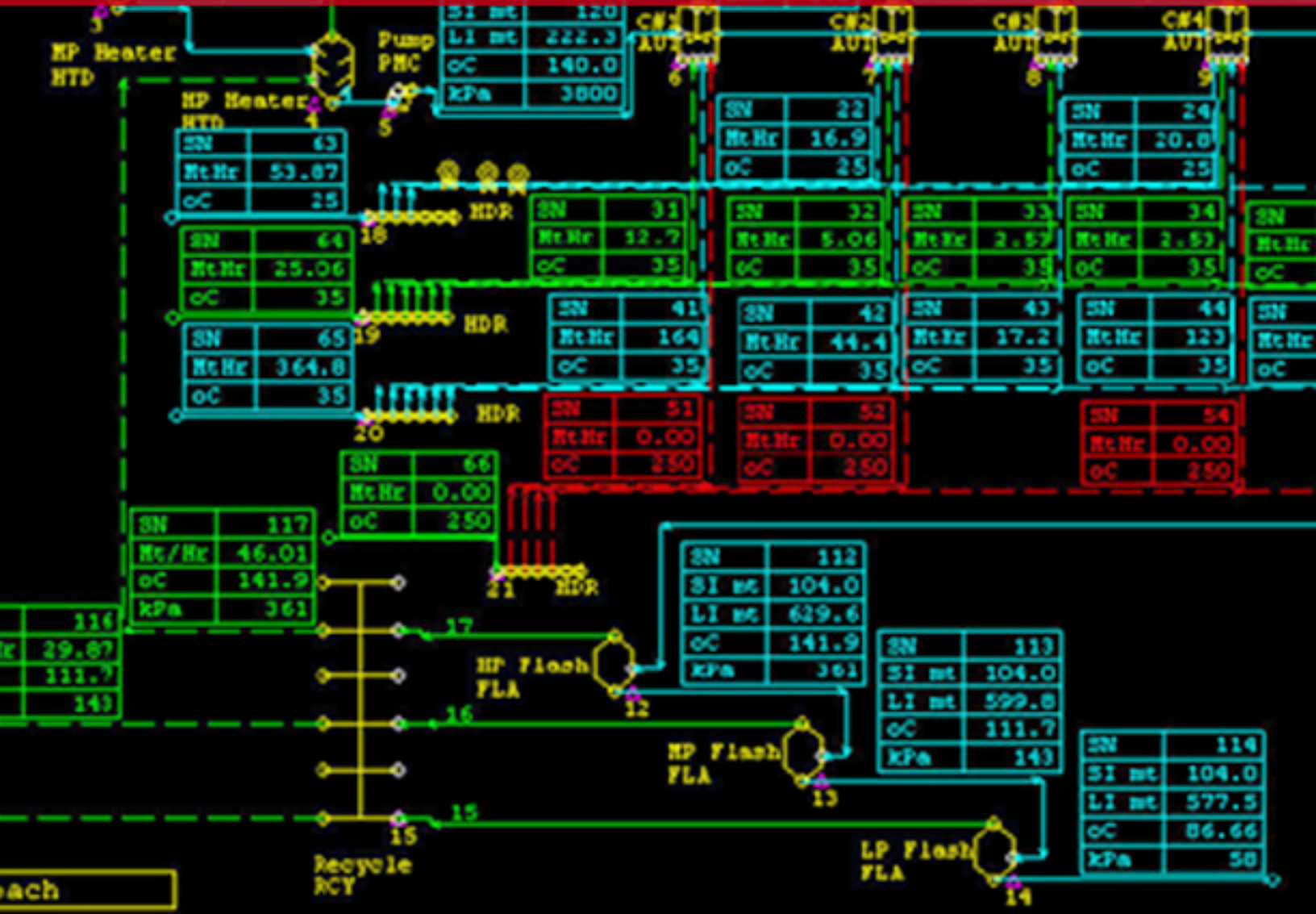
Systems Analysis

Irvin Crenshaw



First Edition, 2012

ISBN 978-81-323-3107-0

Published by:
Research World
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Introduction

Systems analysis is the study of sets of interacting entities, including computer systems analysis. This field is closely related to requirements analysis or operations research. It is also "an explicit formal inquiry carried out to help someone (referred to as the decision maker) identify a better course of action and make a better decision than he might otherwise have made."

Overview

The terms analysis and synthesis come from Greek where they mean respectively "to take apart" and "to put together". These terms are in scientific disciplines from mathematics and logic to economy and psychology to denote similar investigative procedures. Analysis is defined as the procedure by which we break down an intellectual or substantial whole into parts. Synthesis is defined as the procedure by which we combine separate elements or components in order to form a coherent whole. Systems analysis researchers apply methodology to the analysis of systems involved to form an overall picture. System analysis is used in every field where there is a work of developing something.

Information technology

The development of a computer-based information system includes a systems analysis phase which produces or enhances the data model which itself is a precursor to creating or enhancing a database. There are a number of different approaches to system analysis. When a computer-based information system is developed, systems analysis (according to the Waterfall model) would constitute the following steps:

- The development of a feasibility study, involving determining whether a project is economically, socially, technologically and organizationally feasible.
- Conducting fact-finding measures, designed to ascertain the requirements of the system's end-users. These typically span interviews, questionnaires, or visual observations of work on the existing system.

- Gauging how the end-users would operate the system (in terms of general experience in using computer hardware or software), what the system would be used for etc.

Another view outlines a phased approach to the process. This approach breaks systems analysis into 5 phases:

- Scope definition
- Problem analysis
- Requirements analysis
- Logical design
- Decision analysis

Use cases are a widely-used systems analysis modeling tool for identifying and expressing the functional requirements of a system. Each use case is a business scenario or event for which the system must provide a defined response. Use cases evolved out of object-oriented analysis; however, their use as a modeling tool has become common in many other methodologies for system analysis and design.

Practitioners

Practitioners of systems analysis are often called up to dissect systems that have grown haphazardly to determine the current components of the system. This was shown during the year 2000 re-engineering effort as business and manufacturing processes were examined as part of the Y2K automation upgrades. Employment utilizing systems analysis include systems analyst, business analyst, manufacturing engineer, enterprise architect, etc.

While practitioners of systems analysis can be called upon to create new systems, they often modify, expand or document existing systems (processes, procedures and methods). A set of components interact with each other to accomplish some specific purpose. Systems are all around us. Our body is itself a system. A business is also a system. People, money, machine, market and material are the components of business system that work together that achieve the common goal of the organization.

Chapter 1

Event Partitioning & Data Flow Diagram

Event Partitioning

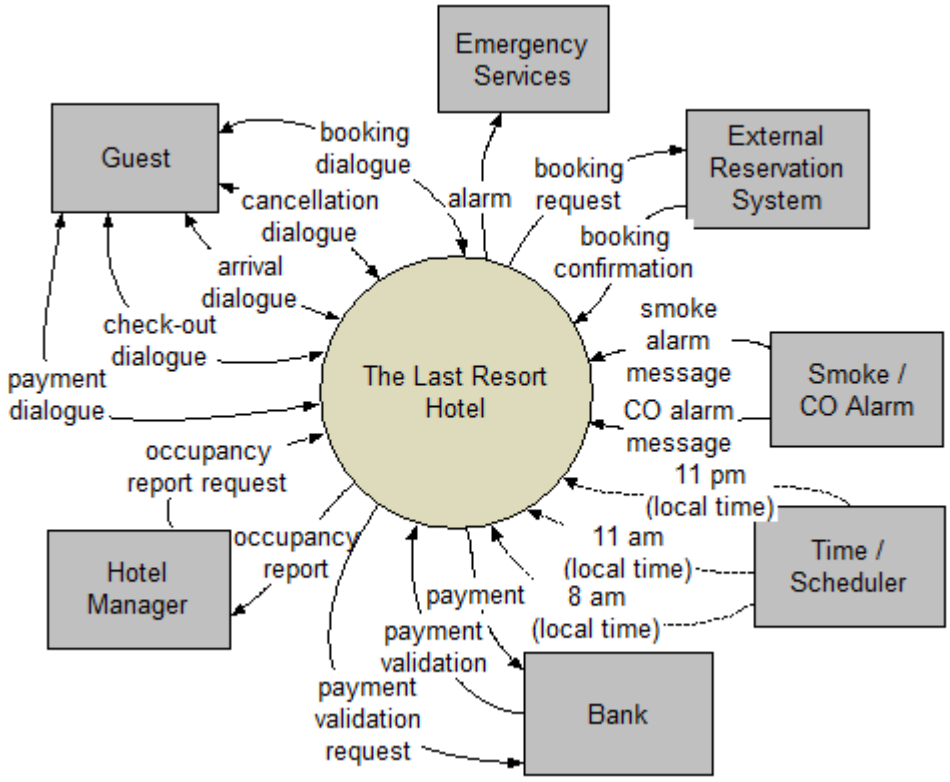
The goal of **event partitioning** is to be an easy-to-apply systems analysis technique that helps the analyst organise requirements for large systems into a collection of smaller, simpler, minimally-connected, easier-to-understand 'mini systems' / use cases. The approach is explained by Stephen M. McMenamin and John F. Palmer in *Essential Systems Analysis* . A brief version of the approach is described in the article on Data Flow Diagrams. A more complete discussion is in Edward Yourdon's *Just Enough Structured Analysis* . The description focuses on using the technique to create data flow diagrams, but it can be used to identify use cases as well.

The premise of event partitioning is that systems exist to respond to external events: identify what happens in the business environment that requires planned responses, then define and build systems to respond according to the rules of the business. In particular, a business system exists to service the requests of customers. A customer, in the jargon of the UML, is an 'actor'.

Actor → Event → Detect → Respond

The method has the following steps.

- **1.** Identify the external systems by brainstorming a list of the '**actors**' (external systems), which are the sources of external events. If you find a graphic to be helpful, create a context diagram showing the actors outside of the system under study and the flows/signals between them.



System context diagram for a fictitious hotel. (By convention, bidirectional flows, with arrows at both ends, are often used when a dialogue is initiated externally. For example, "booking dialogue" contains the flow "booking request", which is the initial trigger; "booking confirmation", the result, is sent back.)

- **2.** Putting oneself in the shoes of an 'actor' (or working with actor representatives), brainstorm a list of the '**external events**' / 'triggers' that they want the system to have a planned response to. (Note that the system cannot originate *external* events; only an actor can.)
- **3.** Identify what will enable the system to **detect** the external events:
  - the arrival of one or more pieces of **data** (possibly in the form of a message)
  - the arrival of one or more points in **time**
- **4.** Identify the **planned response(s)** that the system may carry out when the events occur. It's the response(s)/use case(s) that will enable the system to achieve its goals.

The technique was extended with 'non-event' events by Paul T. Ward and Stephen J. Mellor in *Structured Development for Real-Time Systems: Essential Modeling Techniques* .

> "Since the terminators [actors] are by definition outside the bounds of the system-building effort represented by the model, the implementors cannot modify the terminator [actor] technology at will to improve its reliability. Instead, they must build responses to terminator [actor] problems into the essential model of the system. A useful approach to modeling responses to terminator [actor] problems is to build a list of 'normal' events and then to ask, for each event, 'Does the system need to respond if this event *fails to* occur as expected?' " [emphasis added]

Identifying Requirements and Their Reasons

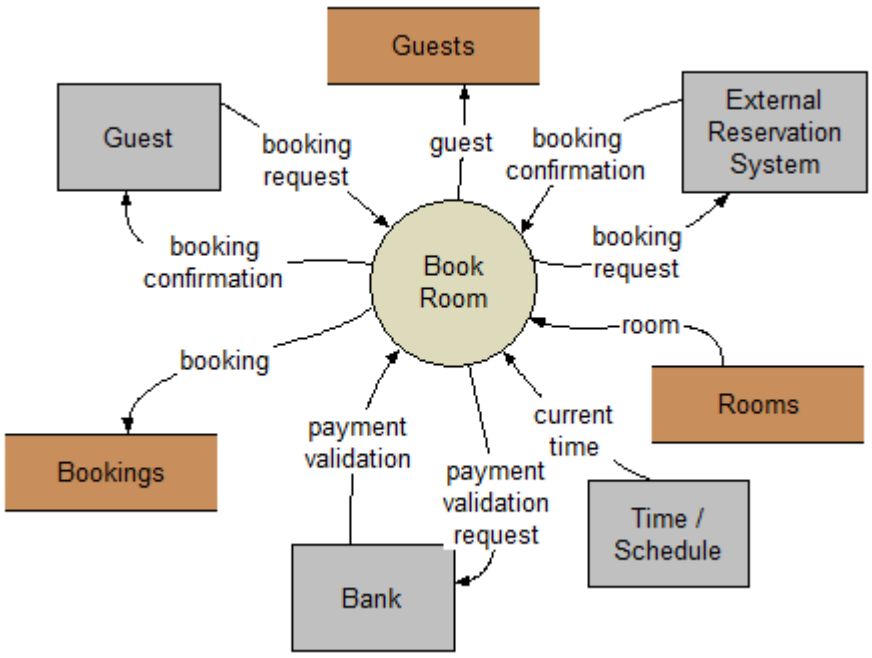
The event-response information may be captured in a table. The event is the *raison d'être* for the response, which gives 'traceability' from the response back to the environment.

1. Actor	2. External Event / Trigger	3. Detected by	4. Response(s) / Use Case(s)
Guest	Guest requests a room of a certain type, for a particular arrival date, departure date, at a certain rate, etc.	booking request + (payment validation) + (*external reservation system* booking confirmation)	Book Room (may include guaranteed booking, alternate hotel booking, waitlisted booking)
Guest	Guest asks to cancel room booking.	cancellation request	Cancel Booking
Guest	Guest arrives at hotel.	arrival message = * * = [guest name ; booking reference]	Check in Guest
Time / Scheduler	Guest *fails to* arrive at hotel. [This is a 'non-event' event.]	11 pm (local time) [A 'non-event' event is detected by the arrival of a point in time, a deadline.]	Create Guest Bill, Update Booking
Guest	Guest asks to check out of hotel.	check-out request = * * = [guest name ; room number]	Create Guest Bill, Update Room Occupancy
Time / Scheduler	Guest *fails to* check out of hotel. [This is a 'non-event' event.]	11 am (local time) [A 'non-event' event is detected by the arrival of a point in time, a deadline.]	Create Guest Bill

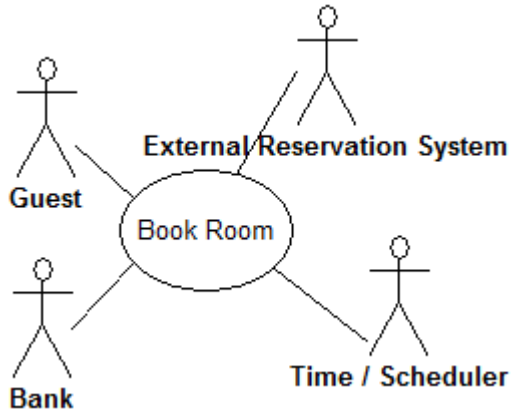
Guest	Guest offers payment of bill.	payment vehicle = * * = [cash ; cheque ; credit card ; debit card] + (guest id)	Accept Guest Payment
Time / Scheduler	Time to prepare Room Occupancy Report for previous night.	8 am (local time)	Report on Room Occupancy
Hotel Manager	Hotel Manager requests Room Occupancy Report.	occupancy report request	Report on Room Occupancy
Smoke / CO Alarm	Alarm detects smoke.	smoke alarm message	Report Smoke Alarm
Smoke / CO Alarm	Alarm detects CO (carbon monoxide).	CO alarm message	Report CO Alarm

Defining requirements

This approach helps the analyst to decompose the system into 'mentally bite-sized' mini-systems using events that require a planned response. The level of detail of each response is at the level of 'primary use cases'. Each planned response may be modelled using DFD notation or as a single use case using use case diagram notation.



Single process in a fictitious hotel using data flow diagram notation.



Single use case in a fictitious hotel using use case diagram notation.

The *basic flow* within a process or use case can usually be described in a relatively small number of steps, often fewer than twenty or thirty, possibly using something like 'structured English'. Ideally, all of the steps would be visible all at once (often a page or less). The intention is to reduce one of the risks associated with short-term memory, namely, forgetting what is not immediately visible ('out of sight, out of mind').

Alternatively, using the notations of structured techniques, an analyst could create a 'Nassi–Shneiderman diagram'. In the UML, the use case could be modelled using an activity diagram, a sequence diagram, or a communication diagram. This could be problematic if there are many complex scenarios of the use case; the analyst may wish to model all or most of the scenarios.

