**4.3.2 Control Flow Diagram**

The control flow diagram (CFD) is superimposed on the DFD and shows events that control the processes noted in the DFD. CFD is used to determine the flow of control objects and processes that act on these objects shows how events move through a system. The CFD contains the same processes as the DFD, but shows control flow, rather than data flow. Control flow, events and control items is represented using a dashed or shaded arrow. A process that handles only control flows, called a control process, is represented using a dashed bubble and the dashed double arrow is used to represent the control store and a notational reference (a solid bar) to a control specification (CSPEC) is used. The CSPEC is used to indicate (1) how the software behaves when an event or control signal is sensed and (2) which processes are invoked as a consequence of the occurrence of the event. A process specification is used to describe the inner workings of a process represented in a flow diagram. Control flow can be input directly to a conventional process or into a control process; figure 4.8 illustrates the data and control flow.

**4.3.3 Relationship between data and Control Flow**

Data flow diagrams are used to represent data and the processes that manipulate it. Control flow diagrams show how events flow among processes and illustrate those external events that cause various processes to be activated. The interrelationship between the process and control models is shown schematically in Figure 4.8. The process model is "connected" to the control model through **data conditions**. The control model is "connected" to the process model through **process activation** information contained in the CSPEC. A data condition occurs whenever data input to a process result in control output.

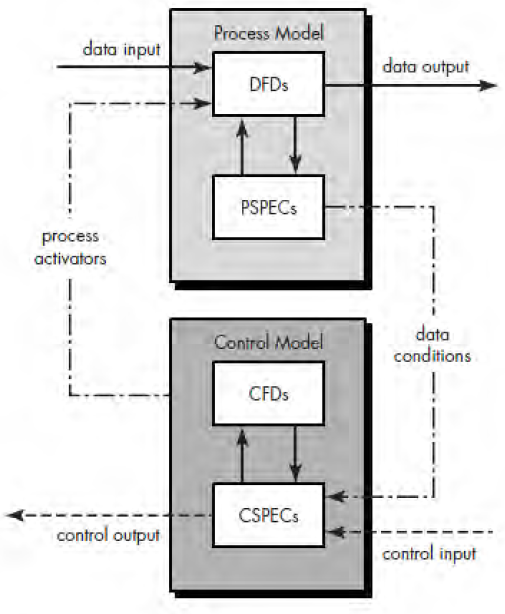


Figure 4.8 the Relationship between Data and Control models

**4.4 Behavioral Modeling**

Behavioral modeling is used to describe the behavior of the system according to different control objects Behavioral modeling is an operational principle for all requirements analysis methods.

**4.4.1 State-Transition Diagram (STD)**

The state transition diagram represents the behavior of a system by depicting its states and the events that cause the system to change state. In addition, the STD indicates what actions (e.g, process activation) are taken as a consequence of a particular event. A state is any observable mode of behavior. A state transition diagram indicates how the system moves from state to state. The basic form of the STD is:-

1. Labeled rectangle represents system state.
2. Arrow represents transition between states.
3. Each arrow is labeled with a ruled expression; the top value indicates the event (control object) that causes the transition to occur and the bottom value indicates the action that occurs as consequences of the event.

**4.5 Object-Oriented Analysis (OOA)**

OO model of computer software must exhibit data and procedural abstractions that lead to effective modularity. A class is an OO concept that encapsulates the data and procedural abstractions required to describe the content and behavior of some real world entity. The data abstractions (attributes) that describe the class are enclosed by a "wall" of procedural abstractions (called operations, methods, or services) that are capable of manipulating the data in some way. The only way to reach the attributes (and operate on them) is to go through one of the methods that form the wall. Therefore, the class encapsulates data (inside the wall) and the processing that manipulates the data (the methods that make up the wall). An object encapsulates data (represented as a collection of attributes) and the algorithms that process the data. These algorithms are called operations, methods, or services and can be viewed as modules in a conventional sense. Messages are the means by which objects interact. A message stimulates some behavior to occur in the receiving object. The behavior is accomplished when an operation is executed. Figure 4.9 shows the object oriented classes and message passing between objects.



Figure 4.9 Object-oriented classes and message passing between objects.

**Inheritance** is one of the key differentiators between conventional and OO systems. A subclass Y inherits all of the attributes and operations associated with its superclass, X. This means that all data structures and algorithms originally designed and implemented for X are immediately available for Y—no further work need be done. Reuse has been accomplished directly. **Polymorphism** is a characteristic that greatly reduces the effort required to extend an existing OO system.

OOA is grounded in a set of basic principles that were introduced. In order to build an analysis model, five basic principles were applied: (1) the information domain is modeled; (2) function is described; (3) behavior is represented; (4) data, functional, and behavioral models are partitioned to expose greater detail; and (5) early models represent the essence of the problem while later models provide implementation details.

The intent of OOA is to define all classes that are relevant to the problem to be solved-the operations and attributes associated with them, the relationships between them, and behavior they exhibit. To accomplish this, a number of tasks must occur:

1. Basic user requirements must be collected through communication between the customer and the software engineer.
2. Classes must be identified (i.e., attributes and methods are defined).
3. A class hierarchy must be specified.
4. Object-to-object relationships (object connections) should be represented.
5. Object behavior must be modeled.
6. Tasks 1 through 5 are reapplied iteratively until the model is complete.

**4.5.1 DOMAIN ANALYSIS**

The objective of domain analysis is to define a set of classes (objects) that are encountered throughout an application domain. These can be reused in many applications. This activity, called domain analysis, is performed when an organization wants to create a library of reusable classes (components) that will be broadly applicable to an entire category of applications. During domain analysis, object (and class) extraction occurs.

**i. Reuse and Domain Analysis**

Object-technologies are leveraged through reuse. Consider a simple example. The analysis of requirements for a new application indicates that 100 classes are needed. Two teams are assigned to build the application. Each will design and construct a final product. Each team is populated by people with the same skill levels and experience. Team A does not have access to a class library, and therefore, it must develop all 100 classes from scratch. Team B uses a robust class library and finds that 55 classes already exist. It is highly likely that

1. Team B will finish the project much sooner than Team A.
2. The cost of Team B's product will be significantly lower than the cost of Team A's product.
3. The product produced by Team B will have fewer delivered defects than Team A’s product.

**ii. The Domain Analysis Process**

The domain analysis process can be characterized by a series of activities that begin with the identification of the domain to be investigated and end with a specification of the objects and classes that characterize the domain.