Jens Kaasbøll

FIRE report no. 14
5. January 1994

Object-Oriented Models of Functionally Integrated Computer Systems

This report is accepted for publication in Journal of Object-Oriented Programming

FIRE
Department of Informatics, University of Oslo
P.O.Box 1080 Blindern, N-0316 Oslo, Norway
Phone: +47 22 85 24 10
Fax: +47 22 85 24 01
Email: fire@ifi.uio.no

Abstract

Functional integration is the compatibility between the structure, culture and competence of an organization and its computer systems, specifically the availability of data and functionality and the consistency of user interfaces. Many people use more than one computer program in their work, and they experience problems relating to functional integration. Various solutions can be considered for different tasks and technology; e.g. to design a common user-interface shell for several applications, or to merge the user programs. The solutions may require different types of technical integration.

Object-oriented methods for application development are said to be well suited for developing components that are useful in several applications. A framework for capturing functional integration in object-oriented analysis and design is proposed. The framework distinguishes between the users, the parts of the computers that are perceivable during use, and the inner parts of the computer system. In addition, distinctions between layers of implementation are introduced.

It is demonstrated how object-oriented models of information systems should be modularized according to the framework. Typical solutions are characterized by object-oriented models in the framework, including the shell and copy and paste.

Three object-oriented approaches are examined to see whether they address problems of functional integration. It is found that none of these approaches cover all relevant aspects.

1 Introduction

Users of computer systems often use more than one program in their work because no single program supplies the desired functionality. A *user program* can be an application, i.e. a program developed for a specific purpose, or a general tool, e.g. a text processor, a spread sheet, an operating system. System development and redesign projects often seek to integrate user programs.

Technical integration of user programs takes place within the environment provided by the operating system and other basic software. When two user programs run in the same environment, exchanging data and invoking functions become easier than when applications run on different operating systems.

The design issue of how to integrate user programs generates several questions at the user side of the systems too. The research project Functional Integration through Redesign develops techniques for improving integration between work and computer systems and perceived integration of computer systems (Braa et.al, forthcoming). Functional integration is defined to be the compatibility between the structure, culture and competence of an organization and its computer systems, specifically the availability of data and functionality and the consistency of user interfaces. The aim of this paper is to demonstrate how conditions for functional integration may be included in analysis and design with object-oriented methods.

1.1 Outline

Functional integration is elaborated in section 2, and a framework will be proposed in section 3. It will be shown in section 4 how various solutions to functional integration can be expressed by means of the framework. The frameworks of the three approaches to object-oriented modelling will be evaluated in section 5 to see whether they are suitable for object-oriented models that support design of functionally integrated systems.

2 Functional Integration

The claims on the computer systems for being functionally integrated are that data and functionality are available and that the user interfaces are consistent. These issues were also mentioned by Newman (1986). A case will be outlined to illustrate typical solutions to functional integration (derived from Kaasbøll and Øgrim, 1989).

> Consider two officials in charge; Tora in the Town Planning Office and Henry in the Highways Service. Currently, there exists a database of properties with their owners, addresses and areas, which is also used by other departments and by banks. There is a vision of a common computerized case file for the technical departments, based on a database system in their common United network. This is to be integrated in their work in the following manner. Tora receives the request to divide property, scans the request into a new record in the common case file, retrieves data about the area of the property from the property database, calculates the areas planned for parking and playground with her spreadsheet, compares the request to current political resolutions, and transmits the case file to Henry. He has the Fine operating and window system in his computer. In order to access the case files, he has to run United on a remote server in addition. Henry compares the request and Tora's calculations to the planned public roads and other infrastructure. If everything is OK, he returns the case to Tora, and he updates the database of traffic estimates by cut and paste from United to Fine.

2.1 Adequate support

Newman emphasizes that one aspect of functional integration is to ensure that the system provides an adequate range of interactive functions (Newman, 1986, p.346).

> Even if Henry would have been better off with a United based computer while working with the case files, his current computer yielded better support for the remaining part of his job.

This example indicates that a functionally integrated system would support both the tasks and the users.

Determining the functionality requires knowledge about the work tasks for which a computer system will be used. However, if the programs are only adjusted to the tasks and the current division of tasks among the users, opportunities for improving work practices may be missed. Therefore, the system development analysis should yield knowledge of the tasks and jobs in the which the functionally integrated system are to be used. This will ensure consideration of work

tasks and the division of tasks among users.

Designing the functions of computer systems also requires knowledge about the subjects to be represented by the data, called the *problem domain*. It has been assumed that functionality of computer systems changes more rapidly than data definitions. This is because work tasks are changed and reorganized more frequently than the problem domain. Even if recent research shows that data models are less stable than expected (Marche 1993), the assumption has not been rejected. To prepare the programs for future changes, object-oriented modelling for functional integration should be based upon the problem domain.

However, when integrating several programs, each of them may have its own problem domain.

> Tora's problem domain includes the areas and shapes of properties in a region, while the central property register has owners, addresses and areas for all properties in the county as its domain.

When considering functional integration, the problem domains of common systems and the problem domain associated with a single task may interfere.

2.2 Coherent user programs

The claim that computer systems should match the tasks applies to all computer usage. When designing user programs to function satisfactorily together, Newman (1986) mentions three questions to be considered. These are:

1. Can two or more different functions operate on the same data, and if so, how is this implemented?
2. How does the user move from one function to another, and how much "state" information is saved when he or she temporarily leaves a function?
3. What overall user-interface conventions are observed across all functions?

These issues will be considered in further detail.

Data sharing

Effort has been put into system integration to avoid retyping of data; e.g., company databases, CIM. However, new user programs are purchased and put into use at an increasing rate. When programs need access to the same data, and the data can neither be transferred nor referred to electronically, the users have to re-register the data by retyping, scanning, or by other means.

In a survey, user problems of inconsistent interfaces, isolated data, and poor task support were compared to more well-known problems of poor training and user documentation, long response time, breakdowns, and physical strain (Kaasbøll, Braa, and Bratteteig, 1993). The users reported that problems of inconsistent interfaces, isolated data, and poor task support were equally serious problems as those which were already well known.

For the user to experience data sharing, it is not necessary to have one database. A shell of functionality, consisting of common functions at a common interface to several databases is an alternative solution. When the data resides outside the organization, a transfer function can provide the desired functionality.

Data integration means standardized data structures according to a conceptual schema (Goodhue et.al., 1992, p.294). Data integration is necessary for transferring data between data-

bases. The empirical findings of Goodhue et.al. indicate that excessive data integration may turn out to be counterproductive. This is because a large, integrated computer system is more difficult to adjust to local needs, adapt and keep up to date than cheap software products an organization can afford to dispose of. Security and privacy may also be in conflict with integrated systems.

Tora utilizes data integration when she retrieves area from the property database.

Users may also experience that data can be shared without common data definitions. Export/import or copy/paste may yield sufficient sharing of data (Hannemyr 1992). Such functions often copy raw data and disregard structures, and are called *context free data sharing.*

Henry copies data in this way from the case file to his traffic estimate program.

