1 Introduction

Mental models and data

This article is about people, more specifically about people who understand and even specify how their computers shall achieve their tasks. Their occupations range from occasional users through schoolchildren to professionals in business and industry. Our focus is on making computer programming accessible to people at large by matching the end user's mental model to what happens in the computer. This makes it imperative to distinguish clearly between what is in the user's mind and what is represented in the computer. These IFIP definitions of 1966 have withstood the test of time:

> "DATA. A representation of facts or ideas in a formalized manner capable of being communicated or manipulated by some process."
>
> "INFORMATION. In automatic data processing the meaning that a human assigns to data by means of the known conventions used in its representation."

There is no information in a computer system, not even in the World Wide Web; there is only data. Any human who can find the data and understand the conventions has reached the first stage of computer literacy.

DCI

We propose a new programming paradigm that we call *DCI, Data, Context, and Interaction,* to close the gap between mind and computer. Its goal is to make programming easy to learn and understand for the beginner while it scales up to satisfy the sophisticated needs of the expert. This is not a trivial goal. Indeed, in the introduction to the Design Patterns book [pp. 22-23], the authors write: "*it's clear that code won't reveal everything about how a system will work.*" It is frightening to read that there are mission critical systems in use today where the code does not reveal how the systems actually work. The end users are not alone in their illiteracy; even system maintainers and other experts have problems understanding what goes on in the computer.

> *This problem challenges us to find a way to write code that clearly expresses the system's runtime behavior.*

Any human who can read and write such code has reached the second stage of computer literacy. We claim that the DCI paradigm can form the foundation for our community at large to reach this stage.

Critical events

DCI builds on the cumulated history of the human use of computers. Three critical events are particularly relevant. The *first event* happened on the 21st June 1948 at the Victoria University of Manchester, England when the world's first stored program computer executed its first instructions. The *second event* was in May 1981 at the National Computer Conference in Chicago, Ill, when Xerox unveiled its new user interface with a desktop and a direct manipulation interface. The *third event* is still in the future. It will be the birth of Personal Programming (PP) which will empower users to program their Personal Computer as naturally and simply as they use laptops. tablets, and smartphones today.

Stored program computer

The appearance of the stored program computer in 1948 marked the birth of software. A program is stored as a bounded chunk of data in a computer's memory so that it can be processed as any other data. This facilitates programming languages and their attendant support software. A program is manifested as a bounded chunk of text called its source code.

Communicating computers

The stand-alone computer of 1948 has gradually been augmented by connecting the computers through communication networks and the boundary of a program has gradually dissolved. The program is becoming a dynamic collection of interconnected capabilities and the whole is no longer controlled by an isolated chunk of source code. The Data consists of the available capabilities. A Context musters the capabilities needed to realize a given system behavior and its Interaction describes the algorithm that ties the capabilities together to achieve this behavior.

Business communication

Business organizations are held together by communication between its people. Most of them are equipped with their own computer. These computers are interconnected so that their owners can delegate some of their communication to their computers. Digital communication is now an essential part of the business processes. An example is a proposed system for distributed planning and control. (Section 3.1: *Prokon's Distributed System)*. The managers have a personal computer with their personal planning system that is tailored to the technology of their departments. These computers are connected in a network and a distributed algorithm ensures that the overall plan is consistent. There is no identifiable Planning Program; the logic was to be distributed in an open-ended structure of people and computers.The DCI paradigm supports this new approach to programming.

