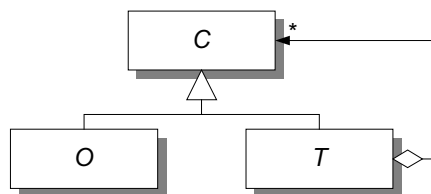


# *Development of a Component for Searching and Filtering*

*COT/3-20-V1.1*



Centre for Object Technology

Revision history:	v0.1	11-11-1998	Initial version
	v0.9	17-03-1999	Version for review
	v1.0	25-03-1999	Internal case 3 review comments incorporated
	v1.1	17-05-1999	COT review comments incorporated

Author(s): Henrik Røn, Systematic Software Engineering A/S

Status: Final

Publication: Public

Summary:

This report is an experience report on the analysis, design, and implementation of a reusable search component in a real life project. The document describes the experiences and lessons learned with the used techniques and technology, i.e., use cases, group work, design and architectural patterns and a meta-model.

The search component was constructed considering requirements from another project that is scheduled to reuse the component in the near future.

The main hypothesis of the project is that using a few extra resources, for taking requirements from other potential reusers into consideration, will ease reuse. However this report only describes the construction of the reusable asset and the reusable asset itself, not the reuse in a second project. This will follow in a subsequent report.

Furthermore, the overall design of the component is described.

© Copyright 1998, 1999 Systematic Software Engineering A/S

# Contents

<b>1. INTRODUCTION .....</b>	<b>4</b>
1.1 PURPOSE OF THIS DOCUMENT .....	4
1.2 EXECUTIVE SUMMARY .....	4
1.3 PILOT PROJECT SCHEDULE .....	5
1.4 WORK METHOD .....	5
1.5 OUTLINE OF THE REPORT .....	6
1.6 IRIS/MFS .....	6
1.7 EWARE .....	7
1.8 POSSIBLE COMPONENT TECHNOLOGIES .....	8
<b>2. OVERALL PROJECT GOALS .....</b>	<b>9</b>
2.1 PROJECT RISKS .....	9
<b>3. SEARCHING AND FILTERING .....</b>	<b>9</b>
<b>4. CUSTOMER REQUIREMENTS .....</b>	<b>10</b>
4.1 IRIS/MFS DATA MODEL .....	10
4.2 EWARE DATA MODEL .....	10
<b>5. ANALYSIS .....</b>	<b>11</b>
5.1 WORK METHOD .....	11
5.2 THE USE CASES .....	12
5.3 EXAMPLE OF USE CASE: DEFINE FILTER .....	13
5.4 EXPERIENCES AND LESSONS LEARNED .....	14
<b>6. DESIGN .....</b>	<b>14</b>
6.1 WORK METHOD .....	14
6.2 GUI DESIGN .....	15
6.2.1 Filter Book .....	15
6.2.2 Filter Editor .....	16
6.2.3 Search Window .....	18
6.3 SYSTEM DESIGN .....	19
6.4 DETAILED DESIGN .....	22
6.5 LEVEL OF ABSTRACTION DURING THE DESIGN .....	22
6.6 ILLUSTRATING INTERACTIONS BETWEEN COMPONENTS .....	23
6.7 LESSONS LEARNED AND OPEN ISSUES .....	25
<b>7. IMPLEMENTATION .....</b>	<b>26</b>
7.1 THE CLASSES IMPLEMENTED .....	27
7.2 LESSONS LEARNED .....	28
7.3 EXPERIENCES WITH PATTERNS .....	28
<b>8. RESTRUCTURING OF THE CODE .....</b>	<b>28</b>
<b>9. PHASE TWO: REUSING IN IRIS/MFS .....</b>	<b>29</b>
<b>10. PHASE THREE: REUSING IN EWARE .....</b>	<b>30</b>
<b>11. CONCLUSION .....</b>	<b>30</b>
<b>12. REFERENCES .....</b>	<b>31</b>
<b>A. ABBREVIATIONS .....</b>	<b>33</b>

# **1. Introduction**

This document describes phase one and two of a project carried out as a pilot project when Systematic Software Engineering A/S, hereafter referred to as SSE, joined the Centre for Object Technology. The pilot project was the design of a new search and filtering mechanism for IRIS/MFS, a SSE product, which is described below. The overall goal was to design a reusable asset with low context dependency, which should be reused partly or in whole in other SSE projects and products.

## **1.1 Purpose of this Document**

The purpose of the document is to describe lessons learned and observations made during the process of analysing, design, and implementing (and restructuring) a component for searching and filtering in a SSE product, IRIS/MFS version 2.2, i.e., phase one of the pilot project. The next phase was reusing the search and filter component in IRIS/MFS 3.0.

The report describes experiences, lessons learned, and observations combined with a description of the product, i.e., the reusable search component. Various aspects of the search and filter component will be described, but only at an abstract level as a detailed description is outside the scope of this report.

The target group is system developers experienced in object technology. Some knowledge about software patterns is preferable, but the document is also accessible to developers, project managers or managers without this knowledge. The reader will learn about experiences with developing software with an eye on reuse and learn about application of design and architectural patterns.

## **1.2 Executive Summary**

The hypothesis of the pilot project is that spending a little additional effort during analysis and design in form of having potential reusers present at the analysis and design meetings will ease the reuse as the requirements for the reusing projects have been taken into consideration.

The hypothesis has not been tested yet, but it will be in phase three.

Although no specific component technology was used, we tried to keep the design on an interface level and thus achieved a high degree of loose coupling.

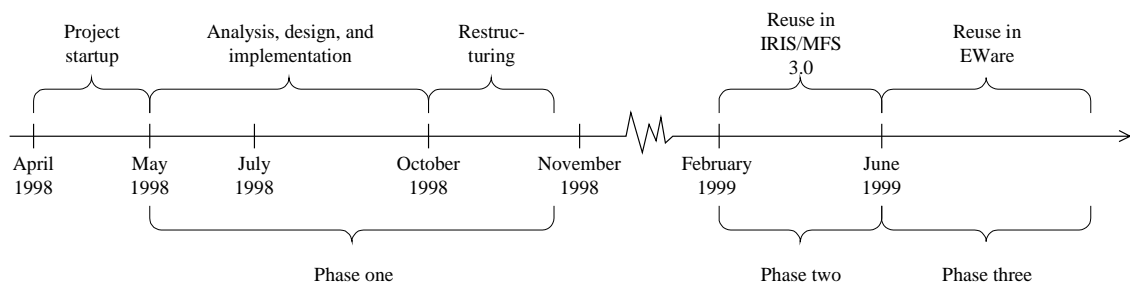
Design patterns were used extensively and successfully during the design phase and their use ensured low coupling, high cohesion and a sensible architecture, which was stable through the entire implementation phase. The architecture is fairly simple when one only sees the high level sub-components.

Furthermore a meta-model was used for uncoupling the search and filter component from the data on which it operates. This should hopefully ease reuse of the search and filter component in future projects, as it is independent and unaware of the domain data.

During the writing of this report it was discovered that the meta-model also is an architectural pattern as described in Section 6.3.

## 1.3 Pilot Project Schedule

Phase one is the analysis, design, and implementation of the search and filter component and initial use in IRIS/MFS version 2.2. The term “component” is used in the sense of a reusable chunk of code with low context dependency that supplies some form of functionality, not a (binary) component in the sense of COM or JavaBeans. Phase two of the pilot project is reusing the component for new functionality scheduled for version 3.0 of IRIS/MFS. Phase three will be reusing the component in another SSE product EWare in the summer and autumn of 1999. Phase one started mid April 1998 and was finished mid October 1998. Phase two started February 1999 and will be completed early June 1999. Phase three is scheduled to start early June 1999.



**Figure 1 - Project Timeline**

It is planned that the search and filter component will be made totally independent of IRIS/MFS after phase two and will be maintained by a person independent of the IRIS/MFS project team. Therefore neither developers from the IRIS/MFS project or from the EWare project are allowed to fix bugs or in any way modify the code that constitutes the reusable search and filter component.

EWare is scheduled to reuse the search and filter component as a white-box framework by adding sub-classes. As part of phase three the search and filter component will be documented using a special template for documenting reusable code. This template will be developed during phase three. The emphasis is on code reuse without cloning, i.e., all reusers will use the same code and benefit from all bug fixes made. However the reuse of the design for implementing a similar search and filter component in, e.g., Java is also a possibility, but it is not the goal of this project to reuse the design. Basic reuse terms are explained in [Goldberg and Rubin 95] and [Jacobson et al. 97].

## 1.4 Work Method

A number of people were involved in the pilot project, i.e., two SSE employees and participants from DTI and AU. The actual participants were:

- Peter Andersen, AU
- Ole Lehrmann Madsen, AU
- Peter Ørbæk, AU

- René Elmstrøm, DTI
- Kristian Lippert, DTI
- Thomas J. Hansen, SSE
- Christian Jeppesen, SSE
- Henrik Røn, SSE

The work group met at regular intervals for work meetings and one developer refined the result of these meeting in between. In the following the use of “we” or “our” relates to the people present at these meetings. Not all participants were present at the meetings. There were usually 5-6 persons present, and four of the eight participants constituted the core of the work group present at all meetings.

The external participants, i.e., the participants from AU and DTI, did not have the role of mentor or consultant, as all involved participated on the same footing and everybody contributed to the work in all phases.

## 1.5 Outline of the Report

The outline of the report basically follows the process during phases one and two. The rest of Section 1 describes the two domains in which the search and filter component at least will be used and why a specific component technology was not chosen. This is followed by Section 2, which contains a short description of the project goals and description of the risks when doing research within a project with a hard deadline. Section 3 contains a definition of the terms filtering and searching and is followed by Section 4, which describes the customer requirements for IRIS/MFS and EWare.

The description of the process starts in Section 5 in which the analysis is described followed by Section 6 describing the design of the search and filter component. Then in Section 7 the implementation is described, while Section 8 describes the restructuring of the code that happened after the delivery to the customer and this concludes the description of phase one.

Phase two is described in Section 9 and the plans for phase three are described in Section 10. The conclusion is in Section 11 and references can be found in Section 12. Abbreviations that are used throughout this document are in Appendix A.

