

To: Larry Ridge, Engineering Manager
From: Jonathan R. Saliers
Subject: Tutorial on Data Flow Diagram Modeling

In response to your request, a brief discussion of the data flow diagram (DFD) modeling technique and its advantages and disadvantages as a system modeling technique has been formulated. In addition, an example of DFD modeling using our Smart Fuel Pump system has been assembled.

There are four central items present in Data Flow Diagram modeling: external entities, processes, data stores, and data flows. An external entity is either the source of a system level input or the destination of a system level output. A process is a function with both inputs and outputs. A data store acts as a storage device for data. A data flow is the actual movement of data from an entity or data store to a process, or vice versa.

There are several common notations used in DFD modeling, including the Yourdon / Constantine and the Gane / Sarson notation. Figure 1 illustrates each of the central items in both notations.

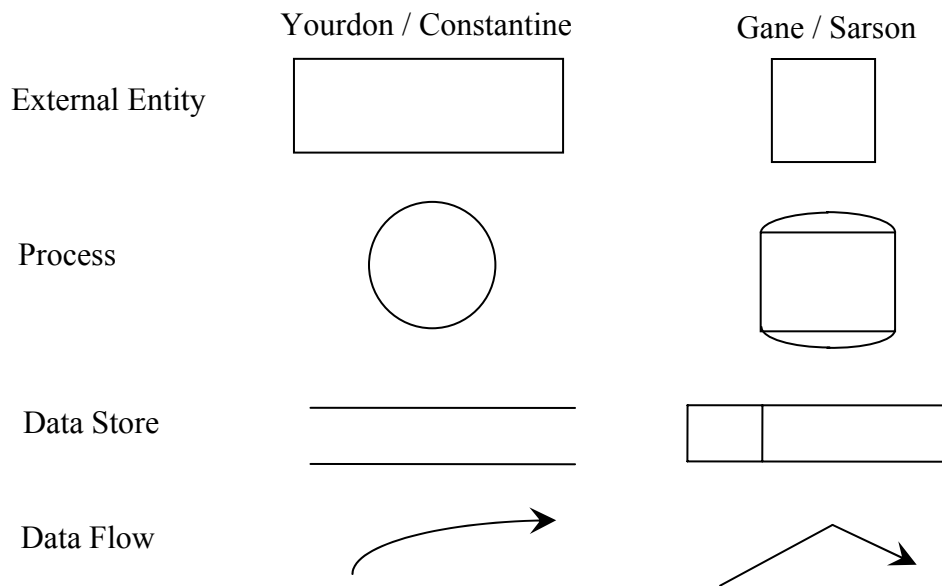


Figure 1. DFD Modeling Central Items with Different Notations

The main purpose of data flow diagrams is to depict the flow and transformation of data throughout a system. These diagrams can capture a broad-based understanding and insight of the system. In fact, DFD modeling is so useful that it is the most frequently used process modeling technique.

Generally, a top-level DFD model is created. This is known as the Context Diagram. Each process can then be decomposed to arbitrary levels of detail. These new detail

levels are called functional decompositions of the process. A sample of the context diagram and its decomposition is illustrated in figure 2.

