

Software Engineering

Third Class

Lecture 8

2- Hatley and Pirbhai Extensions

The Hatley and Pirbhai extensions to basic structured analysis notation focus less on the creation of additional graphical symbols and more on the representation and specification of the control-oriented aspects of the software. The dashed arrow is once again used to represent control or event flow. Unlike Ward and Mellor, Hatley and Pirbhai suggest that dashed and solid notation be represented separately. Therefore, a *control flow diagram* is defined. The CFD contains the same processes as the DFD, but shows control flow, rather than data flow. Instead of representing control processes directly within the flow model, a notational reference (a solid bar) to a *control specification* (CSPEC) is used. In essence, the solid bar can be viewed as a "window" into an "executive" (the CSPEC) that controls the processes (functions) represented in the DFD based on the event that is passed through the window. The CSPEC is used to indicate

1. How the software behaves when an event or control signal is sensed.
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.

Using the notation described in Figures 3 and 4 (these figures are in lecture 7), along with additional information contained in PSPECs and CSPECs, Hatley and Pirbhai create a model of a real-time system. 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 5.

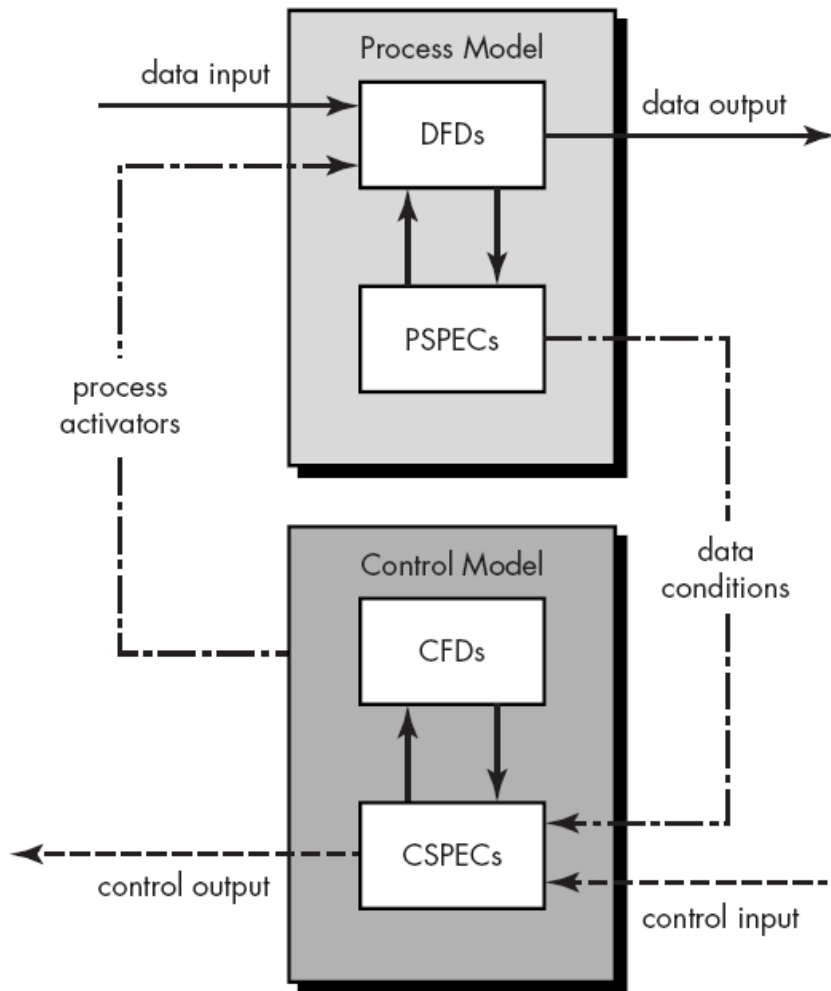


Figure 5: The relationship between data and control models

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.

This situation is illustrated in Figure 6, part of a flow model for an automated monitoring and control system for pressure vessels in an oil refinery.

