2 Project Strategy

The motivation of the Helios project is to alleviate the development of medical application software. This is to be enabled by the development of a pre-industrial Software Engineering Environment[1] - especially dedicated to the medical world. The expected result of Helios is:

- the open and modular “medical SEE”, which will facilitate production of medical applications.
- a unifying mechanism used to integrate medical applications software.
- a demonstration of the usefulness of the implemented tools by porting a significant part of the Ward Information System (WIS).
- a second demonstration showing of the reusability and improvement of the medical applications - with regard to both cost and time consumption in the construction process.

1. A computer system that provides support for the development, repair and enhancement of programs, and for the management and control of these activities.

2.3 Partners

2.3.1 Paris

The Paris partner is the Broussais University Hospital's medical informatics department (BMID). BMID is responsible for the specification, and realization of the SEE kernel, but will mainly contribute to the development of the information system. Also, development tools and the medical documentation facility are involved in their work.

2.3.2 Grenoble

The Grenoble partner is Cap Gemini Innovation, a R&D company of the group Cap Gemini Sogeti - the largest European software house today. Cap will contribute to the object orientation part.

2.3.3 Amsterdam

The Amsterdam partner is Digital Equipment Corporation (DEC) and will be responsible for multimedia activities and composite document generation.

2.3.4 Genève

The Geneva partner is the Centre d'Informatique Hospitalière (CIH) of the Hôpital Cantonal Universitaire. CIH is developing the natural language processing tools, both implementation and integration within the SEE.

2.3.5 Heidelberg

The Heidelberg partner is the Deutsches Krebsforschungszentrum (DKFZ) - the German Cancer Research Center. DKFZ will participate in the image processing parts.

2.3.6 Linköping

The Linköping partner is the department of medical informatics at Linköping University and will concentrate on knowledge-based decision support.

2.3.7 Uppsala

The Uppsala partner is CMD, the Center of Human-Computer Studies at Uppsala University. CMD is responsible for the design of the interface, or rather the Style Guide, and the integration of the interface parts with the other Helios components.

