

### **Creating an Entity/Relationship Diagram**

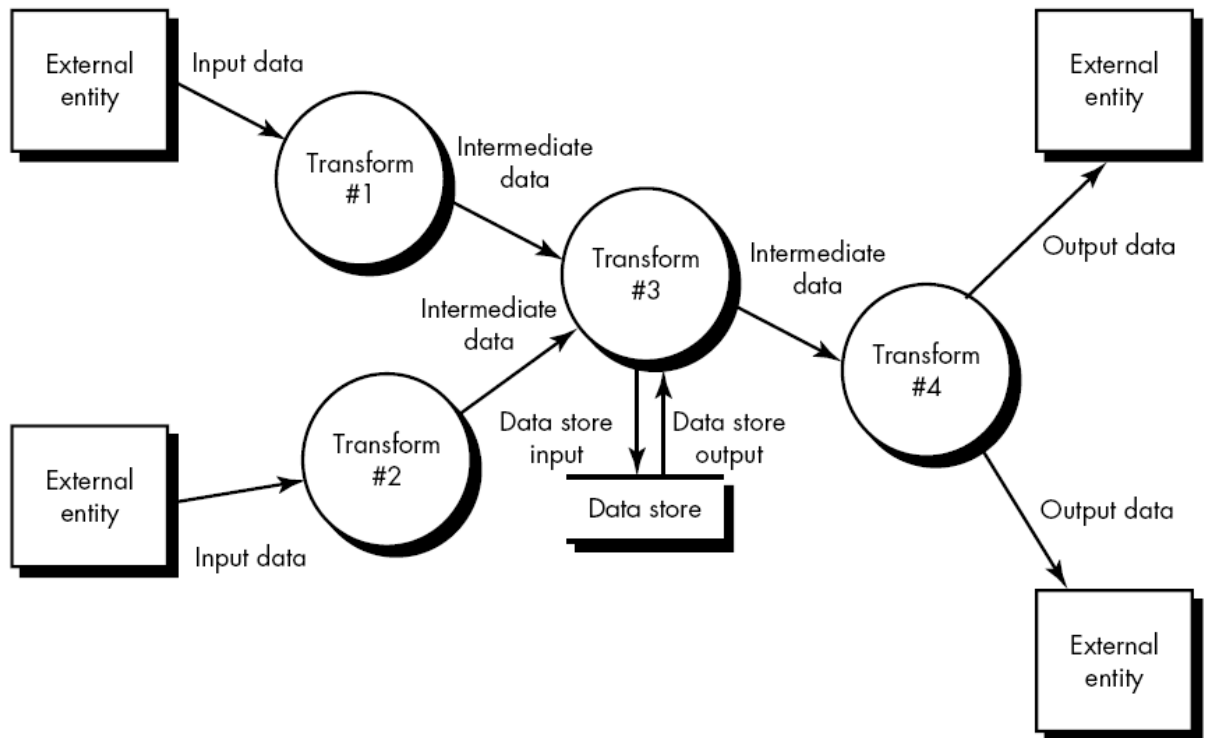
The ERD is constructed in an iterative manner. The following approach is taken:

1. During requirements elicitation, customers are asked to list the “things” that the application or business process addresses. These “things” evolve into a list of input and output data objects as well as external entities that produce or consumer information.
2. Taking the objects one at a time, the analyst and customer define whether or not a connection (unnamed at this stage) exists between the data object and other objects.
3. Wherever a connection exists, the analyst and the customer create one or more object/relationship pairs.
4. For each object/relationship pair, cardinality and modality are explored.
5. Steps 2 through 4 are continued iteratively until all object/relationships have been defined. It is common to discover omissions as this process continues. New objects and relationships will invariably be added as the number of iterations grows.
6. The attributes of each entity are defined.
7. An entity relationship diagram is formalized and reviewed.
8. Steps 1 through 7 are repeated until data modeling is complete.

### **Functional Modeling and Information Flow**

Information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms; applies hardware, software, and human elements to transform it; and produces output in a variety of forms. Input may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage. The transform(s) may comprise a single logical comparison, a complex numerical algorithm. Output may light a single LED or produce a 200-page report. In effect, we can create a *flow model* for any computer-based system, regardless of size and complexity.