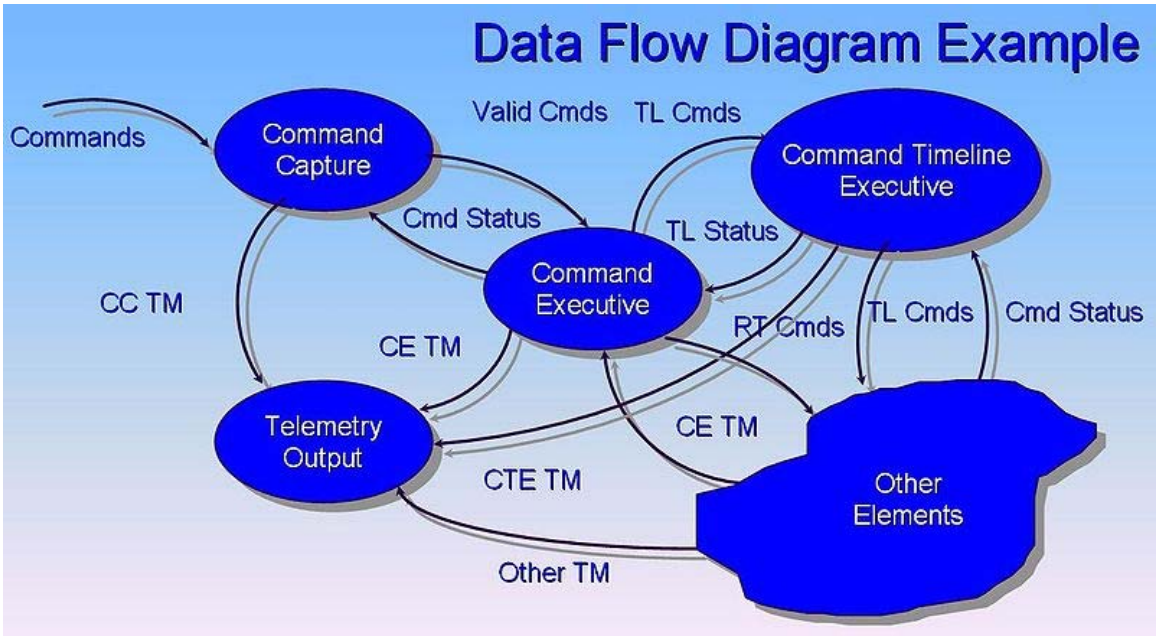
Complexity versus fragmentation

If the response is lengthy or complex (i.e., more than a page of text), an analyst may decompose ('factor out' or deduplicate) into smaller 'secondary use cases' to keep the 'parent' primary use case smaller and simpler. These secondary use cases may prove to be reusable as well. (In a UML use case diagram, they would be drawn as extended or included use cases, which are related to one or more primary use cases.)

While describing a use case, an analyst may also uncover 'business rules'. Some analysts suggest capturing business rules in a separate document using the Object Constraint Language or some other formal notation. Then when a business rule must be obeyed in a use case, the analyst makes reference to it. This minimises repetition within a specification, but risks fragmentation of a specification. One technique that may reduce this tension is to use hyperlinks in the specification document.

In addition to functional requirements captured in a use case description, an analyst may include such non-functional requirements as response time, learnability, etc.

Data Flow Diagram



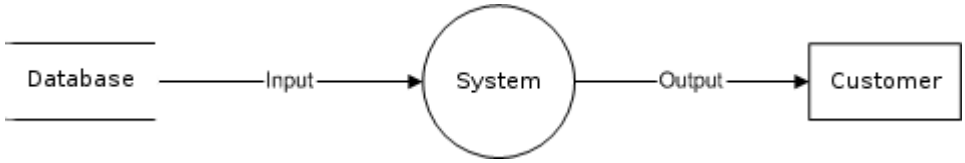
Data flow diagram example.

A **data flow diagram** (**DFD**) is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design).

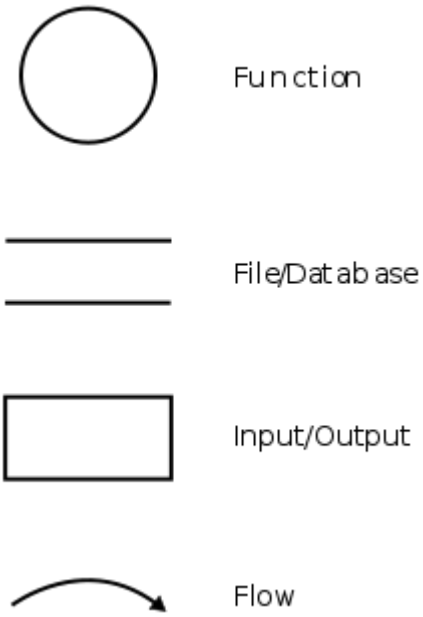
On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process.

A DFD provides no information about the timing of processes, or about whether processes will operate in sequence or in parallel. It is therefore quite different from a flowchart, which shows the flow of control through an algorithm, allowing a reader to determine what operations will be performed, in what order, and under what circumstances, but not what kinds of data will be input to and output from the system, nor where the data will come from and go to, nor where the data will be stored (all of which are shown on a DFD).

Overview



Data flow diagram example.



Data flow diagram - Yourdon/DeMarco notation.

It is common practice to draw a context-level data flow diagram first, which shows the interaction between the system and external agents which act as data sources and data sinks. On the context diagram (also known as the 'Level 0 DFD') the system's interactions with the outside world are modelled purely in terms of data flows across the *system boundary*. The context diagram shows the entire system as a single process, and gives no clues as to its internal organization.

This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.

Data flow diagrams were proposed by Larry Constantine, the original developer of structured design, based on Martin and Estrin's "data flow graph" model of computation.

Data flow diagrams (DFDs) are one of the three essential perspectives of the structured-systems analysis and design method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users are able to visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's dataflow diagrams can be drawn up and compared with the new system's data flow diagrams to draw comparisons to implement a more efficient system. Data flow diagrams can be used to provide the end user with a physical idea of where the data they input

ultimately has an effect upon the structure of the whole system from order to dispatch to report. How any system is developed can be determined through a data flow diagram.

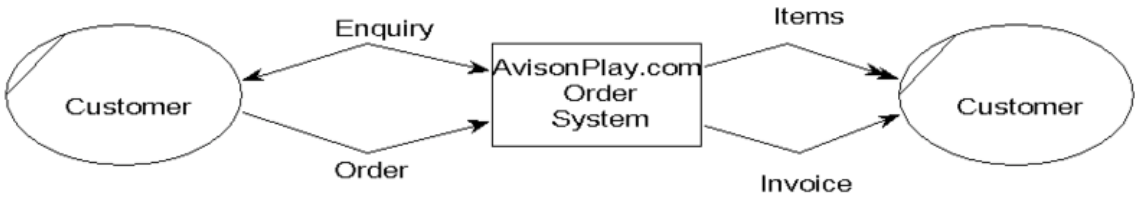
In the course of developing a set of *levelled* data flow diagrams the analyst/designers is forced to address how the system may be decomposed into component sub-systems, and to identify the transaction data in the data model.

There are different notations to draw data flow diagrams (Yourdon & Coad and Gane & Sarson), defining different visual representations for processes, data stores, data flow, and external entities.

Developing a data flow diagram

Event partitioning approach

Event partitioning was described by Edward Yourdon in *Just Enough Structured Analysis*.



A context level Data flow diagram created using Select SSADM.

This level shows the overall context of the system and its operating environment and shows the whole system as just one process. It does not usually show data stores, unless they are "owned" by external systems, e.g. are accessed by but not maintained by this system, however, these are often shown as external entities.

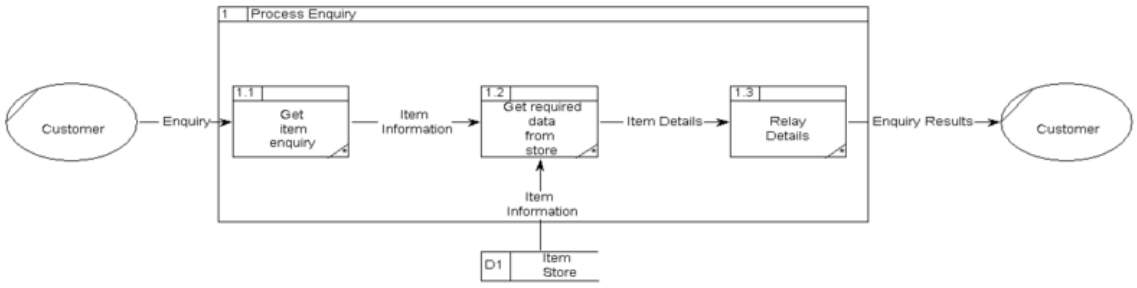
Number the Processes As a convenient way of referencing the processes in a DFD, most systems analysts number each bubble. It doesn't matter very much how you go about doing this — left to right, top to bottom, or any other convenient pattern will do -- as long as you are consistent in how you apply the numbers. The only thing that you must keep in mind is that the numbering scheme will imply, to some casual readers of your DFD, a certain sequence of execution. That is, when you show the DFD to a user, he may ask, "Oh, does this mean that bubble 1 is performed first, and then bubble 2, and then bubble 3?" Indeed, you may get the same question from other systems analysts and programmers; anyone who is familiar with a flowchart may make the mistake of assuming that numbers attached to bubbles imply a sequence. This is not the case at all. The DFD model is a network of communicating, asynchronous processes, which is, in fact, an accurate representation of the way most systems actually operate. Some sequence may be implied by the presence or absence of data (e.g., it may turn out that bubble 2

cannot carry out its function until it receives data from bubble 1), but the numbering scheme has nothing to do with this. So why do we number the bubbles at all? Partly, as indicated above, as a convenient way of referring to the processes; it's much easier in a lively discussion about a DFD to say "bubble 1" rather than "EDIT TRANSACTION AND REPORT ERRORS." But more importantly, the numbers become the basis for a hierarchical numbering scheme when we introduce leveled dataflow diagrams in Section 9.3.

Level 1 (high level diagram)

This level (level 1) shows all processes at the first level of numbering, data stores, external entities and the data flows between them. The purpose of this level is to show the major and high-level processes of the system and their interrelation. A process model will have one, and only one, level-1 diagram. A level-1 diagram must be balanced with its parent context level diagram, i.e. there must be the same external entities and the same data flows, these can be broken down to more detail in the level 1, example the "enquiry" data flow could be split into "enquiry request" and "enquiry results" and still be valid. This is all about using your creativity. So why do we number the bubbles at all? Partly, as indicated above, as a convenient way of referring to the processes; it's much easier in a lively discussion about a DFD to say "bubble 1" rather than "EDIT TRANSACTION AND REPORT ERRORS."

Level 2 (low level diagram)



A Level 2 Data flow diagram showing the "Process Enquiry" process for the same system.

This level is a decomposition of a process shown in a level-1 diagram, as such there should be a level-2 diagram for each and every process shown in a level-1 diagram. In this example, processes 1.1, 1.2 & 1.3 are all children of process 1. Together they wholly and completely describe process 1, and combined must perform the full capacity of this parent process. As before, a level-2 diagram must be balanced with its parent level-1 diagram.

Chapter 2

Control-Feedback-Abort Loop

Too often **systems** fail, sometimes leading to significant loss of life, fortunes and confidence in the provider of a product or service. It was determined that a simple and useful tool was needed to help in the analysis of interactions of groups and systems to determine possible unexpected consequences. The tool didn't need to provide every possible outcome of the interactions but needed to provide a means for analysts and product/service development stakeholders to evaluate the potential risks associated with implementing new functionality in a system. They needed a brainstorming tool to help ascertain if a concept was viable from a business perspective. The ***Control-Feedback-Abort Loop*** and the analysis diagram is one such tool that has helped organizations analyze their system workflows and workflow exceptions.

The concept of the Control-Feedback-Abort (CFA) loop is based upon another concept called the '*Control – Feedback Loop'*. The Control – Feedback Loop has been around for many years and was the key concept in the development of many electronic designs such as *Phase-Lock Loops*. The core of the CFA loop concept was based on a major need that corporate executives and staff can anticipate the operation of systems, processes, products and services they use and create before they are developed.

History of CFA Loop concept

The concept of the CFA loop was developed by T. James LeDoux, 'Jim', a Senior Consultant and software QA / test expert and owner of Alpha Group 3 LLC, a test management consulting company. In 1986, Mr. LeDoux, with assistance from Mr. Warren Yates, a former engineer from General Dynamics, Inc., found that using a Control and Feedback concept for analyzing group and system dynamics was not providing them with the full picture when systems were going out of control. In 1996, Jim LeDoux and Dr. Larry W. Smith, Ph.D., president of Remote Testing Services, Inc., discussed the issue at length and came to the conclusion that some other form of control must be present when a system goes out of control, even if the control is unintended.

In 1997, Mr. LeDoux used the change of behavior a person exhibits when driving a car at the time a police car pulls in behind them to describe how a change of control occurs. He demonstrated this phenomenon at a 2003 Product Development and Management Association (PDMA) meeting in Denver by showing the action of the first control (traffic, signs and speed) being aborted by the driver and a second control (police car, signs and speed) becoming the primary control. In 2004, Mr. LeDoux worked with Dr. Susan Wheeler, Ed. D., a former Instructional Design Consultant with Nims, Inc. and the present Director of Technology Services at Illinois Central College, to identify the range of uses for the CFA Loop. The CFA Loop is now being used to analyze system activities in several Fortune 100 companies. A discussion of its use is also included in the management book "Takeoff!: The Introduction to Project Management Book that Will Make Your Projects Take Off and Fly!" by Dr. Dan Price, D.M. ISBN 9780970746115

It was found that strong similarities existed between the concept of '*Control Charts*' and the CFA Loop. The difference in the two concepts was that control charting is used as a dynamic measurement of present conditions. The CFA Loop is used to analyze how a closed-loop system is supposed to work and what are the expectations when alternate controls take over by either intent or accident. A comparison of the CFA Loop and its relationship to Control Charts are presented in a later section of this discussion.

The Control-Feedback concept

The Control – Feedback concept consisted of a 'Control' that gave information on the way the component was to perform and then adjustments to the control's present operation based upon the feedback. It used a concept called 'Sampling' to determine how often the 'Control' used the 'Feedback' information so that the 'Control' could modify instructions to the component.

What is the CFA Loop

Figure 1 shows a model of the CFA loop. The CFA loop consists of three main elements - The Control element, the Feedback element and the Abort element. Within any system, the lack of any one of these three elements will result in the system failing at some point in time. The term 'system' used in this document can represent any environment, task, process, procedure or system in a physical, organizational or natural structure where an entity will respond to influences. It has been found, through experience, that even trees appear to follow the CFA model. The diagram in Figure 1 can be used as an analysis diagram by inserting functions of the controls, feedbacks and aborts in each of the related circles defining the system being analyzed. (Example: Control - Workflow requests, Feedback - Results of requests, Aborts - Requests that failed, workflow exception path)

The CFA model can be used effectively with 3-sigma Control Charts. CFA loops and Control Charts share the same functionality, which will be discussed later in this document.

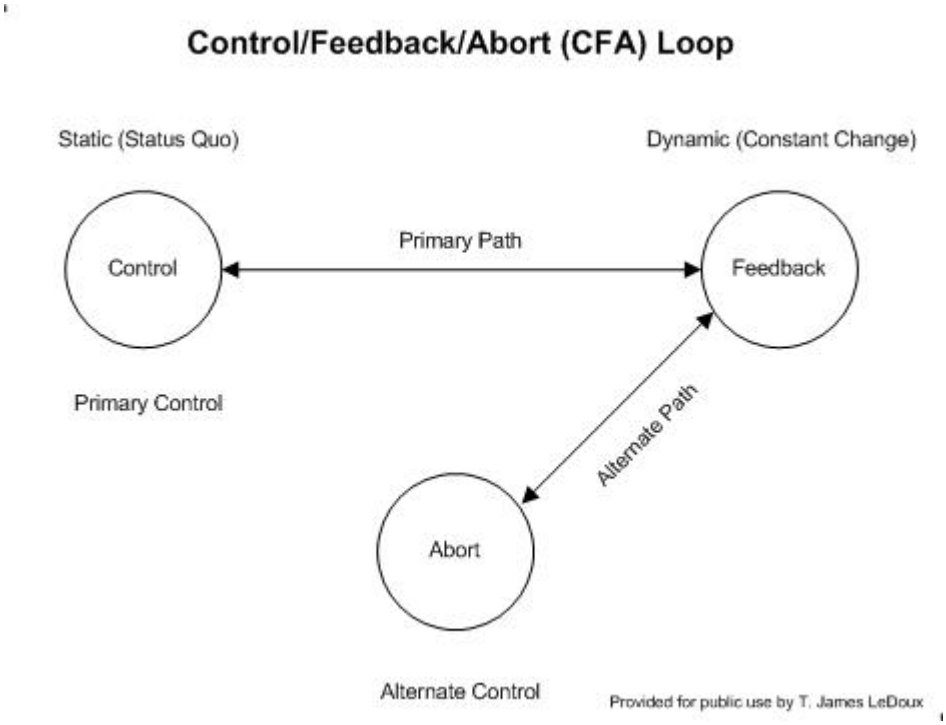


Figure 1 – The CFA Loop

A description of the Control-Feedback-Abort (CFA) Loop

As mentioned, the CFA Loop consists of three elements – Control, Feedback and Abort. First, we will discuss the Control element of the loop.

The control element

The Control element of the CFA loop, as highlighted in Figure 2, controls the activity of the system in question. A basic characteristic of the Control element is that it is always in a static state until it receives new information from the feedback. This static state is, in reality, the Control element holding the system in a status quo condition. Using an automobile as an example, if the previous instruction provided by the Control to the auto was to accelerate, it would continue to accelerate until a feedback reading would indicate to the Control that the Control should issue an instruction to stop accelerating.

Remember, the idea of the static condition is not saying that nothing is happening but rather to say that nothing is changing in the instructions given to the system since the last instruction from the Control. If the last instruction by the Control is to accelerate, the system will continue to accelerate until told otherwise.

The Control element is the ‘primary control’ for the system. While everything is operating within a ‘normal’ operational mode, the Control element remains the primary control.