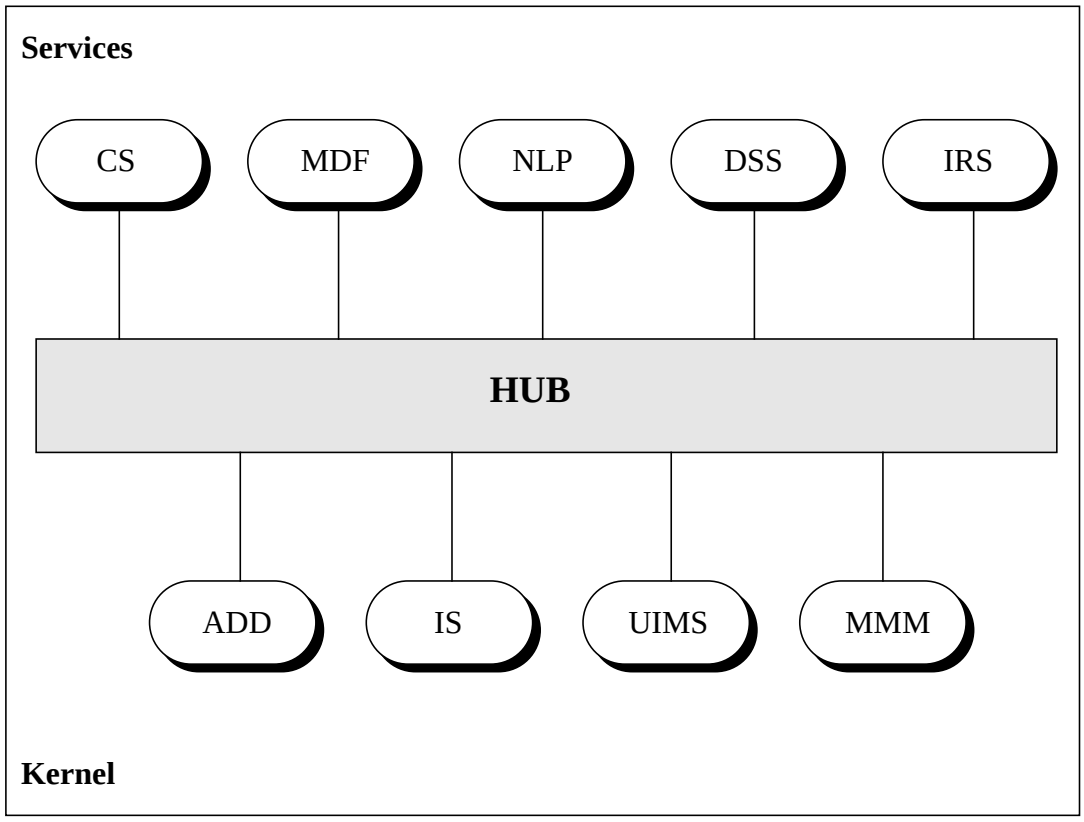
2.4 Method

Current medical applications are becoming more and more complex and tend to grow much in size, and cannot be realized by a single programmers team using traditional software development methods. It is therefore essential for the architecture to be able to benefit from other tools, well encapsulated and packaged. In fact, most of the new medical applications are widely distributed, reflecting the natural organization of their targets - and tools integration is recognized as the basement of future SEE's that aim to at providing cohesive support of the software life cycle.

The Helios' different parts will be implemented in a strict object oriented way to facilitate and uniform integration. The unifying mechanism is the Helios Unification Bus, a communication software bus used together with a well-defined communication language to which the plug-in's must conform in order to fit in the environment.

2.5 Modules

FIGURE 3–1 The top-down object design of the different modules within Helios.



2.5.1 Helios Unification Bus - HUB

This is the communicating part, integrating different parts of the medical application in a *dynamic* client-server fashion; the software components are considered as a *server*, when another component asks for one of its resources, and as a *client* when it asks another component for a service.

2.5.2 Connection Services - CS

It is very important for the project to be compliant with market and research standards, in both software engineering and medical fields. A special case of this is the communication and exchange field, where this "gateway" will enable future integrations.

2.5.3 Medical Documentation Facility - MDF

In the medical field, information updating is a daily activity. These activities require access to medical information resources whether it is a domain expert, the medical literature or several patient records stored in data bases.

The medical SEE must be able to handle functions to facilitate the construction of tools specialized in: knowledge management, "intelligent" browsing, documentation of medical decisions, easy update of medical data bases, etc.

2.5.4 Natural Language Processing - NLP

The goal is to build a domain specific language processor, based on the broad existing experience in this field. The main idea is to take advantage of the closed domain of knowledge. This could be obtained by formal grammars and parse-tree analysis, or semantic analysis of free text using what is called "proximity processing".

2.5.5 Decision Support System - DSS

An essential feature of the environment is to provide a mechanism for data-driven decision support. This part will conform to the Arden Syntax Medical Logic Form (MLM), allowing multiple users to create, criticize and share pieces of medical knowledge, thus taking advantage of the concurrent work in this field.

2.5.6 Image Related Services - IRS

One of the most informative parts of any medical environment is images, such as X-rays, ultra sound and positron emission tomography. In the past, all these data where handled separately from the medical records, as the

technology was not capable of handling this kind of integrated tasks. This is not the case nowadays however.

The IRS is responsible for the image facilities, with respect to handling, storing and processing the images. The image facilities also includes the following areas: image manipulation, 3D visualization and image analysis.

2.5.7 Analysis Design and Development - ADD

ADD will be built upon existing tools, and will cover the different phases of the software development life-cycle by providing methods and support for object-oriented analysis, design and programming.

Special attention will be paid to the software re-usability, and on greater productivity by streamlining the software development process.

2.5.8 Information System - IS

The IS, or the data base, will be pure object oriented, storing both the development data base and the operational data base used by the application end user.

2.5.9 User Interface Management System- UIMS

The UIMS will provide needed tools for a suitable and efficient interface building environment, including the Style Guide for design of the end-user interface.

