Visual Paradigm

Data Flow Diagram with Examples - Securities Trading Platform

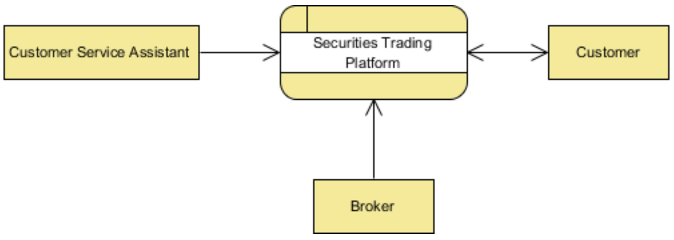
Written Date : February 16, 2015

Data Flow Diagram (DFD) provides a visual representation of the flow of information (i.e. data) within a system. By drawing a Data Flow Diagram, you can tell the information supplied by and delivered to someone who take parts in system processes, the information needed in order to complete the processes and the information needed to be stored and accessed. This article describes and explain Data Flow Diagram (DFD) by using a securities trading platform as an example.

The Securities Trading Platform Example

Context DFD

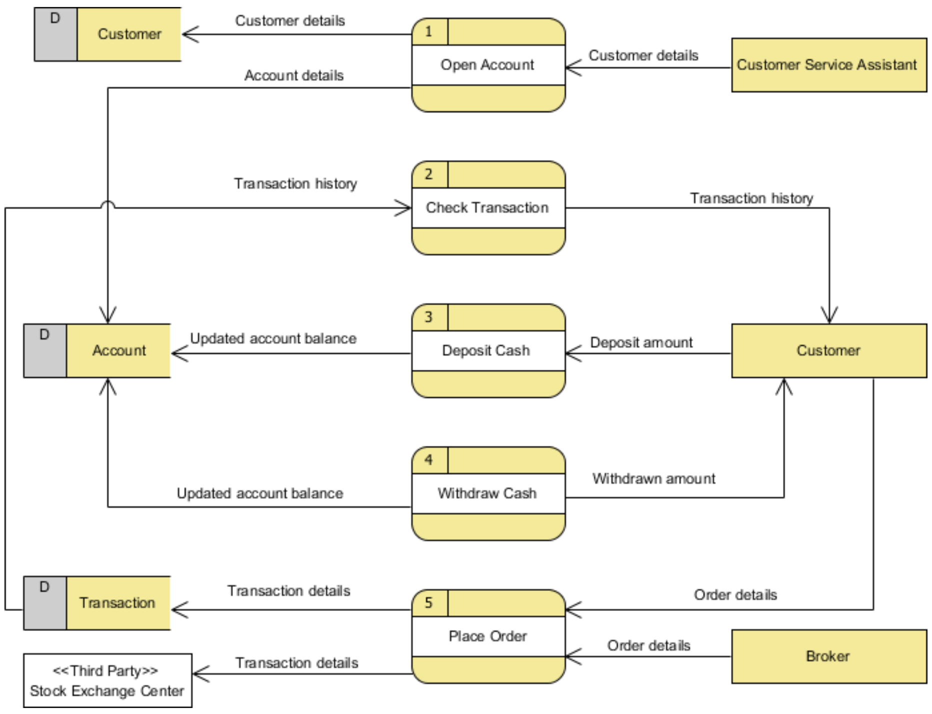
The figure below shows a context Data Flow Diagram that is drawn for a security trading platform. It contains a process (shape) that represents the system to model, in this case, the "*securities trading platform*". It also shows the participants who will interact with the system, called the external entities. In this example, *CS Assistant*, *Customer* and *Broker* are the entities who will interact with the system. In between the process and the external entities, there are data flow (connectors) that indicate the existence of information exchange between the entities and the system.



Context DFD is the entrance of a data flow model. It contains one and only one process and does not show any data store.

Level 1 DFD

The figure below shows the level 1 DFD, which is the decomposition (i.e. break down) of the securities trading platform process shown in the context DFD. Read through the diagram, and then we will introduce some of the key concepts based on this diagram.



The securities trading platform Data Flow Diagram example contains five processes, three external entities and three data stores. Although there is no design guidelines that governs the positioning of shapes in a Data Flow Diagram, we tend to put the processes in the middle and data stores and external entities on the sides to make it easier to comprehend.

Based on the diagram, we know that a *Customer Service Assistant* provides customer details to the *Open Account* process. The result is the *Customer details* being stored in *Customer* data store and *Account details* being stored in *Account* data store. Although we said that the attempt to store customer and account details happens after the details are being provided by the *Customer Service Assistant*, the Data Flow Diagram implies no such thing. It is our common sense that lead us to interpret the diagram in the way that we understand it naturally. Strictly speaking, the diagram only tells us the *Open Account* process receives *customer details* and produce customer and account details, with no order specified. Note that Data Flow Diagram does not answer in what way and in what order the information is being used throughout a system. If this information is important and worth mentioning, consider to model it with diagrams like BPMN Business Process Diagram or UML Activity Diagram.