In many cases where data sharing is possible, data must be reformatted to be exportable from one program and imported into another. Specific conversion programs may be needed to transfer data from one environment to another.

The first condition for functional integration is:

1. Data sharing: data integration and context-free data sharing should be possible with minimal operation by the user and without having to pay attention to data formats.

Available functions

Many modern computers allow for multiprocessing of programs, and the user can interact with each program through one or several windows. Even in these settings, remote access to other computers may be restricted to batch processing, or awkward login procedures and limited interaction time may hamper concurrent use of several applications (e.g. Fagermoen et.al., 1991). The second condition for functionally integrated systems is:

2. Available functions: immediate access to several programs, making the programs remain in any desired state after resuming them, with minimal operations by the user.

Interface consistency

Shneiderman proposes “strive for consistency” as his first golden rule for dialogue design (Shneiderman, 1992, p.72). He states that consistency is the most violated of all principles. When several programs are in use, it seems likely that the total interface of all programs will be even more inconsistent than when only one user program is applied.

Transfer of learning from one situation to another is hampered by inconsistencies (Rosenberg, 1989, p.25). Consistency creates expectations of program behaviour, and if a new program fulfils these expectations, it will increase the possibility of being accepted (Nielsen, 1989, p.5). However, “ease of use” may conflict with principles of consistency, so the purpose of using the computer programs has to be taken into account before insisting on consistency (Grudin, 1989).

In a functionally integrated system, the interfaces of the individual user programs should be consistent. This means that when two programs have the same functionality, the corresponding interface should also be similar. For example, for closing a process, the close button should have the same colour, shape, placement, and manner of activation in all programs.

3. Interface consistency: it should be possible to implement the same functionality in different user programs in the same interface patterns.

2.3 User Knowledge

In the organizations studied in the survey of functional integration (Kaasbøll, Braa, and Bratteteig, 1993), inconsistency, isolated data, and poor task support correlated with complaints about training and user documentation. The users wanted to learn more about the computer systems, but complained that they had not enough time to learn.

The user interface of an integrated system should be designed to enable optimal interaction with the system. An inquiry into four redesign projects indicated that users are very keen to continue using the interface skills they had developed with their former user programs, when converting to the new, integrated systems (Toft, 1992, p.91). Users' knowledge and skills should thus also be considered when designing the interface of the integrated systems.

Training and documentation should be designed for enabling the users to master the integrated system. Therefore, the knowledge needed for interacting with the new system must be determined.

2.4 The Claims of an Object-oriented Model

The design issues to consider in an object-oriented model are the sharing of data, the availability of functions, and the consistency of the interface. The issues in the environment of the computer system include the problem domain, tasks, users' work, and the users' computer skills.

3 A Framework for Functional Integration

The proposed framework separates the *problem domains*, the *users* and the *computers*, see figure 1. The computer is divided into two parts, the *representation of the problem domain*, and the *shell providing functions and data.* The available functions and data are also called the *functionality* of the system. The representations of the problem domains should be shared by many users, while a specific shell is dedicated to a specific user and task.

Since functional integration is based on the user experience, the user's view of the computer must be included in the framework. Since the users operate their computer through the user interface, the user's view of these parts of the computer should be included in the framework. However, designers also need to consider the objects that provide the functions, data, and the interface. Therefore, the technical perspective of the designer on these issues are also included. The "Functions and data for task and user" and the "Shell providing functions and data" in figure 2 are thus two perspectives on the same, functionality component of the computer.

The user interface is omitted in the basic framework. It will be included in an extended version by means of the relation defined in the following section.

3.1 The Realization Relation

Signs are entities that can represent phenomena distinct from themselves. For example, the word "road" represents a physical structure apart from the written text in which the word itself appears. The data and transformations of data which occur in a computer can be regarded as signs, because they represent the problem domain. A theory of signs is therefore applicable to data and data transformations, and the sign concept in structural semiotics will be applied to ex-

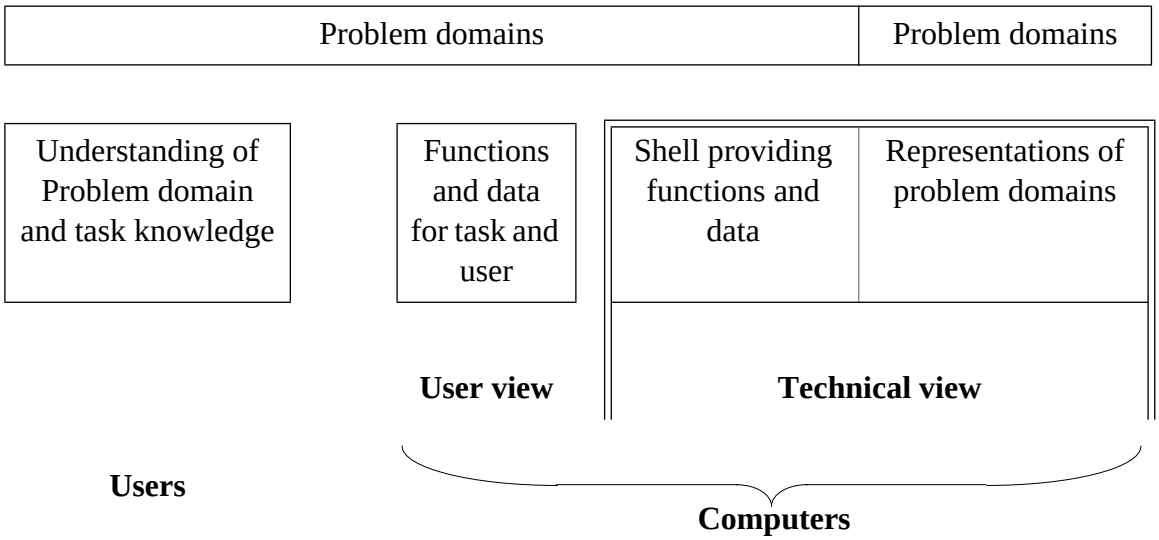

Figure 1: The basic framework for Functional Integration. The area inside double borders can be developed by object-oriented modelling. The "Functions and data for task and user" is the user view and the "Shell providing functions and data" is the technical view of what is usually called the *functionality* of the computer.

press several of the claims for functional integration in a simpler way.

In structural semiotics, a *sign* is a relation between the contents (the signified) and the expression (the signifier) (Andersen, 1990). The *contents* are what the sign is about, corresponding to the problem domain. The *expression* in computer systems is the data and their transformations.

An expression has a substance that acquires a form (Andersen, 1990, p.69). The *form* is the distinctive properties of the sign, while its *substance* is a particular way of realizing the form (Andersen, 1992, p.17). For example, black lines in the shape of letters may be a forms on a surface (the substance). The form is *realized* in the substance. The words "realize" and "realization" will be used to denote the technical meaning defined here.

> The contents in the problem domain may be traffic estimates of a road. The figures typed in the computer and appearing on the screen are the expressions that represent the traffic estimates. The figures are in the form of arabic numbers in a spread sheet. The substance is the keys (when typed) and the screen (when displayed).