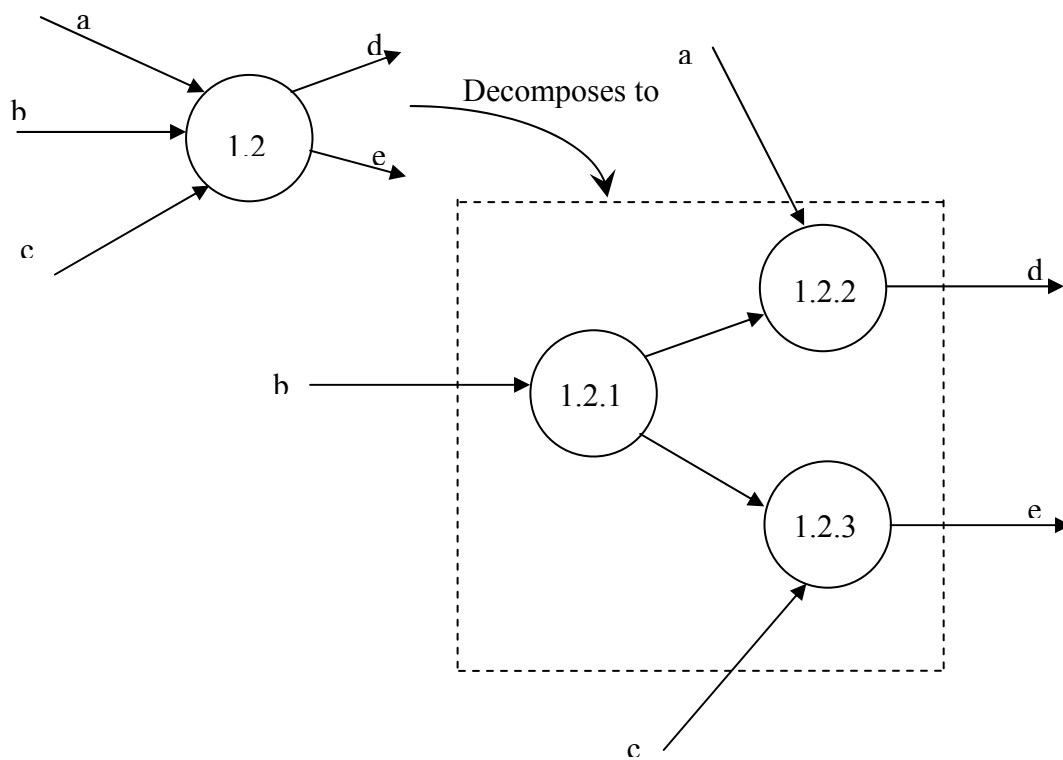
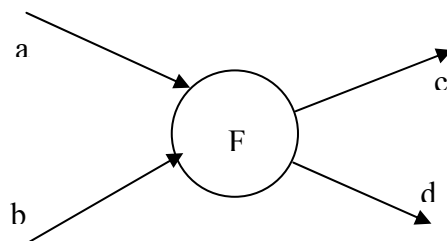


Figure 2. Sample Context Diagram and Functional Decomposition

One drawback of DFD models is that there may be some ambiguity. For example, consider the sample model in figure 3.



What does the process F do?

1. $F(a, b) = (c, d)$
2. $F(a) = (c), F(b) = (d)$
3. If (a) THEN $F(b) = (c)$, ELSE $F(b) = (d)$
4. $F(a) = (d), F(b) = (c)$

Figure 3. DFD Modeling Ambiguity

To illustrate the usefulness of data flow diagram modeling, an example data flow diagram has been assembled, using our smart fuel pump system. Included in figure 4 is the context diagram. Figure 5 contains the first functional decomposition, and figure 6 contains the second functional decomposition. In addition, some pseudo-code has been formulated. This pseudo-code is shown following the data flow diagram.

