N91-10617

Representing Object Oriented Specifications and Designs with Extended Data Flow Notations

Jon Franklin Buser

Paul T Ward

Abstract

This paper addresses the issue of using extended data flow notations to document object oriented designs and specifications. Extended data flow notations, for the purposes of this paper, refer to notations that are based on the rules of Yourdon / DeMarco data flow analysis. The extensions include additional notation for representing real-time systems as well as some proposed extensions specific to object oriented development. The paper will state some advantages of data flow notations, investigate how data flow diagrams are used to represent software objects, point out some problem areas with regard to using data flow notations for object oriented development, and propose some initial solutions to these problems.

Introduction

Data flow diagramming is a general graphic-based modeling notation that has gained wide industry acceptance as a software specification and design tool. The proponents of object oriented techniques claim that systems built using these techniques have a natural system architecture that allows easier system modification and software component reuse. The authors support a method of system building that follows an object oriented development strategy and uses extended data flow notations to document the specification and design. There are many reasons for using data flow notation as the documentation medium:

- The notation is supported by a large number of Computer Aided Software Engineering (CASE) tools.
- Data flow models are not specific to any particular computer language, operating system, or hardware configuration making the necessary investment in training and tools useful over a wide spectrum of projects.
- Data flow modeling has a relatively long and successful record within the computer industry; many software engineers already have a working understanding of the notation.

Data flow diagrams use circles to represent processes, or units of work within a system, and arrows to represent data that is supplied to and produced by the processes [1]. Data Flow diagrams can be used for modeling general problem domains. These domain models are then evolved into software system specifications and designs. Figure 1 is a data flow diagram describing a Data Storage and Reporting System. The system produces reports on stored data and has a menu driven user interface for adding and updating records. A complete specification for the system would also include a detailed description of each process explaining how it will produce its output given the input data supplied. The Ward / Mellor[2] and Boeing / Hatley [3] real-time extensions introduce additional graphic symbols that are used to integrate finite state machine logic into the model. These state machine models strictly define the relationship of operations within a model and can potentially be executed to demonstrate the correctness of the model.

Object Oriented Partitioning

One of the key features of a data flow model is that it may be partitioned and leveled. This means that a number of processes can be grouped together into a single higher level process that represents the combined operations of the lower level processes. The highest level diagram in the model (the context diagram) represents the system as a single process and uses rectangular boxes to represent entities that are external to, but interact with, the system being modeled. Figure 2 is a

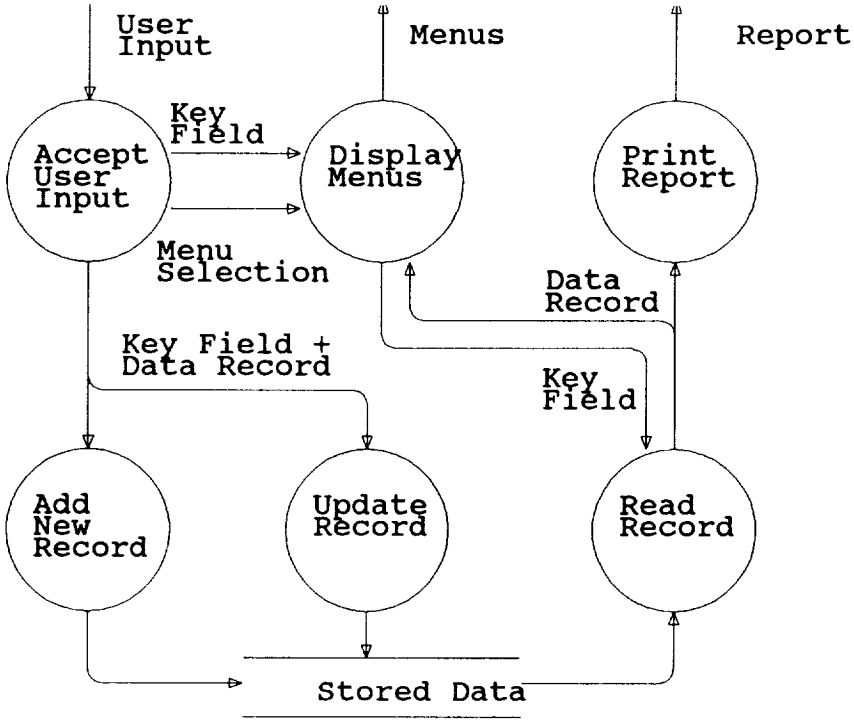


Figure 1

context diagram for the Data Storage and Reporting System.