## 1.6 IRIS/MFS

IRIS/MFS is a commercial-of-the-shelf (COTS) product for handling formatted messages in the defence community. A simplification would be viewing IRIS/MFS as an e-mail tool for military messages.

Formatted messages can be prepared, validated and distributed according to standards and rules.

As several standards exist for formatting both the message “envelope” and the message “text” there is also a problem of interoperability between different formats. Furthermore messages can be sent from and to many different sources as shown in Figure 2 and this also raises issue of interoperability.

The main focus of IRIS/MFS is to:

- Help the user prepare, validate, and manage formatted messages.
- Solve the problems arising when addressing both kinds of interoperability described above.

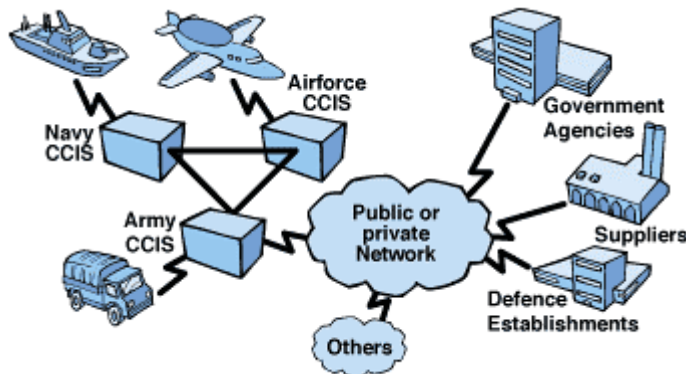


Figure 2 - Senders and recipients of formatted messages

Similarly to many e-mail tools, e.g., Microsoft Outlook™, the IRIS/MFS main window, called the “Repository”, consists of a number of folders and a view of the messages in the selected folder.

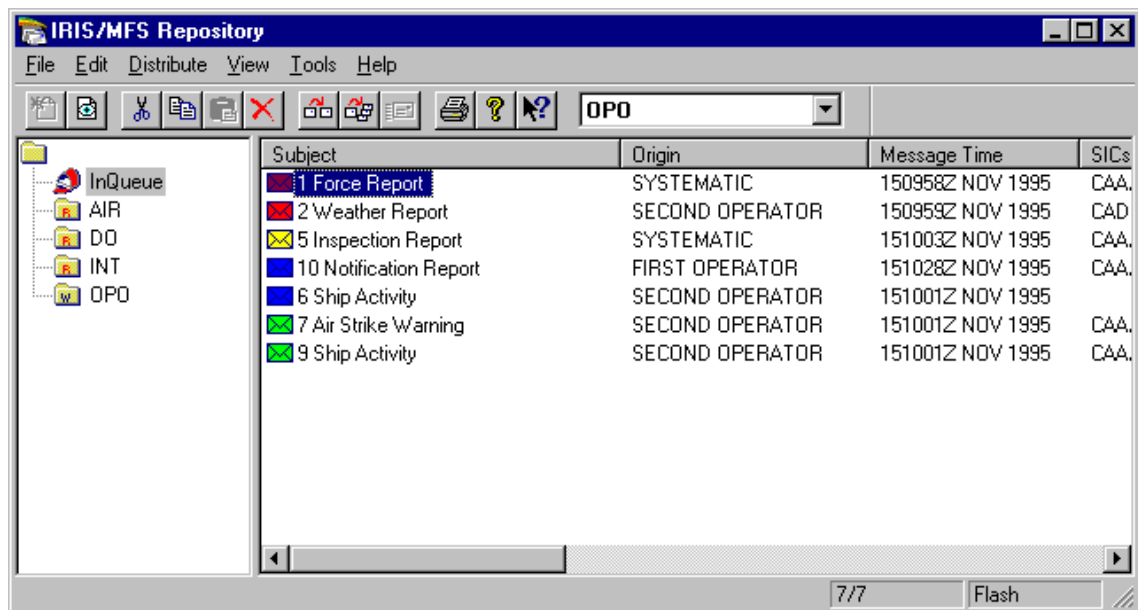


Figure 3 - IRIS/MFS repository

## 1.7 EWare

EWare is another SSE COTS product consisting of several independent tools for the management of Electronic Warfare (EW) information located in a relational database. Intuitively, EW information consists of electromagnetic signatures, e.g., radar signatures. Each type of emitter (radar) has a unique signature consisting of several measured values, e.g., scan and frequency.

When a military unit, e.g., a ship, is sent on a mission it is fitted with a library containing a subset of the information in the database. The EW information in the library is selected so that it only contains information for threats that can be found in the part of the world where the mission takes place. When the ship is on the mission it “listens” for electromagnetic signatures and can then use the library to identify the type of radar with this specific signature. By identifying the type of radar on a possible threat (missile, aircraft, ship, etc.), one can deduce what other kinds of threat one possibly is dealing with.

The EWare product provides support to EW analysts in preparing such an EW library for operational use and maintaining large collections of EW information in the database.

## 1.8 Possible Component Technologies

When the pilot project was initiated it was decided not to use any component technologies, such as COM/DCOM, JavaBeans, or CORBA. There are at least two aspects of IRIS/MFS that make it very difficult to use any of these technologies:

- It is available on multiple platforms, which include several different UNIX versions and Wintel platforms.
- It is implemented in C/C++, which excludes Java Beans for reasons stated below.

EWare has a similar aspect:

- It is also implemented entirely in C++.
- Runs on a Solaris UNIX platform.

The multi-platform aspect eliminates COM/DCOM as there only exists one implementation on UNIX, which has recently been introduced on the market and it is very expensive. Furthermore SSE did not want to tie itself to a sole vendor. Having said this we tried to use a COM like approach to design, where we considered which parts of a component’s interfaces other components would use. This is, however, an observation made in retrospective, but this was nonetheless what we did.

The C/C++ implementation of IRIS/MFS eliminated JavaBeans for performance reasons. Studies limited to the JDK documentation and monitoring the performance of Java program led to the conclusion that JavaBeans could be used in the following way:

1. IRIS/MFS starts a virtual machine using methods present in Sun’s JDK™.
2. Whenever IRIS/MFS should call the methods of the search and filter bean it would use the methods of the virtual machine created under 1.
3. The communication from the search and filter bean to the rest of IRIS/MFS could be done using Java’s native interface.

However Java Beans were eliminated for the following reasons:

1. Starting a virtual machine would increase memory consumption drastically.
2. From a maintenance point of view it is not very practical to have an application consisting of parts written in two very different programming languages.

CORBA was not used directly as it was considered to be an over-kill to restructure large parts of IRIS/MFS in order to produce a reusable component. Furthermore all reusers of the component would to some degree have to use CORBA, a solution that is not feasible, as this would potentially mean using CORBA in all SSE projects.

## 2. Overall Project Goals

The pilot project had three main goals from SSE's point of view:

1. Development of improved search and filtering capabilities in IRIS/MFS.
2. Development of a generic component for implementation of search and filtering facilities in other SSE products and projects.
3. Use the component to re-engineer search in EWare.

The IRIS/MFS developers in co-operation with the EWare project manager did the analysis and design, while the implementation was done solely by IRIS/MFS developers.

### 2.1 Project Risks

Intertwining a research project and a real life project with a strict deadline is a risky decision as it invariably will be the research project that will suffer in the event that something unforeseen happens. In our case the intertwining was that the research project was done within a real world project and the unforeseen event was that the delivery deadline for IRIS/MFS was moved from November 1<sup>st</sup> to October 1<sup>st</sup> by SSE's upper management. The result was that there was almost no communication between the external participants and the SSE during the implementation because of time pressure.

## 3. Searching and Filtering

This section will define the terms used in connection with filtering and searching. Basically filtering is searching combined with the ability to specify action that should occur if an entity passes a filter. These actions can be used for many different tasks, for example distribution, e.g., moving a message to a folder, and notification, e.g., opening a pop-up window. The terms "filtering" and "searching" are defined in the following way:

- **Searching.** The task of searching consists of two steps:
  1. Specification of the criteria, i.e., the rules describing the entities searched for. This is also called the search query. An example could be searching for e-mail messages where the Sender is 'René Elmstrøm' and he sent the mail to the address 'cot-all@cit.dk'.
  2. Specification of the data to be searched. In the example this could be 'Inbox' or the entire 'Mailbox'.
- **Filtering.** the task of filtering consists of four steps:

1. Specification of the criteria, i.e., the rules describing the entities passing the filter. Again the Sender is 'René Elmstrøm' and he sent the mail to the address 'cot-all@cit.dk'
2. Specification of which entities that should be filtered. Again this could be a folder in a mail folder hierarchy or the incoming messages.
3. Specification of the events and actions that occur if an entity passes the filter. This could be that the subject of the messages matching the filter is displayed using red as the font colour.
4. Execution of these events and actions when an entity actually passes through the filter. When a message passes the filter the font colour of the subject changes to red.

In the above list the items 1. and 2. under searching and filtering are identical.

“Condition” or “criteria” are used as synonyms for the rules that entities are checked against. Many different formats can be used for presenting criteria, e.g., logical expression on the form  $((\text{Sender} = \text{'René Elmstrøm'}) \wedge (\text{To} = \text{'cot-all@cit.dk'}))$  or in “natural” language: *Sender is 'René Elmstrøm' AND Receiver is 'cot-all@cit.dk'*.

Even though the tasks of searching and filtering, as noted above, are closely related, most COTS products distinguish between the two when it comes to the user interface.

## 4. Customer Requirements

The initial requirements for the search and filter facilities were dictated by a contract between SSE and one of its customers.

The initial requirements with the biggest impact on the design were:

- **Named criteria.** The user must be able to name criteria and make them persistent (save them).
- **Recursive search.** Searching in the result of the previous search.
- **Logical Operators.** Combination of queries using logical operators, i.e., “and, “or” and “not”.
- **Wildcards.** When searching or filtering on a string it must be possible to use wildcards, e.g., “\*” for zero or more characters or “?” for exactly one character.

### 4.1 IRIS/MFS Data Model

The IRIS/MFS data model is a very simple data model consisting only of one object: a message. The message has a number of attributes like originator, subject, etc.