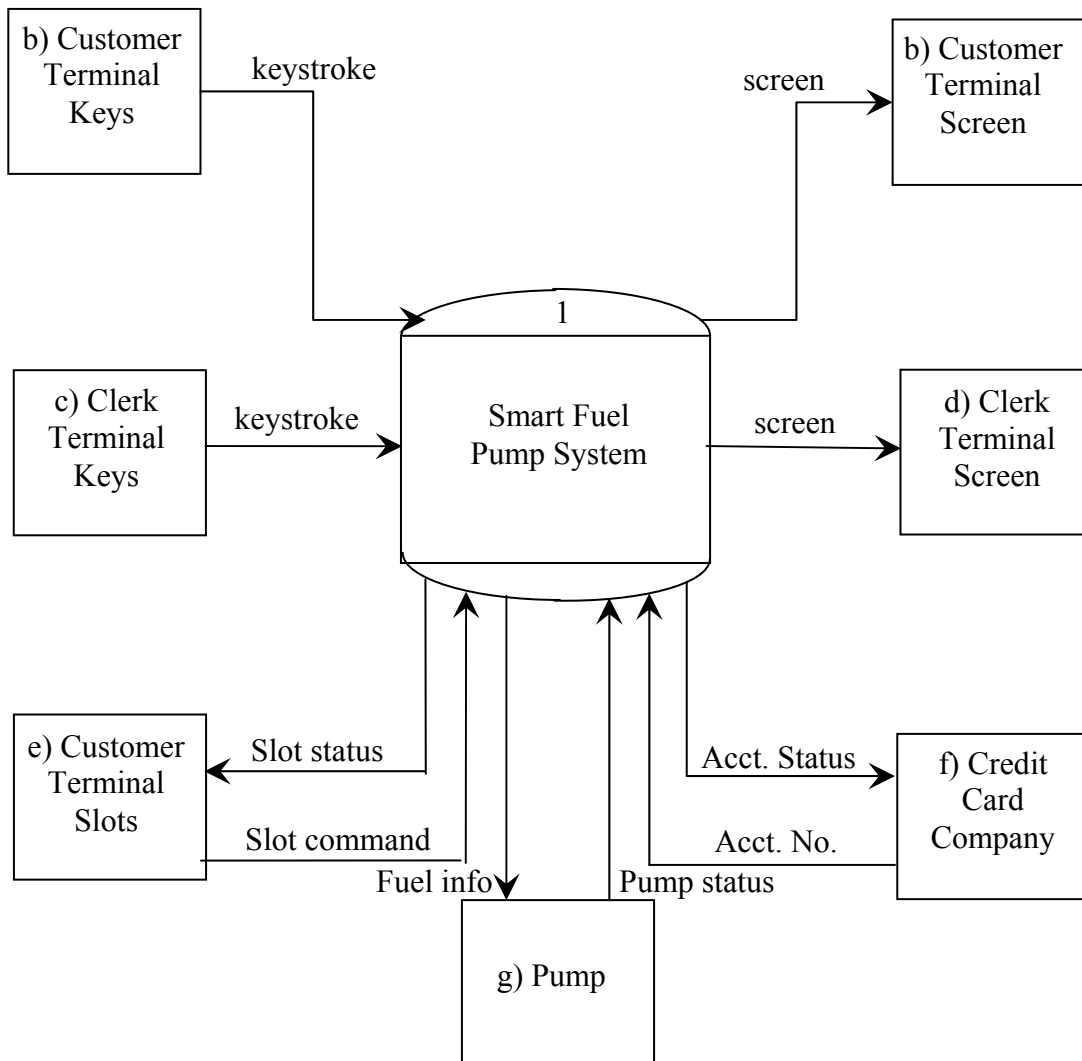


Figure 4. Context Diagram of Smart Fuel Pump

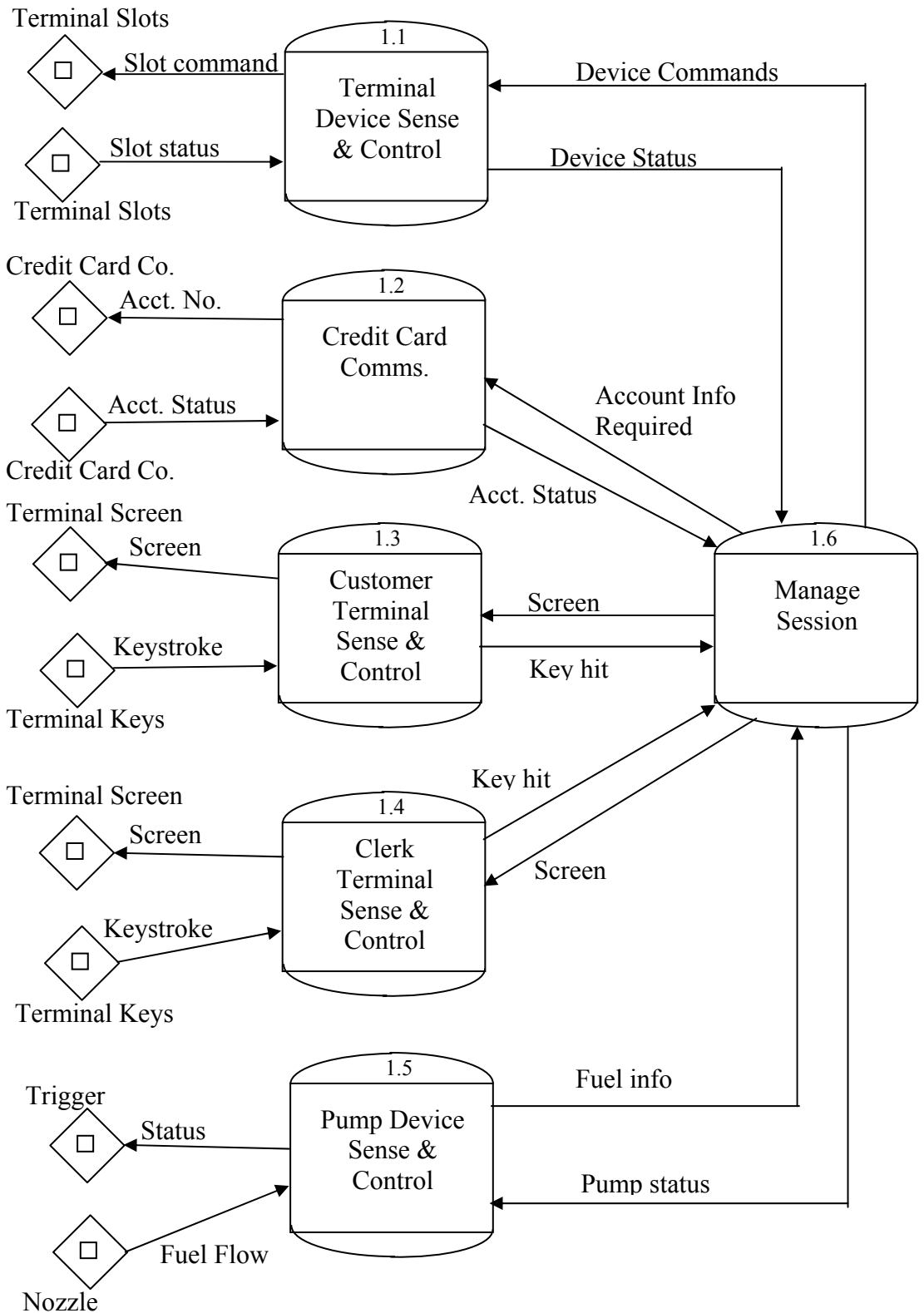


Figure 5. First Decomposition of Smart Fuel Pump

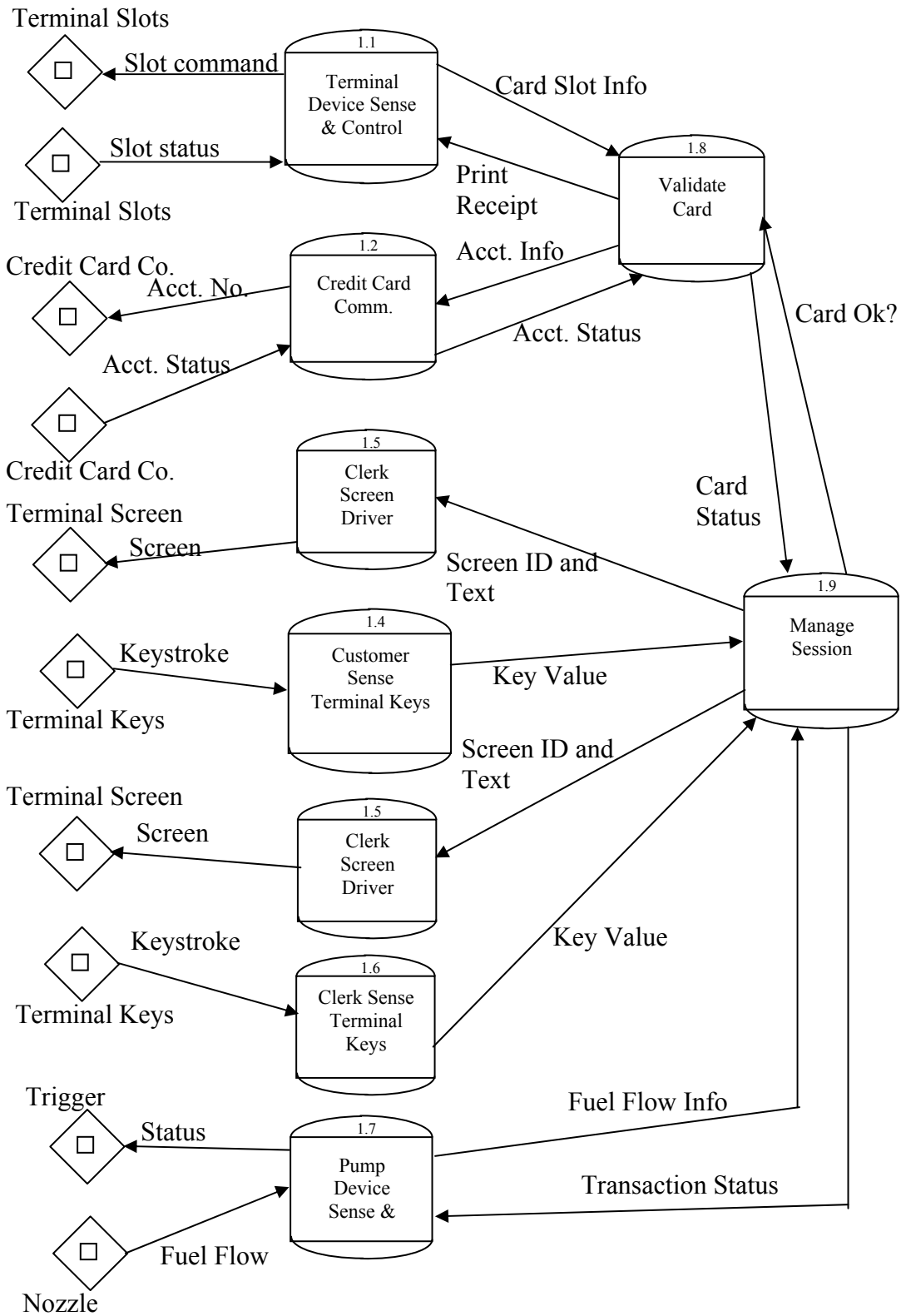


Figure 6. Second Decomposition of Smart Fuel Pump

```
/* Pseudo-Code for Smart Fuel Pump System */
```

```
/* Main Program Pseudo-Code */
```

```
Procedure AwaitStart:
```

```
    Customer sees welcome screen
    IF (customer to pay inside) THEN
        Begin AwaitEnable stage, return pump status
    ELSE
        Begin ValidateCard stage, return status
        IF (card is valid) THEN
            Begin AwaitEnable stage, return pump status
        ELSE
            Eject card
            Show 'eject card' screen
            Goto beginning of main
    IF (pump status = OK) THEN
        Show 'Select Grade' screen
        Get fuel grade
        Switch to correct fuel tank
        Show 'Begin pumping' screen
        WHILE (Holster = empty)
            IF (Trigger = active) THEN
                Pump fuel
            IF (credit card payment) THEN
                Show 'print receipt?' screen
                IF (printReceipt = true) THEN
                    Print receipt
                Goto beginning of main
            ELSE
                Show 'Please pay inside' screen
                Set alert in clerk terminal
                Get reset from clerk keypad
                Goto beginning of main
    ELSE
        Show 'Please see clerk...' screen
```

```
/* AwaitEnable Pseudo-Code => waits for clerk to enable/disable pump */
```

```
Procedure AwaitEnable:
```

```
    Set alert in clerk terminal
    Get enable/disable from clerk keypad
```

```
/* ValidateCard Pseudo-Code => Validates card with credit card company */
```

```
Procedure ValidateCard:
```

```
    Get Acct. # from card
    Get Acct status from credit card company
    Return status to calling procedure
```