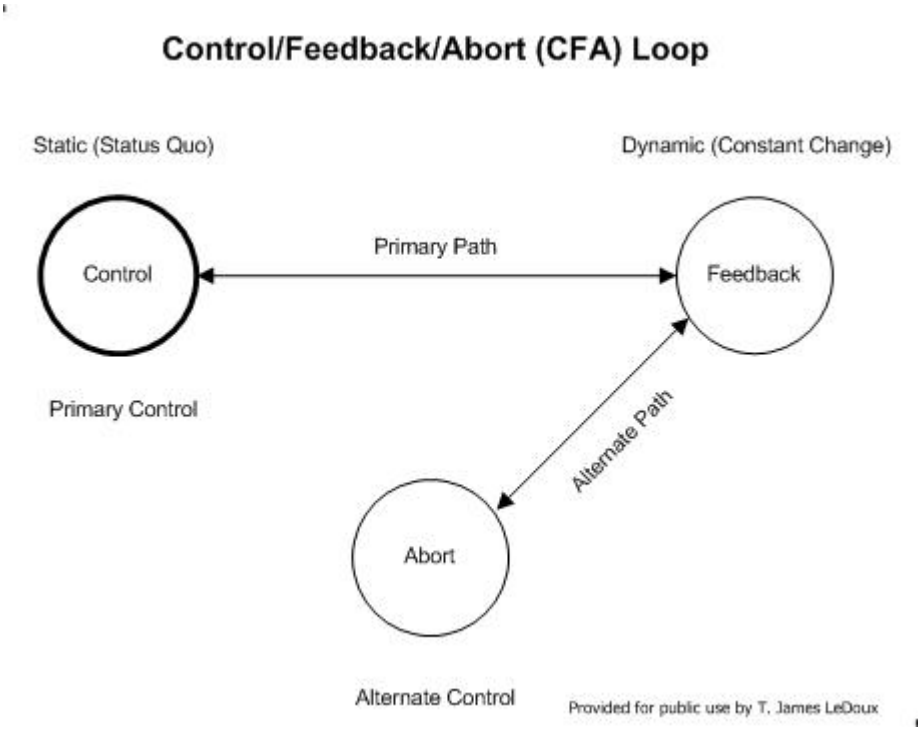


Figure 2 – CFA Loop – Control Element

The feedback element

The Feedback element feeds back information on the present state of the system. Due to the fact that the Feedback element is always reading the present state of the system, the feedback element has the basic characteristic of always being in a 'dynamic' state. This means that the feedback is reading constantly changing conditions. No system is ever in a non-changing condition except if it is off, no longer functioning or dead. Look at a computer in a wait state. It is still performing administrative activities even while it waits for some activity to happen. Change is the constant state of the Feedback element.

For this reason, the Feedback element needs to provide information to the Control element at intervals necessary to provide the Control element time to adequately respond to the changing environment.

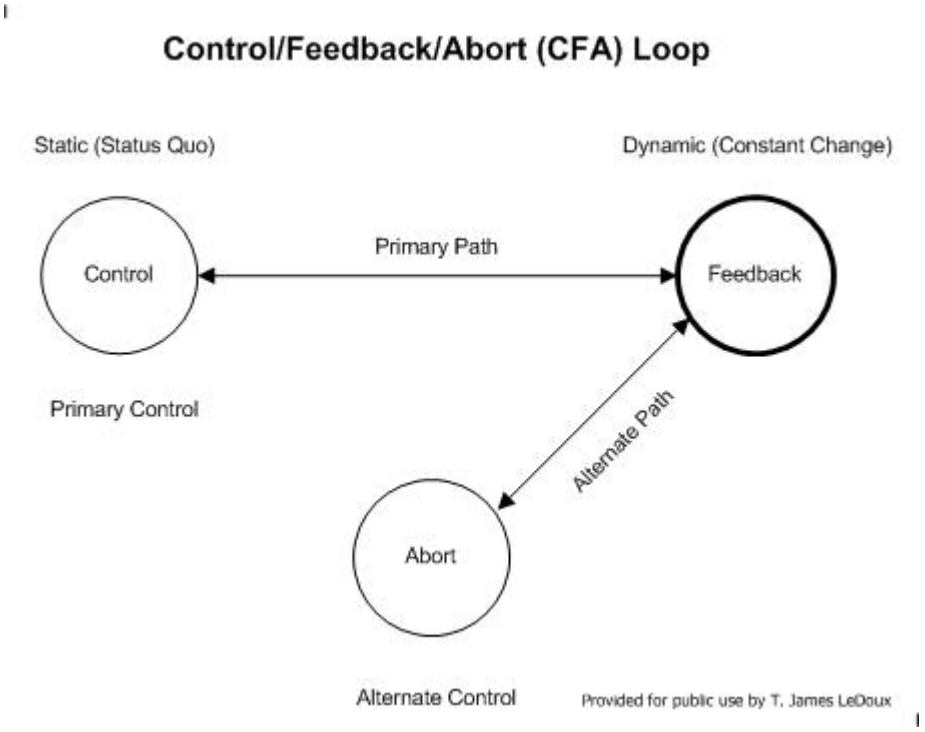


Figure 3 – CFA Loop – Feedback Element

The communication between the Control element and the Feedback element is performed by way of the 'Primary Path' (see Figure 4). The Primary Path is a bidirectional path that allows for the Control Element to request a sample of the information and the Feedback element to respond.

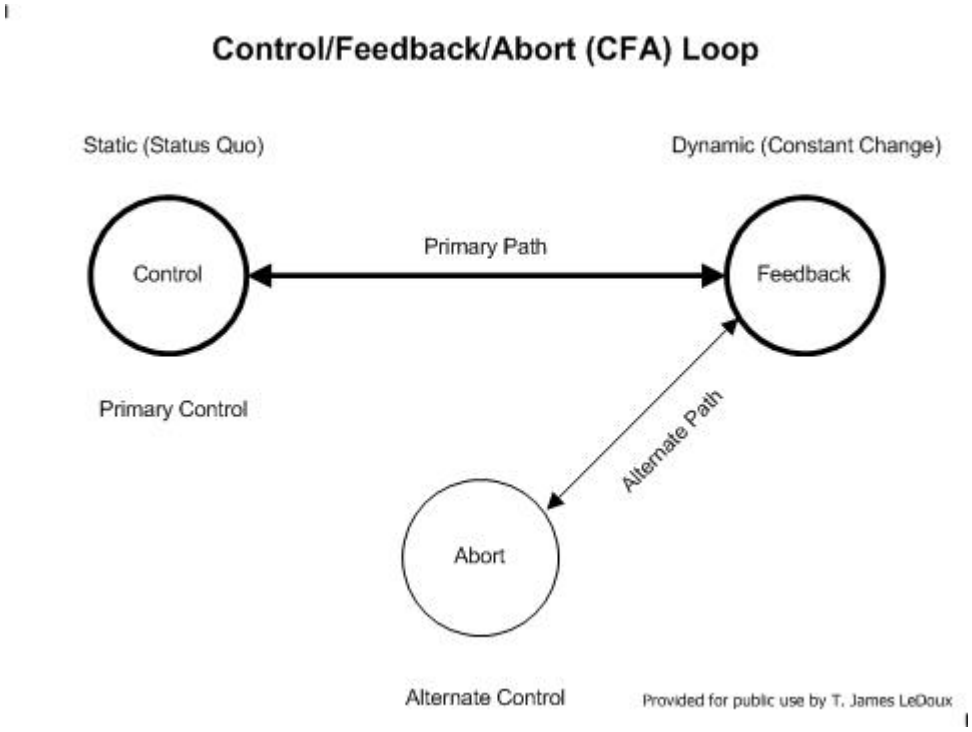


Figure 4 – CFA Loop – Primary Path

The abort element

The Abort element (see Figure 5) is so named because it responds to conditions that resulted in the primary path being 'aborted'. The Abort element then takes over the act of control until conditions can be brought back into acceptable parameters.

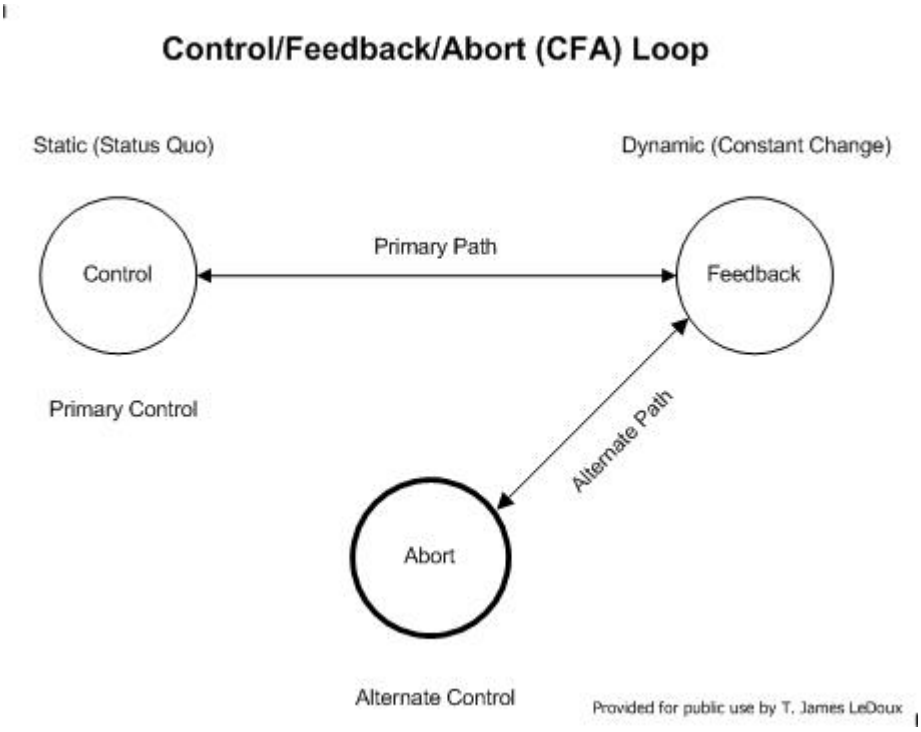


Figure 5 – CFA Loop – Abort Element

The 'Alternate Path' (see Figure 6) is used for communication between the Alternate Control (Abort) and the Feedback. The Feedback at this point may be a different set of feedbacks than was defined for the primary path.

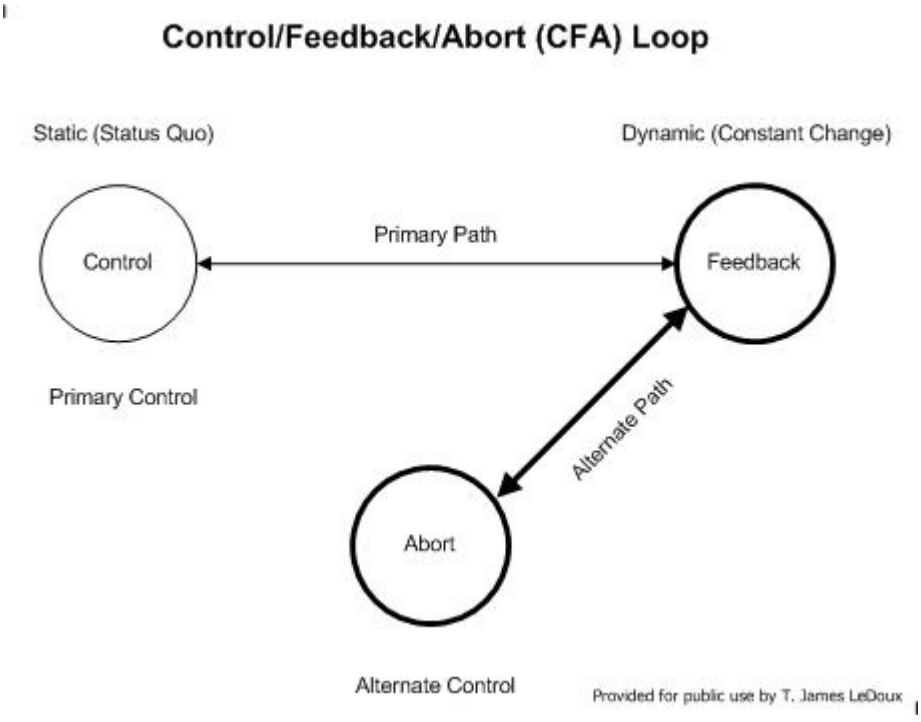


Figure 6 – CFA Loop – Alternate Path

To demonstrate that the Feedback may be another set of feedback elements, we look at the following example.

Let's use the act of driving the auto once more for our example (see Figure 7). When a driver is driving the car, the primary path is the Control element (gas pedal) and the Feedback element (speedometer and street signs). Once a stop sign is detected ahead, the driver will take the foot off the gas pedal (primary control) and press the brake pedal (alternate control). Note that the driver is no longer looking at the speedometer or street signs once the auto gets to the stop sign. The driver is looking for other cars that may cross his path. In other words, the driver is looking for a different set of feedback sources. Once he feels it is safe to go, he will go back to the primary control and feedback and the primary path.

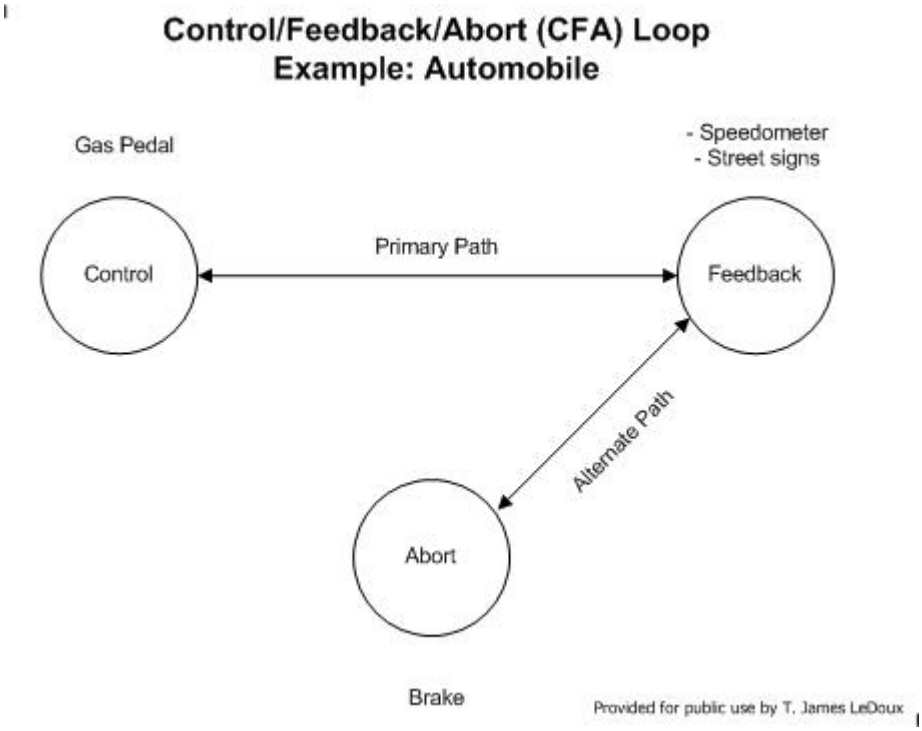


Figure 7 – CFA for Automobile Speed Adjustment

Sampling and the feedback element

In order for the Control element to be able to give proper instructions on what the system needs to do next, the information provided by the Feedback needs to be a true representation of the present conditions. If the feedback information is sampled by the Control element too often, it can put unnecessary demands on the system. If the information is not read often enough, considerable error can occur resulting in system

failure. The solution to this dilemma is to sample when needed, at a rate that allows us to have confidence that we can still maintain control over the system.

Going back to our auto. The rate we sample the street signs for information is going to be different than when we look at the speedometer. We may also change our sampling rate when certain outside influences introduce themselves into the feedback mix. If we have a police car behind us, odds are that we will be sampling the speedometer much more often than if a police car was not there.

Creating the control loop diagram using the CFA Loop

The Control Loop Diagram is a chart that provides a list of each of the conditions we discover during the analysis of the interaction of the specific item in question. A basic Control Loop Diagram is shown in Table 1.

Control Element Conditions	Feedback Element Conditions	Abort Element Conditions
Control Element Name	Feedback Element Name	Abort Element Name
Numbered list	Numbered list	Numbered list

Table 1 – Control Loop Diagram Template

The Control Loop Diagram provides a vehicle for the CFA Loop to be used effectively. The following is a sequence that allows for us to create the CFA Loop analysis information and convert it into a Control Loop Diagram. The process is:

A. Identify the perspective of the CFA Loop.

It is important to know what the perspective is. We may be looking at the environment from a specific perspective (i.e. from the viewpoint of a Test Manager looking at defects or a Development Manager looking at versions.) The perspective will determine what is to be the Control and what is providing the Feedback for the analysis.

B. Identify what is controlling the environment.

C. Identify the Feedback components.

By identifying the controlling environment and the feedback elements, we can identify the parameters of the primary path.

D. Identify the conditions that would lead to an abort of the primary path.

The abort conditions can give us an insight into the limitations and boundaries the primary path must operate within.

E. Identify the processes the Control will use to manage the environment.

The interaction between the control and feedback elements can now be analyzed and the resulting information can be mapped into the Control Loop Diagram.

F. Identify the processes used when the Abort is given control.

An Example of the CFA Loop – Control loop diagram relationship

The following CFA Loop and Control Loop Diagram exhibit the relationship between a Version Control/Defect Reporting CFA Loop (Figure 8) and its associated Control Loop Diagram (Table 2).