The process *Check Transaction* receives *Transaction details* from the *Transaction* data store and pass it on to *Customer*.

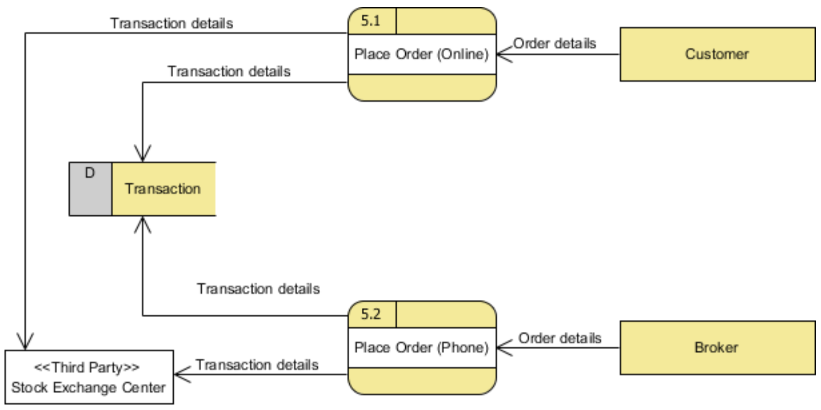
A *Customer* can *Deposit Cash* by providing the *Deposit amount* and the result is the *Updated account balance* being stored in the *Account* data store.

Similarly, a *Customer* can *Withdraw Cash*. The result is that he will receive the *Withdrawn amount* and the *Updated account balance* will be stored in the *Account* data store.

Finally, both the *Customer* and *Broker* can initiate the *Place Order* process, which results in the *Transaction details* being stored in the *Transaction* data store. The *Place Order* process also passes the *Transaction details* to the *Stock Exchange Center*, which is an entity out of the system scope. In the next section, we will introduce a way to represent this kind of entity.

Level 2 DFD

Just like the process in context DFD, processes in level 1 DFD can also be decomposed into a deeper level or even levels of process details. The figure below shows the level 2 DFD of the *Place Order* process.



The external entities and data stores in this DFD correspond with those shown in the upper level (i.e. the diagram above). What makes it different is the breakdown of *Place Order* process into *Place Order (Online)* process and *Place Order (Offline)* process.

Based on this diagram, we know that a *Customer* can perform *Place Order (Online)* by supplying *Order details* while a *Broker* can perform *Place Order (Phone)* also by supplying *Order details*; in either case causing *Transaction* details to be stored in the *Transaction* data store and passed to the *Stock Exchange Center*.

Using stereotype for modeling a "special kind of" entity

Stereotype and tagged values are kind of extensibility mechanisms introduced by Object Management Group (OMG). It allows designers to extend the vocabulary of UML in order to create new model elements. As a software design tool, Visual Paradigm extends the support of stereotype to non UML standards like DFD and ERD. Take the securities trading platform as example, we can define a stereotype Third Party for external entity. External entities with the stereotype assigned are said to be "a kind of third party entity".

Be aware of the level of details

In this Data Flow Diagram example the word "details" is used many times when labeling data. We have "customer details", "transaction details", etc. What if we write them explicitly as "customer name, email address, job, address" and "stock number, amount, bid price"? Is this correct? Well, there is no definite answer to this question but try to ask yourself a question when making a decision. Why are you drawing a DFD?

In most cases, Data Flow Diagram is drawn in the early phase of system development, where many details are yet to be confirmed. The use of general terminologies like "details", "information", "credential" certainly leave room for discussion. However, using general terms can be kind of lacking details and make the design lost its usefulness. So it really depends on the purpose of your design.

Don't overdrawn

In a Data Flow Diagram, we focus on the interactions between the system and external parties, rather than the internal communications among interfaces. Therefore, data flows between interfaces and the data stores used are considered to be out of scope and should not be shown in the diagram.

Don't mix up data flow and process flow

Some designers may feel uncomfortable when seeing a connector connecting from a data store to a process, without seeing the step of data request being shown on the diagram somehow. Some of them will try to represent a request by adding a connector between a process and a data store, labeling it "a request" or "request for something", which is wrong.

Keep in mind that Data Flow Diagram was designed for representing the exchange of information. Connectors in a Data Flow Diagram are for representing data, not for representing process flow, step or anything else. When we label a data flow that ends at a data store "a request", this literally means we are passing a request as data into a data store. Although this may be the case in implementation level as some of the DBMS do support the use of functions, which intake some values as parameters and return a result, in Data Flow Diagram we tend to treat data store as a sole data holder that does not possess any processing capability. If you want to model the system flow or process flow, use UML Activity Diagram or BPMN Business Process Diagram instead. If you want to model the internal structure of data store, use Entity Relationship Diagram.

Resources

1. Securities-Trading-Platform.vpp