The process *check and convert pressure* implements the algorithm described in the PSPEC pseudocode shown. When the **absolute tank pressure** is greater than an allowable maximum, an **above pressure** event is generated. Note that when Hatley and Pirbhai notation is used, the data flow is shown as part of a DFD, while the control flow is noted separately as part of a control flow diagram. As we noted earlier, the vertical solid bar into which the **above pressure** event flows is a pointer to the CSPEC.

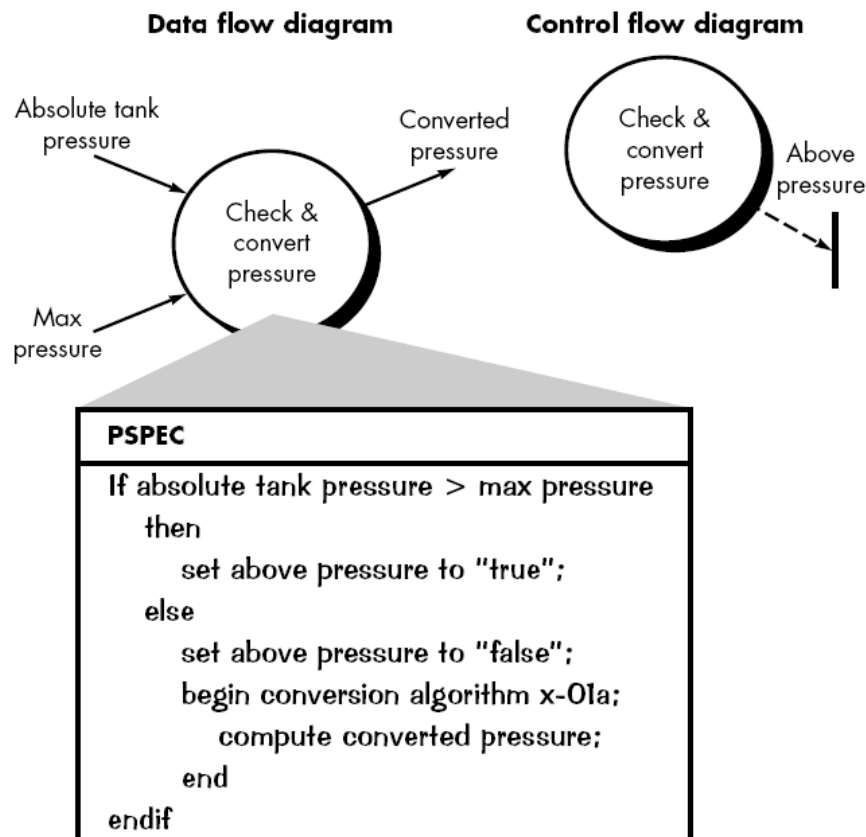


Figure 6: Data conditions

Therefore, to determine what happens when this event occurs, we must check the CSPEC.

The control specification (CSPEC) contains a number of important modeling tools. A *process activation table* is used to indicate which processes are activated by a given event. For example, a process activation table (PAT) for Figure 6 might indicate that the **above pressure** event would cause a process *reduce tank pressure* (not shown) to be invoked. In addition to the PAT, the CSPEC may contain a state transition diagram. The STD is a behavioral model that relies on the definition of a set of system states.

Behavioral Modeling

Behavioral modeling is an operational principle for all requirements analysis methods. Yet, only extended versions of structured analysis provide a notation for this type of modeling. 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.

To illustrate the use of the Hatley and Pirbhai control and behavioral extensions, consider software embedded within an office photocopier machine. A simplified representation of the control flow for the photocopier software is shown in Figure 7