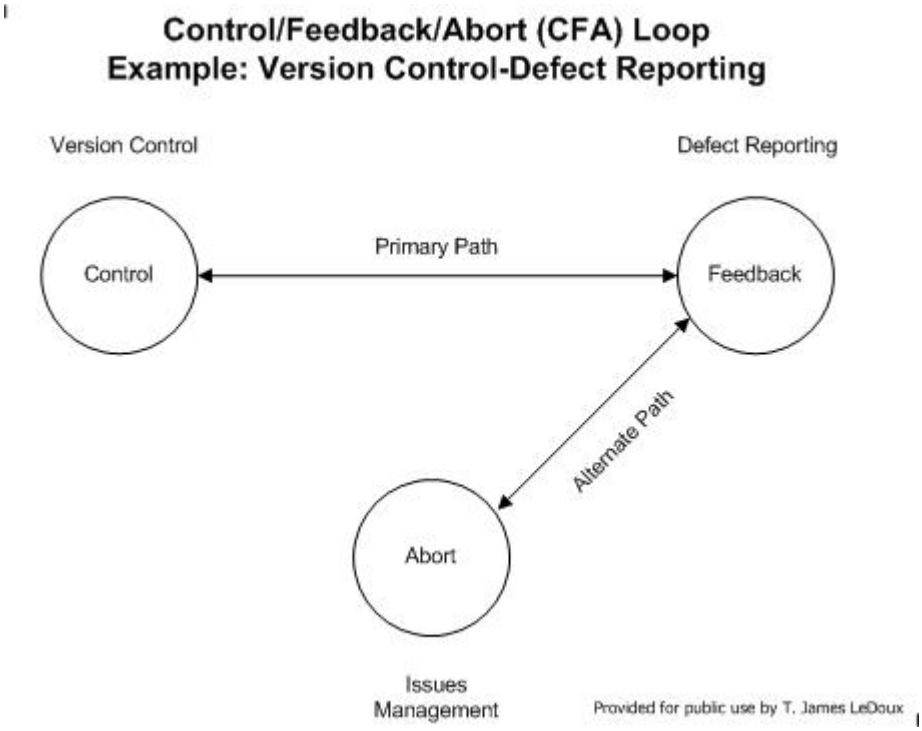


Figure 8 – CFA Loop for Version Control/Defect Reporting

A Control Loop Diagram for the CFA Loop with a focus on Version Control as the Control element (see Table 2) should look similar to the following table (mapped during an analysis brainstorming session):

Control Element Conditions	Feedback Element Conditions	Abort Element Conditions
Version Control (A)	Defect Reporting (B)	Issues Management (C)

1. Version must match the incremental sub-version number expected to fix the next set of defects	1. Defects reported back by critical levels	1. Defects that cannot be fixed within a predefined time must be escalated
2. Defects that are fixed, tested and passed cause the sub-version count to be incremented	2. Critical defect count	2. Defects violating the critical defect count or the age limits on critical defects will automatically create an Abort
3. Defects not above a predefined count		

Table 2 – Control Loop Diagram

Control charts

Control Charts have a very close relationship to the CFA Loop. Control Charts are used to provide a means of tracking the trend and condition of a specific measured item. The Control Chart (see Figure 9) uses the standard deviation of sampled items to determine whether the item is in-bounds (within acceptable conditions) or out of bounds (outside of acceptable conditions). The +3s is also identified as the Upper Defined-Control Limit or UDL. The -3s is also known as the Lower Defined-Control Limit or LDL.

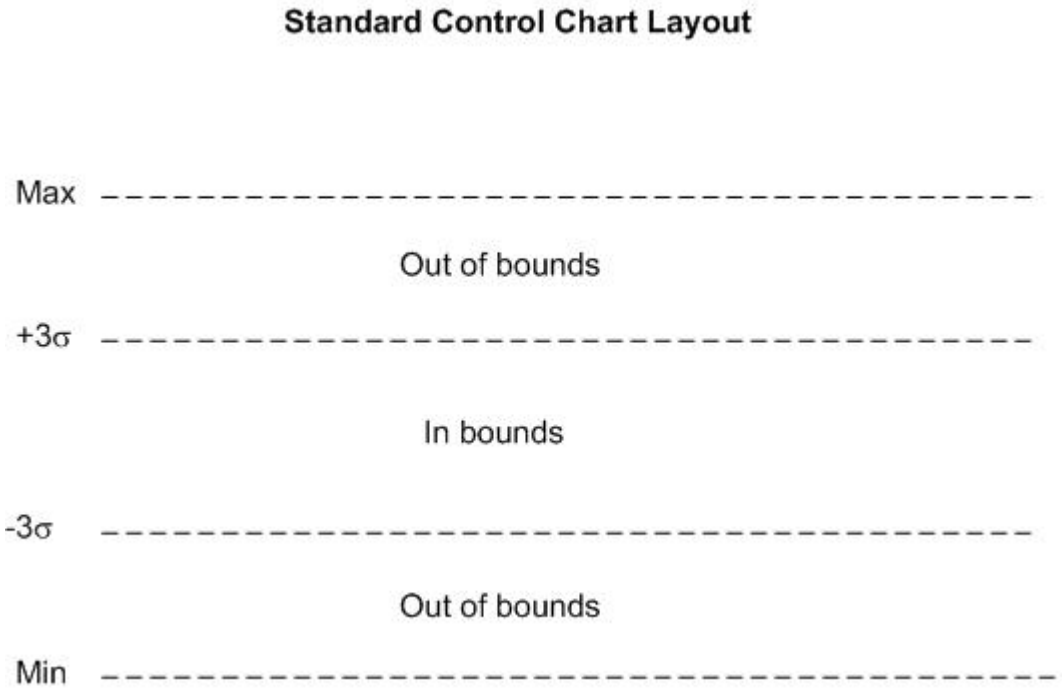


Figure 9 – Control Chart Limits

Those items that are in bounds are considered to be in control (see Figure 10). They can be the Control element of the CFA Loop.

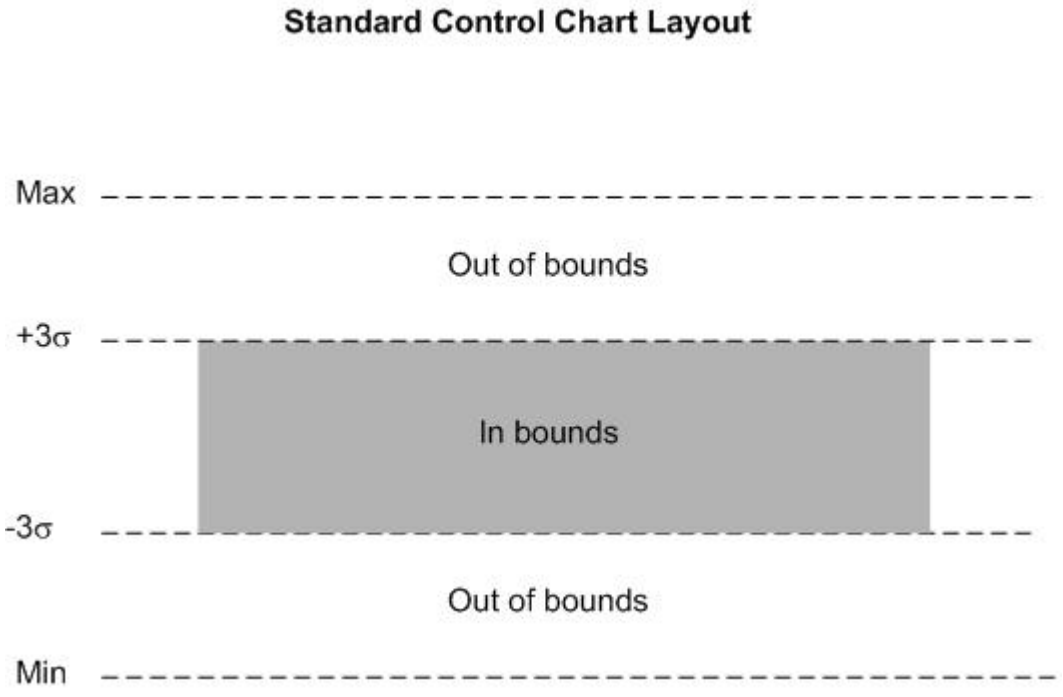


Figure 10 – Control Chart In-Bounds Area

Those items that are out of bounds are said to be out of control (see Figure 11). The out of bounds areas can also be identified as the Abort element of the CFA Loop.

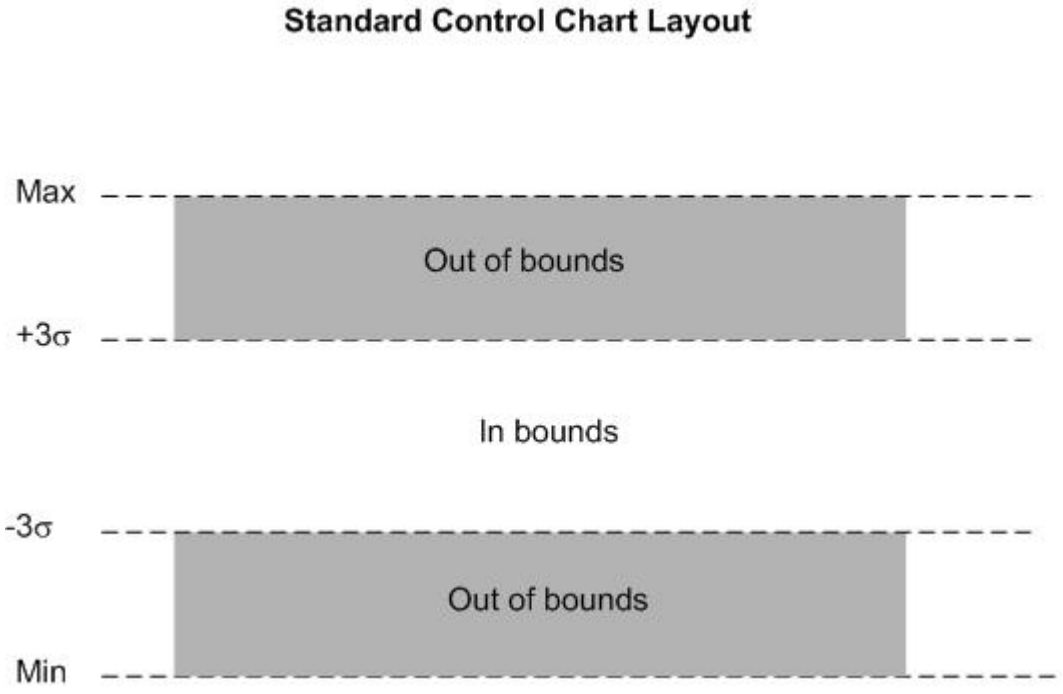


Figure 11 – Control Chart Out-Of-Bounds Area

Remember that it was mentioned earlier in this document that the CFA Loop and the Control Chart share the similar functions, the difference is in the use and objectives. We have already seen the Control and Abort similarities.

Let's look at a Control Chart (see Figure 12) and compare the information in the Control Chart with the CFA Loop elements.

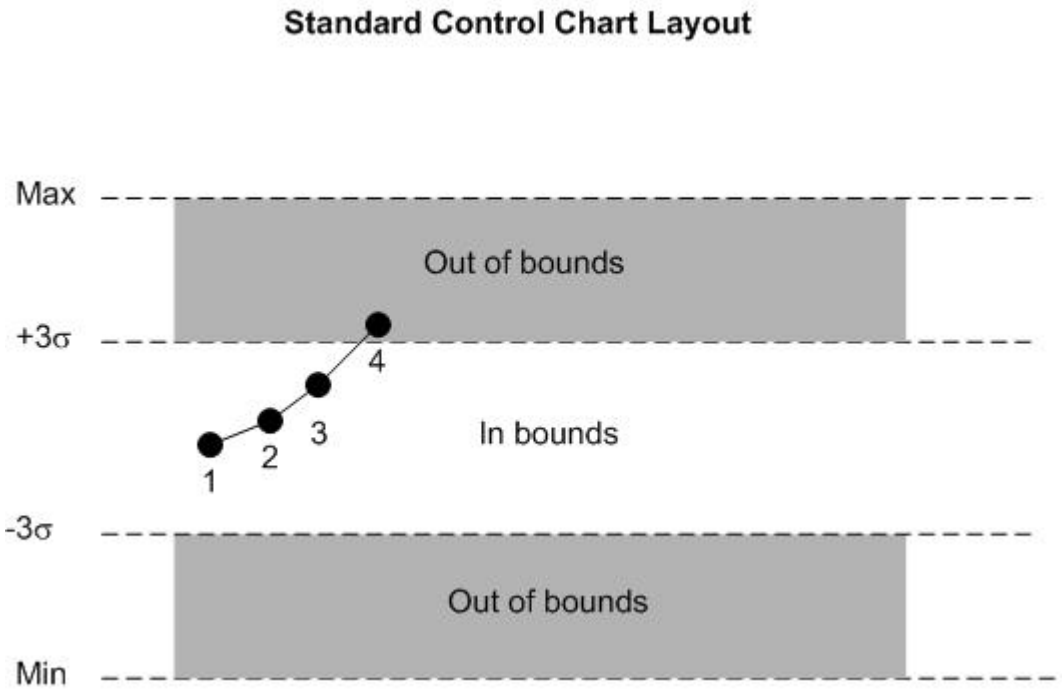


Figure 12 – Control Chart Showing Use

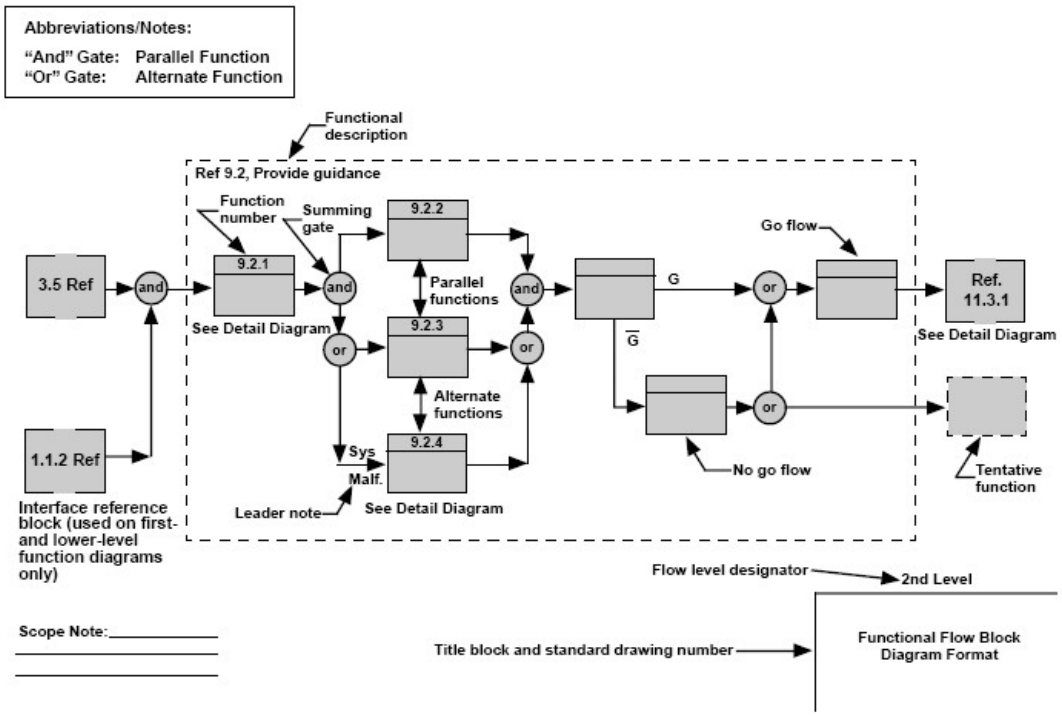
The 'in bounds' area is our Control element. As long as our data points, sometimes called items, are within the 'in bounds' area, we are said to be in control. The data points are the Feedback element. The 'out of bounds' areas are the Abort elements. Notice that data point 4 is in the 'out of bounds' area, which should lead to control being passed to the Abort element in order to take action to bring the future data points back into control. During the analysis of the system operation using the CFA Loop, the abort mechanism should have been clearly identified so that when the system goes out of bounds during operation, the alternate control should have been activated and the alternate action should be no surprise to the system designers.

The benefit of using control charts is due to its ability to report dynamic conditions of a system in operation. By data point 2, we should be able to see that if the data follows the trend set by the previous data points, the data will go out of control at some point. This ability to see the trend allows for the chart user to take early action to ensure that the

system stays in control or to monitor automated abort processes used to bring the system back into control.

Chapter 3

Functional Flow Block Diagram



Functional Flow Block Diagram Format.

A **Functional Flow Block Diagram** (FFBD) is a multi-tier, time-sequenced, step-by-step flow diagram of a system's functional flow.

The FFBD notation was developed in the 1950s, and is widely used in classical systems engineering. FFBDs are one of the classic business process modeling methodologies, along with flow charts, data flow diagrams, control flow diagrams, Gantt charts, PERT diagrams, and IDEF.

FFBDs are also referred to as *Functional Flow Diagrams*, *functional block diagrams*, and *functional flows*.

History

The first structured method for documenting process flow, the flow process chart, was introduced by Frank Gilbreth to members of American Society of Mechanical Engineers (ASME) in 1921 as the presentation “Process Charts—First Steps in Finding the One Best Way”. Gilbreth's tools quickly found their way into industrial engineering curricula. In the early 1930s, an industrial engineer, Allan H. Mogensen began training business people in the use of some of the tools of industrial engineering at his Work Simplification Conferences in Lake Placid, New York. A 1944 graduate of Mogensen's class, Art Spinanger, took the tools back to Procter and Gamble where he developed their Deliberate Methods Change Program. Another 1944 graduate, Ben S. Graham, Director of Formcraft Engineering at Standard Register Corporation, adapted the flow process chart to information processing with his development of the multi-flow process chart to displays multiple documents and their relationships. In 1947, ASME adopted a symbol set as the ASME Standard for Operation and Flow Process Charts, derived from Gilbreth's original work.

The modern FFBD was developed by TRW Incorporated, a defense-related business, in the 1950s. In the 1960s it was exploited by NASA to visualize the time sequence of events in space systems and flight missions. FFBDs became widely used in classical systems engineering to show the order of execution of system functions.