Traditionally, data flow models have been partitioned by using a strategy called functional decomposition. This is a top down method that identifies high level system functions and then details, at the next level of the model, what processes will be required to perform each function. This process is repeated until all of the system's primitive components have been identified. Figure 3 shows a possible functional partitioning of the Data Storage and Reporting System. The system is partitioned into two sub-systems: one for managing data input and the other for data reporting. Both sub-systems have direct access to the data store.

There are other partitioning methods. One alternate strategy groups together processes that are parts of the response to a given external event. Another organizes the model so that the number of data flows between the higher level processes are minimized. The choice of system partitioning is important because it will define the major subsystem interfaces and, in the case of large software projects, it will probably define the management structure of the organization that builds the system.

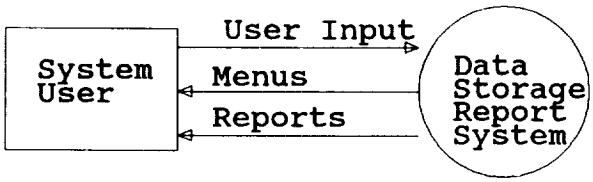


Figure 2

Object oriented specifications are produced by changing the criteria used when partitioning the model. With the help of information modeling techniques, classes of real world objects are identified in the problem domain [4]. Then the data flow model is partitioned by grouping together the processes associated with each object or class. In the case of the Data Storage and Reporting System we will identify a user interface object, a report object, and a data store object. These specification objects may be useable directly as

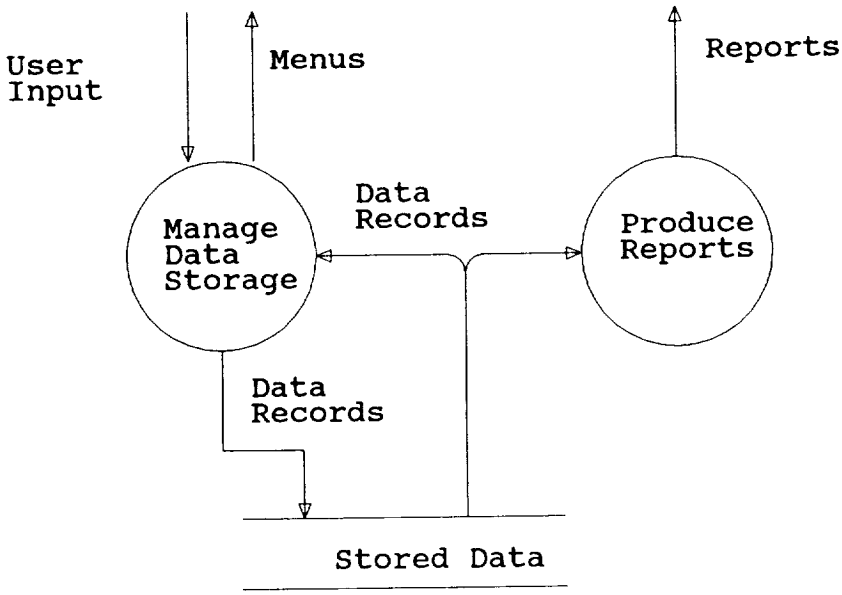


Figure 3

design objects, or they may have to be modified to transform them into design objects (e.g., to meet system performance constraints). These design objects can then be implemented as information hiding modules or Ada packages.

Data Flow Problems

We have found the object oriented partitioning strategy useful, however some of the rules governing traditional data flow diagrams and the CASE tool implementations of these rules conflict with object oriented goals.