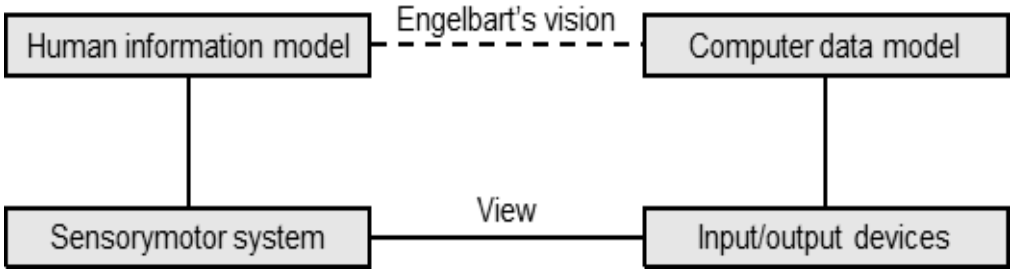
Direct manipulation interfaces

Before 1981, the prevalent style of human-computer interaction was the command-line interface where textual commands alternated with the computer's response. These interfaces put a heavy load on the user's memory and required a significant investment in learning time to master. Contrast with the direct manipulation interfaces. Douglas Engelbart first created this style of interaction as part of his strive for "making the computer an extension of the human intellect". He divided the screen into several windows and invented the mouse to be able to replace the remember-and-type style with see-and-select [NLS]. This style was refined at Xerox Palo Alto Research Center (PARC) through the sixties and culminated with our second event in Chicago. This marked the beginning of a new era when computing came within the reach of more than a billion people around the world. Engelbart's vision is now reality. The equipment is out of sight and there is an apparent direct contact between mind and data.

MVC

In retrospect, we can describe the new style of user interfaces by MVC, a conceptual model that describes the connection between mind and data. The user's mental model of relevant information is mirrored in the data *Model* stored in the computer. The user observes and interacts with these data through *Views* that connect the human sensorymotor system with the computer's I/O channels to bridge the gap between the human and the computer. (figure 1) We will use MVC to bridge the gap between the user's conception of a program and the actual program in the computer.
(Section 2: MVC - the Model, View, Controller Paradigm).

Figure 1: MVC bridges the gap between mind and computer.



Object

DCI is founded on Alan Kay's definition of object orientation: "*I thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages*" [a]. In this model, an object is an entity with a globally unique and immutable identity that encapsulates state and behavior. (Section 3.2: *The First Object)* and (Section 3.3: *Kay's Object Orientation)*. The definition says what an object oriented system *is*. DCI adds code that says what a system *does* when it achieves its goals through message interaction between participating objects.

OOram

OOram (Object Oriented Role Analysis and Modeling) describes how objects interact when realizing a use case. (Section 3.4: *OOram Role Modeling)*. OOram introduces the notion of the role that an object plays when interacting with other objects to achieve a use case. The word *role* is taken from a theatre metaphor; an actor plays a role when interacting with other roles in the performance of a play. OOram describes the messages (like cues) that flow between the roles during a performance. OOram does not say anything about what the objects actually do in response to the messages they receive. We go a step further in DCI where we equip the roles with scripts that specify how they fulfil their responsibilities as participants in an Interaction (like in the performance of a play). Communication is now a first class citizen of programming. (Section 4: *DCI, the new Programming Paradigm)*.

Fuzzy program model

The notion of a program as a closed and static chunk of code can now be augmented by the dynamic notion of system behavior as an interaction between communicating objects. The contours of a program becomes hard if not impossible to describe in closed form. What used to be application programs with their closed plethora of capabilities can now more fruitfully be extended to an open Model that contains all the objects that are available at a given time. Different Views on this Model specify how selected objects interact at runtime to realize different system operations. Personal programming can be achieved by inviting a person to specify a new operation through a View on the Model.

Example

An example illustrates our vision. Consider a woman, Ellen, who plans a long hike for the morrow. She wants to start early, and to sets the alarm to wake her at 06:00 but only if the meteorologists forecast a dry day. She needs an ad hoc program to do this, and opens an empty workspace on her new program (figure 2).

Ellen's objects

Ellen then considers the capabilities she needs to get the job done. Fist, she needs her program to wake up at six o'clock. She pokes around the built-in services on her device and finds a Time object that can do the job. She drags it into the workspace and calls it the Clock role. Second, she needs to check the weather forecast. It takes some time with the internet browser to find an object in the local weather service that will give the expected precipitation. She drags it into the workspace and calls it the Forecaster role. Third, she needs to activate the alarm that sits on her bedside table. She finds the alarm as an object in her local Internet Of Things, drags it into the workspace, and calls it the Alarm role.