The Style Guide is to be domain-specific, based both on an analysis of how information is used in the involved health care, and on a set of predefined design rules. Also, it will adopt to the existing standards regarding GUI's as far as possible. We will conform to the OSF/Motif Style Guide whenever possible.

2.5.10 Multi Media Manager - MMM

The MMM will provide the sound and video components.

3 Graphical User Interface

The design of an interface can be compared to the work of an architect when a building is designed and constructed [5]; the task is to design a building that is both functional, aesthetic and which can be constructed efficiently using existing techniques.

3.1 Domain dependency

The design of a user interface must always be based on an analysis of both the work situation, and of the end user working with the application. As the computer is merely a tool for the user, emphasis must be put on that the design and behaviour of the computer system is optimized for the thought work situation and its activities, and not for the computer use in itself. Such a design must be based on an analysis of how information is being used in the work context so that the interface will be “transparent” and the user can concentrate on the work activities. [12]

There are big differences in the way professionals use the computer compared to the novice, and therefore the requirements on the systems also differ - both in capabilities and kind of capabilities. What it takes to make the novice cope is quite well known and not very complex, but for the professional it is the very contrary; much is expected and little is known about how to make the professional cope. Optimizing a system, and specially the complex one’s found in the medical field, for daily professional use, is a task where not much is known and suitable working methods are few. Much of this is considered being solved only by skill of craftsmanship. The Helios Medical Style Guide is an attempt to overcome some of these problems, taking domain aspects into account when stating rules for the design. [9]

3.2 Interface Components

3.2.1 Lack of components

When making a design one may find that some data is difficult to present in a way that makes sense in the actual work situation, or that there is no suitable interface component that will do the job.

In the context of interface components “do the job” here denotes several things; presenting some sort of data in (i) a **relevant** way for the current domain, and (ii) in a way that makes the interpretation of it **immediate**.

Part (i) is a rather simple problem, but still requires some explanation. For instance, presenting data in a text field is very common, although it may not be relevant to present data ASCII encoded - often some sort of graphical presentation would be more accurate. This of course addresses the lack of suitable interface components in current engineering systems, plus the difficulties in making general enough components that also fit in very specialized domains. In addition, the components must also fulfil the requirements stated below. Much could be gained by implementing new, more task-dedicated components; there is certainly room for new ways to presenting data.

Part (ii) is more complex because the properties that make an interpretation fast or simple is often not known at all. The properties may perhaps even vary from domain to domain, and not be applicable in general. This statement is almost true; the perceptual properties (such as colour, movement and pattern) are of course general, but when I say properties here I mean in a higher level, e.g. how to present temperature or text - a task that may not have general properties in different domains. Some common tasks, such as deciding whether values are high/normal/low, could naturally be considered having general value but still it is not certain that different domains could share the same implementation.

There is a need for domain-specific interface elements as well as for elements on a high conceptual level. The demands for these elements is not obvious until one tries to actually specify how the end-users performs their daily work and what kind of support they need.

3.2.2 Composed components

Due to limitations in generality and space, the current Helios graphical environment[1] - or any other environment for that matter - can only support a limited set of interface primitives for the designer of interfaces.

As these primitives may not be sufficient for the desired design, one has to build own elements, composed of the existing graphical primitives. This leads to hand-crafted components that may, or may not, suite the current application - much code and maintenance must be put behind the composed components, a fact which might not be desired. Thus, the design and implementation of the desired component grows in size and it is therefore not of practical use. For such a component to be useful the functionality and behaviour of it must be automated within the component itself, and not be maintained explicitly by the designer. Maintaining two large complex structures, (i) the composed component part and (ii) the code managing control of flow, is most inefficient due to the **cross-dependencies** between these two parts. Changes in one of the parts often result in necessary non-trivial changes in the other.

However, if complex elements with highly adapted functionality is desired, composed components may be the only solution available. These drawbacks mentioned above are much avoided by the support of **templates** and the graphical editor **VIP** of TeleUSE.