One goal of object oriented design methods is to identify reusable objects. These objects may be reused within the same model or in different but related problem domains. Many of the CASE tools have a problem with regard to reusing these objects in the same model because the CASE tools typically enforce that all processes have unique names. If we want a process to be reused within a single model, naming conventions have to be devised to specify that different instances of the process are really the same. Of course, without additional tool support it is impossible to prevent different instances of each object from being modified so that they are no longer the same.

Another problem is that objects designed with reuse in mind will often be built in a more general manner than ones that have been engineered for a specific use. The result of this is that all of the object's access functions or methods may not used in a specific instance of the object. One of the primary model validation criteria applied to data flow diagrams is that all of the input and output flows entering a process must exist in the lower level description of the process. The existing CASE tools will report errors when general reusable objects are used in a model that does not make use of all the object's capabilities. For example, a

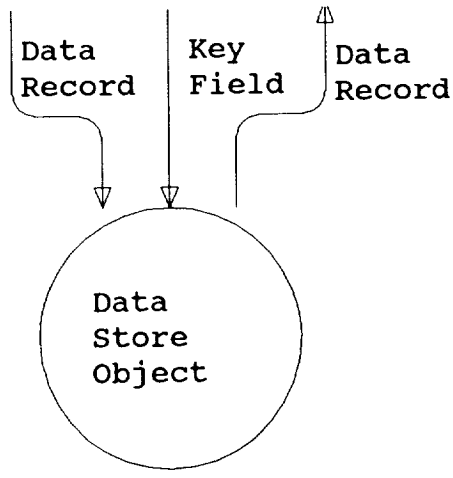


Figure 4

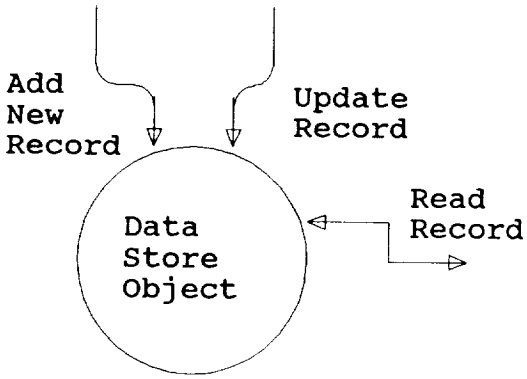


Figure 5

Add New Record =

Input Key + Input Data Record

Update Record =

Input Key + Input Data Record

Read Record =

Input Key + Output Data Record

Figure 6

more general data store object for the Data Storage and Reporting System might have a process for deleting records from the store. If this object is instantiated in an application that does not require a delete capability the analysis routines in the current CASE tools will report an error. To successfully level-balance the model, the delete process and its associated flows will have to be removed. A CASE tool designed to support importation of reusable objects must have a facility for deactivating specific access routines.

Representing Access Functions

Data flow models can be partitioned so that processes are grouped together in an object oriented fashion. The rules of data flow notation also allow data flows to be grouped together. This is commonly done to reduce the clutter of data flows entering and leaving higher level processes. We propose that the data flows should be grouped together so that all of the input and output parameters of each access routine are combined,