Computers do not only store data, they change data as well. The realization relation between form and substance can be extended to the transformation of data.

> When deleting a road from the database of traffic estimates, the selection of the road and of the text `Delete` that appears in a menu constitutes the form of the transformation, while the pointing and clicking constitute the substance.

The realization relation between form and substance is also used for categories of signs. A class of objects representing a category in the problem domain can therefore be a form. Another example is a function available for users to invoke. When considering classes, user functions, or other parts of computer software, their *specification* may be regarded as the form. A program

that realizes a specification is called an *implementation.* Therefore, there is also *a realization relation between specification and implementation.*

The functionality specifies which data can be manipulated and by which functions, and various user interfaces can be implementations of the functionality. Therefore, the functionality is realized in the user interface.

The conditions for functional integration can now be expressed as follows:

1 and 2: **Data sharing** and **availability of functions** should be realized such that the users have to pay minimal attention to the implementation.

3. **Interface consistency:** The same functionality specified in different user programs should be realized in the same implementations.

There is a strong correlation between specification and implementation in computers. When an implementation is executed, it is highly predictable that the corresponding specification is followed, because errors tend not to appear at random.

The users also have to master the realization relation. They have to know that to make the computer fulfil a specific function, specific behaviour is required. This is part of the skills required to master a system.

However, people's knowledge and behaviour is not predictable to the same extent as computer processing. A user may push a button that triggers an unintended function. Therefore, there is a weaker regularity between the user's knowledge of the function and its implementation than between specification and implementation in the computer. Similarly, the relation between the form of a user's action and the substance of her/his behaviour is not as predictable as the realization relation of computers. Training and documentation should strengthen the users' knowledge of the relation between specification and implementation.

To be able to deal with only one concept, the realization relation is defined in general for both computer processing and people's behaviour to be a regular correspondence between form and substance of signs, data, transformations of data, and categories of these.

The framework derived from the realization of the user knowledge and of the computer is shown in figure 2. The representations of problem domains can be implemented in other objects, e.g. in objects which determine data formats.

3.2 Detailed Framework

The encapsulation concept captures the realization relation between specification and implementation for a class of objects (Andersen, 1992, p.16–17). An object providing data and functions for the user may be organized as an encapsulation of an interface part specifying data and behaviour, and a hidden data and procedure part implementing the specification.

However, encapsulation is not the only way realization is obtained. For example, the object providing data and functions may also be realized in a user interface object, which again may be realized in objects provided by a user interface management system. Sometimes this is referred to as layers of implementation in the computer. Even if the layers might be fuzzy, the realization of software must follow the pattern of a directed graph down to e.g. a relational database, operating system, and finally to hardware. The object orientation may be abandoned at any layer. Figure 3 shows the framework with realization down to physical matter.

When an object is realized in another object, the relation between them takes place through message passing, being calls of services, methods, procedures, or whatever mechanism for

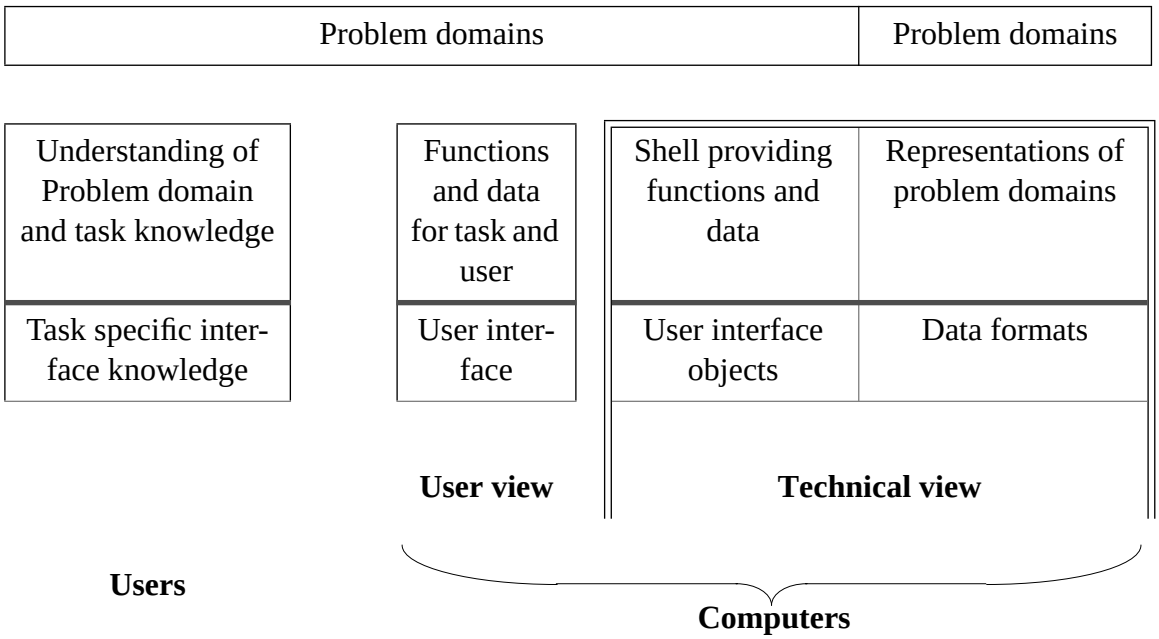

Figure 2: One layer of Realization in the framework for Functional Integration. The ——— line denotes the realization relation. The area inside double borders can be developed by object-oriented modelling.

transfer of program control and parameters the language may provide.

The realization relation may also be implemented through inheritance. For example, the objects providing functions and data may be programmed as subclasses under general user interface classes.

The realization relation can thus be realized through

- encapsulation,
- message passing (call service, method, procedure), and
- inheritance.

For the general framework, it is not significant which of these forms that is used. The property of the relation that is useful in object-oriented modelling and design is that several layers of realization can be separated, or that realization relations between classes can be described in a directed graph.

The realization of the functions and data available to the user can be regarded both from a user and from a technical point of view. The technical implementation may involve window objects, graphical objects, etc., and these may again be realized in the facilities of a user interface management tool.

Since the realization relation is extended to any number of layers in the technical view of computers, it may turn out useful to regard realizations along several layers from the user view as well.