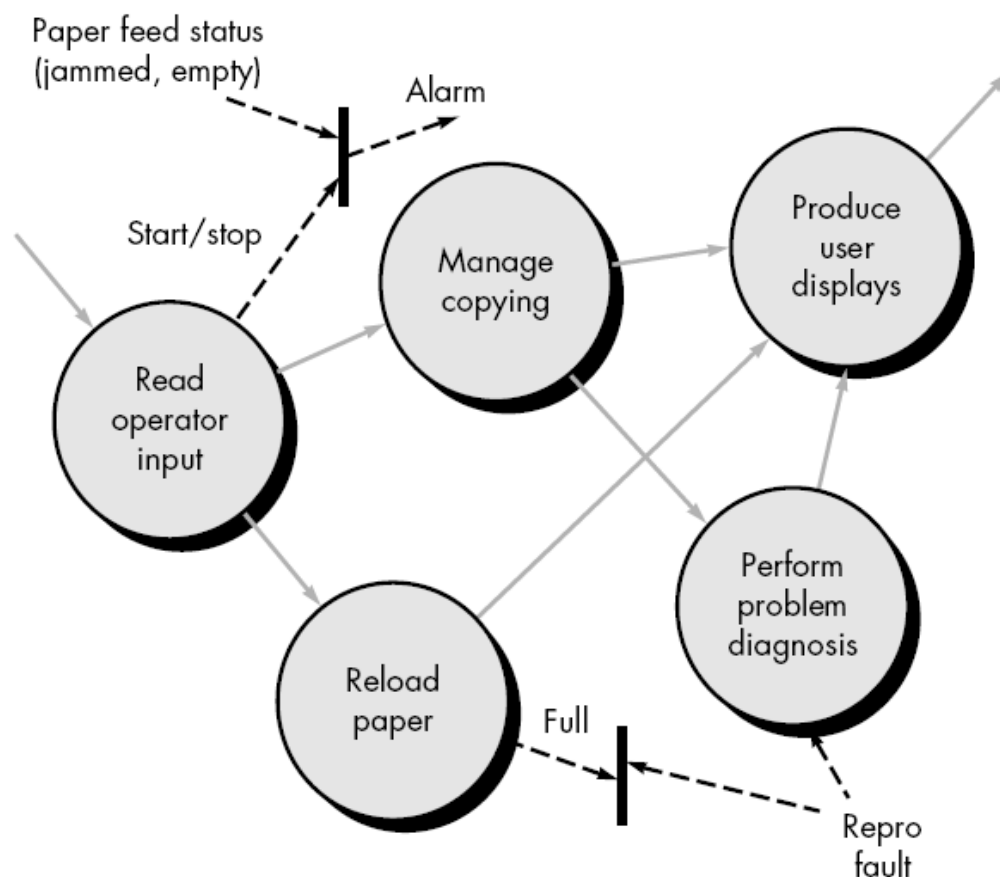


Figure 7 : Level 1 CFD for photocopier software

Data flow arrows have been lightly shaded for illustrative purposes, but in reality they are not shown as part of a control flow diagram. Control flows are shown entering and exiting individual processes and the vertical bar representing the CSPEC "window."

For example, the **paper feed status** and **start/stop** events flow into the CSPEC bar.

This implies that each of these events will cause some process represented in the CFD

to be activated. If we were to examine the CSPEC internals, the **start/stop** event

would be shown to activate/deactivate the *manage copying* process. Similarly, the

jammed event (part of **paper feed status**) would activate *perform problem diagnosis*.

It should be noted that all vertical bars within the CFD refer to the same CSPEC. An

event flow can be input directly into a process as shown with **repro fault**. However, this flow does not activate the process but rather provides control information for the process algorithm.

A simplified state transition diagram for the photocopier software is shown in Figure 8