Generally, it is not possible, due to time consumption and cost issues, to develop specialized primitives for a single task. When making interface primitives one has to bear in mind realities as **re-usability** and **time-cost** factors.

3.2.3 Primitives, Interface Elements and Forms

3.2.3.1 Component levels - abstractions

When designing interface components for computer screens one must define different levels, or **abstractions**, for the components in question. The abstractions span from the basal X primitives, such as lines, dots or widgets, to the most advanced ones - the final application.

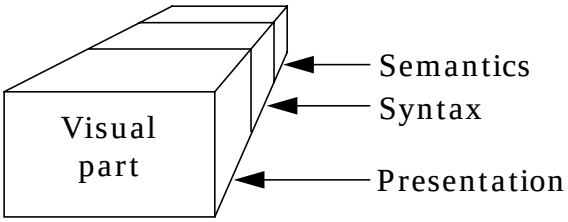
Rather than naming the different levels at this point I would first like to discuss them on an abstract level. One may bear in mind the purpose of making the abstractions match the defined concepts in the target domain. The level of concepts related must obviously, later at the stage of implementation, somehow have a corresponding “item” in the real world of interface components. The reasons for distinguishing different levels is obvious; e.g. a nurse does not relate to his/hers work in terms of radio buttons or labels, but rather in terms of medical records and prescriptions.

---

1. The Helios graphical environment consists of the standard X-Windows system and the OSF/Motif set of widgets.

- The lowest level of abstraction does not contain any other knowledge than about strict appearance. It consists therefore of a syntax level that handles graphical requests from its user, in this case the X level, and of the semantics that apply upon these requests.

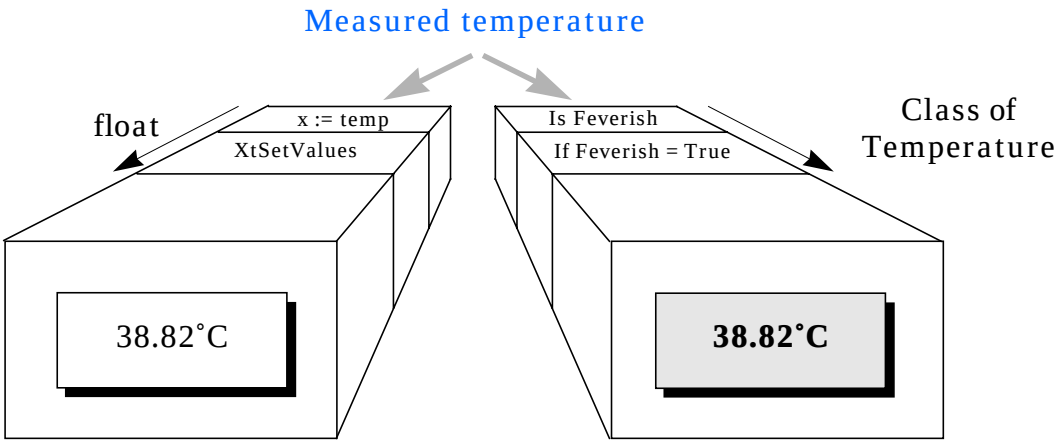
FIGURE 4–1: Interface component parts



- An imaginary next level contains a syntactic part on a higher level as it has got more of (semantic) knowledge. The knowledge may be of the kind able to handle an identity card with picture, name, address, etc.