Users may regard the realization of the interface in different ways. Even if the technical interpretation of the computer is object-oriented, the user evaluates a computer system in terms of its usability. The following four aspects may, e.g., be considered:

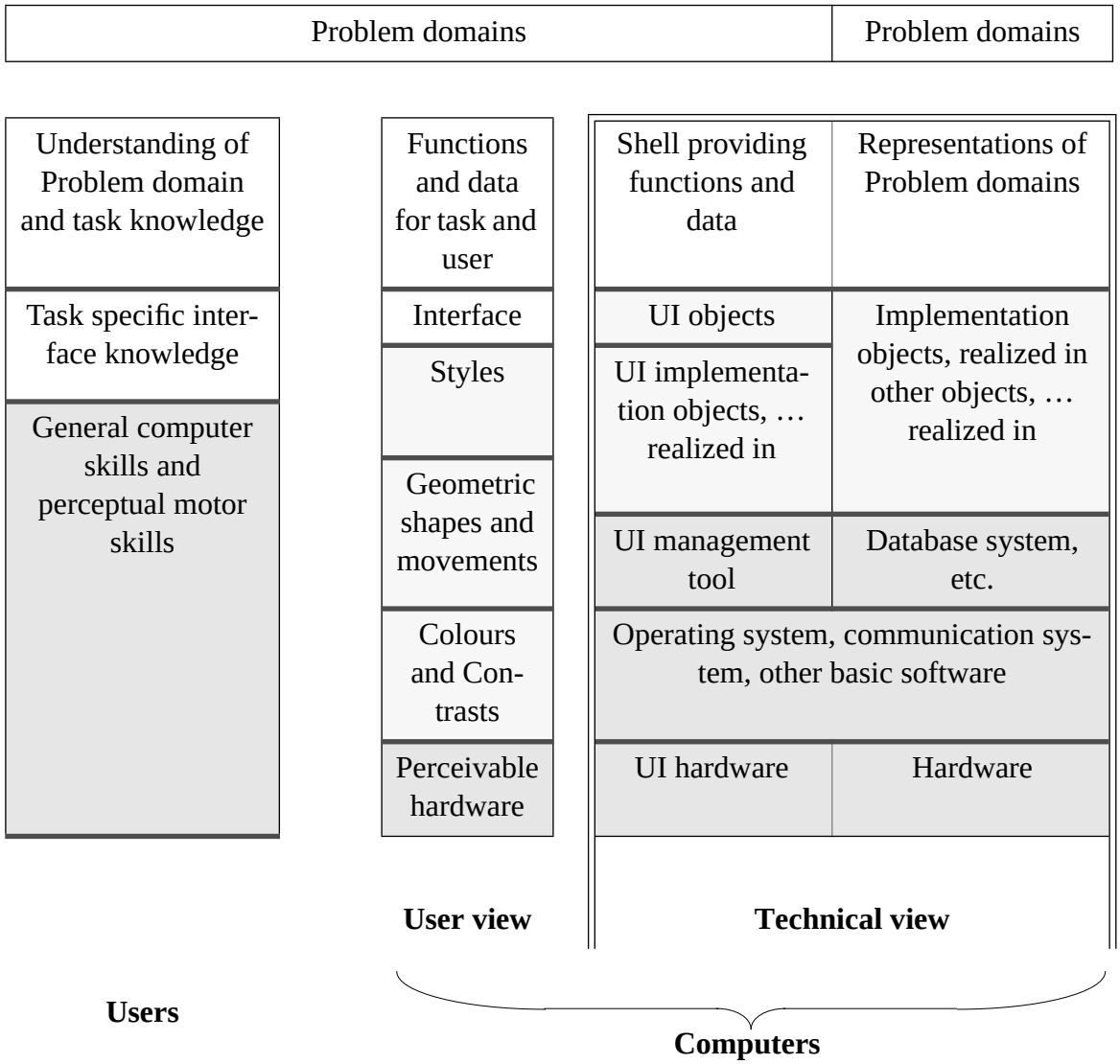

Figure 3: The Framework for Functional Integration. The ——— line denotes the Realization relation. Shaded areas of computers are independent of user programs. Shaded areas of users are independent of tasks. Light shaded are partly independent. The area inside double borders can be developed by object-oriented modelling.

(a) Styles. The expressions in the interface are realized in graphical and linguistic styles. If the styles of two user programs differ, the interfaces are not consistent. Windows may be realized in specific styles, e.g. Motif, OpenLook, Mac.

(b) Geometric shapes and movements. A style is realized in geometric shapes and movements, e.g., the lines and curves of a letter, the lines and patterns of a window, the speed of the cursor.

(c) Colours and contrasts. Shapes and movements are again realized in colours and differences between colours, i.e. contrasts.

(d) Perceivable hardware. The three above-mentioned aspects are simply ways in which properties of perceivable hardware are interpreted.

Several issues of user competence are also considered. It is not assumed that these should be described in any object orientation way.

Users have detailed knowledge of the functionality, interface, styles, shapes, colours, and hardware they use frequently. They also have general understanding of computing, and they have general skills in speech, writing, pattern recognition, perception, detailed movements, etc. Even if it is not possible to identify realization relations between these skills in general, there may be correspondence between, e.g., a certain meaning, a written expression, its shape and colour.

4 Evaluation of the Proposed Framework

The framework is intended to cover the issues of functional integration. This claim will be evaluated for each of the three conditions.

1. Data sharing

Data integration

> In the example of Tora and Henry, data is to be transmitted between the property database and the case file. One desired function is to get the area of a property. The property database can be considered to have an object for each property, with a service that returns its area. The case-file system could have an object for each case, and each case object contains one old property object and any number of new property objects, illustrated in figure 4. The object that supports Tora could be a specialization of the property object in the case file, with an added service for fetching the area from the property database.

The condition for functional integration was also that the users should have to pay minimal attention to the user interface.

> In this data integration example, the area could be fetched from the property database automatically upon generation of the case file objects. If user control is desired, there could be a Calculate choice in a menu, like the one indicated in figure 4.

Context-free data sharing

> Tora and Henry also copy and paste frequently. Tora moves data between the case file and her spreadsheet, and Henry copies from the case file into his traffic estimates data.

Copy and paste is a way to transfer data by means of a cutbuffer in a user interface management tool. It has the benefit of being independent of particular user programs when their interface is realized by means of the tool.

> Henry has a more complex way of copying and pasting than Tora, because he has to use two user interfaces. He can copy from the case file simply by dragging the text. The text is placed in a cutbuffer, which he can paste by the `Paste`