Development of functional flow block diagrams

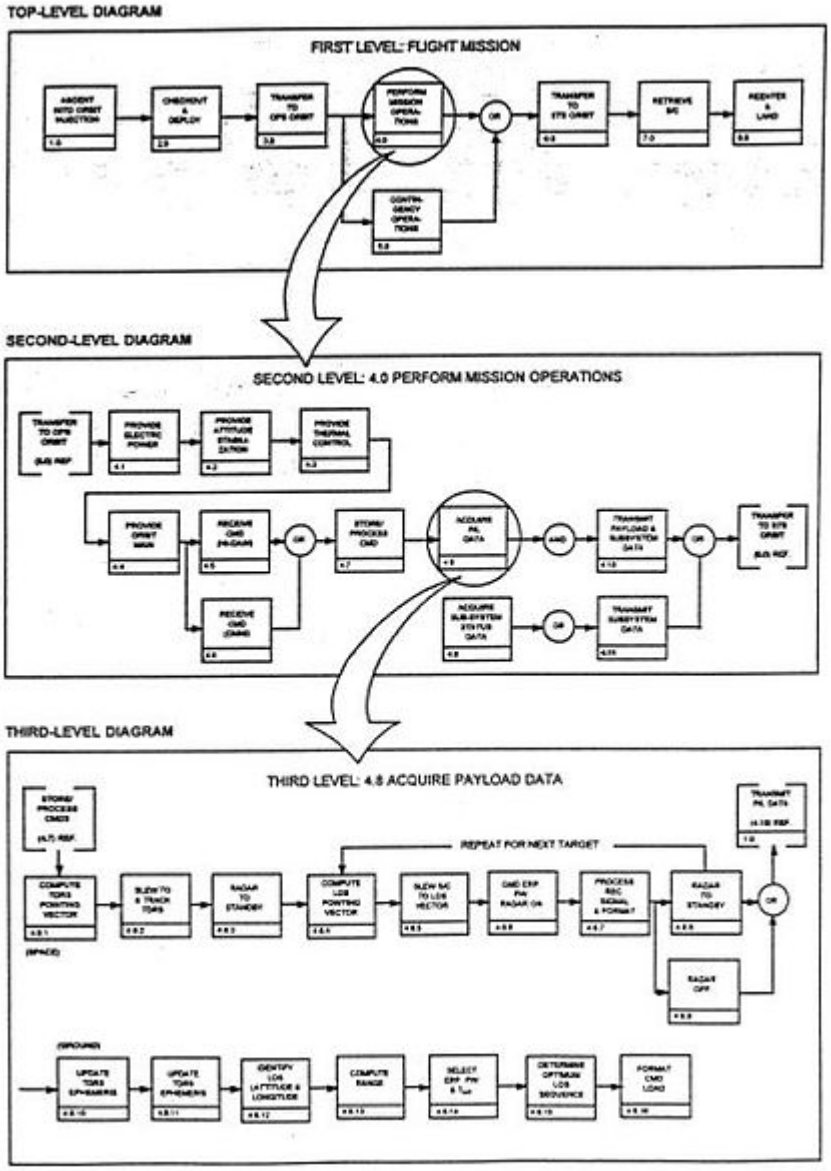


Figure 2: Development of Functional Flow Block Diagrams

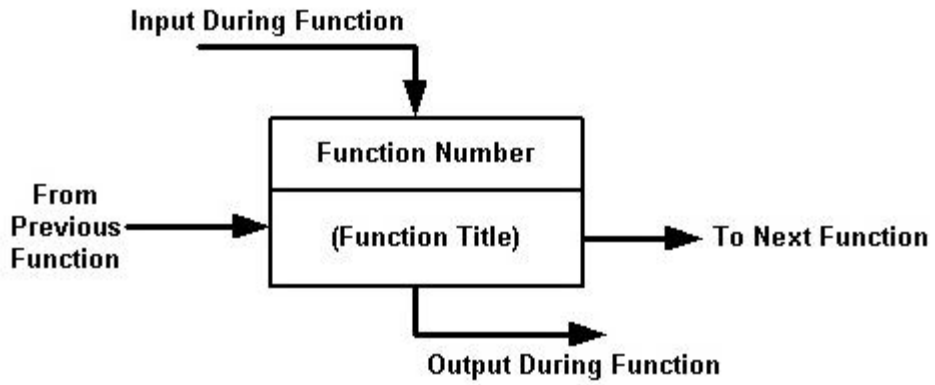
FFBDs can be developed in a series of levels. FFBDs show the same tasks identified through functional decomposition and display them in their logical, sequential relationship. For example, the entire flight mission of a spacecraft can be defined in a top level FFBD, as shown in Figure 2. Each block in the first level diagram can then be expanded to a series of functions, as shown in the second level diagram for "perform mission operations." Note that the diagram shows both input (transfer to operational orbit) and output (transfer to space transportation system orbit), thus initiating the interface identification and control process. Each block in the second level diagram can be progressively developed into a series of functions, as shown in the third level diagram on Figure 2.

These diagrams are used both to develop requirements and to identify profitable trade studies. For example, does the spacecraft antenna acquire the tracking and data relay satellite (TDRS) only when the payload data are to be transmitted, or does it track TDRS continually to allow for the reception of emergency commands or transmission of emergency data? The FFBD also incorporates alternate and contingency operations, which improve the probability of mission success. The flow diagram provides an understanding of total operation of the system, serves as a basis for development of operational and contingency procedures, and pinpoints areas where changes in operational procedures could simplify the overall system operation. In certain cases, alternate FFBDs may be used to represent various means of satisfying a particular function until data are acquired, which permits selection among the alternatives.

Building blocks

Key attributes

An overview of the key FFBD attributes:



Graphical explanation of a "function block" used in these diagrams. Flow is from left to right.

- *Function block*: Each function on an FFBD should be separate and be represented by single box (solid line). Each function needs to stand for definite, finite, discrete action to be accomplished by system elements.
- *Function numbering*: Each level should have a consistent number scheme and provide information concerning function origin. These numbers establish identification and relationships that will carry through all Functional Analysis and Allocation activities and facilitate traceability from lower to top levels.
- *Functional reference*: Each diagram should contain a reference to other functional diagrams by using a functional reference (box in brackets).
- *Flow connection*: Lines connecting functions should only indicate function flow and not a lapse in time or intermediate activity.

- *Flow direction*: Diagrams should be laid out so that the flow direction is generally from left to right. Arrows are often used to indicate functional flows.
- *Summing gates*: A circle is used to denote a summing gate and is used when AND/OR is present. AND is used to indicate parallel functions and all conditions must be satisfied to proceed. OR is used to indicate that alternative paths can be satisfied to proceed.
- *GO and NO-GO paths*: "G" and "bar G" are used to denote "go" and "no-go" conditions. These symbols are placed adjacent to lines leaving a particular function to indicate alternative paths.

Function symbolism

A function shall be represented by a rectangle containing the title of the function (an action verb followed by a noun phrase) and its unique decimal delimited number. A horizontal line shall separate this number and the title, as shown in see Figure 3 above. The figure also depicts how to represent a reference function, which provides context within a specific FFBD.

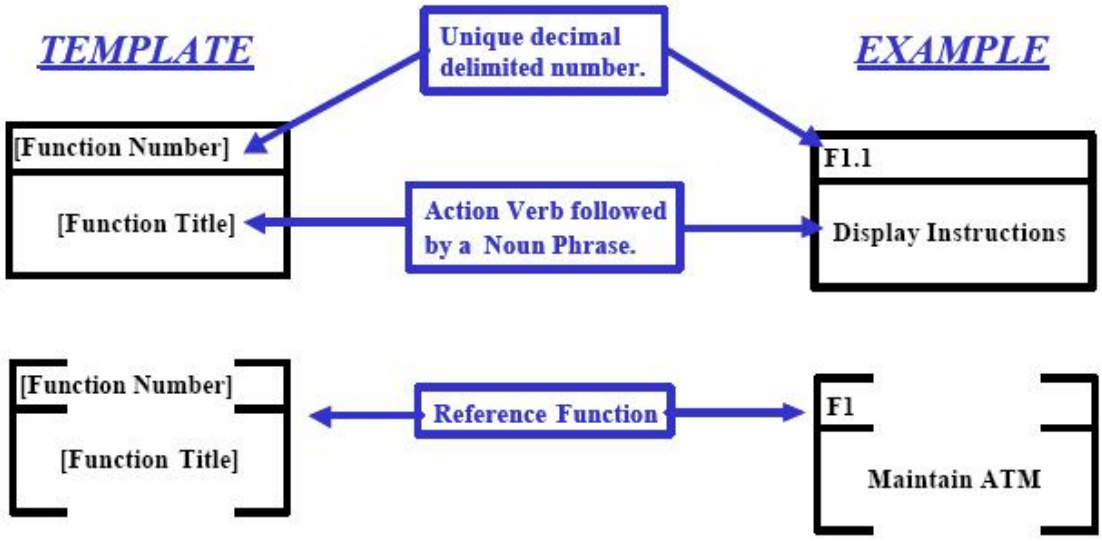


Figure 3. Function Symbol

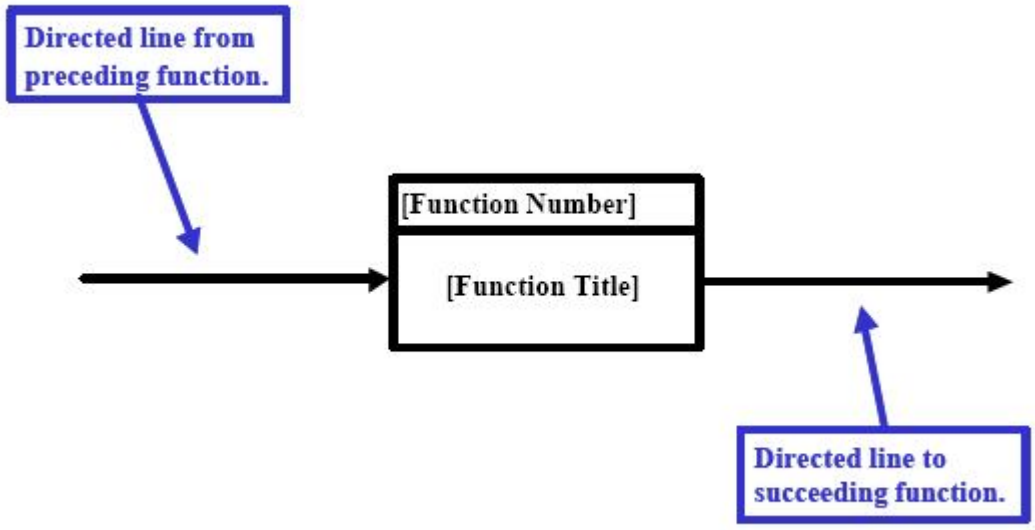


Figure 4. Directed Lines

Directed lines

A line with a single arrowhead shall depict functional flow from left to right, see Figure 4.

Logic Symbols

The following basic logic symbols shall be used.

- AND: A condition in which all preceding or succeeding paths are required. The symbol may contain a single input with multiple outputs or multiple inputs with a single output, but not multiple inputs and outputs combined (Figure 5). Read the figure as follows: F2 AND F3 may begin in parallel after completion of F1. Likewise, F4 may begin after completion of F2 AND F3.

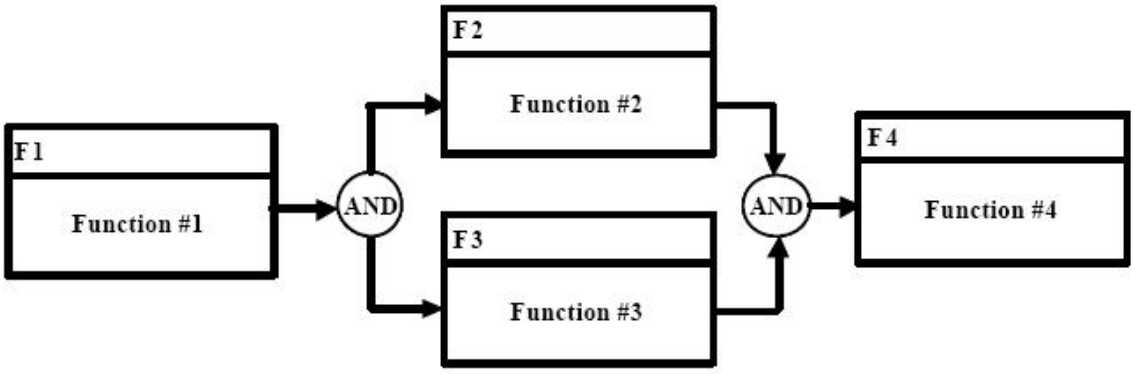


Figure 5. "AND" Symbol

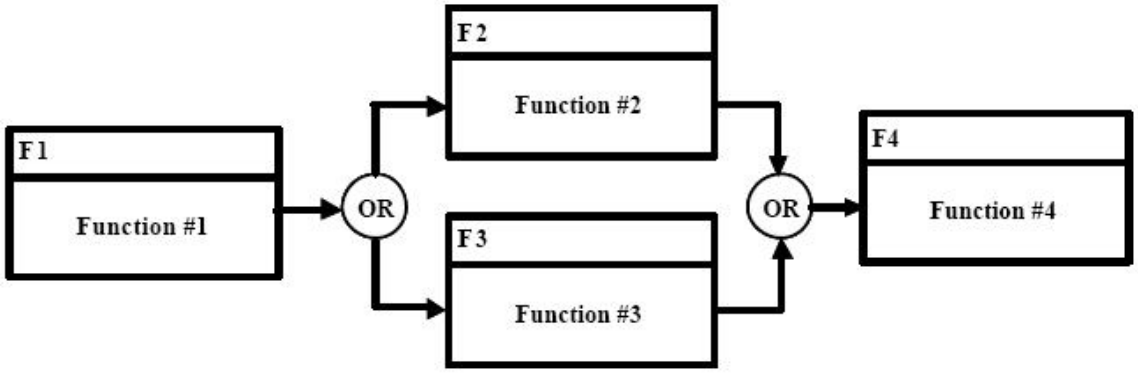


Figure 6. "Exclusive OR" Symbol

- Exclusive OR: A condition in which one of multiple preceding or succeeding paths is required, but not all. The symbol may contain a single input with multiple outputs or multiple inputs with single output, but not multiple inputs and outputs combined (Figure 6). Read the figure as follows: F2 OR F3 may begin after completion of F1. Likewise, F4 may begin after completion of either F2 OR F3.

- Inclusive OR: A condition in which one, some, or all of the multiple preceding or succeeding paths are required. Figure 7 depicts Inclusive OR logic using a combination of the AND symbol (Figure 5) and the Exclusive OR symbol (Figure 6). Read Figure 7 as follows: F2 OR F3 (exclusively) may begin after completion of F1, OR (again exclusive) F2 AND F3 may begin after completion of F1. Likewise, F4 may begin after completion of either F2 OR F3 (exclusively), OR (again exclusive) F4 may begin after completion of both F2 AND F3

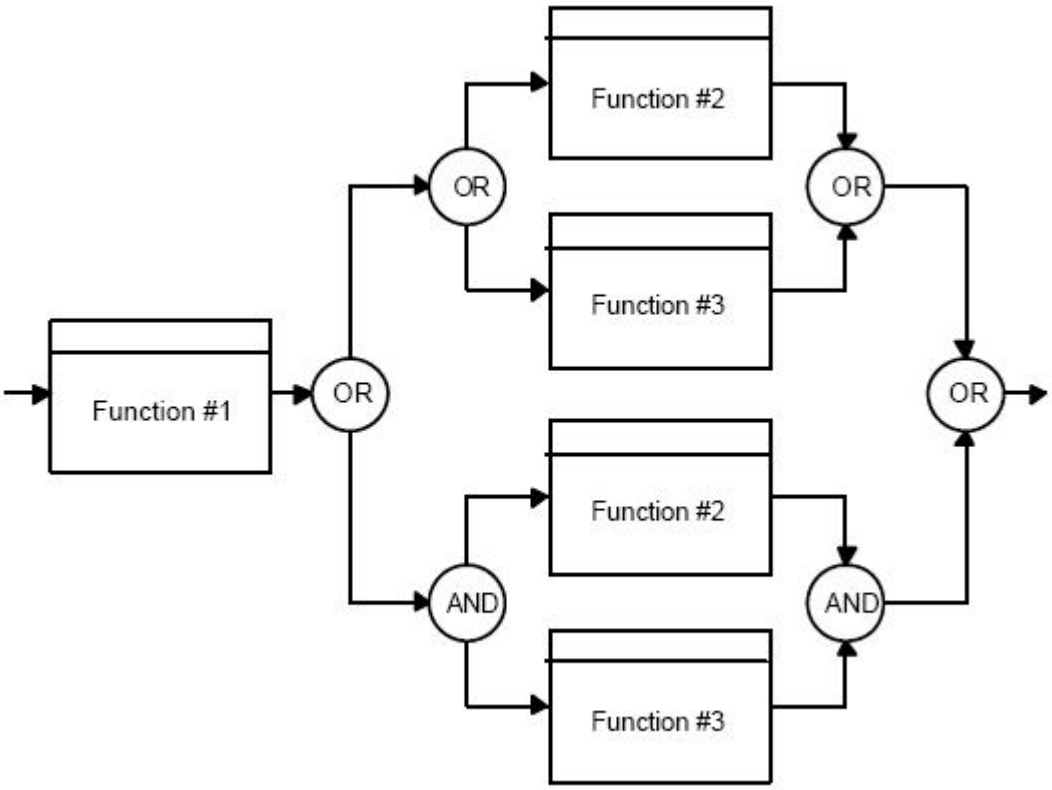


Figure 7. “Inclusive OR” Logic

Contextual and Administrative Data

Each FFBD shall contain the following contextual and administrative data:

- Date the diagram was created
- Name of the engineer, organization, or working group that created the diagram
- Unique decimal delimited number of the function being diagrammed
- Unique function name of the function being diagrammed.

Figure 8 and Figure 9 present the data in an FFBD. Figure 9 is a decomposition of the function F2 contained in Figure 8 and illustrates the context between functions at different levels of the model.