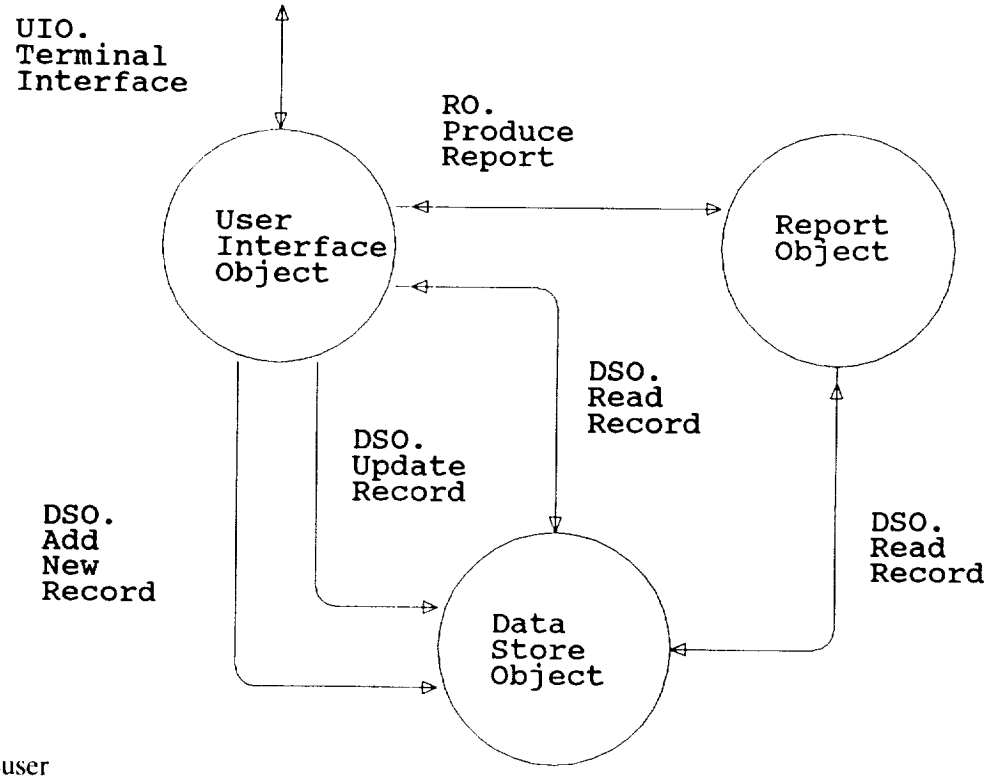


Figure 7

and that the combined flow is named for the access routine that it represents. If this approach is not followed it is impossible to determine which data flows operate together. Figure 4 shows the data store object from the Data Storage and Reporting System. Notice that all information that correlates input and output data with specific object capabilities has been lost. Compare this to figure 5 which groups the object's input and output flows together according to which access routine they are associated with. Information about the object's access routines is now retained. Figure 6 shows the composition of the each of the flows from figure 5.

Some CASE tools allow a data flow to have arrows on both ends indicating a two way flow of information. We suggest that this is a useful convention for representing flows that have both an input and output component. This notation is not completely adequate though, because it will not be clear from this diagram which object is using the other. This problem could be alleviated by introducing a new graphic symbol to indicate the direction of these combined flows or by applying naming conventions. One naming convention could name the flow by concatenating the objects name with the access function name, another convention could specify whether a particular flow component was an input or output (e.g., "input data record" as opposed to just "data record"). Figure 7 shows how the data store object integrates with the rest of the Data Storage and Reporting System using the double arrow head convention.

Future Work

Data flow diagrams can be used to model object oriented specifications and designs, however additional conventions may be needed for this to work well. Further work is needed to identify all of these conventions and to integrate them into CASE tools. Two areas of particular need are tools that will support the concept of inheritance, and browsers that can scan reusable software object libraries documented with data flow diagrams.

References

[1] T. DeMarco, *Structured Analysis and System Specification*, New Jersy: Prentice-Hall, 1978

[2] P.Ward and S. Mellor, *Structured Analysis for Real-Time Systems*, New Jersy: Prentice-Hall, 1985.

[3] D. Hatley and E. Pirbhai, *Strategies for Real-Time System Specifications*, New York: Dorset House, 1987.

[4] S. Mellor and S. Shlaer, *Object Oriented System Analysis*, New Jersy: Prentice-Hall, 1988.

THE VIEWGRAPH MATERIALS

FOR THE

J. F. BUSER PRESENTATION FOLLOW

PAGE 6 INTENTIONALLY BLANK

Representing Object Oriented

Specifications and Designs

with

Extended Data Flow Notations

by

Jon Franklin Buser

Paul T Ward

Software Development Concepts Background Information

- **Real-Time Data Flow Diagram Extensions**
- **Develop Courses and Teach Real-Time Specification and Design Methods**
- **Work with CASE vendors**
- **Continued Research into Real-Time Development and Object-Oriented Methods**

Goal

Develop ways to represent object oriented designs and specifications with Data Flow Diagram based notations.