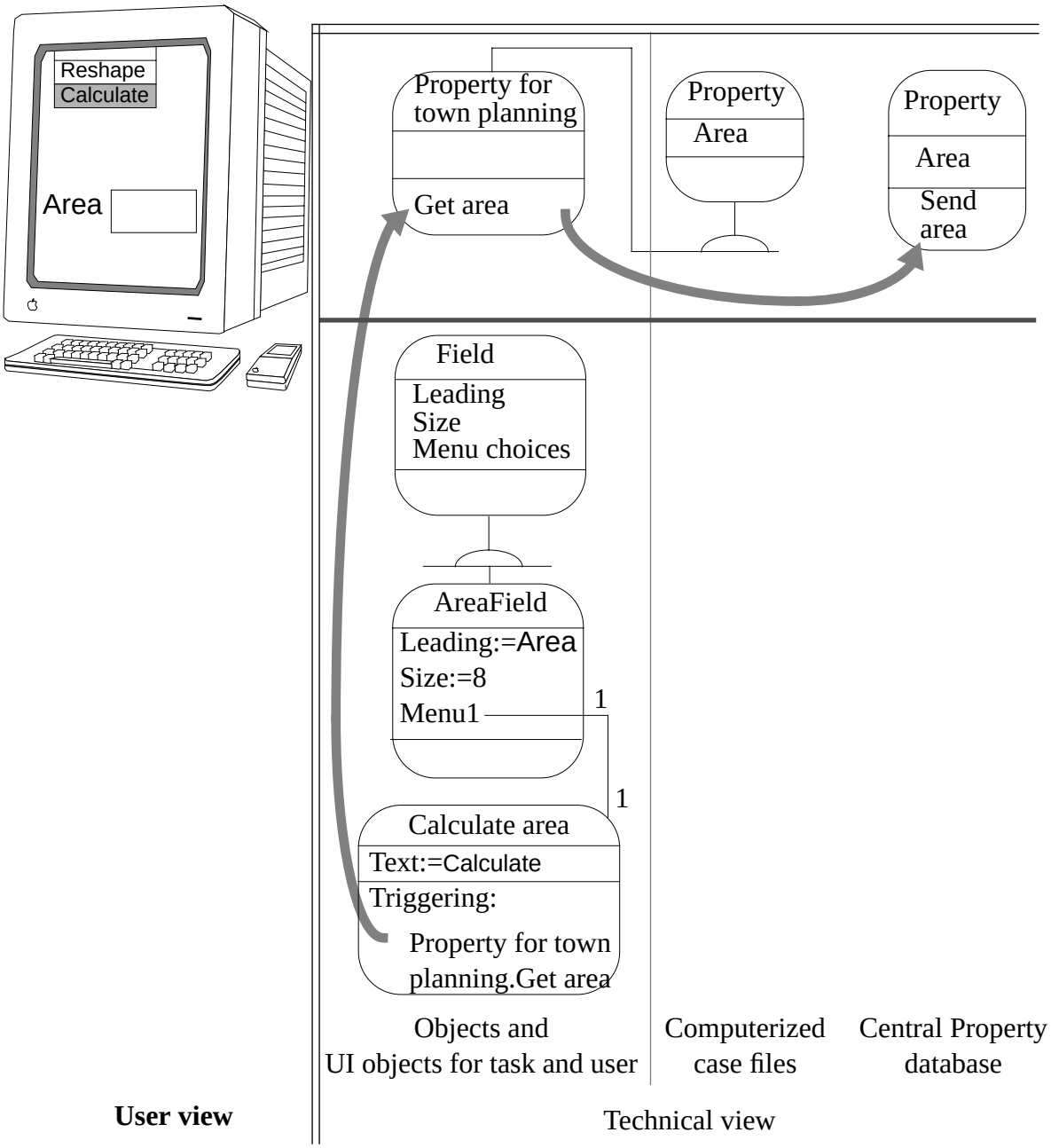

Figure 4: The objects involved in retrieving area data from the property database to the case file system. The objects below the dotted line are the user interface objects. Coad and Yourdon's notation is used. User and Problem domains are omitted from the figure.

command in his traffic estimate program under Fine, illustrated in figure 5. The text can also be pasted in United programs with command/click.

The figure suggests that there is no message passing directly between the user programs. The data to be transferred is passed through two cutbuffers with which the programs can interact. Even if the two user programs had used a single cutbuffer, the separation would still have existed.

The total separation of the programs implies that changes made to them will not affect their

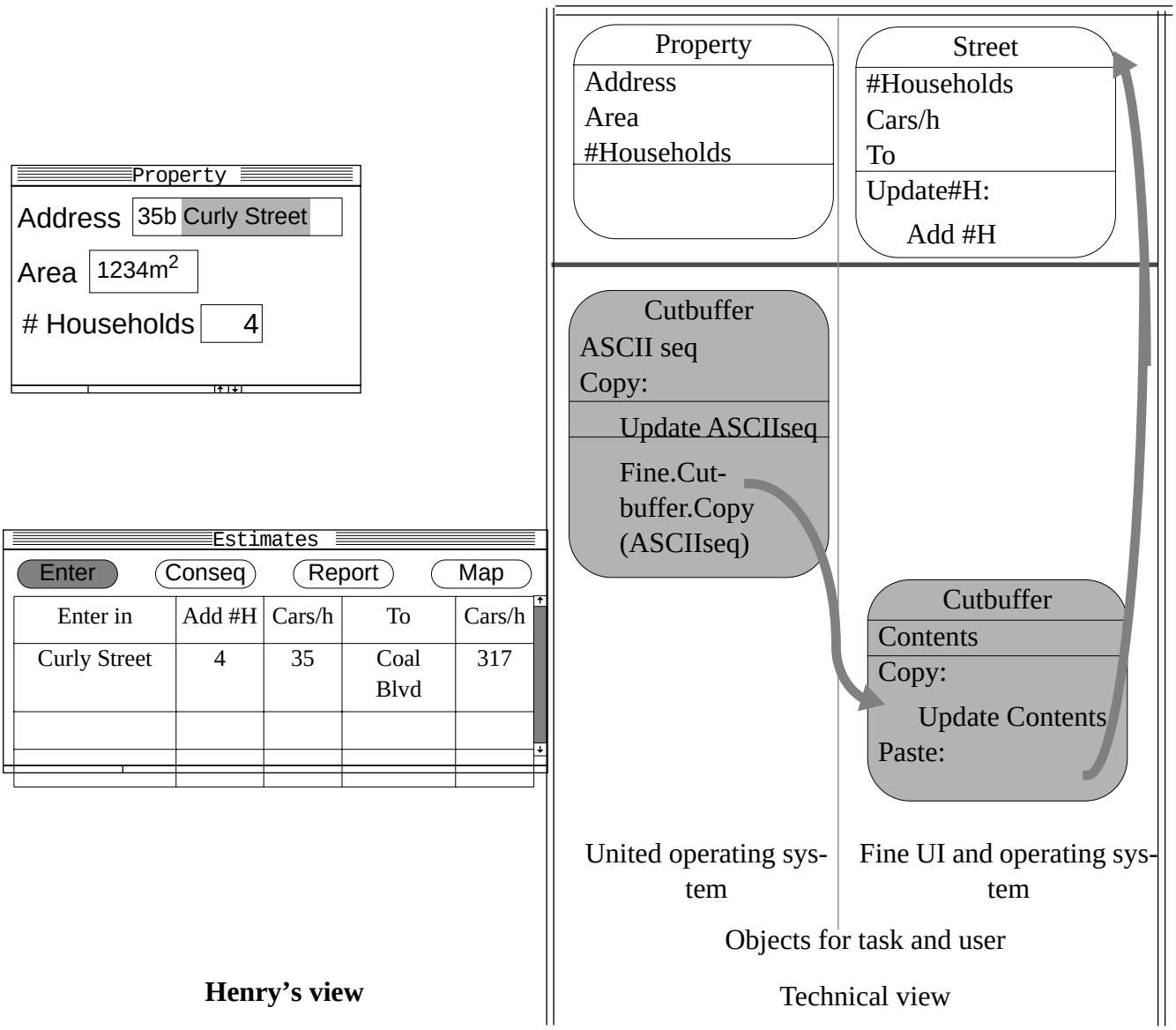

Figure 5: Copy and Paste. The data is transferred from one program to another through cutbuffers in the window systems. The two user programs are completely independent of each other. Intermediate objects and message passing are omitted for clarity. The user, problem domains, and representation of problem domains are omitted from the figure.

ability to transfer data in this way. However, the cutbuffers allow only restricted formats, which means that some data structures get lost.