### 4.2 EWare Data Model

The search facility in EWare is not only able to search in the data for emitters, but also in the data for the possible threats on which the emitters are fitted. An example could be an emitter with signature *x*. It is possible to find out which threats have such an emitter mounted by specifying a query like “All threats *y* that have an emitter *x*”. The search

does not only operate on attributes of an object, but also on the relations between an object and other objects, which in turn may have relations to other objects and so on.

A typical example is an aircraft carrier, which has its own radars and furthermore carries, e.g., cruise missiles, surface-to-air missiles, and aircrafts. The aircrafts also have, e.g., radar, air-to-air missiles, and air-to-surface missiles attached. All these entities most likely have electromagnetic signatures.

An example of usage of the library could be that a ship picks up the signature of an emitter mounted on a missile that is heading towards it. From the signature of the radar the ship's crew can determine which kind of missile it is by using the library and take countermeasures. Furthermore the can use the library to determine which types of aircrafts have this type of missile fitted and then again which aircraft carriers carry this type of aircrafts. This way a lot of information can be obtained from one intercepted emitter signal.

Internally in EWare a meta-model is used for describing the data on which the EWare applications operate, i.e., the data stored in the relational database. The meta-model maps between the object-oriented model used in the EWare application and the relational data model. When EWare initially was developed it was not an option to use an object-oriented database, as the customer demanded that an Oracle™ database should be used, therefore the meta-model approach was used.

## 5. Analysis

The analysis phase consisted of two sub-phases:

1. **Survey.** A survey of searching and filtering in existing COTS products as Outlook™, Netscape™, EWare™, IRIS/MFS™, Timeline™, and Internet Explorer™.
2. **Use cases.** The use cases described (and clarified) how we imagined that the users of IRIS/MFS would use the functionality of the search and filter component. We employed use cases, as one of their benefits is that they force the developer into thinking in more or less concrete scenarios. This was useful for two reasons:
  - None of the participants had a clear idea about exactly what functionality is desirable for the improved filtering and searching capabilities nor how should be implemented.
  - The non-SSE participants had to be introduced to the IRIS/MFS domain.

The work products of the analysis phase were two documents. The first document describes searching and filtering in various COTS products, and the second document contained the use cases describing the functionality of the search and filter component.

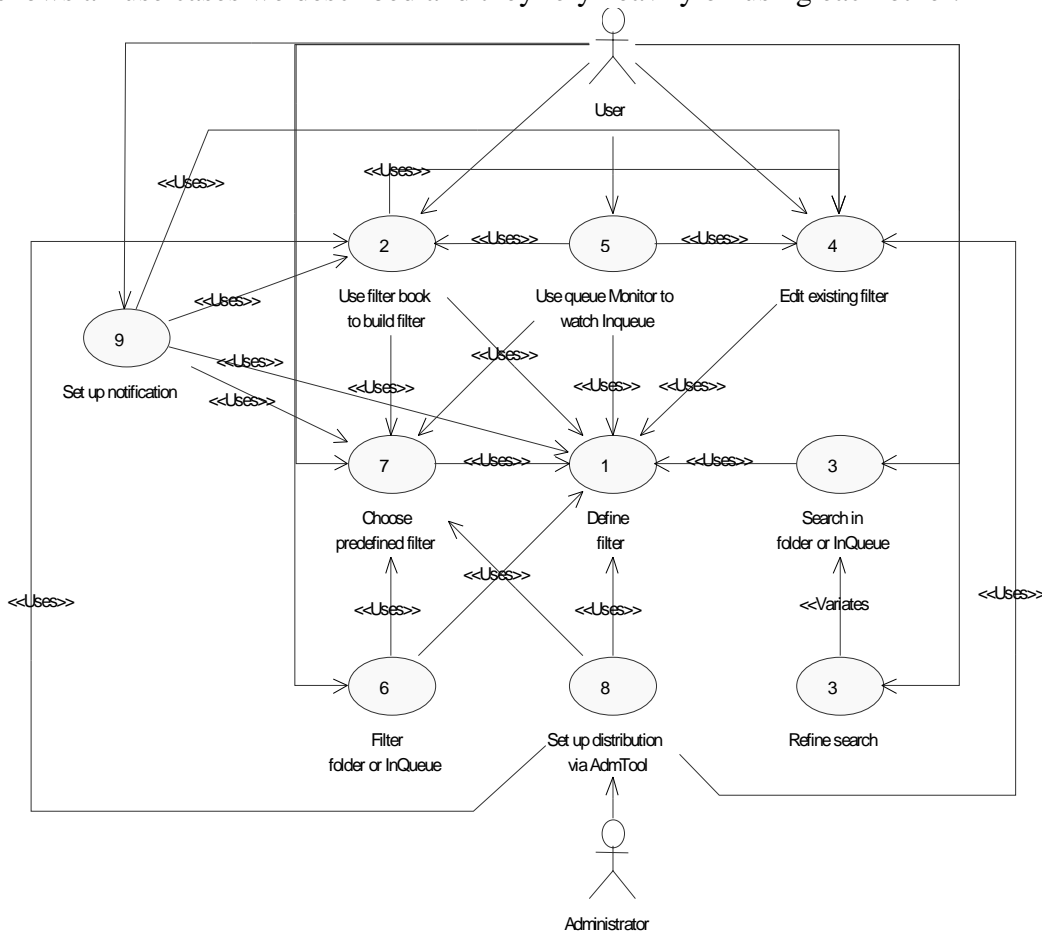
### 5.1 Work Method

One SSE employee documented both phases and presented the work to the other participants in the pilot project on a regular basis at the work meetings described in Section 1.4 and received feedback.

## 5.2 The Use Cases

The use cases were done in the context of using search, filter, and notification in IRIS/MFS. As mentioned in Section 3 a notification consists of a filter and an action that is performed upon the arrival of a message that matches the filter. The example in Section 3 was changing the colour of the subject font.

We modelled several work situations involving searching and filtering. Step by step making atomic use cases that were combined to form the actual work situations. The use cases are in UML [UML 97] notation. Often use cases are used in conjunction with collaboration or sequence diagrams for linking the performed actions to the internal sub-components of the system. This, in our opinion, a useful technique, but it was not an option because the customer wanted a GUI design for approval before development of the rest of the system commenced and therefore we had no internal structure yet. Figure 4 shows all use cases we described and they rely heavily on using each other.



**Figure 4 - Use case diagram for the filter/search facility**

The author's previous experience with use cases had left a feeling that use cases are useful, but that it is hard to come up with the right structure of use cases. Therefore we applied use cases following the techniques in [Cockburn 97a] and [Cockburn 97b] for structuring the use cases.

## 5.3 Example of Use Case: Define filter

This is the description of the definition of a filter. The description includes a short description of the use cases that trigger the use case, which are the vast majority of the use cases as “Define filter” is the most fundamental use case.

<b>Use case: 1</b>	<b>Define filter</b>	
<i>Goal in context</i>	Definition of a filter as part of another use case	
<i>Scope &amp; Level</i>	The filter definition GUI	
<i>Primary actor(s)</i>	User	
<i>Success postcondition</i>	The primary actor has defined a filter	
<i>Failed postcondition</i>		
<i>Trigger(s)</i>	<p>One of the following use cases will trigger the definition of a filter:</p> <ul style="list-style-type: none"> <li>a) The administrator wants to define a filter for distributing messages to the roles.</li> <li>b) The primary actor wants to use a filter on a folder or the InQueue.</li> <li>c) The primary actor wants to edit an existing filter.</li> <li>d) The primary actor wants to initiate a search.</li> <li>e) The primary actor wants to refine a search, i.e. search on search results.</li> <li>f) The primary actor wants to apply an existing filter, but does not find a suitable filter.</li> <li>g) The primary actor wants to use the InQueue Monitor to watch out for a special kind of message in the InQueue and defines a filter for this purpose.</li> <li>h) Set up notification on new messages.</li> </ul>	
<b>Description</b>	<i>Step</i>	<i>Action</i>
	1	The primary actor defines a parameter he wants to filter on. This step is repeated until the primary actor considers the filter as complete.
<b>Extensions</b>	2	The primary actor wants to name the filter and put it in the filter book for later use.
<b>Variations</b>		
<b>Related information</b>		
<i>Status</i>		
<i>Priority</i>		
<i>Performance</i>		
<i>Frequency</i>		
<i>Channels to actors</i>	Interactive	
<b>OPEN ISSUES</b>	The GUI	
<i>Superordinates</i>	All other use cases	
<i>Subordinates</i>		

**Table 1 - An example of a use case according to Cockburn**

## 5.4 Experiences and Lessons Learned

Our experiences with use cases were that they are a very useful tool for acquiring knowledge of a domain, in our case the domain of search queries, filters, and management of these. The specific use cases helped visualise the potential work processes of the users, clarify the functionality of the search and filter component, and identify the GUI sub-components needed such as editors for criteria and a book for storing the criteria. The use cases also formed a good concrete base for discussion at the meetings.

Cockburn's techniques were tried and they proved useful because most scenarios for filtering and searching in IRIS/MFS can be constructed by combining a number of use cases and the technique therefore provided a way to structure our use cases. As Table 1 indicates most of our use cases did not use the entire schema, but Cockburn also suggests that a user should tailor his template to suit his own needs. However, as this was a first try we did not have sufficient knowledge to eliminate the unused entries.

Beyond this the use cases did not give any benefits and were not utilised in the rest of the development process. After having reached a level where the use cases described the general functionality we found ourselves unable to refine the use cases further and make them more concrete.

Within the group present at these meetings we do not agree whether the reason is that:

- SSE has no (very little) contact with the end-users of our product, as SSE mainly delivers to system integrators who integrate IRIS/MFS into the system they are developing. This lack of end-user contact caused the use case to remain very abstract and they were not used for the rest of the project.
- The search and filter component is a generic component that can be used in any context where searching or filtering is necessary and therefore it is intrinsically hard to make the use cases for this particular part of the system concrete.

## 6. Design

This section describes the design process and the result of the process, i.e., the design of the GUI, the system design, and the detailed design. The employee who designed the GUI had no special knowledge in areas such as GUI design and usability.