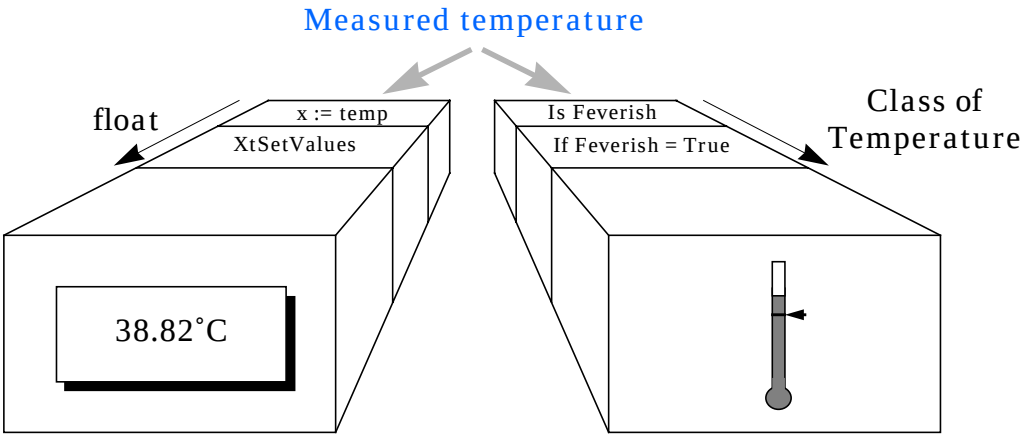
3.2.3.2 Examples

FIGURE 4–2: Two mappings of temperature



The temperature can be defined as a digital number, represented by a text field as above. But just the fact that it has a representation does not automatically make it suitable for the domain. The interpretation of the temperature, given a medical context, is several abstractions higher than that of digits in a text field. The box here on the left is just a text field and the box to the right is an interface component. The interface component can be coded to show more information as it contains more of domain knowledge. In this example the background and a bold typeface are used to help the user in the interpretation of the patient status.

FIGURE 4–3: Thermometer interface component



Here is maybe a better example. The thermometer is a specialized presentation component for displaying temperature, here used to, not only to show the degrees, but also if the patient is feverish or not. The thermometer component relates to the patients temperature much in the same way that for example a nurse would. We can say that the thermometer interface component reaches a much higher degree of abstraction or is more *domain related*.

3.2.4 Primitives, Interface Elements and Forms in Helios

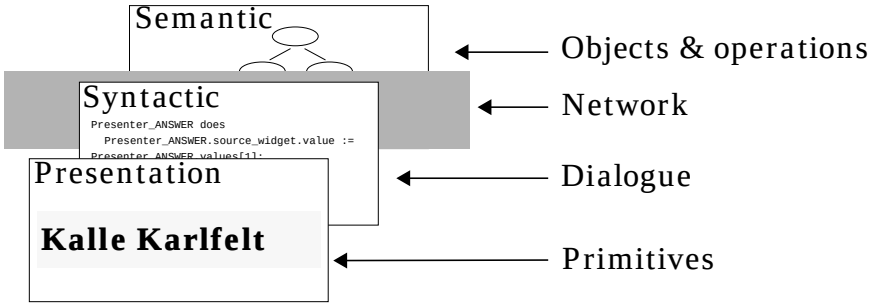
The visual objects in a Helios GUI can be divided into three categories: Primitives, Interface Elements (IEs) and Forms. The main difference between the three types is the functionality:

Primitives: In the Helios context, widgets or other “native” objects such as lines and rectangles. In Helios-SEE the OSF/Motif widget toolkit and customized widgets are available. Primitives are used for layout handling and formatting of the user interface.

IEs: An IE is dedicated to represent a certain object class. An IE has a visible presentation part, a **template**, and another part describing the connection to objects and operations.

Forms: A collection of designed IEs that covers a work task for the end user. A form can be placed within other forms or as a top level form in its own window. It is desirable to design the forms to fulfil all the user requirements to perform a complete work task.

FIGURE 4–4: The parts of an IE:



Semantic: Functionality; objects and operations, e.g. other services in the Helios-SEE.

Syntactic: Dialogue level with added external functionality.

Presentation: Output and input representation, primitives such as OSF/ Motif toolkit and other customized widgets.

In the figure above, the IE has a presentation part that is a Text widget, a syntactic part written in the dialogue language and a semantic part implemented in an operation somewhere in the Helios-SEE. The shadowed box implicates the network and the fact that the object can be located anywhere on the network.

4 Interface Elements

The need of new, better and more specialized interface components has always been a problem due to the fact that it is difficult to make an exact specification of the desired components, and far more difficult to implement them; the interface components has to be re-usable, and fit well to the existing environment.