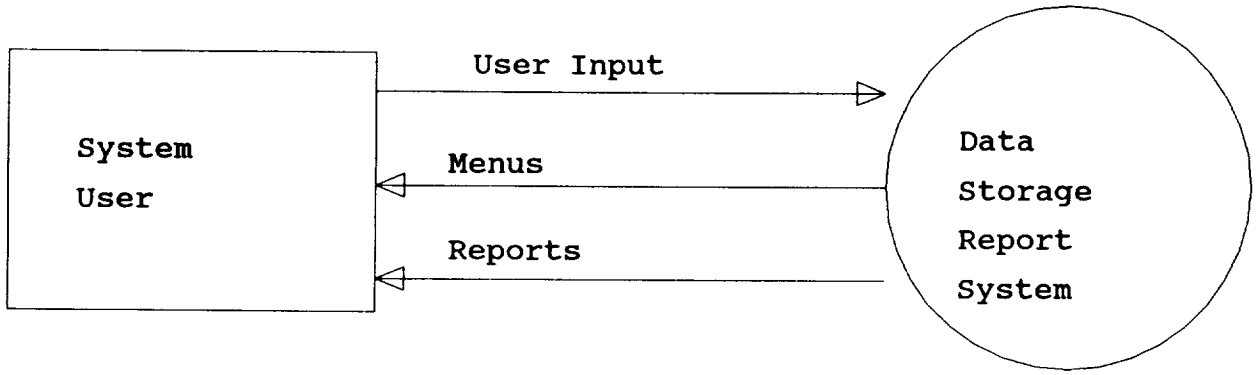
Advantages of Data Flow Diagrams

- Supported by many CASE tools
- NOT specific to any computer language or operating system
- Many Software Engineers already have a working understanding

Data Flow Problems

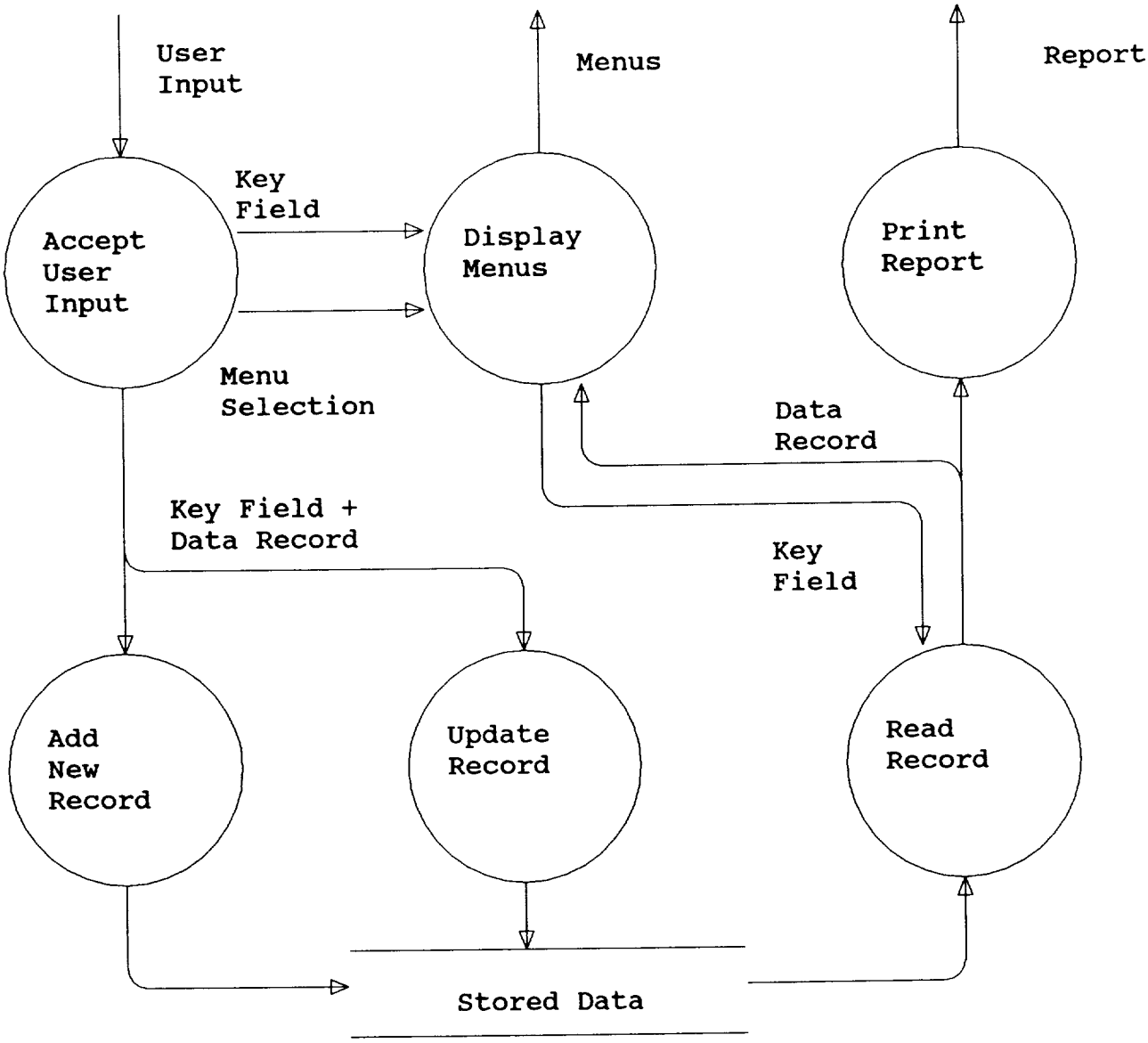
- CASE tool enforced unique names conflict with component reuse
- Level-Balancing conflicts with building general reusable components that have unused access functions
- Commonly used partitioning strategies do NOT reinforce the concept of Software Objects