Structured analysis began as an information flow modeling technique. A computer-based system is represented as an information transform as shown in Figure 1



**Figure 1:** Information flow model

A rectangle is used to represent an *external entity*; that is, a system element (e.g., hardware, a person, another program) or another system that produces information for transformation by the software or receives information produced by the software. A circle (sometimes called a *bubble*) represents a *process* or *transform* that is applied to data (or control) and changes it in some way. An arrow represents one or more *data items* (data objects). All arrows on a data flow diagram should be labeled. The double line represents a data store—stored information that is used by the software. The simplicity of DFD notation is one reason why structured analysis techniques are widely used.

**Note:**

The DFD is not procedural. That is, do not try to represent conditional processing or loops with this diagrammatic form. Simply show the flow of data.

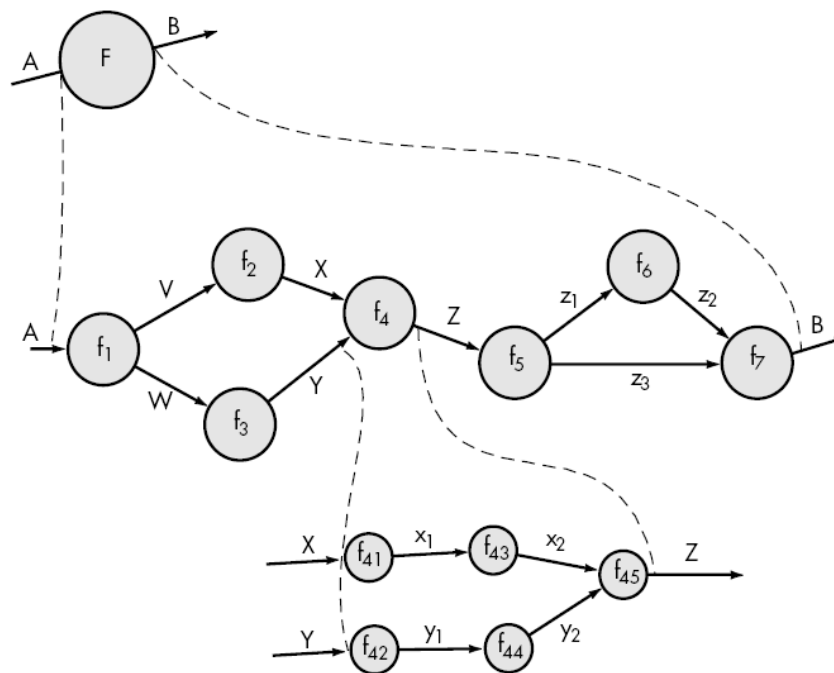
- **Data Flow Diagrams**

As information moves through software, it is modified by a series of transformations. A data flow diagram is a graphical representation that depicts information flow and the transforms that are applied as data move from input to output. The basic form of a data flow diagram, also known as a data flow graph or a bubble chart, is illustrated in Figure 1.

The data flow diagram may be used to represent a system or software at any level of abstraction. In fact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Therefore, the DFD provides a mechanism for functional modeling as well as information flow modeling. In so doing, it satisfies the second operational analysis principle (i.e., creating a functional model).

A level 0 DFD, also called a *fundamental system model* or a *context model*, represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively. Additional processes (bubbles) and information flow paths are represented as the level 0 DFD is partitioned to reveal more detail. For example, a level 1 DFD might contain five or six bubbles with interconnecting arrows. Each of the processes represented at level 1 is a subfunction of the overall system depicted in the context model.

As has been noted earlier, each of the bubbles may be refined or layered to depict more detail. Figure2 illustrates this concept.



**Figure 2:** Information flow refinement

A fundamental model for system  $F$  indicates the primary input is  $A$  and ultimate output is  $B$ . We refine the  $F$  model into transforms  $f1$  to  $f7$ . Note that *information flow continuity* must be maintained; that is, input and output to each refinement must remain the same. This concept, sometimes called *balancing*, is essential for the development of consistent models. Further refinement of  $f4$  depicts detail in the form of transforms  $f41$  to  $f45$ . Again, the input ( $X, Y$ ) and output ( $Z$ ) remain unchanged.