4.1 Widgets

A solution to the above is what in X Window terms is called *widgets*, highly re-usable and portable components containing a complex data structure that combines an X window with a set of procedures that performs actions on the window. [7]

There are several collections of widgets available, including commercial and free libraries [4]. Unfortunately, there is not currently a sufficiently diverse set of widgets available for the wide variety of user interface needs. Additionally, some widgets have licensing costs and distribution restrictions that make them inappropriate for inclusion in certain software projects.

The task of making your own widgets is therefore appealing, but has several drawbacks:

- It is hard! X programming is not easy.
- The development process is an iterative one, and lasts in best case for years.
- A widget is never finished off, but requires maintenance for a long time.

4.2 Selected widgets

4.2.1 HtField - input/output with constraints

For many applications, there is a need to put constraints on input, e.g. when requesting the user for a phone number or date of birth. In these cases some validity checks must be made on the values entered, and by OSF/Motif means there is no way to handle this in the interface component itself. The means for defining constraints must obviously not be specific to a certain type of input, as general use and different target domains must be handled by the very same widget.

There are two different ways one might want the validity checks made on input:

Static	On whole input, i.e. a final check or verification.
Dynamic	Check as the user types, made character by character.

These two methods do conflict with each other in a semantic point of view, although it is possible to combine them. The dynamic checking is the most difficult one, as it is not easy to make a reasonable *semantic* check on data which is not complete. Some types of data do not allow this method to be applied at all.

For certain cases of input it is desirable to present some options for the user in a menu, or to simply indicate that there is a set of valid input to choose from. This is in PC-terms called a combo-box. [16]

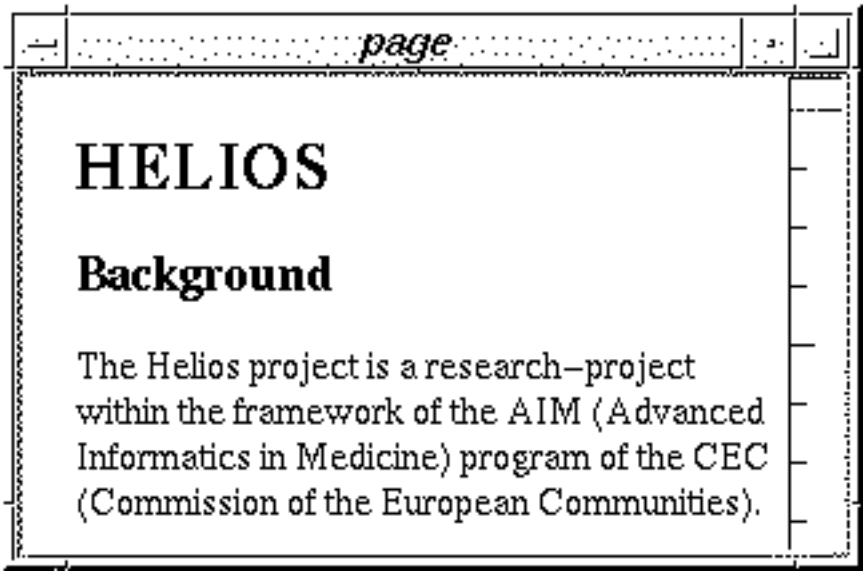
FIGURE 5–1 The CDE combo-box with an arrow showing the existence of a pre-defined set of possible input values.

4.2.2 HtPage - static document layout

How to present information that normally is paper bound is a non-trivial task. Since humans have the ability to fast get an overview and orientation of large amounts of data - given that the presentation is well known and carries this sort of orientational information - we must take advantage of this fact. Also, it is important that the built-in information is presented in such a way that encoding can be automated in a low cognitive level.

Exactly how to benefit from this in the best way is not known, but some principles may be listed. [9]

FIGURE 5–2 The HtPage containing eight pages showing the current page one.



4.2.2.1 Size

The form of presentation must (visually) contain information about its current size. This size property may then e.g. be used to identify and separate documents from each other. It is desirable if the size property is to be recognized immediately (or at a glance) as this help automate the use. For this to happen, what is called *secondary information* must be present - this has to do with spatial aspects, i.e. the appearance of information by pattern, color or some other interpretation.