### 6.1 Work Method

The two first phases of the design were carried out by having group meetings once or twice every fortnight where the design was done jointly. In between the meetings one developer elaborated on the work done and presented the work at the beginning of the next meeting. An incremental and iterative approach to development was used in the design phase. The work was done in the following order:

1. **GUI design.** In the contract with the customer it was specified that the GUI design should be a separate deliverable.
2. **System design.** The next part was the system design during which we identified the main components and some interactions between them. Design and architectural patterns were used extensively in this phase.
3. **Detailed design.** In the detailed design phase the system design was elaborated and refined.

## 6.2 GUI design

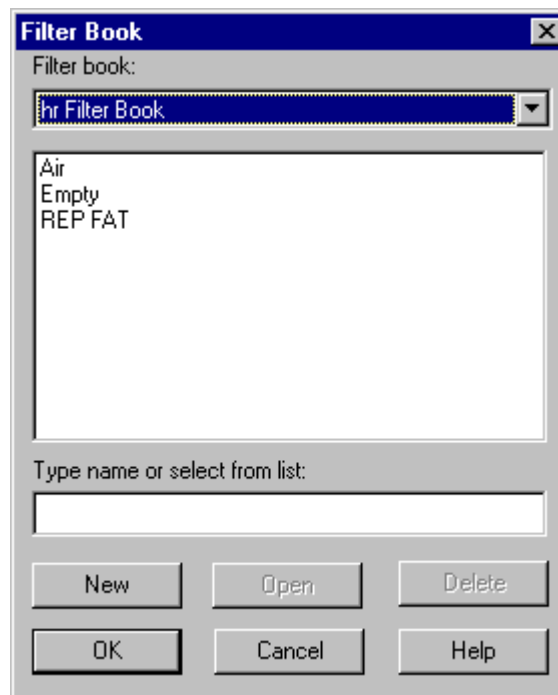
The GUI for improved filtering and searching consisted of three main parts:

- **Filter book.** For managing filters and search queries. The filter book corresponds to management of addresses in an address book as in, e.g., Microsoft Outlook™.
- **Filter editor.** For editing filters and search queries.
- **Search window.** The search window is comprised of the filter editor, a list box showing the result of the last search, and buttons for opening a filter book.

In the following the IRIS/MFS specific instantiation of the three main GUI sub-components is shown and shortly described.

### 6.2.1 Filter Book

One of the customer requirements was functionality to name and store criteria used in search queries or for filtering. Therefore a book for managing the criteria was designed. The user is able to make a new criterion and delete or modify existing criteria.



**Figure 5 - The filter book**

When the user makes a new criterion or modifies an existing one the filter editor is opened. It is described below.

## 6.2.2 Filter Editor

The filter editor is used for constructing and modifying criteria. It consists of three kinds of editors. The last two are partly constructed using the first.

- **Expression editors.** Each expression editor is for a specific type of attributes. There are separate editor for types like, e.g., number, string, date and set.
- **Simple filter editor.** For each instantiation of the search and filter component it is optional whether a simple filter editor that operates on a subset of the searchable attributes should be specified.
- **Advanced filter editor.** The advanced filter editor is generic as it operates on all attributes in the specified meta-model.

### 6.2.2.1 The Expression Editors

The attributes of the domain data can be grouped into several types, e.g., the same editor, the “String Editor”, handles all queries for text attributes. For each type of attribute there is a corresponding type of editor. There are currently five types of editors, but more will be added during phase three of the pilot project. Below is a UML [UML97] diagram depicting the classes.

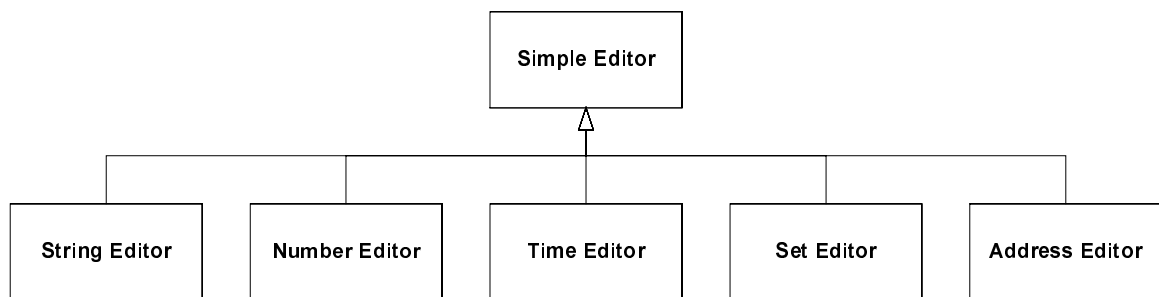
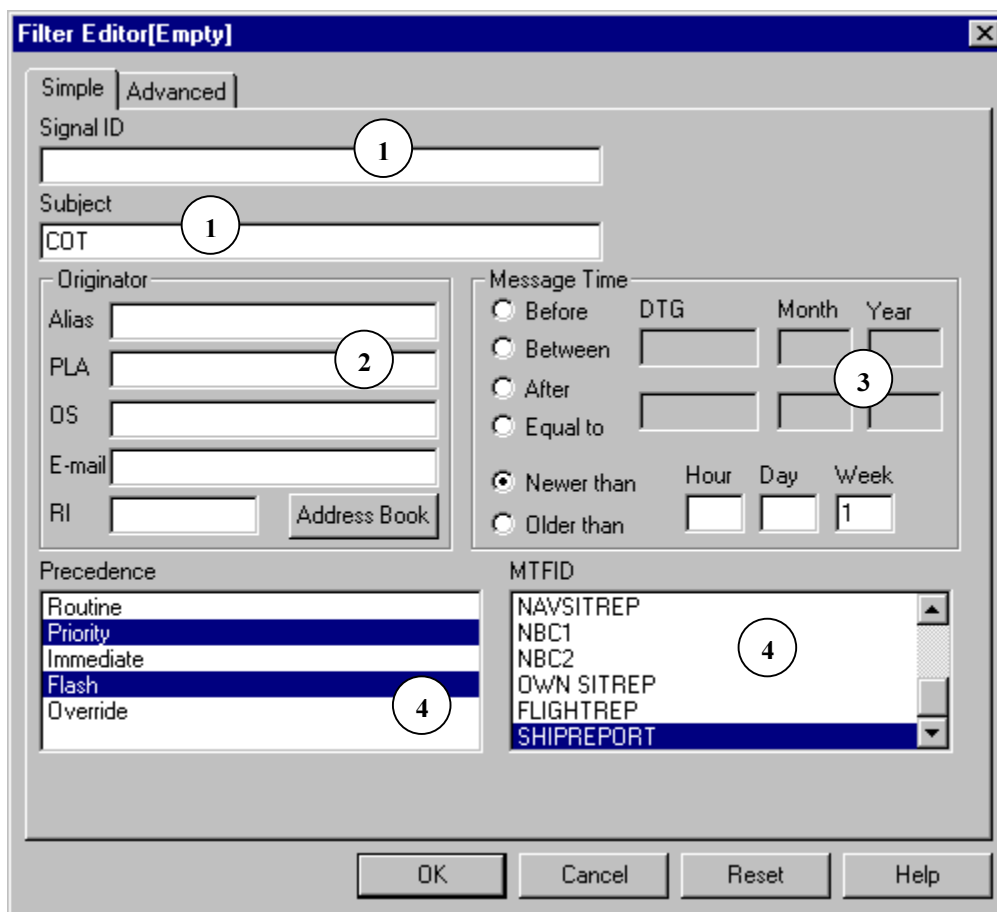


Figure 6 - The expression editors

The simple editors are used in both the “Simple” and the “Advanced” pages in the filter editor as explained below.

### 6.2.2.2 The Simple Page

The simple page was designed with the objective that it should be used for the majority of the criteria. It is comprised of a number of instances of subclasses of the “Simple Editor”. The editors marked ① in Figure are all “String Editors”, ② is an “Address Editor”, ③ is a “Time Editor”, while ④ are “Set Editors”.



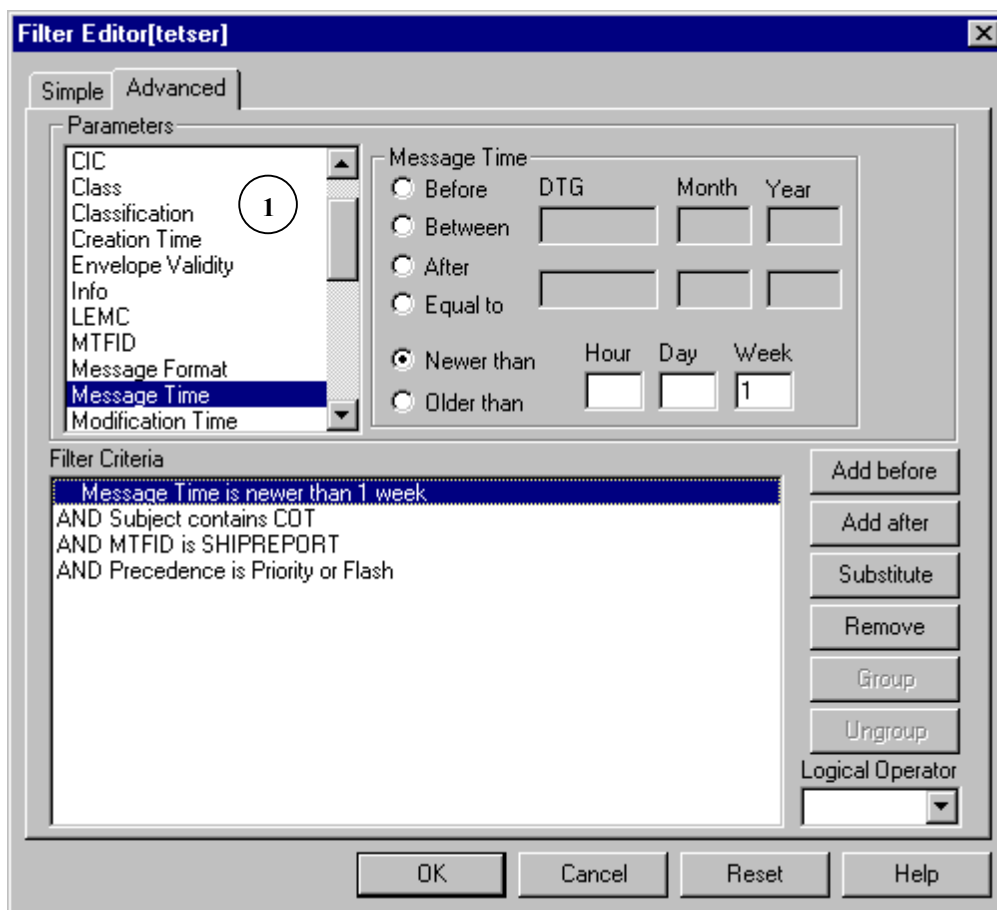
**Figure 7 - The simple filter editor**

Therefore it is up to the implementor to pick which attributes are shown and this is why “Simple” page has to be specified for each domain that the search and filter component is used in. When using the search and filter component it is possible not to have a simple page. It is currently planned not to have a “Simple” page when the search and filter component is used in EWare.