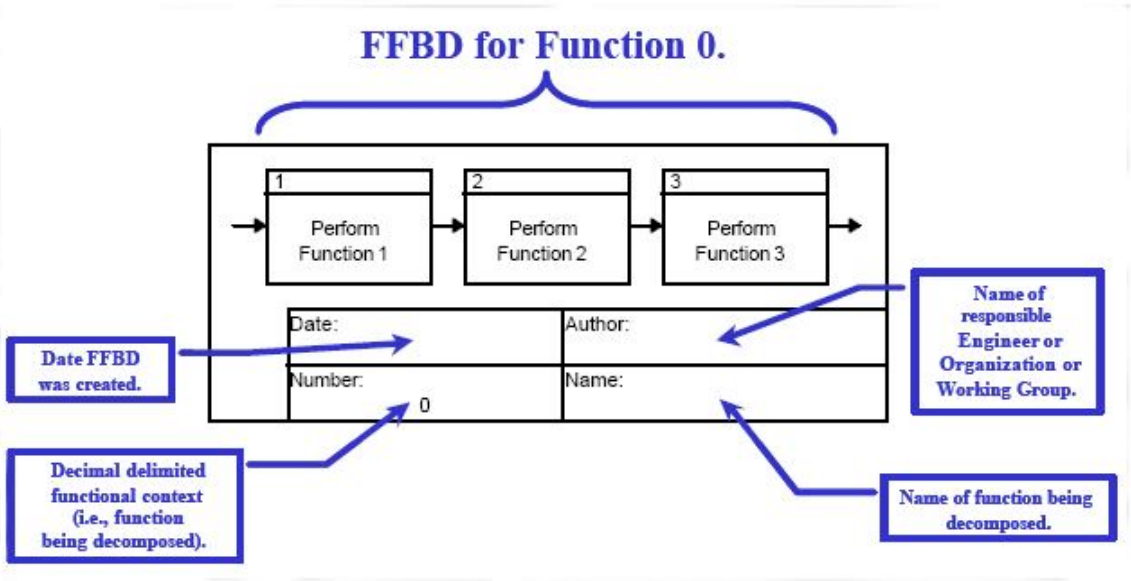


Figure 8. FFBD Function 0 Illustration

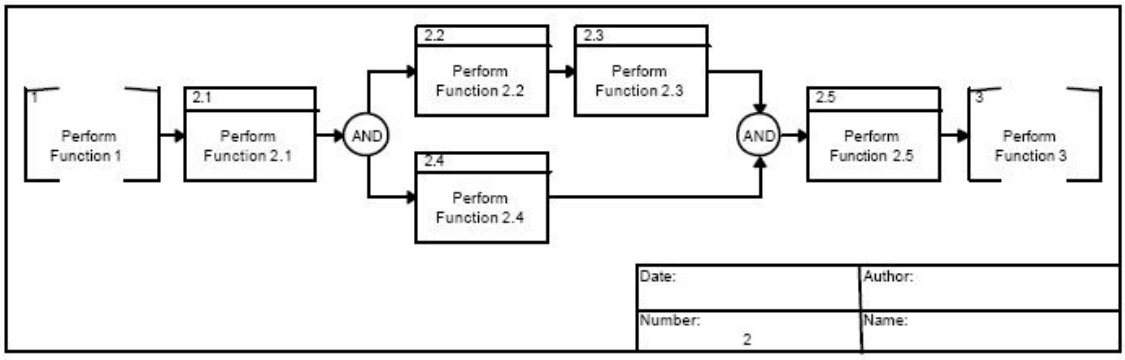
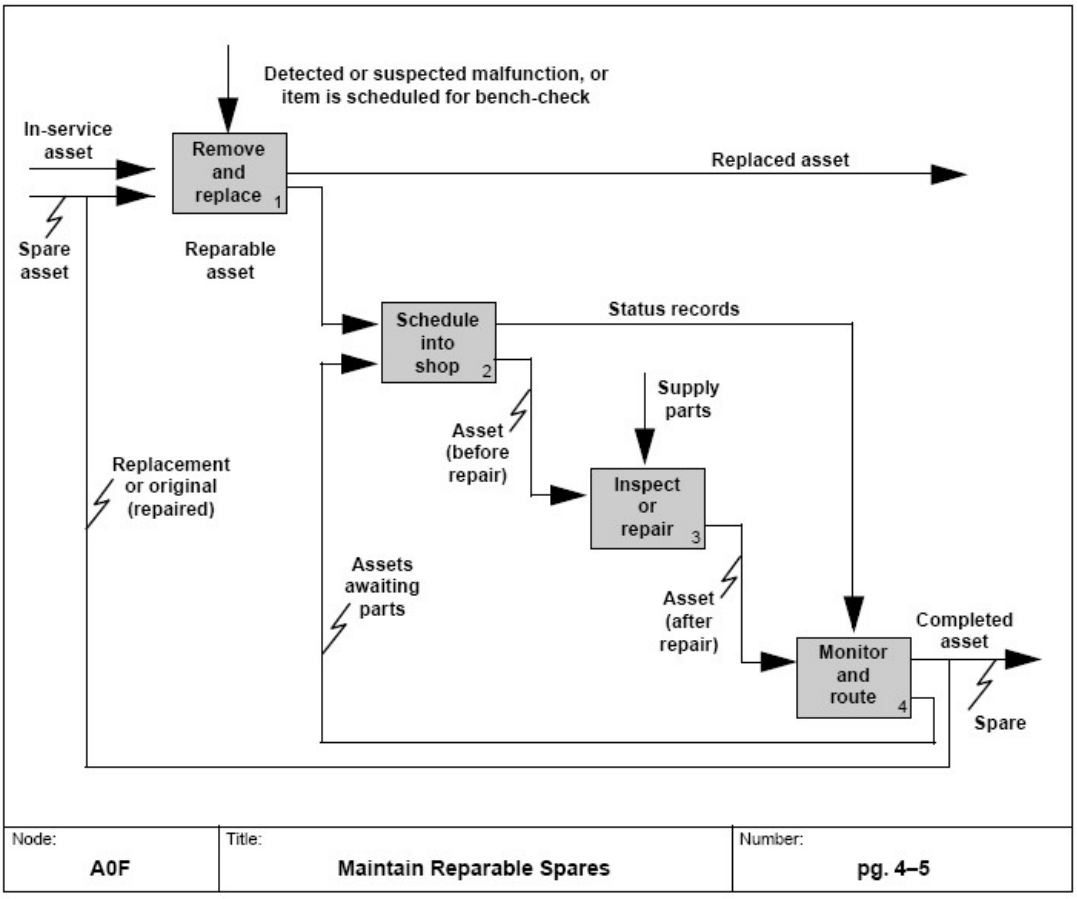


Figure 9. FFBD Function 2 Illustration

Chapter 4

IDEF0 & IDEF1X

IDEF0



IDEF0 Diagram Example

IDEF0 (*Integration Definition for Function Modeling*) is a function modeling methodology for describing manufacturing functions, which offers a functional modeling language for the analysis, development, reengineering, and integration of information systems; business processes; or software engineering analysis.

IDEF0 is part of the IDEF family of modeling languages in the field of software engineering, and is built on the functional modeling language Structured Analysis and Design Technique (SADT).

Overview

The IDEF0 Functional Modeling method is designed to model the decisions, actions, and activities of an organization or system. It was derived from the established graphic modeling language Structured Analysis and Design Technique (SADT) developed by Douglas T. Ross and SofTech, Inc.. In its original form, IDEF0 includes both a definition of a graphical modeling language (syntax and semantics) and a description of a comprehensive methodology for developing models. The US Air Force commissioned the SADT developers "to develop a function model method for analyzing and communicating the functional perspective of a system. IDEF0 should assist in organizing system analysis and promote effective communication between the analyst and the customer through simplified graphical devices".

Where the Functional flow block diagram is used to show the functional flow of a product, IDEF0 is used to show data flow, system control, and the functional flow of life cycle processes. IDEF0 is capable of graphically representing a wide variety of business, manufacturing and other types of enterprise operations to any level of detail. It provides rigorous and precise description, and promotes consistency of usage and interpretation. It is well-tested and proven through many years of use by government and private industry. It can be generated by a variety of computer graphics tools. Numerous commercial products specifically support development and analysis of IDEF0 diagrams and models.

An associated technique, Integration Definition for Information Modeling (IDEF1x), is used to supplement IDEF0 for data intensive systems. The IDEF0 standard, Federal Information Processing Standards Publication 183 (FIPS 183), and the IDEF1x standard (FIPS 184) are maintained by the National Institute of Standards and Technology (NIST).

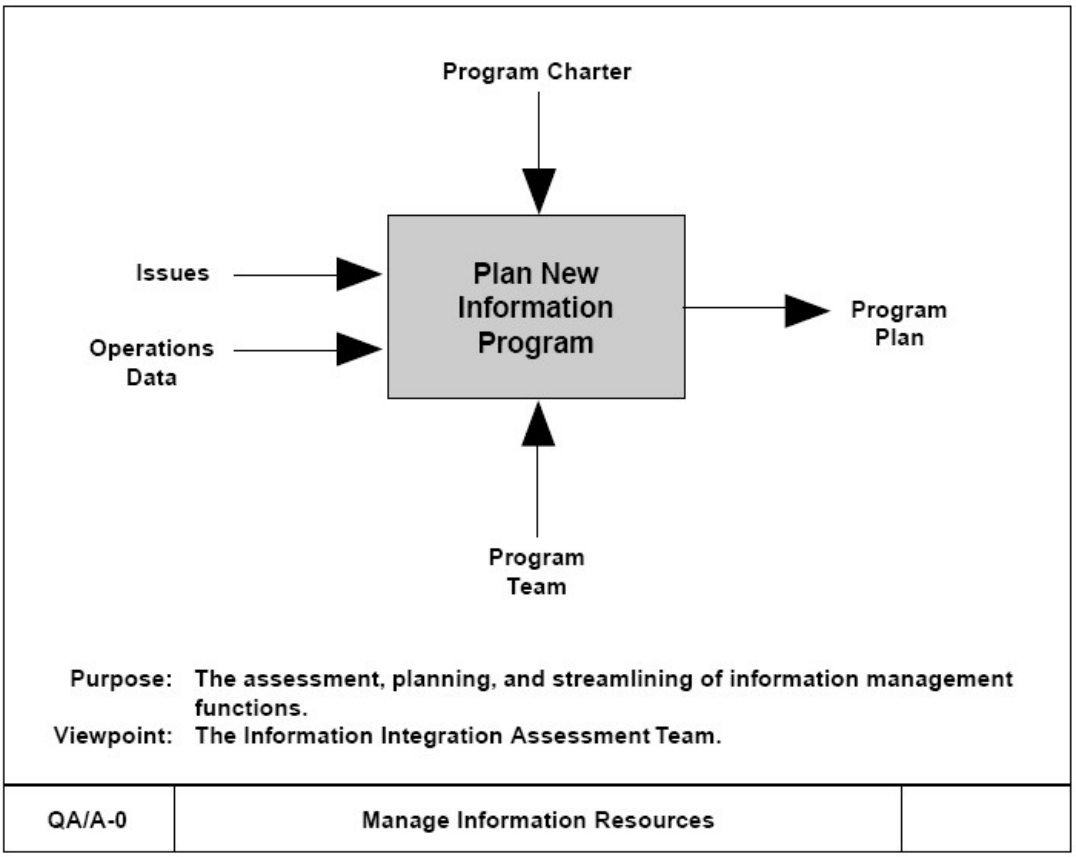
History

During the 1970s, the U.S. Air Force Program for Integrated Computer Aided Manufacturing (ICAM) sought to increase manufacturing productivity through systematic application of computer technology. The ICAM program identified the need for better analysis and communication techniques for people involved in improving manufacturing productivity. As a result, in 1981 the ICAM program developed a series of techniques known as the IDEF (ICAM Definition) techniques which included the following:

- IDEF0, used to produce a "function model". A function model is a structured representation of the functions, activities or processes within the modeled system or subject area.
- IDEF1, used to produce an "information model". An information model represents the structure and semantics of information within the modeled system or subject area.
- IDEF2, used to produce a "dynamics model". A dynamics model represents the time-varying behavioral characteristics of the modeled system or subject area.

In 1983, the U.S. Air Force Integrated Information Support System program enhanced the IDEF1 information modeling technique to form IDEF1X (IDEF1 Extended), a semantic data modeling technique. By the 1990s, IDEF0 and IDEF1X techniques are widely used in the government, industrial and commercial sectors, supporting modeling efforts for a wide range of enterprises and application domains. In 1991 the National Institute of Standards and Technology (NIST) received support from the U.S. Department of Defense, Office of Corporate Information Management (DoD/CIM), to develop one or more Federal Information Processing Standard (FIPS) for modeling techniques. The techniques selected were IDEF0 for function modeling and IDEF1X for information modeling. These FIPS documents are based on the IDEF manuals published by the U.S. Air Force in the early 1980s.

IDEF0 Topics

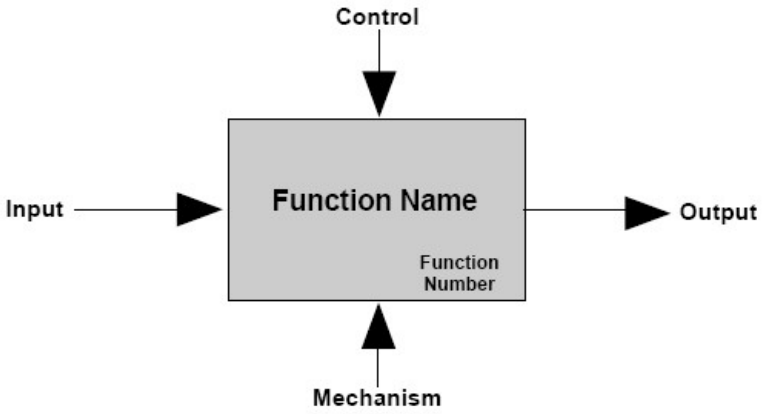


Top-Level Context Diagram

The IDEF0 Approach

IDEF0 may be used to model a wide variety of automated and non-automated systems. For new systems, it may be used first to define the requirements and specify the functions, and then to design an implementation that meets the requirements and performs the functions. For existing systems, IDEF0 can be used to analyze the functions the system performs and to record the mechanisms (means) by which these are done. The result of applying IDEF0 to a system is a model that consists of a hierarchical series of diagrams, text, and glossary cross-referenced to each other. The two primary modeling components are functions (represented on a diagram by boxes) and the data and objects that inter-relate those functions (represented by arrows).

IDEF0 Building blocks

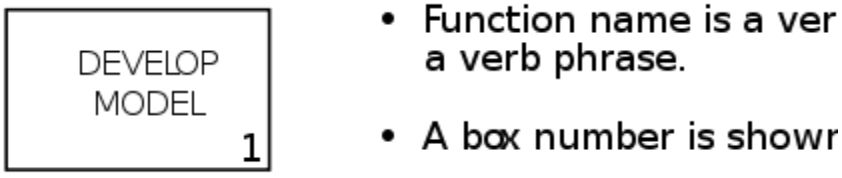


Integration Definition for Function Modeling (IDEF0) Box Format

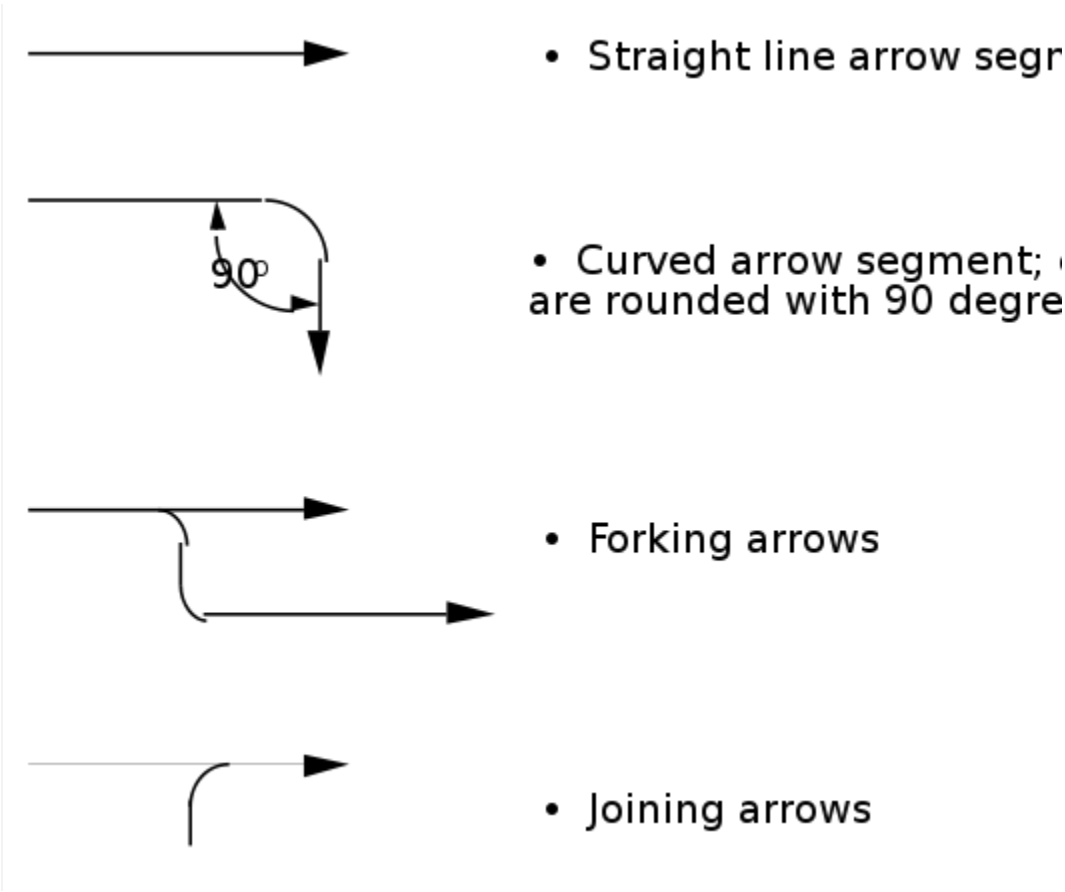
The IDEF0 model displayed here on the left is based on a simple syntax. Each activity is described by a verb based label placed in a box. Inputs are shown as arrows entering the left side of the activity box while output are shown as exiting arrows on the right side of the box. Controls are displayed as arrows entering the top of the box and mechanisms are displayed as arrows entering from the bottom of the box. Inputs, Controls, Outputs, and Mechanisms are all referred to as concepts.