4.2.2.2 Orientation

The user must be able to identify his/hers position within a document, and this must also be constructed in a way that allows immediate recognition. To enable this a static layout of the pages is essential [15], meaning that the data on a certain page is fixed in contents and may not change. A user may then recognize the page by the layout, a fact that is never possible when pages dynamically change the way they look - as when scrolling data for instance.

It is important for the efficiency of the work that as much as possible of these activities (size, orientation, recognition, etc.) can be automated.

4.2.2.3 Manoeuvring vs. Presentation

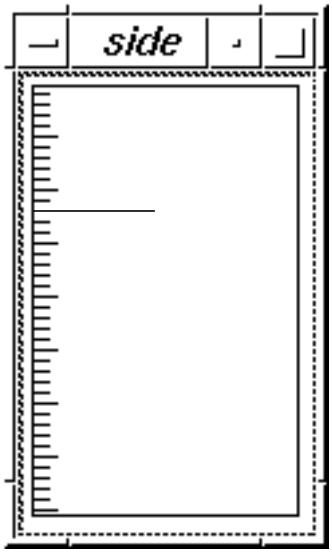
There is a potential conflict between optimizing the manoeuvring skills versus the presentation factors; a good presentation may not be very good at manoeuvring and vice versa. The trade-off must be made with respect to what kind and size of data that is to be handled. When dealing with large amounts of data one may emphasise ease-of-handling rather than possibly better displaying methods. How this trade-off is measured is beyond the scope of this document, but such a trade-off decision must clearly be based on an analysis of the target field containing an analysis of target components. Just guessing what the user needs, or what will work best, is deemed to fail in the long run.

It may not be obvious how the trade-off is to be measured; is it important to optimize the most frequent tasks, or is it vital that the same approach can be used in all possible occurring states? For instance, should the same method be used to navigate a document of two pages and one of several hundred pages? The importance may not lie in the expected good behaviour in all cases, but rather in merely coping at all in an acceptable fashion.

4.2.3 HtSide - a manoeuvring widget

The traditional **ScrollBar** is often used to display the amount of current data in some presentation and at the same time it is used for manoeuvring. To accomplish a more *static look* to the amount-displaying function, and to provide a *better manoeuvring* skills, the **HtSide** widget was implemented.

FIGURE 5–3 The HtSide widget containing 40 items where number 12 is selected.



4.2.3.1 Linear growth

When the data amount viewed increases (dynamically) the feedback indicator must increase in a linear way to avoid any dynamic effects to it's appearance.

Obviously, one could choose to have some data amounts encoded to a certain presentation to improve handling of small changes to the amount. For instance, you could define a couple of ranges, 0 - 10, 11 - 100, 100 - 1000, and have them all have a more or less separate look and behaviour. In this way a more optimized situation can be obtained in each case, but would suffer in consistency and possibly cause problems when switching between the defined ranges.

4.2.3.2 Size presentation

To be frank, the usual **ScollBar** has got an awful way of presenting data amount; the more data, the smaller the indicator... The **HtSide** represents each "item of data" with a line. Thus, if the data amount increases, so does also the indicator.

There is also an attempt to indicate size in a *ruler-style*; every fifth line is a bit longer than the others.

4.2.3.3 Manoeuvring

As the **HtSide** widget tries to represent each held data item with an own line, it is possible to directly select the desired item. I case of too much data to distinguish between the items, it is still likely the user could select an item close to the desired one. The selection is done by clicking with the leftmost key on the mouse, or, using defined keys on the keyboard such as **Prev** and **Next**.

4.3 TeleUSE

The **TeleUSE** system is based on the **OSF/Motif** standard set of widgets. As this set of widgets is not sufficient for Helios applications, the TeleUSE system has to be extended with additional functionality.