DFD graphical notation must be augmented with descriptive text. A *process specification* (PSPEC) can be used to specify the processing details implied by a bubble within a DFD. The process specification describes the input to a function, the algorithm that is applied to transform the input, and the output that is produced. In addition, the PSPEC indicates restrictions and limitations imposed on the process (function), performance characteristics that are relevant to the process, and design constraints that may influence the way in which the process will be implemented.

The process specification (PSPEC) is used to describe all flow model processes that appear at the final level of refinement. The content of the process specification can include narrative text, a *program design language* (PDL) description of the process algorithm, mathematical equations, tables, diagrams, or charts. By providing a PSPEC to accompany each bubble in the flow model, the software engineer creates a "mini-spec" that can serve as a first step in the creation of the *Software Requirements Specification* and as a guide for design of the software component that will implement the process.

- **Creating a Data Flow Model**

The data flow diagram enables the software engineer to develop models of the information domain and functional domain at the same time. As the DFD is refined into greater levels of detail, the analyst performs an implicit functional decomposition of the system, thereby accomplishing the fourth operational analysis principle for function. At the same time, the DFD refinement results in a corresponding refinement of data as it moves through the processes that embody the application.

A few simple guidelines can aid immeasurably during derivation of a data flow diagram:

1. The level 0 data flow diagram should depict the software/system as a single bubble.
2. Primary input and output should be carefully noted.
3. Refinement should begin by isolating candidate processes, data objects, and stores to be represented at the next level.
4. All arrows and bubbles should be labeled with meaningful names.
5. Information flow continuity must be maintained from level to level.
6. One bubble at a time should be refined.

#### ▪ **Extensions for Real-Time Systems**

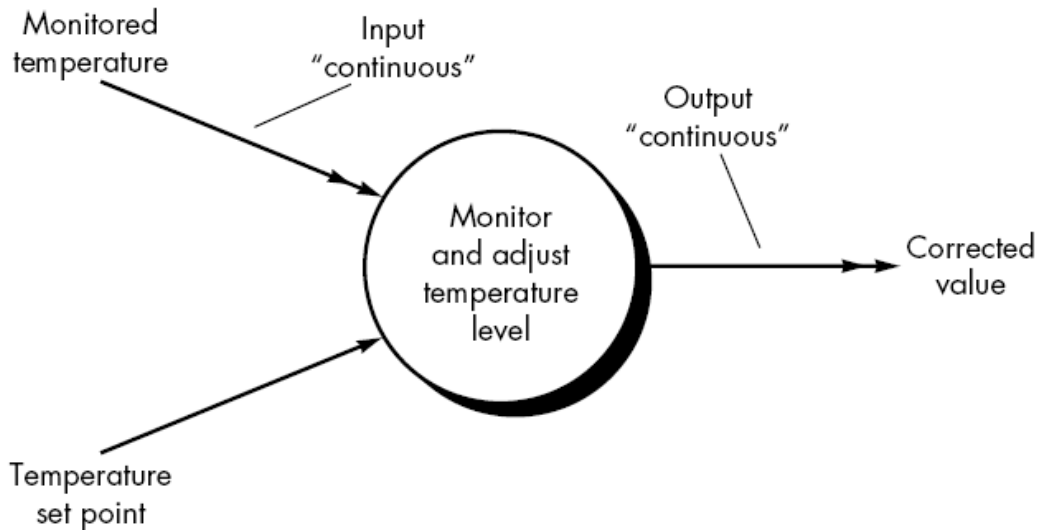
Many software applications are time dependent and process as much or more control-oriented information as data. A real-time system must interact with the real world in a time frame dictated by the real world. manufacturing process control, and industrial instrumentation are but a few of hundreds of real-time software applications.

To accommodate the analysis of real-time software, a number of extensions to the basic notation for structured analysis have been defined. These extensions, developed by Ward and Mellor **and** Hatley and Pirbhai, enable the analyst to represent control flow and control processing as well as data flow and processing.

#### **1-Ward and Mellor Extensions**

In a significant percentage of real-time applications, the system must monitor time continuous information generated by some real-world process. For example, a real-time test monitoring system for gas turbine engines might be required to monitor turbine speed, combustor temperature, and a variety of pressure probes on a continuous basis. Conventional data flow notation does not make a distinction between discrete data and time-continuous data. One extension to basic structured analysis notation, shown in Figure 3, provides a mechanism for representing *time-continuous data flow*.