The user simply enters the filter and presses “OK”. The filter will then consist of the values entered in all expression editors combined with the AND operator, e.g., the filter for Figure 7 would be “(Subject = “COT”) AND (Message Time newer than 1 week) AND (Precedence = “Priority” or “Flash”) AND (MTFID = “SHIPREPORT”)”.

### 6.2.2.3 Advanced Page

The functionality of the “Advanced” page is more general than that of the “Simple” page. While it is only possible to access a limited number of attributes from the “Simple” page the “Advanced” page offers access to all the attributes of the entity being searched.



**Figure 8 - The advanced filter editor**

The attributes are listed in the listbox labelled ① in the “Parameters” area in Figure 8. The simple editor corresponding to the selected attribute is displayed to the right of the listbox. In the “Filter Criteria” area the entire filter expression is pretty printed. The filter in Figure 8 is the same as in Figure 7. In Figure 8 a criterion for the “Message Time” attribute has been selected and therefore the “Time Editor” is presented with the values from the criterion and the corresponding line in the pretty printed expression is highlighted.

The user can manipulate the selected line in the pretty printed expression in several ways. He has the ability to change the logical operator in front of the selected criterion by choosing another value in the “Logical Operator” combobox. The selected criterion can be a) removed, b) edited and the selected line is then replaced with the edited version. Furthermore a new criterion can be inserted either before or after the selected criterion. Grouping is used for ensuring that certain parts of an expression are evaluated before other, i.e., grouping is used instead for parentheses.

### 6.2.3 Search Window

The search window as shown in Figure 9 is used for searching. It consists of three major parts:

- **Query editor.** The “Query editor” consists of the “Simple” and “Advanced” pages, which have been described above.

- **Result pane.** Here the messages that are the result of the last search are listed. It is possible to perform a number of basic actions on each message, e.g., opening or deleting. The result pane is marked with a ①.
- **Search buttons.** The buttons are the user interface to search, refine a search, and get queries from the filter book. The buttons are marked with a bracket and a ②.

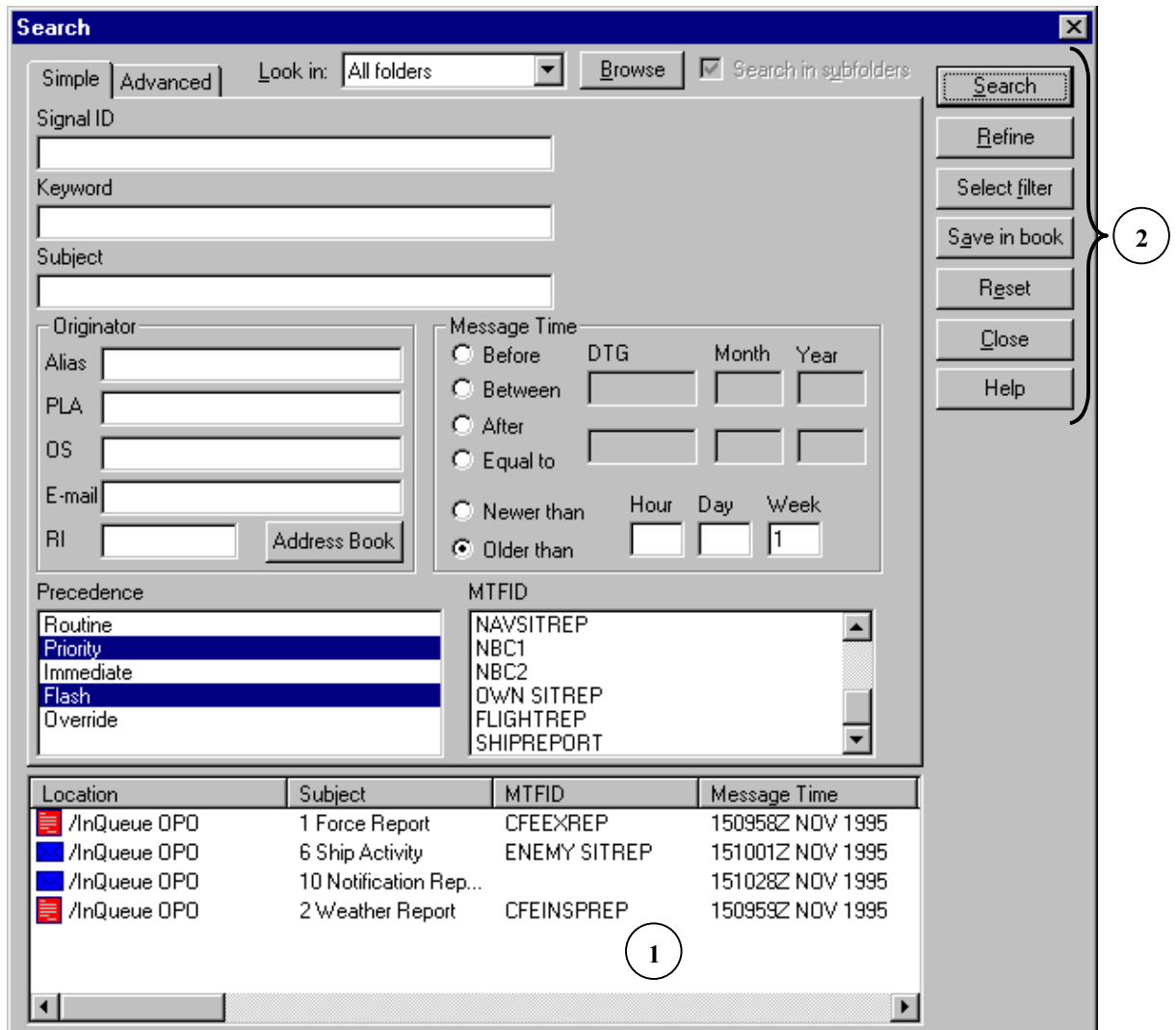


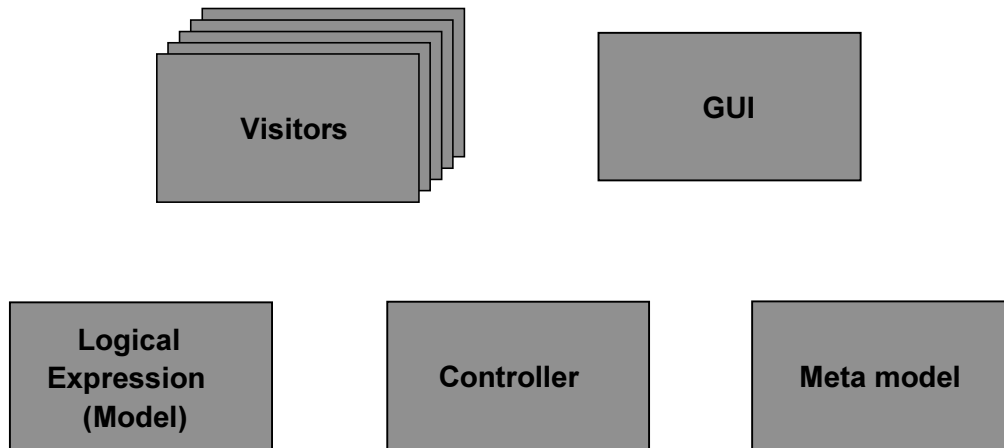
Figure 9 - The search window

## 6.3 System Design

The system design activity had the purpose to break the search and filter component down into a number of sub-components and describe the interaction between these sub-components. We use the term “sub-component” signifying a part that is responsible for delivering some specific functionality or behaviour. As the main focus of the project was to design the component so that it was reusable the work group focused the design on loose coupling and low context dependency. The objective was to ensure that the various sub-components of the search and filter component could be replaced by alternative variants conforming to the same interface and therefore each sub-component

in the search and filter component knows about as few other sub-components as possible.

The system design was initiated by a brainstorm session, which was followed by grouping the identified entities to get some structure and identify sub-components. The following key sub-components were quickly identified:



**Figure 10 – Sub-components in the search and filter component**

Each of the sub-components had the following purpose and responsibilities:

- **Logical Expression.** The work group chose to represent the query as a tree. One of the applications of the “Composite” pattern from [Gamma et al. 95] is the provision of the infrastructure for a tree. Therefore, the logical expression tree is a representation of the criterion, where the nodes are logical operators, e.g., AND. The tree only has methods for simple manipulations, construction and destruction. It should be possible to use the tree to represent any logical expression, not just a search query.
- **Visitors.** When extracting information from a tree there are basically two approaches: Having methods in the nodes of the tree or grouping these methods externally. In [Gamma et al. 95] these two approaches are called the “Interpreter” and “Visitor” pattern respectively.

The “Visitor” pattern was chosen in order to avoid “polluting” the interface of the tree with application specific methods, as it is the intention that the logical expression tree should be used in other contexts than the search and filter component.