In the case of two buffers that are operated with different keys and that provide different functionality, the user interface should also reflect this. One of the golden rules for user interface design is to "minimize the memory load on users" (Shneiderman, 1992, p.79). Having to handle two invisible cutbuffers, each with its own functionality and interface (keystrokes and mouse movements) violates this rule. The memory load could be lessened if the buffers were visible together with the codes for handling them. Greater emphasis on training should also be considered in this case.

2. Available functions

Assuming an object-oriented method which allows each object to have its own action sequence, each functionality object can have its own process underway or suspended. Each process could also have, e.g., a window interface object attached, enabling the user to perceive and manipulate each process. Current window technology enables switching between windows by moving a mouse, etc.

Access to remote computers or restricted data could also be available as separate processes, requiring passwords to open, but otherwise in line with the claims for minimal user action.

3. Interface consistency

Two user programs will, in a general sense, have some common functionality, e.g., to find an item that satisfies certain conditions; to insert an object; to provide assistance; to modify text. To support transfer of learning, it should be possible to implement equal functionality in the same user interface.

Interface objects of the same class that can be used by several user programs will provide sufficient consistency. A user-interface management tool could have classes for buttons, fields, tables, etc.

Several “layers” of implementation are indicated at the User view: styles, geometric shapes and movements, colours and contrasts. User-interface objects can be designed for any such aspect. It may be useful to vary the consistency in any of these aspects, e.g., to separate the presentation of two programs by using different colours.

This demonstration shows that the three conditions for functional integration can be handled in the framework.

5 Current approaches

Other frameworks for object-oriented modelling have also been proposed. A recent comparison of object-oriented analysis and design methods identifies 28 key aspects of the methods (Monarchi and Puhr, 1992). The methods differ in terms of which parts of the world they are being used to model. The aspect of identification of user interface classes was found in three of the 23 methods in the comparison. The coupling of functionality and user interface was not mentioned in the comparison of methods.

One of the three methods addressing user interface covered the development process only, and not the OOA/D product. Since user-interface properties are central to functional integration, only the two approaches covering user-interface classes will be considered further (Coad and Yourdon, 1991a and b; Iivari, 1991). An additional method not covered in the comparison addresses user-interface properties (Mathiassen et.al., 1993). This method will be considered as the third approach.

5.1 Coad and Yourdon

Coad and Yourdon propose an object-oriented analysis of the Problem Domain, which is the

part of the world that the data should represent (Coad and Yourdon, 1991a). The result is a Problem Domain Component, consisting of objects, which will represent the problem domain. However, when it comes to object-oriented design, Coad and Yourdon refer to the Problem Domain Component as the structure of the objects inside the computer (Coad and Yourdon, 1991b). No explanation of this shift of interpretation is given.

The object-oriented analysis should also consider the "Systems responsibilities: an arrangement of things accountable for, related together as a whole" (Coad and Yourdon, 1991a, p.9, 53). This seems to be functional requirements derived from the tasks which the computer system is supposed to support.

In object-oriented design, three more components are added to their framework, see figure 2. These are the Human Interaction, the Task Management, and the Data Management Components. In addition, Coad and Yourdon propose to analyse users' Tasks and Skills.

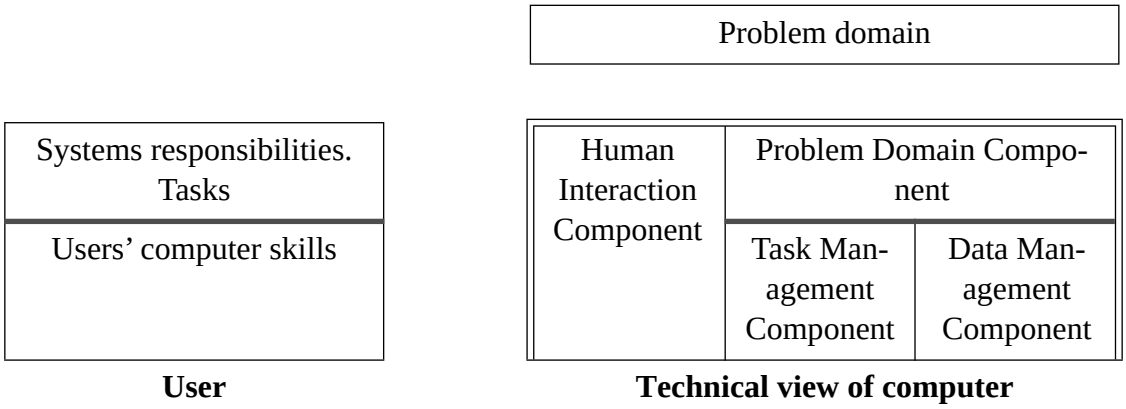

Figure 6: The Framework of Coad and Yourdon (1991a and b). The issues inside the double border are parts of the object-oriented model

The design of the Human Interaction Component starts by categorizing the users and investigating their Tasks and Skills. The design includes selecting the functions, organizing them in a menu hierarchy and formatting them according to principles of interface design (Coad and Yourdon, 1991b, pp.56–64). The Task Management Component should identify events and control the program execution. The Data Management Component creates the relation between the Problem Domain Component and the data management structure in which the data are stored.

Evaluation

Data formats can be modelled in the Data Management Component, and this is satisfying for the technical support for data sharing. Coad and Yourdon have one Problem Domain in their framework. This does not adequately represent one user who integrates some domains covered by some systems and other users who integrate other overlapping domains. Data integration is therefore not covered sufficiently.

The Problem Domain Component seems to be realized in the Task Management Component and the Data Management Component. The Task Management Component provides sequencing of program components. Therefore Coad and Yourdon's method can provide support for program availability.

The interface component of Coad and Yourdon also includes aspects of functionality, since it is determined by selecting the appropriate functions for given tasks. Therefore, the realization relation between functionality and user interface is hidden inside the Human Interaction Component.

Coad and Yourdon also address the issues of tasks and user knowledge of computers. However, this is not included in any systematic manner in their framework. Besides, it is not clear how their System Responsibility interferes with their Tasks. System Responsibility is considered here as the total need for functionality in the user organization, while the Tasks are the individual work operations, each making a contribution towards the need for functionality.

Coad and Yourdon's framework does include data sharing between systems, even though their framework has a component for data implementation. The other issues of functional integration are covered. However, the task and the user knowledge issues are treated superficially, and the functionality is not separated from the user interface. In addition, the user view of the computer is omitted.

5.2 Iivari

Iivari proposes a framework for design of application systems. Three "levels of abstraction," the organizational, the conceptual, and the technical levels constitute the core concepts of the framework (Iivari, 1991).

The organizational level covers the organizational context and the users. The required functionality of the computer system, the IS Use Acts (Iivari p.210), is derived from knowledge of the users, see figure 7.