The double headed arrow is used to represent time-continuous flow while a single headed arrow is used to indicate discrete data flow. In the figure, monitored temperature is measured continuously while a single value for temperature set point is also provided. The process shown in the figure produces a time-continuous output, corrected value.

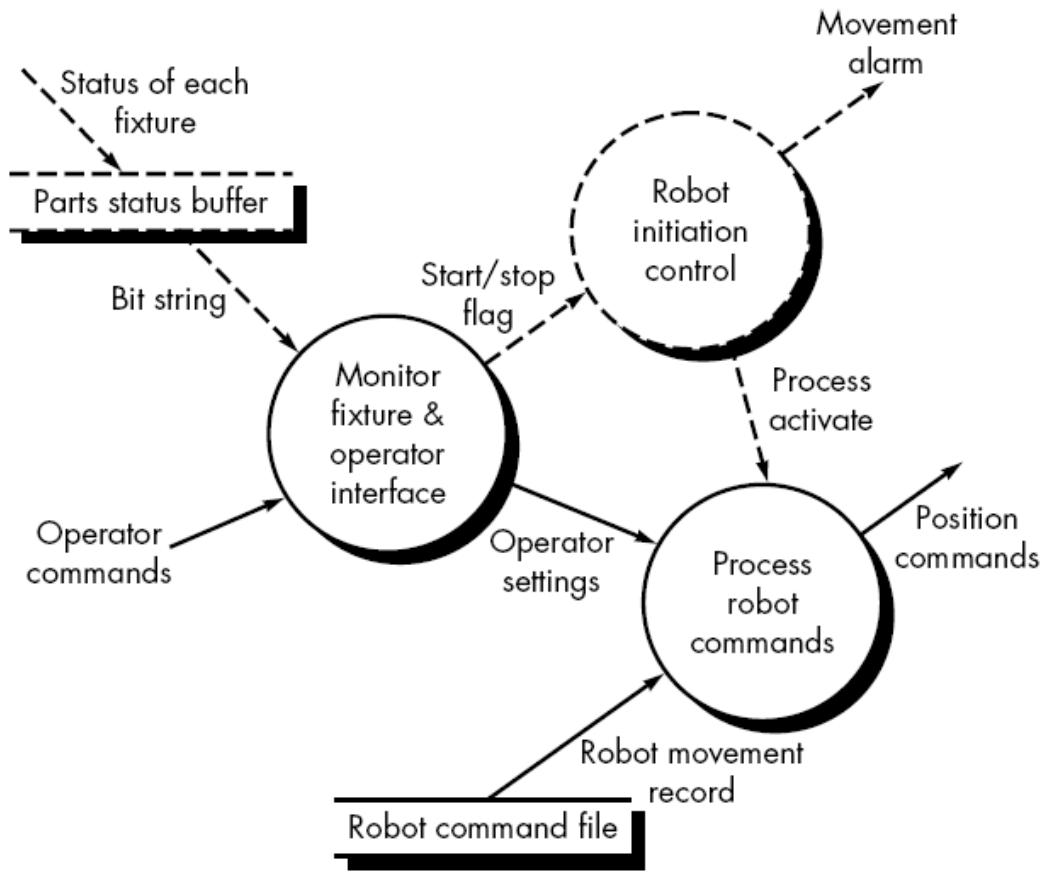


**Figure 3:** Time continuous data flow

Continuing the convention established for data flow diagrams, data flow is represented using a solid arrow. *Control flow*, however, is represented using a dashed or shaded arrow. A process that handles only control flows, called a *control process*, is similarly represented using a dashed bubble.

Control flow can be input directly to a conventional process or into a control process. Figure 4 illustrates control flow and processing as it would be represented using Ward and Mellor notation.

The figure illustrates a top-level view of a data and control flow for a manufacturing cell. As components to be assembled by a robot are placed on fixtures, a status bit is set within a **parts status buffer** (a control store) that indicates the presence or absence of each component. Event information contained within the **parts status buffer** is passed as a bit string to a process, *monitor fixture and operator interface*. The process will read **operator commands** only when the control information, bit string, indicates that all fixtures contain components. An event flag, **start/stop flag**, is sent to *robot initiation control*, a control process that enables further command processing. Other data flows occur as a consequence of the **process activate** event that is sent to *process robot commands*.



**Figure 4:** Data and control flows using Ward and Mellor notation