The visitors are used for traversing the tree and extracting information. The idea is that one writes a visitor for each task that involves extracting information from the tree, e.g., pretty printing the filter in the “Filter Criteria” list box in the “Advanced Page” in the filter editor or transforming it into a textual representation for storage in a database. The functionality for a specific task is thus located in one place instead of being scattered over many classes, as it would have been when using the “Interpreter” pattern.

- **Meta-model.** The meta-model abstracts the data model of the domain for which the system is instantiated. It is the responsibility of the implementor to instantiate a meta-model for the application domain. The basic idea is that the other sub-components in the search and filter component do not know the data model of the domain but instead ask the meta-model for the needed information. The meta-model consists of a number of meta-objects describing each of the entities in the application domain. Each meta-object has meta-attributes describing the attributes of the object. During the design we did not know that our meta-model uses many of the principles of the “Reflection” architectural pattern from [Buschmann et al. 96] and we could have benefited greatly from the knowledge accumulated in the Reflection pattern. Of course there have been costs for our use of the meta-model:
  - Performance: At least an extra level of indirection has been added. This is however not an issue that troubles us as we have no strict demands on performance.
  - Comprehension: There was an initial learning curve when we started to use the meta-model and it takes a while before one has an overview of all the possibilities.
  - Complexity of code: The code has become a bit harder to understand because of the extra level of indirection, but as the meta-model is used in the same way in the entire search and filter component this is not a very big hurdle.

In our opinion the benefits by far outweigh the costs and furthermore we claim that it would be near impossible to reuse the search and filter component without a meta-model to hide the data of the application domain.

- **Controller.** The controller is responsible for manipulation of the logical expression tree. When the user changes the logical expression, then the GUI asks the controller to perform the manipulations on the tree corresponding to the user input.
- **Graphical User Interface (GUI).** The GUI serves two purposes a) it is the user interface for manipulating the logical expression tree and b) it is a view of the information in the tree. The GUI knows nothing about the domain data it is a view of, i.e., the GUI is domain independent and queries the meta-model for knowledge about the domain.

The logical expression tree, the GUI, and the controller form a layered version of the Model-View-Controller pattern from [Buschmann et al. 96]. Although the expression editor (see Figure 8) and the “Filter Criteria” listbox (see also Figure 8) are two different views of the model, our MVC does not include the “Observer” pattern. The reason is that the expression editor is an editor of the model and we have decided that the user has to press the “Substitute” button before changes are added to the model. Therefore there is no need for keeping consistency between several views. The layered version, which is an application of the principles in the “Layers” pattern also from [Buschmann et al. 96], was used in order to minimise dependencies between the sub-components meta-model, GUI, and “Logical expression tree”.

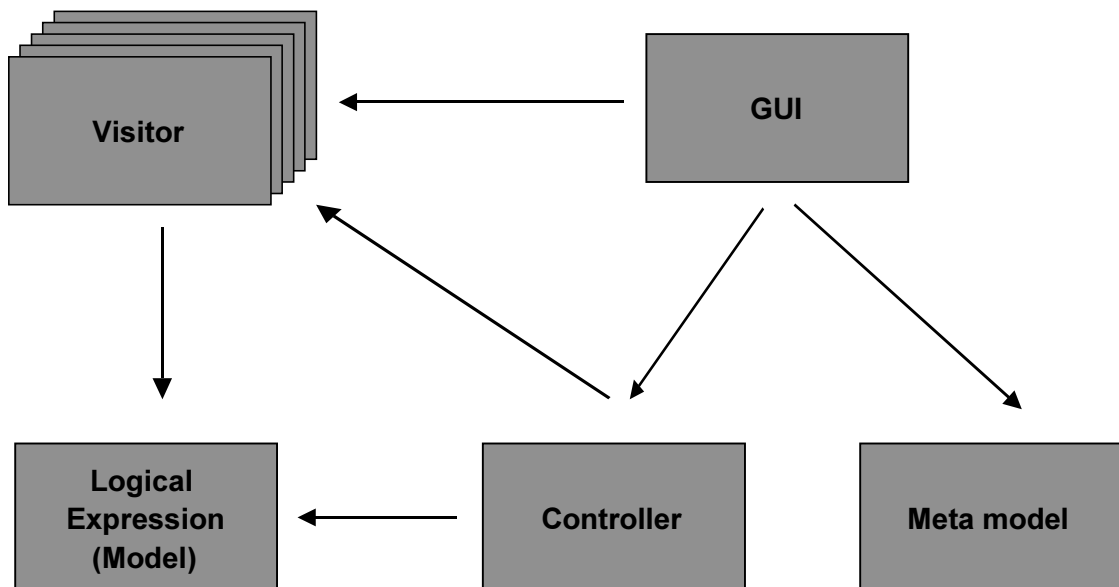
We also tried to think in terms of responsibilities, e.g., the tree is only responsible for the most basic manipulation such as adding or removing sons, while the controller is

responsible for performing the more advanced manipulations by combining the simple operations offered by the tree.

When we say that we have applied a pattern we do not mean that our class diagram exactly matches that from [Gamma et al. 95] or [Buschmann et al. 96]. We mean it in the way that Buschmann advocates [Buschmann 99], namely that we have used the ideas and principles from the pattern and implemented our own variant of the pattern to solve our specific problem. This is way the meat-model does not correspond precisely to the Reflection pattern nor does our Model-View-Controller have an observer pattern attached.

## 6.4 Detailed Design

The detailed design is a refinement of the system design where the sub-components, their interfaces, and the interaction between them is described in greater detail.



**Figure 11 - The client-server relations of the sub-component**

The level of detail in the detailed phase was much closer to the class diagram in Figure 15 than to the diagram of the sub-system parts in Figure 10 or Figure 11. However this level of detail was chosen in this document because it is beyond the scope and purpose of the document to describe the full detailed design.

## 6.5 Level of Abstraction During the Design

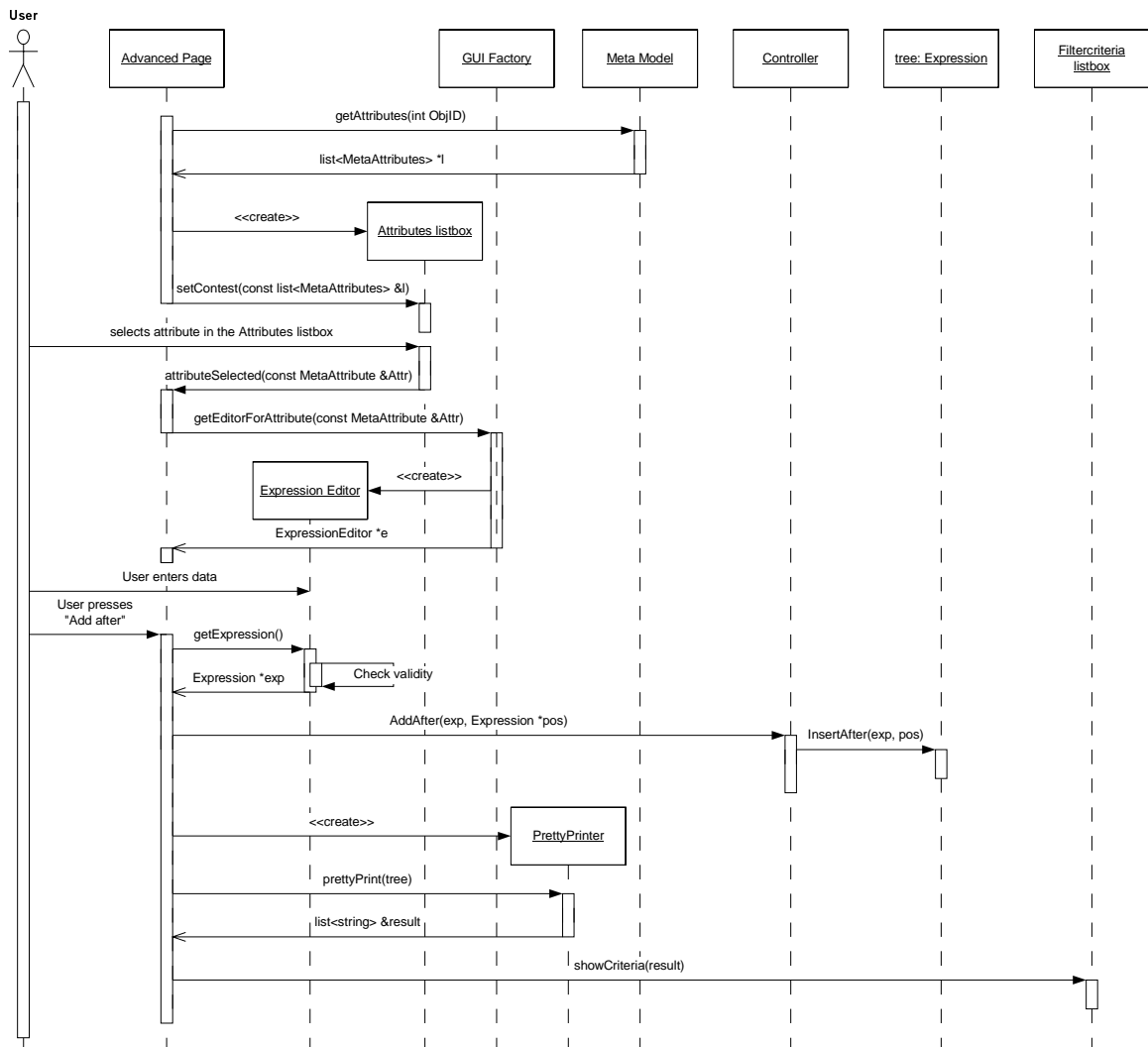
Instead of using the normal waterfall oriented work method mostly used at SSE, we used an iterative approach to the design during which we frequently shifted level of abstraction between discussing the search and filter component at sub-component, class and code level.

Most of the time we worked at an abstraction level close to that of the UML diagram in Figure 15.

## 6.6 Illustrating Interactions Between Components

To give an idea of how the search and filter component works an example featuring all sub-components is described below. The reader will not be given a full explanation of the internal structure and interface of the components, rather a flavour of how the component functions. The example deals with the construction of a new criterion.