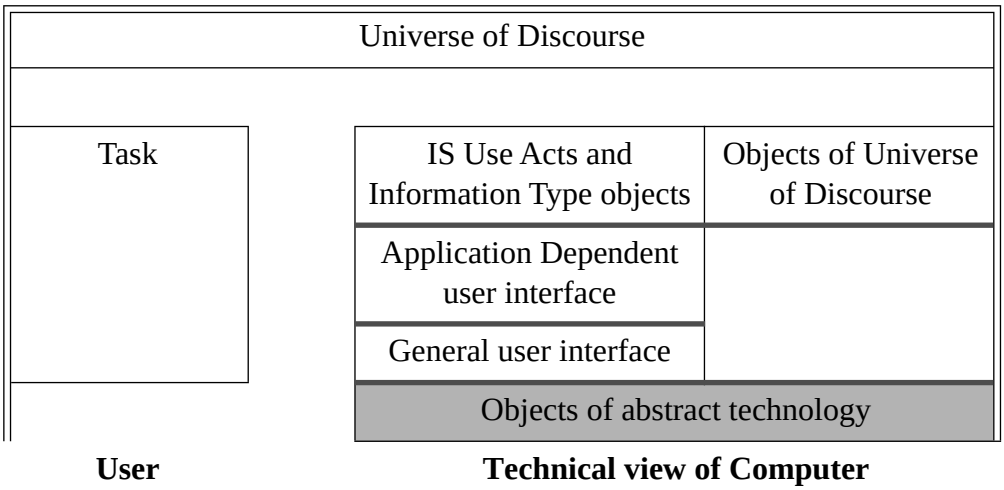

Figure 7: The framework of Iivari (1989 and 1991).

The conceptual level contains the Universe of Discourse, the IS specifications, and the User interface. The Universe of Discourse is the problem domain, which is used to identify software objects. The Information Type objects are intended for handling input/output, queries, etc. This seems to express the functionality, which is also covered by the IS Use Acts. However, in an analogy with structured analysis, Iivari classifies every data store, flow and transformation under the Information Type objects. This may indicate that he also includes some internal processing between the Objects of the Universe of Discourse. His example shows a "Customer-order-info" object of the Information Type, which is to provide output. This object seems to be

a response to an IS use act. The Information Type object is therefore interpreted as a more detailed specification of the IS Use Acts.

Iivari also includes an Application Dependent and a general User Interface Component (Iivari 89). There are also Objects of abstract technology in the Technical level. The latter are intended to cover basic software, and these objects are not included in his example.

The objects may be concurrently active.

Evaluation

Iivari also has only one Universe of Discourse, prohibiting data sharing between systems. The data sharing inside the Objects of the Universe of Discourse seems to follow the same principle as the framework developed to take care of functional integration.

Iivari's objects may be concurrently active, but it is not clear whether they have internal action sequencing. The concurrently active objects support program availability.

The separation between IS Use Acts and Information Type objects on one side, and User Interface on the other, makes up for the realization relation. It seems possible to achieve interface consistency by making functions of the same class be presented by the same interface object.

Iivari's framework seems to lack objects that determine the format of internal data. The user's view of the interface is also missing. No concepts for users' skills were found. Data sharing between systems is not covered in the framework.

5.3 Mathiassen et.al.

Mathiassen, Munk-Madsen, Nielsen, and Stage (1993) have developed a framework for object-oriented analysis and design. In addition to attributes and procedures (services), their objects contain actions, in the tradition of Jackson System Development (Jackson, 1983) and the programming language Beta (Kristensen et.al., 1987).

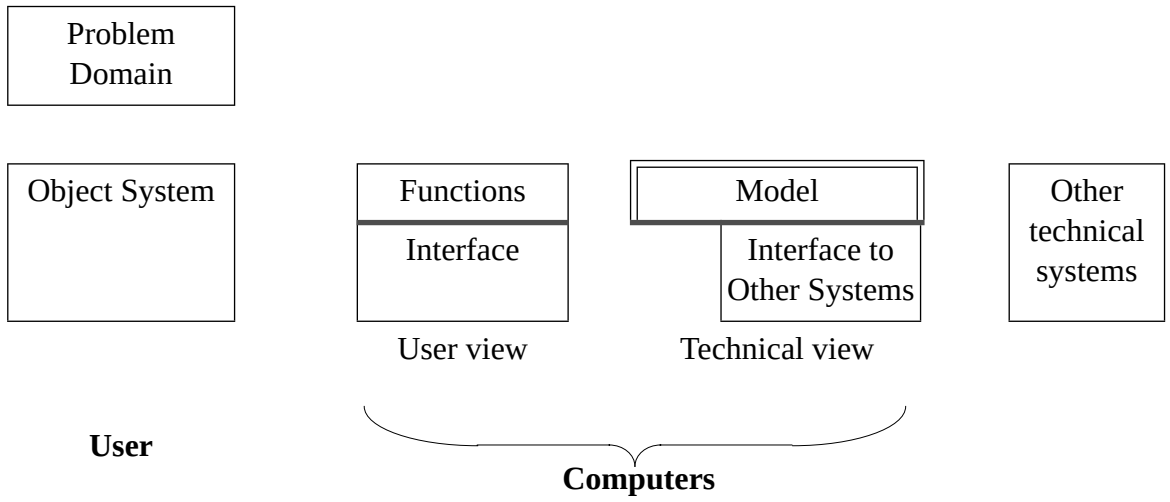

Figure 8: The Framework of Mathiassen et.al. (1991). The Object System is the users' interpretation of the Problem Domain.

The framework is illustrated in figure 8. One Problem Domain occurs. Mathiassen et.al. include both the Problem Domain and the users' interpretation of it (the Object System) in their

framework. During analysis, an object-oriented model of the Object System should be built. This model is used to structure the Model (of the Problem domain) in the computer during design. Analysis also includes requirement specification of Functions and Interface.

The Interface differs from Coad and Yourdon by including both user interface and interface to other technical systems, e.g., sensors and other computers. It is not specified how the interface to other systems should be built. The Functions and the User Interface are seen from a user perspective. It may well be that the Functions and the Interfaces can be modelled object-oriented when determining their technical properties.

Mathiassen et.al. also mention the relation between the Problem Domain and the Application Domain, which is the users and the computer system together (Mathiassen et.al, 1993, p.7–9). This relation seems unclear in the framework, however.

Evaluation

Mathiassen et.al. also lack the Task Management Component. However, each object in their method has its own action sequence, such that the objects in the Model can handle multiprocessing without external control.

The separation between Function and Interface makes up for the realization relation in the same way as for the framework for functional integration.

Data is shared through the Interface to Other Systems. In the framework for functional integration, several representations of problem domains could be included. To achieve functional integration with the Mathiassen framework, the coupling to other systems could be utilized. Their Interface to Other Systems can include data formats for data transfer too.

The framework seems to lack objects that determine the format of internal data, and it also lacks concepts for users' computer skills.