- *Arrow* : A directed line, composed of one or more arrow segments, that models an open channel or conduit conveying data or objects from source (no arrowhead) to use (with arrowhead). There are 4 arrow classes: Input Arrow, Output Arrow, Control Arrow, and Mechanism Arrow (includes Call Arrow).

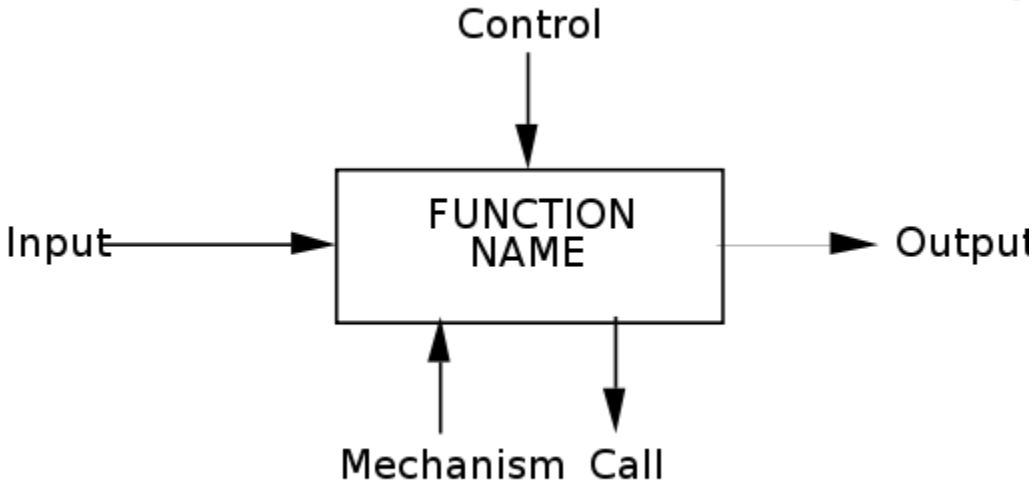
- *Box* : A rectangle, containing a name and number, used to represent a function.



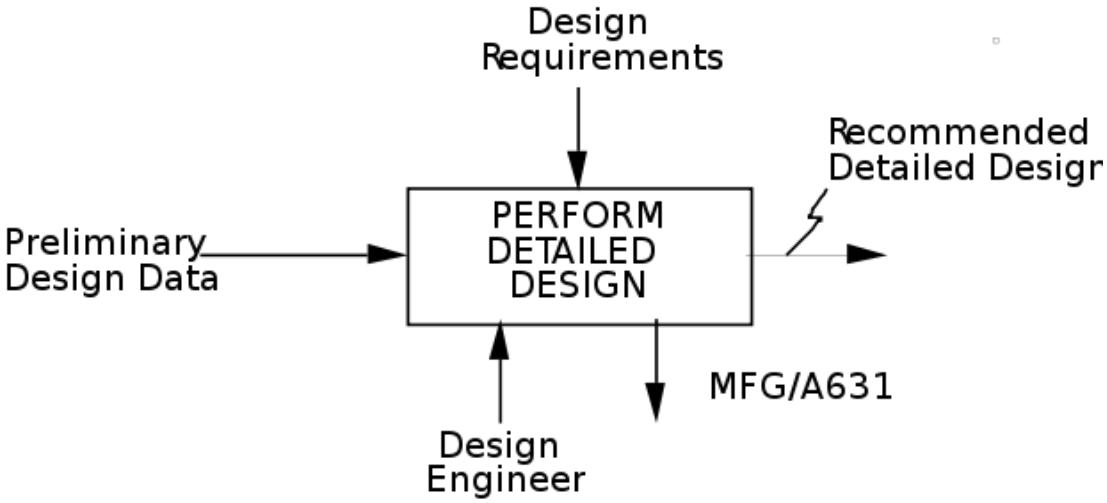
Box Syntax



Arrow Syntax

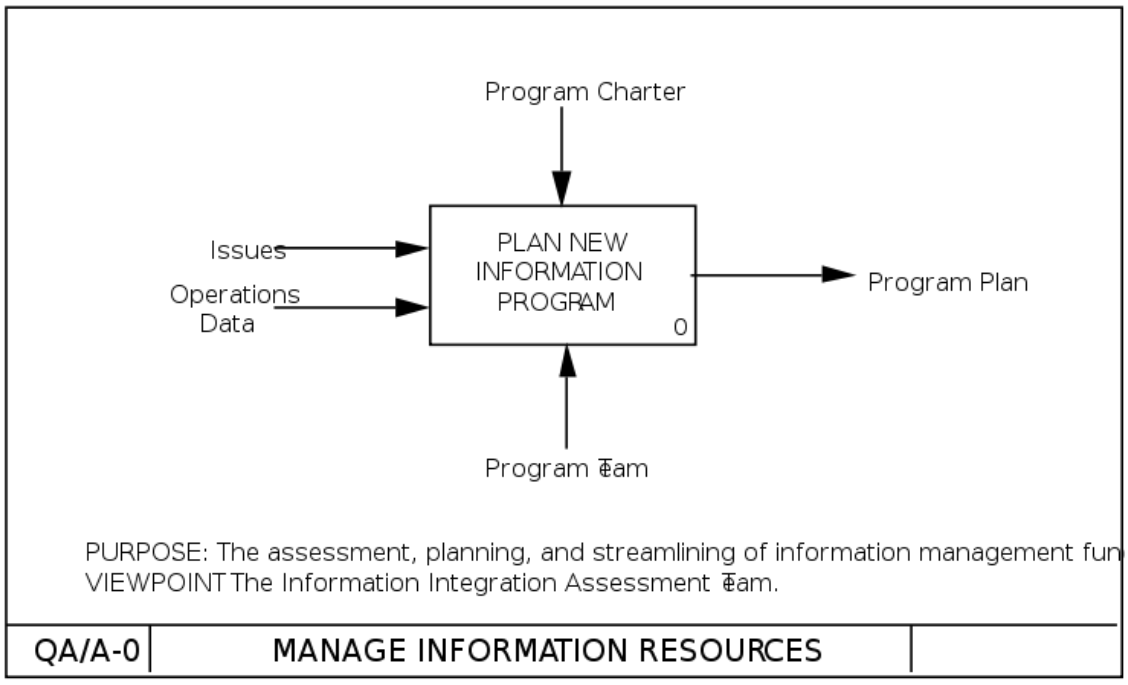


Arrow Positions and Roles

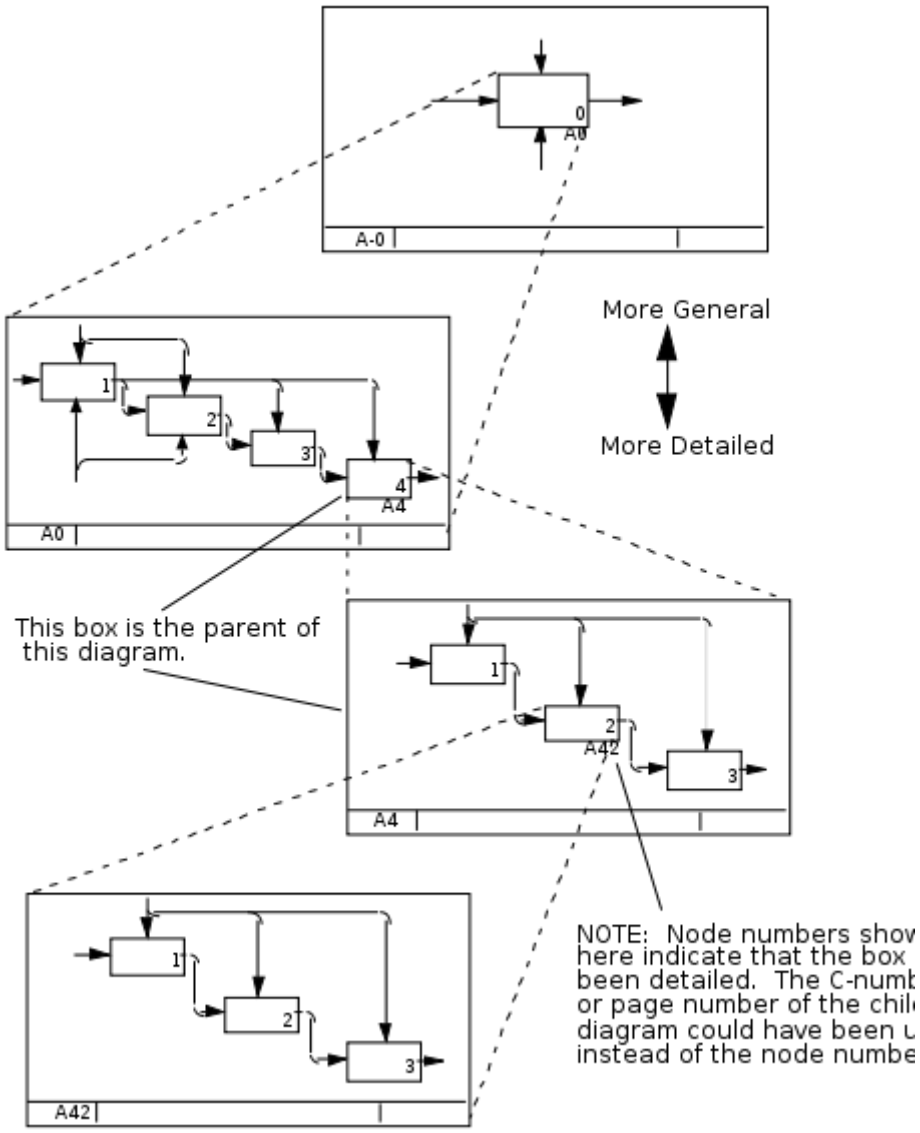


Label and Name Semantics

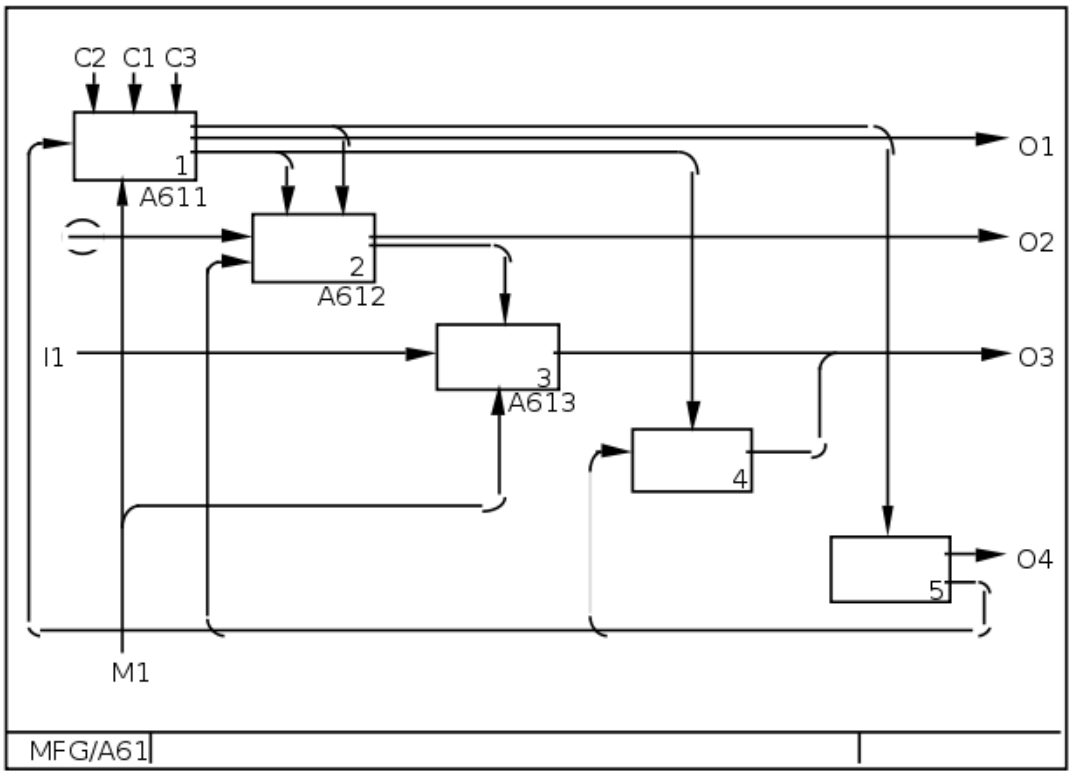
- *Context* : The immediate environment in which a function (or set of functions on a diagram) operates.
- *Decomposition* : The partitioning of a modeled function into its component functions.