The situation is as follows: The user has opened the filter editor and presses the “Advanced” tab thus switching to the “Advanced” page, which is shown in Figure 8. The UML sequence diagram for the example is shown in Figure 12.

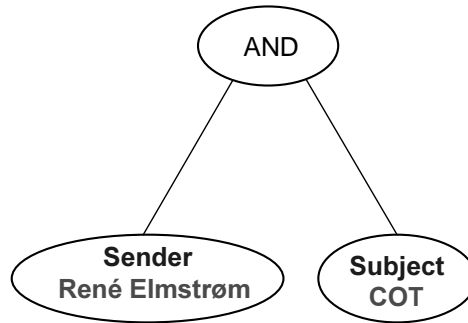


**Figure 12 - Interaction between the sub components**

The example begins with the “Advanced” page getting the meta-attributes for the meta-object on which it is searching. These meta-attributes are put into the “Attributes” listbox (marked ① in Figure 8). The user then presses on an entry in the “Attributes” listbox. The listbox notifies the “Advanced” page. The page then asks the “GUI factory” for the expression editor corresponding to the selected meta-attribute. The user then enters data for the expression and presses the “Add After” button. The “Advanced” page then gets the expression corresponding to the data entered from the “Expression Editor”

after the “Expression Editor” has checked the validity of the entered data. The “Advanced” page then asks the “Controller” to insert it into the “Logical Expression Tree”. The “Advanced” page then creates a pretty printer and pretty printer object and asks it to pretty print the “Logical Expression Tree” and gets a list of strings containing the pretty printed tree as result. The “Advanced” page then inserts the result in the “Filter Criteria” area shown in Figure 8.

An example of using a Visitor is the pretty printing of the expression ((Sender = ‘René Elmstrøm’)  $\wedge$  (To = ‘cot-all@cit.dk’)), which is represented by the following tree:



**Figure 13 - An expression as a tree**

The AND node object is an instantiation of the class `AndNode`, the Sender node is an instantiation of the class `AddressNode`, and the Subject node object is an instantiation of the class `StringNode`. The class hierarchy for the nodes in tree can be found in Figure 15.

The Visitor class has a method visiting each of the node existing node types, e.g., a pretty printer has:

```
class PrettyPrinter: public Visitor {
public:
    const &vector<string> PrettyPrint(const Expression &tree);
    void VisitAndNode(AndNode *node);
    void VisitStringNode(StringNode *node);
    void VisitAddressNode(AddressNode *node);
    ...;
};
```

The ‘...’ contains all the Visit methods for the remaining types of nodes from Figure 15.

The superclass for the node types `Expression` has an abstract method `Accept`, which takes an `Visitor` as argument and then calls back to the `Visitor` reporting to it, which type of node it operates on. The `Visitor` then performs the appropriate actions for that node.

The interaction between the pretty printer and the expression tree happens the following way as expressed in the UML sequence diagram:

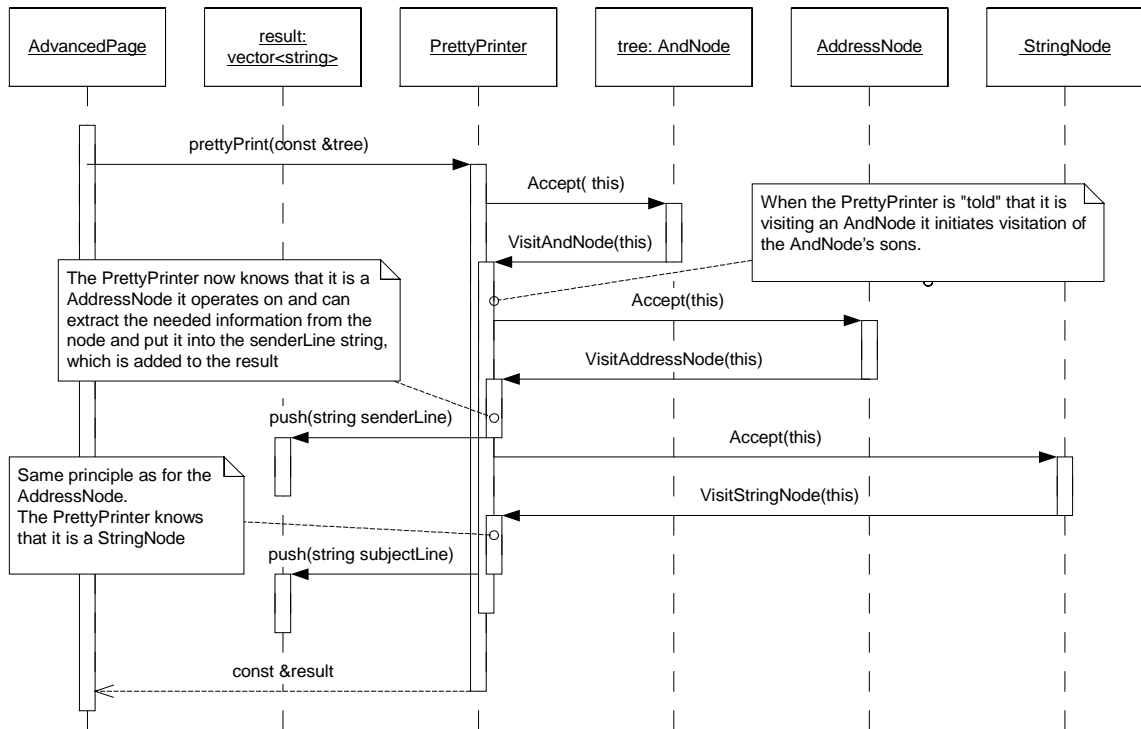


Figure 14 - Pretty printing an expression

## 6.7 Lessons Learned and Open Issues

Most of the lessons learned during the design are not new insights; they rather support observations made by other sources.

The group design provided good drive and many different proposals for solutions to the problems encountered. However, when working this way, the group should not be too large otherwise there will be a tendency that some participants will be inactive. A suitable group size is three or four persons. Furthermore the meetings should not last longer than three or four hours, as the high activity level is quite tiresome.

The overhead when designing for reuse was not very large. The presence of the EWare project manager during the design was the only overhead in the design phase and this amounted to 3-4 man-days. Furthermore the non-SSE participants also were an additional resource that participated in the design. They had no domain knowledge when the project was initiated and throughout the work they added different perspectives and fresh ideas that the SSE employees with the domain knowledge might not have thought of.

The component has low context dependency (low coupling) and high cohesion thus giving a component where all parts can be replaced if desired. However it remains to be seen whether this actually is true. It is an open issue whether use of the "Mediator" pattern [Gamma et al. 95] for handling communication between different parts of the component could have reduced the degree of coupling even further, but at the expense of introducing another level of indirection.

The quality of the documentation for the component and its sub-components is essential when reusing. In our opinion it is not the kind of notation one uses for describing a component that decides whether documentation is good or bad, but rather which aspects of the component that are documented. The normal UML class diagram notation such as dependency and association are not specific enough as they do not specifically state, which parts of a class' interface the other classes use. It may be an idea to use UML's notation for interfaces when documenting for reuse. This would make it explicit which parts of the interface of a class are used by which other classes. This matches well with our retrospective realisation of having unconsciously designed the search and filter component with an eye on which parts of a sub-components interfaces that are used by the other sub-components. A template for documenting reusable components will be developed as part of phase three and the template and our requirements for it will be described in a future COT document.

An observation made during the design was that one should consider which language the design is going to be implemented in. Each programming language has its strengths and weaknesses that will affect not only how the design is implemented, but also the benefits one can gain from a specific design. The Scandinavian school of object-orientation focuses on modelling, and there are often several ways to model the same problem and these alternatives should be considered in light of the intended implementation language. A concrete example from the pilot project will not be given as this would require a much more detailed explanation of the internal structure and this is outside the scope of this report.

Similar experiences have been reported in [Brown et al. 98], where the design patterns from [Gamma et al. 95] have been implemented in Smalltalk. The authors of [Brown et al. 98] wanted to provide a Smalltalk perspective on design patterns complementary to the C++ perspective in [Gamma et al. 94]. This shift in perspective caused changes to some patterns and the implementation of others varied greatly.

## **7. Implementation**

In the beginning of the implementation phase the number of developers was increased from one developer, who was supposed to use iterative and incremental development model to five developers using a waterfall oriented approach. In the implementation phase the detailed design proved to be too abstract and not detailed enough for the four developers who had not participated in the analysis and design. Of the four two had a Master's Degree in Computer Science, one had a Master's Degree in Engineering, and the last had an education equivalent to a Bachelor's Degree in Computer Science.

Three of them are experienced in object technology, but have no knowledge of UML, but knowledge of similar notations such as OMT, and no knowledge of patterns, but they could be expected to learn the concepts of patterns by themselves. The fourth has limited experience of object technology and no knowledge of patterns and UML.

Throughout the entire implementation the architecture of the search and filter component proved stable.



## 7.2 Lessons Learned

Several lessons were learned from the implementation phase:

- The design should be very specific if more developers are added to the project team during the implementation phase. In the project a few unwise decisions were made simply because the design was not concrete enough. Again this will not be exemplified, as this would require a level of detail that is beyond the scope of this report.
- Just as patterns can be a great benefit if all involved have the same amount of knowledge they can be a burden if all involved have different knowledge levels. Just as design patterns make the design easy to understand if one knows them, they can make it very hard to understand if one has no knowledge of design patterns.
- It should be attempted to ensure that the interfaces of the components are stable early in the implementation phase, as it is much easier to work several developers at the same time if the interfaces do not change.

## 7.3 Experiences with Patterns

Patterns quickly became our common language during the design as the members of the work group had a common reference in the design patterns “bible” [Gamma et al. 95] and some had knowledge of [Buschmann et al. 96]. Patterns were therefore used extensively throughout the entire design phase. This resulted in a large speedup of the design and simultaneously produced a soundly structured search and filter component at least on the sub-component level. However, we have no qualitative or quantitative data to back this statement only our own feeling and estimation of the amount of resources that would normally be required to design a component such as the search and filter component. These experiences support the experiences from [Beck et al. 96].

