

# Working with objects in the user interfaces

## Trygve Reenskaug

### Taskon, Oslo

My first exposure to personal information systems was in 1970 when I heard Douglas Engelbart talk about computer augmentation; the idea that computers could provide an extension of the human intellect. I had then devoted ten years of my life developing a system for the computer aided design of ships. It was a shock to realize that a successful system for the design of ships was inadequate as a system for ship designers. Our product provided a basic shipbuilding service supporting the description of the ship and all its parts. The ship designer needs much more; he works with scheduling and building contracts, he interacts with shipowner and classification society, he deals with materials management and production technicalities. Inspired by Douglas Engelbart, I saw the dichotomy between personal information tools tailored to the tasks performed by the individual; and the business services dedicated to supporting functions such as accounting, scheduling, materials management, and design.

I have spent the 25 years since 1970 exploring this dichotomy. The total information system should support people in their individual tasks with intuitive and easy to use information tools; and at the same time manage shared information so as to protect its integrity and make it available to all concerned. To achieve this, three important requirements need to be satisfied. The first requirement is that the system shall appear *intuitive*. This implies that the user's mental model must correspond to the information model exhibited by the system. The second requirement is that the system shall be *easy to use*. This is satisfied by tailoring presentations and operations to the user's tasks and preferences. The third requirement is the need for a *distributed architecture* where the task oriented information tools are integrated in personal work environments in the foreground, and shared information in the form of business objects are managed by a battery of services in the background.

Our solutions to these three requirements are all heavily based on object technology. 'Intuitive' systems are realized by user and system sharing an object model; 'ease of use' by letting the user see and manipulate objects on the screen; and the 'distributed architecture' is expressed as a pattern of interacting objects.

The common sense of objects is to describe phenomena as structures of interacting objects. Separation of concern is a valuable technique for understanding complex systems. If the whole is too complex, we select any *area of concern* and describe the *roles* objects play in that context. Conceptually, a role model resembles an object model. The difference is that the role model is a filtered representation of an underlying, more complex object model. Role models are orthogonal to the popular class models. The main difference here is that the class model focuses on individual objects, while the role models focus on object interaction. A class model could say that Numbers can be specialized into Integers and Reals, and that Numbers are related to other Numbers through Operations. A role model would describe structures of Numbers. For example, a Number playing the role of Fraction could interact with other Numbers playing the roles of Numerator and Denominator, but it could equally well interact with Expression objects playing the same roles.

Understanding the business world as a whole is impossible; but we can still think of it as a very large structure of interacting objects. We then isolate any part of it and describe this part in a role model. Objects and roles are applicable to all levels of business modeling, including modeling of enterprise and processes, tasks and procedures, information systems and data warehouses, and distributed systems such as the World Wide Web.

*First requirement: Intuitive personal information environments.*

In his book *Mind Matters*, the brain scientist Michael S. Gazzaniga claims that our brains are equipped with an interpreter module that tries to make sense out of our sensory input. Indeed, dreams could be the output from this interpreter in response to random neurological noise. {Houghton Mifflin Company 1988. ISBN 0-359-42159-4}. This book is well worth reading for its own sake. Its relevance to our current discussion is that the user will always try to create a mental model. Bizarre models can appear if the interface designer does not provide a nice and intuitive one that the user can readily appreciate. (One example is a user who firmly believed that if his computer got stuck, banging the table would help loosen it up. Reality was that the computation lasted slightly longer than the user's patience.)

The object idea is simple, yet powerful. Reflected users seem to grasp relevant object or role models with ease. Separation of concern permits the creation of models for each specific user. To the user, "his" models appear as a fully rounded, intuitive models that he manipulates directly, or that he manipulates indirectly through a mechanism such as the Model-View-Controller described below. Alan Kay once said that "all computations are illusions". It is the responsibility of the system implementor to ensure that the illusion of concrete models experienced by the user is faithfully reproduced by the programs. Using an object oriented programming language, this is fairly easy in the trivial cases when there is one model common to all. It is somewhat harder in sophisticated systems where different users need to see different models. The total computer system is then specified as a composite; and its classes are defined so that their instances play the roles needed to support the different users.

*Second requirement: 'Easy to use' information environments.*

Over the past ten to fifteen years, we have seen several clumsy approaches to meeting this requirement. The term "user interface" has been defined in terms of basic program library widgets such as menus, lists, and text fields. This is far too primitive; a tool needs a combination of widgets facing the user and a task oriented program that links the widgets to the background business services. *Application* is the term that ParcPlace-Digitalk use for such task oriented programs, and we will use the same term here.

One of the great inventions of the Smalltalk group at Xerox Palo Alto Research Center (PARC) in the seventies was the idea that objects can be made visible on the computer screen so that the user can see and manipulate them directly. This makes the abstract computer data appear concrete and the underlying object model visible. The user can easily adjust his mental model to this computer model and operate on it with confidence. The well-designed direct manipulation object interface is intuitively obvious and therefore easy to learn for the uninitiated.

This strength of the direct manipulation object model is also its main weakness. Each object can only appear once on the screen and must always be presented in the same way to preserve the illusion of concreteness. This is insufficient for large and complex models where we need to view objects in different ways. The Model-View-Controller paradigm extends the power of the user interface at the expense of increased demands on the user's mental model. There are four roles in this user interaction paradigm. The human *User* has a mental model of the information he is currently working with. The object playing the *Model* role is the computer's internal and invisible representation of this information. The computer presents different aspects of the information through objects playing the *View* role, several Views of the same model objects can be visible on the computer screen simultaneously. Objects playing the *Controller* role translate User commands into suitable messages for the View or Model objects as needed.

*Third requirement: Distributed architecture provides personalized task support while preserving centralized management of shared information.*

Client-Server technologies has been in the vogue for many years. Two architectures have been

proposed. One has a lightweight client only containing the user interface widgets; concentrating all application and business logic in the server. The other moves the application and business logic to the client, reducing the server to a database engine. Neither solution supports the basic need for a many-to-many relationship between the task oriented information tools and the business oriented services. Our industry is therefore moving to a four tier architecture, where the widgets and the task oriented applications are in the clients; while the business objects together with the databases are in the servers. But the four layers is an unnecessary complication that only serves to camouflage the unfortunate architectures of the early client-servers. Two main layers separating the task-oriented, personal information environments from the background business services is all that is needed. The further subdivision of these two layers are details that can be chosen freely in the clients and the servers respectively.

If you want to know more about the issues discussed in this column, you could read my new book *Working with objects*. (Manning/Prentice Hall 1996. ISBN 0-13-452930-8). You can reach me at [trygve@taskon.no](mailto:trygve@taskon.no).