---

a. Alan Kay's History of Smalltalk

Figure 2: Ellen's alarm clock workspace; her ad hoc Personal Program:

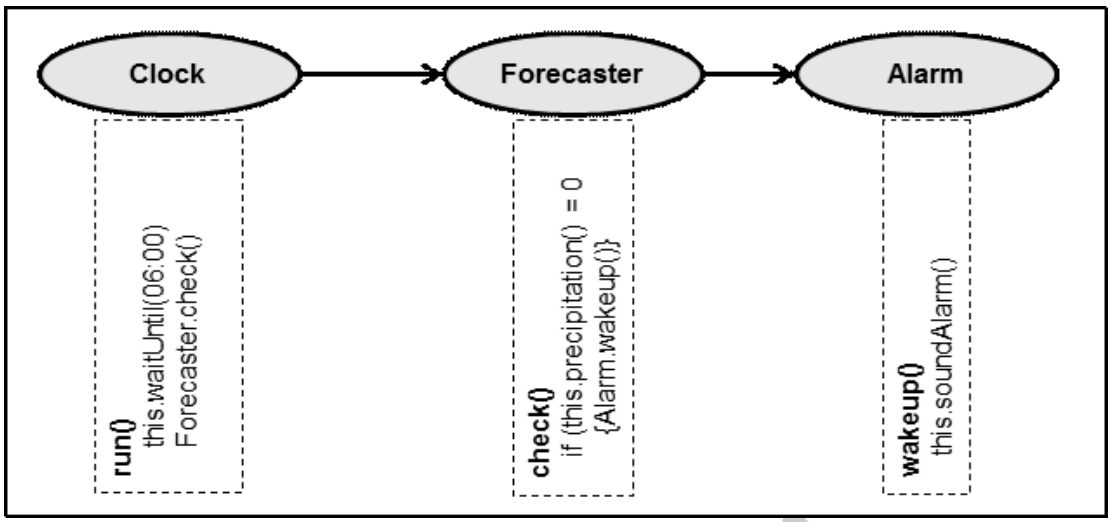


Figure 3: Ellen's alarm clock; her personal program

Ellen's scripts

The next and final stage is to instruct the roles as to what to do for her. The Clock script makes the process wait until morning before it cues the Forecaster to check the weather. If rain is expected, the process stops here and Ellen sleeps on. Otherwise, the Alarm is cued to wake Ellen according to the normal procedure for alarm clocks.

MVC applied to DCI

Ellen's user interface is based on MVC and her conceptual foundation is the DCI paradigm. She has a large and changing reservoir of Data objects and she can pick them from her device, from the internet, and from her internet of things. In addition, advanced programmers will know that Data objects can be instances of the programmer's own classes. Most importantly, a Context object can play a role in outer, higher level contexts. All the objects are Model objects in the MVC sense. Ellen's workspace is a View on a DCI context with its roles and other Views on the role scripts. An MVC Controller ties these Views together to make up Ellen's workspace.

The main goals of DCI are:

MENTAL MODELS. To reflect the way different users conceptualize the objects of their world so that a program feels like an extension of its user's mind.

REASONING. To help software developers reason about system state and behavior in addition to the state and behavior of isolated objects.

READABILITY. To improve the readability of object-oriented code by giving system behavior first-class status.

REUSE. To be able to reuse old solutions for new purposes.

REVISION. To cleanly separate code for rapidly changing system behavior (what the system *does*) from code for slowly changing domain knowledge (what the system *is*), instead of combining both in one class hierarchy.

Ellen's program is discussed in depth (Section 5: *Example)*. In (Section 6: *Related Work)*, we briefly comment on other efforts that are related to DCI. Suggestions for further work are in (Section 7: *Future Work)*. In (Section 8: *Conclusion)*, we conclude with the vision of DCI as a programming paradigm that spans many programming and modeling languages as well as Personal Programming.