A drawback was that the discussions sometimes were a little abstract, as the participants tended to talk in terms of the abstract description of the patterns instead of the actual application of the pattern. As mentioned above the use of patterns actually made the design harder to understand for some of the developers that were added in the beginning of the implementation phase. This observation was not made in [Beck et al. 96] because all authors had performed training beforehand, so this stresses the importance of training before using patterns.

## 8. Restructuring of the Code

As the implementation was done (a) quicker than originally envisioned, (b) by more developers than originally planned, and (c) the detailed design was not sufficiently concrete some unfortunate decisions regarding search and filter component structure and detailed design were made along the way. Therefore a restructuring phase was necessary. The restructuring consisted of two tasks:

- **Re-implementation of the pretty printer.** The pretty printer did not have general enough support for tailoring to a language other than English. Therefore the pretty

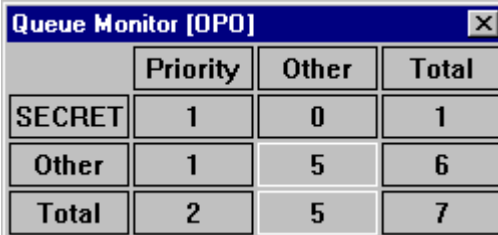
printer had to be re-implemented. Furthermore new insight regarding the usefulness of the meta-model had been gained during the implementation of the rest of the system. Before the reimplementation the pretty printer was 100 % IRIS/MFS specific, but extensive usage of the meta-model reduced the IRIS/MFS dependencies greatly. A small future extension to the meta-model will remove IRIS/MFS dependencies altogether.

- **Restructuring the code.** During the hurried implementation the search and filter component code and IRIS/MFS specific code was placed together in the same files and furthermore IRIS/MFS specific constants were used in some places. All these mishaps had to be corrected after the delivery.

The clean up and the restructuring all in all took two weeks. Even so before the code can be used in EWare another round of restructuring is necessary. The changes necessary for EWare are architectural changes, which will ease the addition and removal of node types and expression editors during run-time. A future COT document describing phase three will contain a description of the EWare changes.

## 9. Phase Two: Reusing in IRIS/MFS

The first design and implementation was done for version 2.2 of IRIS/MFS. In version 3.0 the search and filter component has, of course, been reused in whole in the parts for which it was originally intended, but some of the sub-components have also been reused in parts of the application for which they were not originally envisioned. The same approach is now used for filters, notifications<sup>1</sup>, and an Inbox overview tool called the “Queue Monitor”, which is shown in Figure 16. The leftmost column and the topmost row are filters. It is possible for the user to define filters in both directions. The columns “Other” (how many messages do not match user-defined queries) and “Total” (how many messages that are present in the Inbox) are predefined filters, while “Priority” and “SECRET” are user-defined filters. Each entry in the matrix specifies how many messages match the two filters simultaneously. There is for example one message that matches the “SECRET” and “Priority” filters simultaneously, while there are two messages matching the “Priority” and “Total” filters simultaneously.



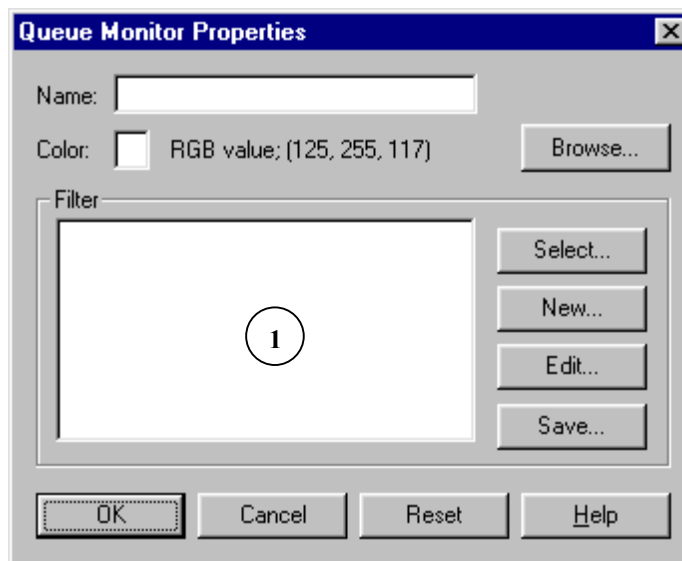
	Priority	Other	Total
SECRET	1	0	1
Other	1	5	6
Total	2	5	7

Figure 16 - The Inbox Monitor

If the Inbox Monitor was used in Outlook a corresponding example could be a filter names “COT René”, where the criteria are “Subject = COT” and “Originator = ‘René Elmstrøm’”. The entry (“COT René”, “Total”) in the matrix would then show how many messages have “COT” as subject and “René Elmstrøm” as the sender.

<sup>1</sup> A notification is an event triggered by the arrival of a message matching a filter.

Before version 3.0 of IRIS/MFS the interface for notification and Inbox Monitor differed greatly, but from version 3.0 and onwards they will have a very similar look and feel. The new interface is shown in Figure 17.



**Figure 17 - New “Queue Monitor” Interface**

The parts that were reused are:

- The pretty printer from the “Filter Criteria” area in the “Advanced” page. The pretty printer is used in the listbox labelled ① in Figure 17.
- The filter editor that is opened when the user presses the “New...” button in Figure 17.
- The meta-model, which is used for describing attributes that can be used for notifications and queue monitors.

After the user has specified the filters, he presses the “OK” button in Figure 17 and the window from Figure 16 is opened.

## 10. Phase Three: Reusing in EWare

Phase three will be integration and use of the search and filter component in EWare. A number of modifications and extensions have been identified at the time of writing. New node types have to be added to the logical expression tree and new expression editors for the new node types have to be designed and implemented. Finally a meta-model for EWare has to be implemented. This work will be described in another COT document as soon as it has been carried out.

## 11. Conclusion

The main conclusion is that the design proved stable and implementable when used in the first project. It remains to be seen what happens when the component is used in EWare.

During the pilot project the most important observations and lessons learned were:

- Even though no real-life scenarios could be established the use cases provide a useful base for discussion and helped clarify the needed functionality.
- It is our experience that working in a group when designing is worthwhile. It gives a better drive and the participants can inspire each other. However, the resources required will often be a hindrance. Instead one could have design presentations where the designer presents his or her design and the participants may comment, but the designer is under no obligation to include the proposals in the final design. If this is combined with reviews, the presentation could serve as a pre-review activity.
- Patterns are indeed a powerful and high level way of communicating if all involved have roughly the same knowledge of patterns. If this is not the case they can have the opposite effect and can make the design much harder to understand and thus confuse the developers implementing the design. In our opinion this problem can be eliminated by training developers in the theory of patterns and letting them apply them in practice.
- The design and architectural patterns ensured the low coupling and high cohesion in our system by providing a sensible infrastructure.
- The overhead for taking requirements from potential reusers into consideration was not very large. The presence of the EWare project manager during the design was the only overhead in the design phase and this amounted to 3-4 man-days. It is much harder to estimate how much extra effort was put into the implementation phase compared to making an IRIS/MFS specific search solution. It is our claim that there was an overhead but that it was relatively small. The non-SSE participants were of course an additional resource that provided fresh ideas and novel views on the domains.
- When introducing new techniques such as a meta-model it takes a while to “see” all the possibilities, strengths, and weaknesses of the new techniques. A benefit of an incremental and iterative software process would be better support for the adoption of new technology as the developers can gradually refine their use of the techniques. This applies equally to the design and implementation phases.

## 12. References

- |                       |  |
|-----------------------|--|
| [Beck et al. 96]      | Kent Beck, James O. Coplien, Ron Crocker, Lutz Dominick, Gerard Meszaros, Frances Paulisch, and John Vlissides: “ <i>Industrial Experiences with Design Patterns</i> ”, 18 <sup>th</sup> International Conference on Software Engineering, 1996, pp. 103-124 |
| [Brown et al. 98]     | Kyle Brown, Bobby Woolf, Sherman R. Alpert: “ <i>The Design Patterns Smalltalk Companion</i> ”, 1998, Addison-Wesley, ISBN 0-201-18462-1   |
| [Buschmann et al. 96] | Frank Buschmann, Regine Meunier, Hans Rohnert,   |

Peter Sommerlad, Michael Stal: *"Pattern-Oriented Software Architecture – A System of Patterns"*, 1996, John Wiley & Sons, ISBN 0-471-95869-7

- [Buschmann 99] Invited talk by Frank Buschmann at “Objektorienteret Softwareudvikling – udfordringer og løsninger”, Copenhagen, March 1999
- [Cockburn 97a] Alistair Cockburn: *“Goals and Use Cases”*, Journal of Object-Oriented Programming (JOOP), September 1997
- [Cockburn 97b] Alistair Cockburn: *“Using Goal-Based use Cases”*, Journal of Object-Oriented Programming (JOOP), November/December 1997
- [Gamma et al. 95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides: *“Design Patterns – Elements of Reusable Object-Oriented Software”*, 1995, Addison-Wesley, ISBN 0-201-63361-2
- [Goldberg and Rubin 95] Adele Goldberg and Kenneth Rubin *“Succeeding with Objects”*, Addison-Wesley, 1995, ISBN 0-201-62878-3
- [Jacobson et al. 97b] Ivar Jacobson, Martin Griss, and Patrick Jonsson: *“Software Reuse – Architecture, Process and Organization for Business Success”*, Addison-Wesley, 1997, ISBN 0-201-92476-5
- [UML 97] Object Management Group standardisation of UML, [http://www.omg.org/techproducts/meetings/schedule/Technology\\_Adoptions.html#tbl](http://www.omg.org/techproducts/meetings/schedule/Technology_Adoptions.html#tbl) UML Specification , 1997

## **A. Abbreviations**

AU.....	University of Aarhus
CCIS.....	Commando Control Information System
COM.....	Common Object Model
CORBA.....	Common Object Request Broker Architecture
COTS.....	Commercial of the Shelf
DCOM.....	Distributed Common Object Model
DTI.....	Danish Technological Institute
GUI.....	Graphical User Interface
IRIS/MFS.....	IRIS Message Formatting System
SSE.....	Systematic Software Engineering A/S
UML.....	Unified Modelling Language