5.4 Summary of Existing Frameworks

None of the existing frameworks cover every aspect of functional integration. Coad and Yourdon's method includes users' computer skills. Iivari includes basic software into his framework. Mathiassen et.al. bring remote communication into their framework. Iivari and Mathiassen et.al. also have the advantage of basing their method on concurrent objects, thereby avoiding additional objects for control.

6 Conclusion and Further Research

The Framework

The conditions for functional integration were:

1. Data sharing: data integration and context-free data sharing should be possible with minimal operation by the user and without having to pay attention to data formats.
2. Available functions: immediate access to several programs, making the programs remain in any desired state after resuming them, with minimal operations by the user.

3. Interface consistency: It should be possible to implement the same functionality in different user programs in the same interface patterns.

It is argued and demonstrated by an example that the proposed framework handles these conditions for functional integration. It is necessary to include user knowledge in the framework to provide a background for the design of training and documentation. However, there is no correspondence between object-oriented modelling of the computer system and the way user knowledge should be treated.

There is no obvious obstacle to adapting current object-oriented methods to the proposed framework. Some of the methods, e.g. Coad and Yourdon (1991 a and b) and Mathiassen et.al. (1993) suggest sequences and iterations in analysis and design corresponding to their frameworks. Therefore, it would be necessary to determine where in the methods the elements of the proposed framework should be included.

The Realization Relation

Even if encapsulation describes the realization relation, there is no specific notation or mechanism in existing object-oriented methods to support the relation in general. Ideas are found in the ensemble/role concepts in Beta (Kristensen et.al., 1987). Since the realization relation supports modularization, it seems worthwhile to investigate how it should be precisely defined.

Context free Data Sharing

Several cutbuffers appear when several operating systems and window systems are integrated on a computer. Making cutbuffers visible on the screen is a way to reduce memory load. Practical experiments with the appearance and form of cutbuffers should be carried out.

Data integration vs. syntactic data sharing may be a choice to make in development projects. Goodhue and Wybo (1992) suggest that data integration could be counterproductive between organizational units that differ substantially, and that greater instability makes the situation even worse. Since copy/paste is independent of the data definitions, this could be solutions to claims for data sharing when data integration is not wanted. Empirical evidence should be gathered on this problem.

7 References

Andersen, P.B. (1990) *A Theory of Computer Semiotics: Semiotic approaches to construction and assessment of computer systems* Cambridge University Press, Cambridge

Andersen, P.B. (1992) “Computer Semiotics” *Scandinavian Journal of Information Systems* Vol.4, pp.3–30

Braa, K; Bratteteig, T.; Kaasbøll, J.; Smørdal, O. and Øgrim, L. “Barriers and Triggers for Development for Functional Integration— A case study” In Bansler, Bødker, Kensing, Nørbjerg, Pries-Heje (eds.) *Proceedings of the 16th IRIS* Rapport Nr.93/16, Department of Computer Science, University of Copenhagen, 1993, pp.361–375

Coad, P. and Yourdon, E. (1991 a) *Object oriented Analysis* 2nd Edition, Yourdon Press, NJ

Coad, P. and Yourdon, E. (1991 b) *Object oriented Design* Yourdon Press, NJ

Fagermoen, F.E.E.; E.M. Lund; L.E.Mathisen; L.N.Rørvik; and H. Østgaard (1992) *External Databases at Stabekk Social Security Office* (In Norwegian) Student Report no.36, Department of Informatics, University of Oslo

Goodhue, D.L.; Wybo, M.D.; Kirsch, L.J.(1992) "The Impact of Data Integration on the Costs and Benefits of Information Systems" *MIS Quarterly*, September 1992, pp.293–311

Grudin, J. (1989) The Case Against User Interface Consistency *Communications of the ACM* 32, 10, pp.1164–1173

Hannemyr, G. (1992) *Open Systems: Technology, strategy, and practice* (In Norwegian) Universitetsforlaget, Oslo

Iivari, J. (1989) "Levels of Abstraction as a Conceptual Framework for an Information System" In E.D. Falkenberg and P. Lindgreen (eds.) *Information System Concepts: An In-depth Analysis* Elsevier, North Holland, Amsterdam, pp.323–352

Iivari, J. (1991) "Object-Oriented Information Systems Analysis: A framework for object identification" In Proceedings of the Twenty-Fourth Annual *Hawaii International Conference on System Sciences* IEEE, pp.205–218

Jackson, M. (1983) *System Development* Prentice-Hall, New Jersey

Kaasbøll, J. and Øgrim, L. (1989) *Memo on the flow of cases in the technical departments in the Municipality of Oslo: Can Norsk Data's "Task Flow" be used for cases concerning division of property?* (in Norwegian) Internal memorandum, Department of Informatics, University of Oslo

Kaasbøll, J., Braa, K. & Bratteteig, T. (1993) "User Problems Concerning Functional Integration in Thirteen Organizations" In D. Avison, J.E. Kendall, and J.I. DeGross (eds.) *Human, Organizational, and Social Dimensions of Information Systems Development* IFIP Transactions A-24, North Holland, Amsterdam, pp.61–81

Kristensen, B.B.; Madsen, O.L.; Møller-Pedersen, B.; Nygaard, K. (1987) "The BETA Programming Language." In Shriver, B.D. and Wegner, P. (eds.) *Research Directions in Object Oriented Programming* MIT Press

Marche, S. (1993) "Measuring the stability of data models" *European Journal of Information Systems* Vol.2, No.1, pp.37–47

Mathiassen, L.; Munk-Madsen, A.; Nielsen, P. A.; and Stage, J. (1992) "Modelling Events in Object-Oriented Analysis" In Bjerknes, Bratteteig & Kautz (eds.) *Precedings of the 15th IRIS* Department of Informatics, University of Oslo, pp.742–757

Mathiassen, L.; Munk-Madsen, A.; Nielsen, P. A.; and Stage, J. (1993) *Object Oriented Analysis* (In Danish) Aalborg, Forlaget Marko

Monarchi, D.A. and Puhr, G. I. (1992) "A Research Typology for Object-Oriented Analysis and Design" *Communications of the ACM* Vol.35, no.9, pp.35–47

Newman, W. (1986) *Designing Integrated systems for the office environment* McGraw-Hill, Singapore

Nielsen, J. (ed.) (1989) *Coordinating User Interfaces for Consistency*. Academic Press, Boston

Nierstrasz, O.; Gibbs, S.; and Tsichritzis, D. (1992) “Component-Oriented Software Development” *Communications of the ACM* Vol.35, no.9, pp.160–165

Rosenberg, D. (1989) A Cost Benefit Analysis for Corporate User Interface Standards: What price to pay for a consistent “look and feel” In: Nielsen, 1989, pp.21-34

Shneiderman, B. (1992) *Designing the User Interface: Strategies for Effective Human-Computer Interaction* Second edition, Addison Wesley, Reading, Mass.

Toft, J. H.B. (1992) *From Old to New Computer Systems: — Data modelling, prototyping, conversions, and training* (Master thesis in Norwegian) Department of Informatics, University of Oslo