8

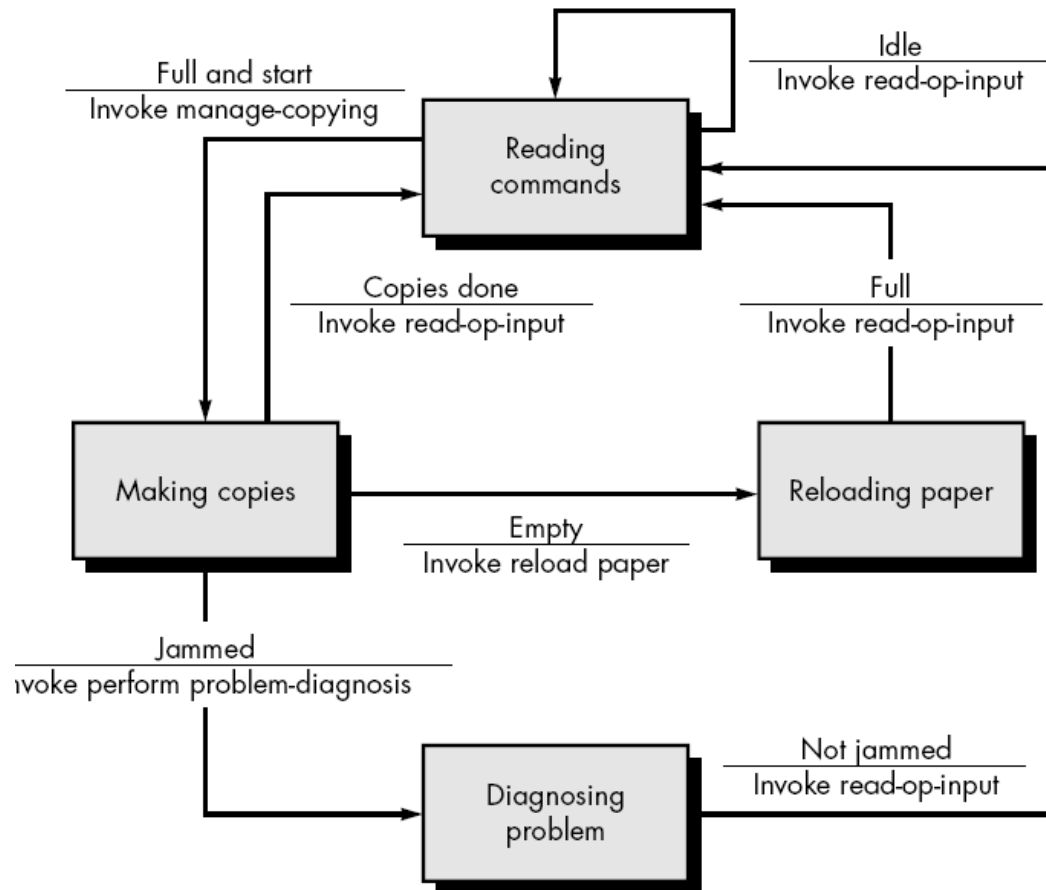


Figure 8: State transition diagram for photocopier software

The rectangles represent system states and the arrows represent transitions between states. Each arrow is labeled with a ruled expression. The top value indicates the event(s) that cause the transition to occur. The bottom value indicates the action that occurs as a consequence of the event. Therefore, when the paper tray is **full** and the **start** button is pressed, the system moves from the *reading commands* state to the *making copies* state. Note that states do not necessarily correspond to processes on a one-to-one basis. For example, the state *making copies* would encompass both the *manage copying* and *produce user displays* processes shown in Figure 7.

The Data Dictionary

The analysis model encompasses representations of data objects, function, and control. In each representation data objects and/or control items play a role.

Therefore, it is necessary to provide an organized approach for representing the characteristics of each data object and control item. This is accomplished with the data dictionary.

The data dictionary has been proposed as a quasi-formal grammar for describing the content of objects defined during structured analysis. This important modeling notation has been defined in the following manner:

The *data dictionary* is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores and [even] intermediate calculations.

Today, the data dictionary is always implemented as part of a CASE "structured analysis and design tool." Although the format of dictionaries varies from tool to tool, most contain the following information:

- *Name*—the primary name of the data or control item, the data store or an external entity.
- *Alias*—other names used for the first entry.
- *Where-used/how-used*—a listing of the processes that use the data or control item and how it is used (e.g., input to the process, output from the process, as a store, as an external entity).
- *Content description*—a notation for representing content.
- *Supplementary information*—other information about data types, preset values (if known), restrictions or limitations, and so forth.

Once a data object or control item name and its aliases are entered into the data dictionary, consistency in naming can be enforced. That is, if an analysis team member decides to name a newly derived data item **xyz**, but **xyz** is already in the dictionary, the CASE tool supporting the dictionary posts a warning to indicate duplicate names. This improves the consistency of the analysis model and helps to reduce errors.

“Where-used/how-used” information is recorded automatically from the flow models. a dictionary entry is created, the CASE tool scans DFDs and CFDs to determine which processes use the data or control information and how it is used. Although this may appear unimportant, it is actually one of the most important benefits of the dictionary. During analysis there is an almost continuous stream of changes. For large projects, it is often quite difficult to determine the impact of a change. Many a software engineer has asked, "Where is this data object used? What else will have to change if we modify it? What will the overall impact of the change be?" Because the data dictionary can be treated as a database, the analyst can ask "where used/how used" questions, and get answers to these queries.