The Data Storage and Reporting System



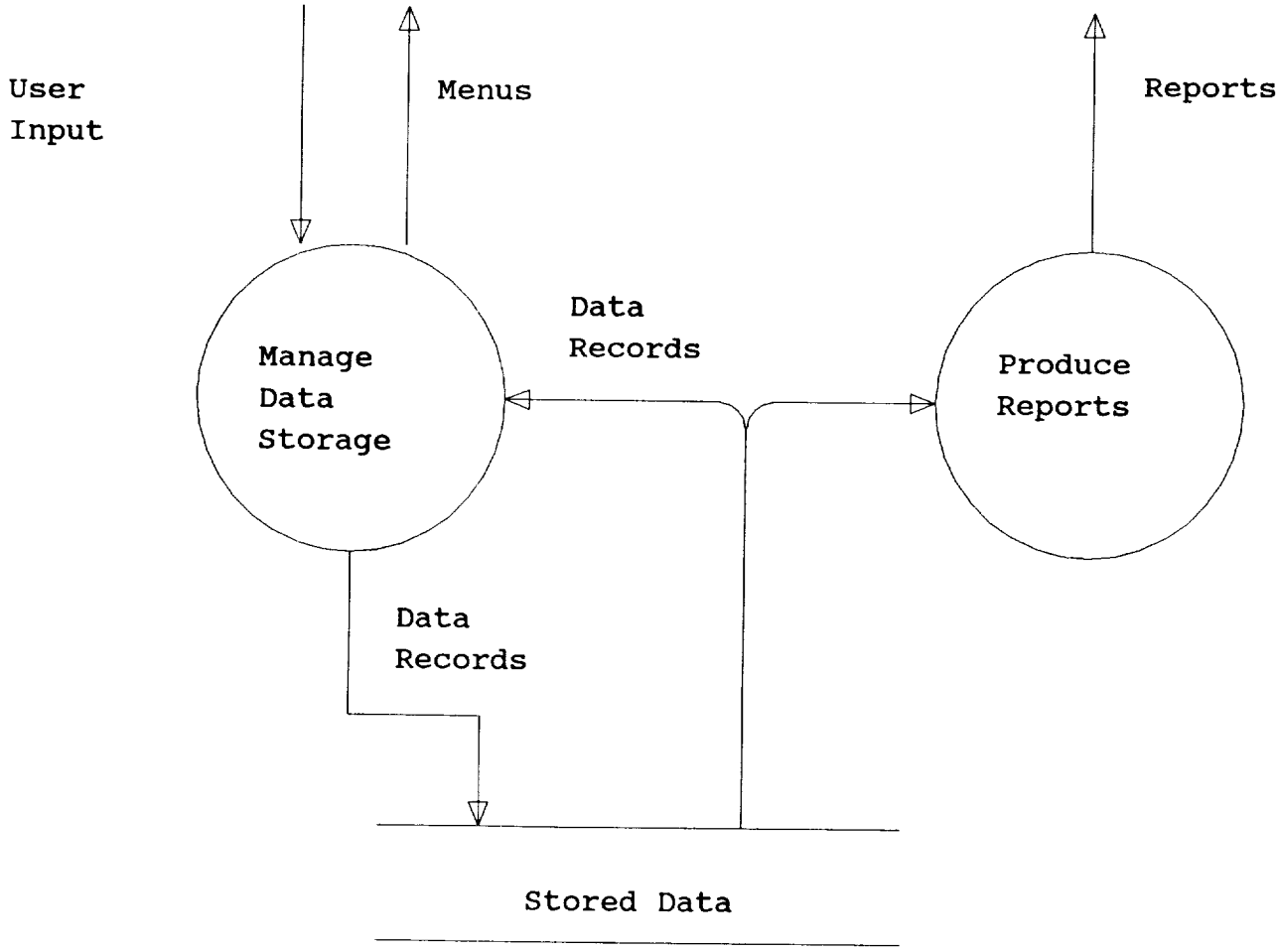
Data Storage and Reporting System

Detailed View



Data Storage and Reporting System

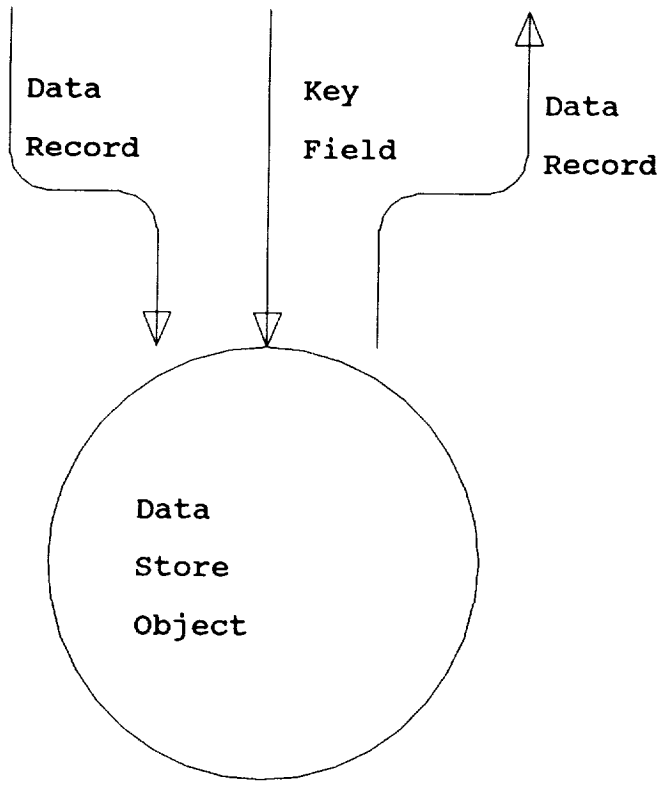
Functional Partitioning



Objects in the Data Storage and Reporting System

- Data Store Object
- Report Object
- User Interface Object

The Data Store Objects grouped together

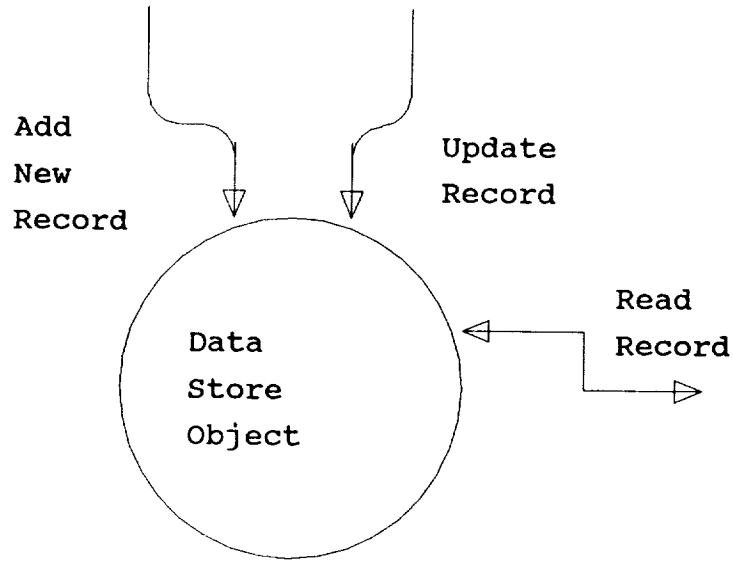