4.3.1 Customizing TeleUSE

The task of customizing TeleUSE is one of adding widgets [8], which requires in-depth knowledge of the widgets to add, experience in C and the X Window system, knowledge about include files and data types in X, and experience using the TeleUSE system. The document *HT-distribution* describes this process in detail. [13]

4.3.1.1 tu_mkmf

`tu_mkmf` is a script that makes it possible to extend the TeleUSE predefined set of widgets in a *semi-automated* fashion. The script is hand-tailored for the Helios project and its environment [11]. The script performs several non-trivial steps in order to be able to add several widgets at the same time to TeleUSE.

It also has the capability to build a portable package of all files, links and sources needed for an installation at some other location. An installing script is generated to ease installation.

4.3.1.2 extract

`extract` is a text parsing program that extracts information about a widget from the widget source file, called *WidgetName.c*, and builds a **Widget Interface Description** file, called *WidgetName.wid*. This wid-file is an important part when extending TeleUSE. [10]

4.3.2 Added widgets

For a more detailed description of the widgets below, see separate widget documentation. [17]

4.3.2.1 Ghost

Ghost is a widget that displays postscript files, using a user specified postscript server. The default server is the wide-spread Ghostscript server **gs**.

4.3.2.2 Layout

The **XmtLayout** widget is a general-purpose manager widget. It uses constraint resources to provide a dynamic interface for positioning child widgets, and also parses a simple layout grammar which describes the widget layout with a single string resource.

4.3.2.3 Paged

The **Paged** widget is a container widget that behaves like a “pile of pages”.

4.3.2.4 Pic

The **Pic** widget is used for displaying images in the format .pic - a format designed and implemented by the *Heidelberg* partner.

4.3.2.5 Resize

Resize allows automatic resizing of its child widgets.

4.3.2.6 Vcr

Vcr is an extended push-button. It has the additional look and behaviour of real-world vcr buttons.

4.3.2.7 Xbae

XbaeMatrix is a widget that presents an editable array of string data in a, possible scrollable, spreadsheet-like grid format.

4.3.2.8 XmtHelpBox

The **XmtHelpBox** displays a *multi-line*, *multi-font* message in a scrolled window. It displays a specified pixmap in the upper left corner of the widget, and displays a title flush-right above the help text. It positions an “okay” button below the help text so that the user can dismiss the XmtHelpBox when it is used as a dialogue box.

4.3.2.9 Mpeg

Mpeg is a version of the MPEG player from the *Berkeley Plateau Research Group* as a widget. It can be used either as a Motif widget subclassed from **XmPrimitive** or as a toolkit-independent widget subclassed from **Core**.

5 References

[1] **Programme AIM** (section 2), *Hospital Object Software Tools*. Technical/Management proposal.

[2] **Programme AIM** (section 3), *Participants' roles and qualifications*.

[3] **CMD progress report**, *Datoranvändning i arbetslivet*. CMD, Uppsala University.

[4] **Free Widget Foundation** (FWF), *Information about the FWF.*

[5] **comp.software-eng** (article 12427), *If architects had to work like programmers*. NCSA Mosaic access is "file://dse.doc.ic.ac.uk/requirements/renl4".

[6] The document **/pub/FWF/README** is available by anonymous ftp at `a.cs.uiuc.edu`.

[7] **X Window System** - *Programming and applications with Xt (OSF/MOTIF Edition)*. Douglas A. Young.

[8] **Adding widgets** - *Programmer's Guide*, TeleUSE.

[9] **Helios**, *Domain specific style guides - Design and implementation*, CMD

[10] **Helios**, *Extract - a WID generator*, CMD.

[11] **Helios**, *tu_mkmf - a VIP building script*, CMD.

[12] **Helios**, *Style Guide*, CMD.

[13] **Helios**, *HT-Distribution*, CMD.

[14] **Helios**, *HtSide*, CMD.

[15] **Helios**, *HtPage*, CMD.

[16] **Helios**, *HtCombo*, CMD.

[17] **Helios**, widget documentation, CMD:
HtField
HtPage
XbaeMatrix
Ghost
Layout
HelpBox
InputField
Mpeg

List of Figures

Index