The notation used to develop a content description is noted in the following table:

Data Construct	Notation	Meaning
	=	is composed of
Sequence	+	and
Selection	[]	either-or
Repetition	{ } ⁿ	n repetitions of
	()	optional data
	* ... *	delimits comments

The notation enables a software engineer to represent composite data in one of the three fundamental ways that it can be constructed:

1. As a sequence of data items.
2. As a selection from among a set of data items.
3. As a repeated grouping of data items. Each data item entry that is represented as part of a sequence, selection, or repetition may itself be another composite data item that needs further refinement within the dictionary.

To illustrate the use of the data dictionary, we consider the level 2 DFD for the *monitor system* process for *SafeHome*, shown in Figure 9.

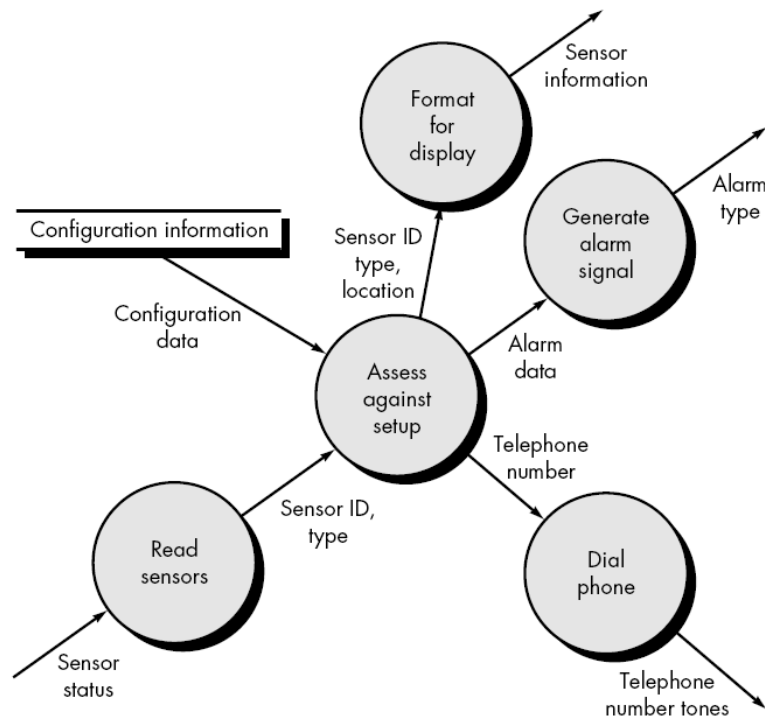


Figure 9: Level 2 DFD that refines the monitor sensors process

Referring to the figure, the data item **telephone number** is specified as input. But what exactly is a telephone number? It could be a 7-digit local number, or a 4-digit extension. The data dictionary provides us with a precise definition of **telephone number** for the DFD in question. In addition it indicates where and how this data item is used and any supplementary information that is relevant to it. The data dictionary entry begins as follows:

<u>name:</u>	telephone number
<u>aliases:</u>	none
<u>where used/how used:</u>	<u>assess against set-up</u> (output)
	<u>dial phone</u> (input)
<u>description:</u>	
telephone number = [local number long distance number]	
local number = prefix + access number	
long distance number = 1 + area code + local number	
area code = [800 888 561]	
prefix = *a three digit number that never starts with 0 or 1*	
access number = * any four number string *	

The content description is expanded until all composite data items have been represented as elementary items (items that require no further expansion) or until all composite items are represented in terms that would be well-known and unambiguous to all readers. It is also important to note that a specification of elementary data often restricts a system. For example, the definition of area code indicates that only three area codes (two toll-free and one in South Florida) are valid for this system.

The data dictionary defines information items unambiguously

For large computer-based systems, the data dictionary grows rapidly in size and complexity. In fact, it is extremely difficult to maintain a dictionary manually. For this reason, CASE tools should be used.