New Partitioning Conventions

for Representing Objects

- Group together processes that operate on the same real-world objects
- Group together Data Flows that are associated with the same process or access routine
- Name the combined flow for the access routine that it is attached to
- Use double arrow head if the flow is composed of both input and output flows

The Data Store Object



Add New Record =

Input Key + Input Data Record

Update Record =

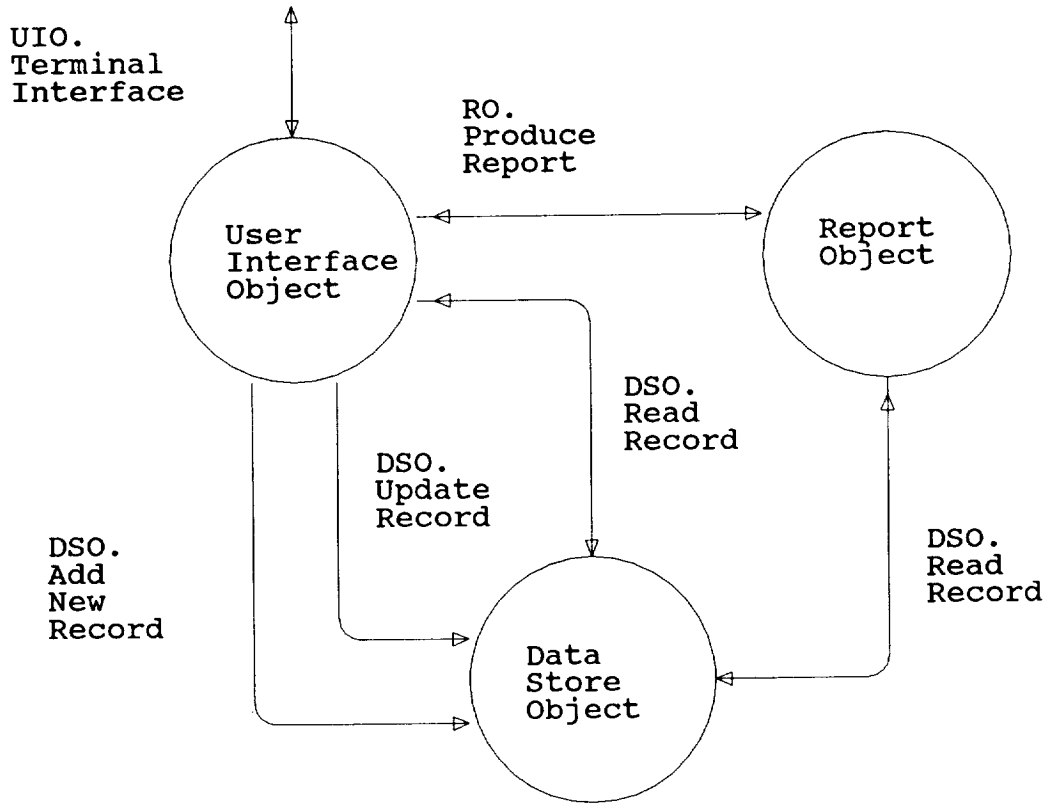
Input Key + Input Data Record

Read Record =

Input Key + Output Data Record

Object Oriented View of the

Data Storage and Reporting System



Future Work

- Work further with these conventions
- CASE tools to support reuse and inheritance
- Browsers to scan libraries of reusable components documented with Data Flow Diagrams