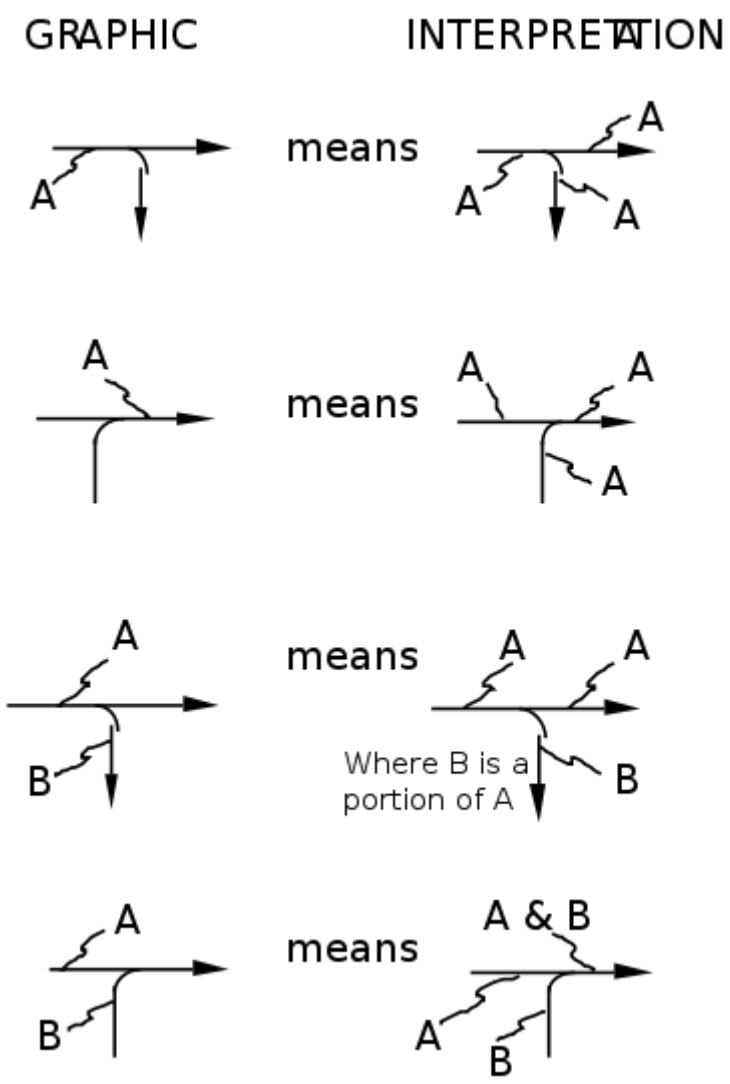
Example Top-level Diagram



Decomposition Structure

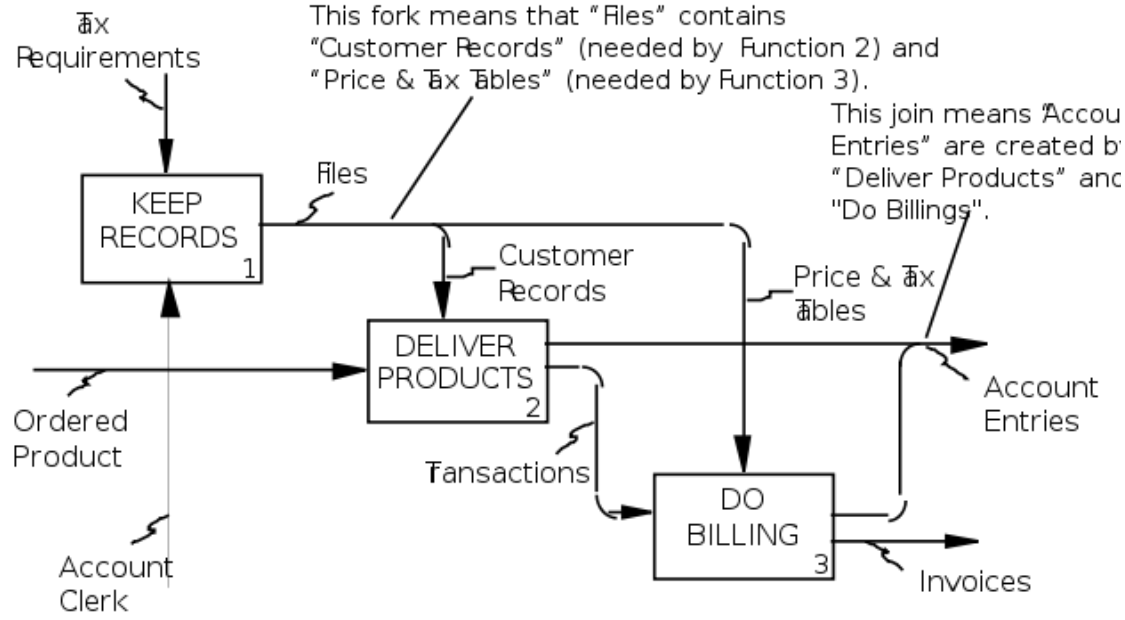


Detail Reference Expression Use

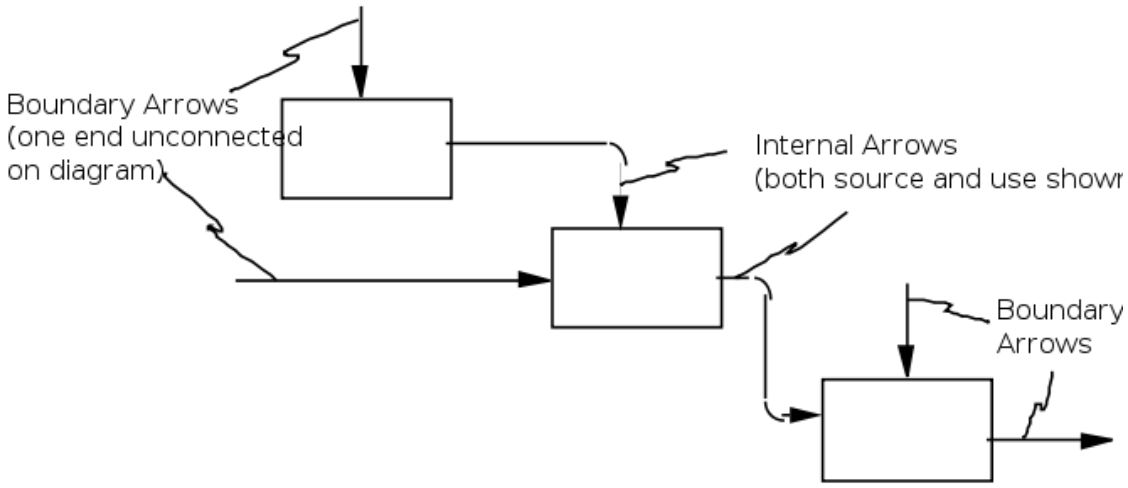


Arrow Fork and Join Structures

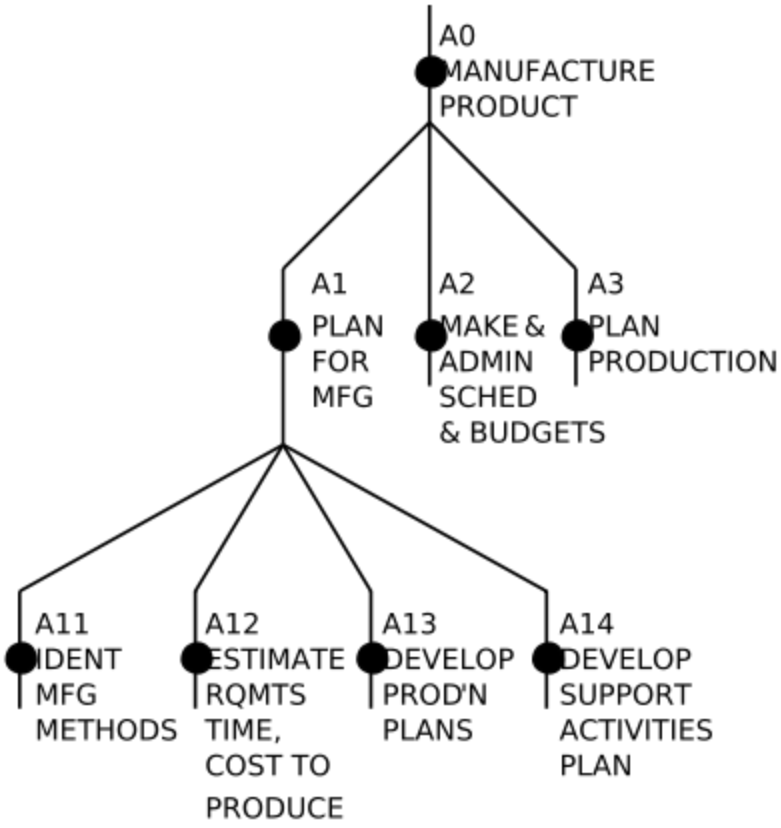
- *Fork* : The junction at which an IDEF0 arrow segment (going from source to use) divides into two or more arrow segments. May denote unbundling of meaning.



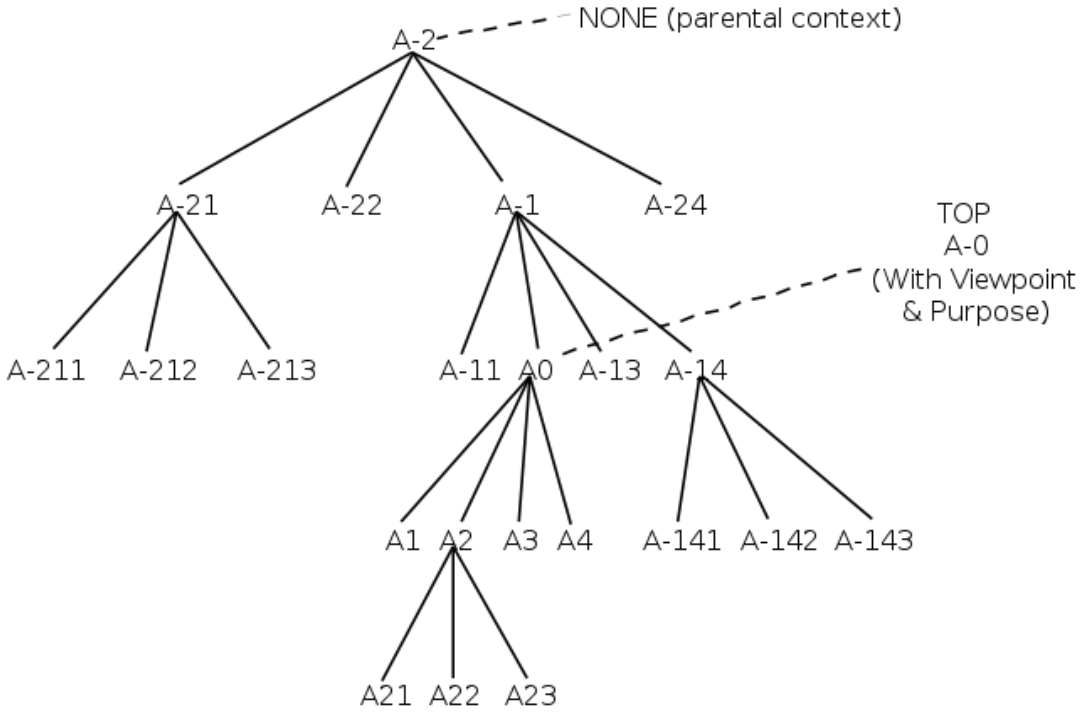
Connections Between Boxes



Boundary and Internal Arrows



Typical Node Tree

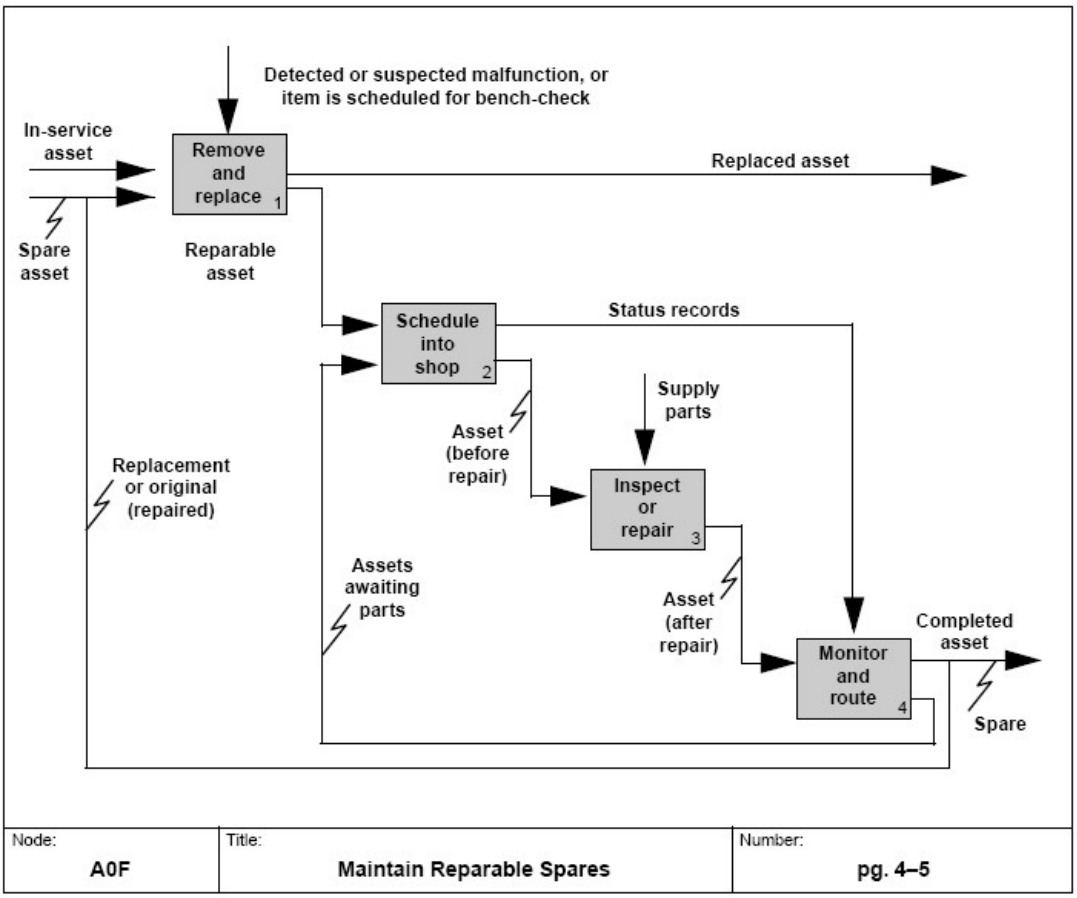


Negative Node-Numbered Context

- *Function* : An activity, process, or transformation (modeled by an IDEF0 box) identified by a verb or verb phrase that describes what must be accomplished.

- *Join* : The junction at which an IDEF0 arrow segment (going from source to use) merges with one or more other arrow segments to form a single arrow segment. May denote bundling of arrow segment meanings

- *Node* : A box from which child boxes originate; a parent box.



IDEF0 Diagram Example

Diagrammic notation

IDEF0 is a model that consists of a hierarchical series of diagrams, text, and glossary cross referenced to each other. The two primary modeling components are:

- functions (represented on a diagram by boxes), and
- data and objects that interrelate those functions (represented by arrows).

As shown by Figure 3 the position at which the arrow attaches to a box conveys the specific role of the interface. The controls enter the top of the box. The inputs, the data or objects acted upon by the operation, enter the box from the left. The outputs of the operation leave the right-hand side of the box. Mechanism arrows that provide supporting means for performing the function join (point up to) the bottom of the box.

The IDEF0 process

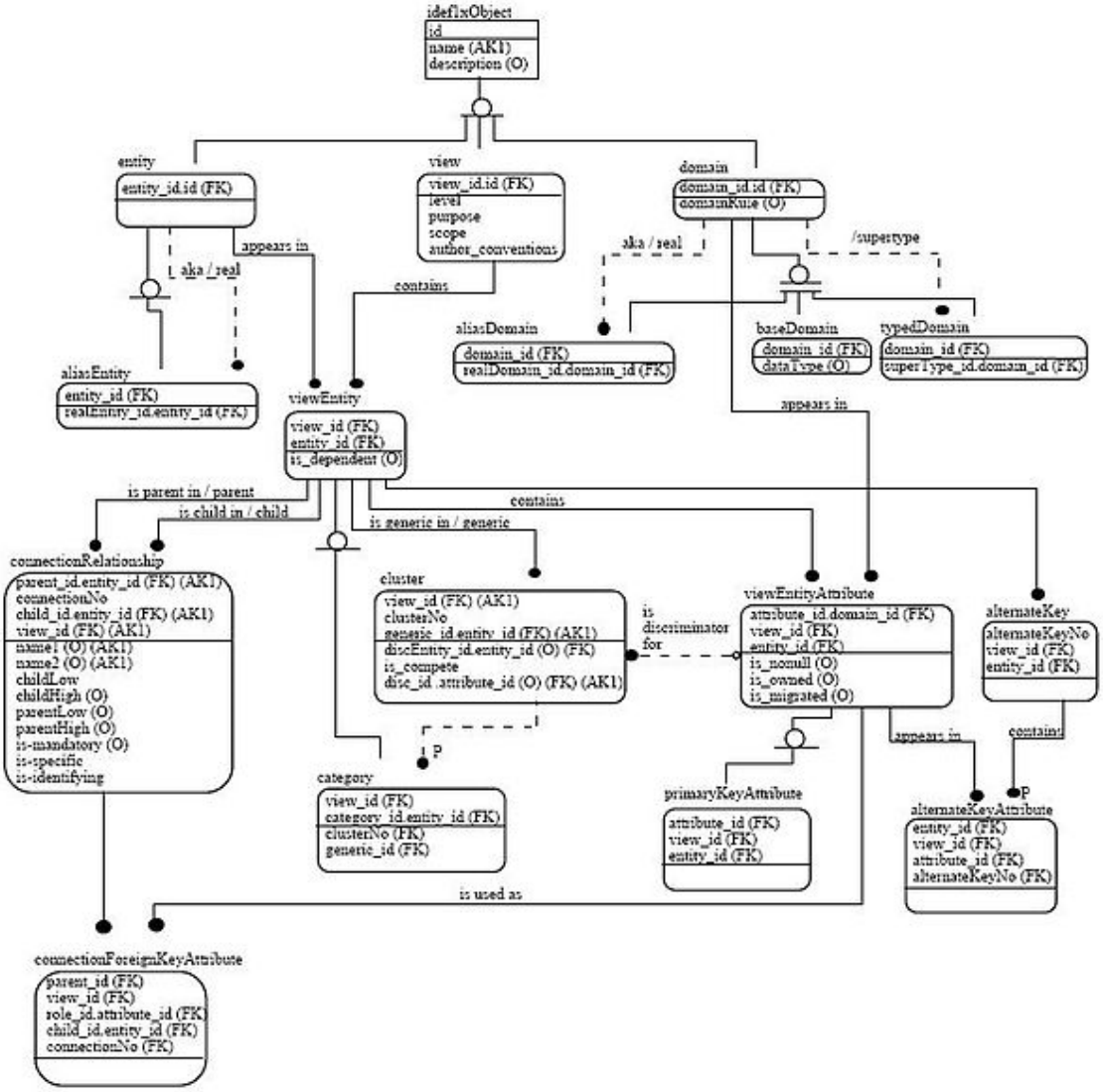
The IDEF0 process starts with the identification of the prime function to be decomposed. This function is identified on a “Top Level Context Diagram,” that defines the scope of

the particular IDEF0 analysis. An example of a Top Level Context Diagram for an information system management process is shown in Figure 3. From this diagram lower-level diagrams are generated. An example of a derived diagram, called a “child” in IDEF0 terminology, for a life cycle function is shown in Figure 4.

Federal Information Processing Standards

In Dec 1993 the National Institute of Standards and Technology announcing the standard for Integration Definition for Function Modeling (IDEF0) in the category Software Standard, Modeling Techniques. This publication announces the adoption of the IDEF0 as a Federal Information Processing Standard (FIPS). This standard was based on the Air Force Wright Aeronautical Laboratories Integrated Computer-Aided Manufacturing (ICAM) Architecture from June 1981.

IDEF1X



Example of an IDEF1X Diagram.

IDEF1X (**Integration Definition for Information Modeling**) is a data modeling language for the developing of semantic data models. IDEF1X is used to produce a graphical information model which represents the structure and semantics of information within an environment or system.

Use of the IDEF1X permits the construction of semantic data models which may serve to support the management of data as a resource, the integration of information systems, and

the building of computer databases. This standard is part of the IDEF family of modeling languages in the field of software engineering.

Overview

A data modeling technique is used to model data in a standard, consistent, predictable manner in order to manage it as a resource. It can be used in projects requiring a standard means of defining and analyzing the data resources within an organization. Such projects include the incorporation of a data modeling technique into a methodology, managing data as a resource, integrating information systems, or designing computer databases. The primary objectives of the IDEF1X standard are to provide:

- Means for completely understanding and analyzing an organization's data resources;
- Common means of representing and communicating the complexity of data;
- A technique for presenting an overall view of the data required to run an enterprise;
- Means for defining an application-independent view of data which can be validated by users and transformed into a physical database design; and
- A technique for deriving an integrated data definition from existing data resources.

A principal objective of IDEF1X is to support integration. The approach to integration focuses on the capture, management, and use of a single semantic definition of the data resource referred to as a “Conceptual schema”. The “conceptual schema” provides a single integrated definition of the data within an enterprise which is unbiased toward any single application of data and is independent of how the data is physically stored or accessed. The primary objective of this conceptual schema is to provide a consistent definition of the meanings and interrelationship of data which can be used to integrate, share, and manage the integrity of data. A conceptual schema must have three important characteristics. It must be:

- Consistent with the infrastructure of the business and be true across all application areas.
- Extendible, such that, new data can be defined without altering previously defined data.
- Transformable to both the required user views and to a variety of data storage and access structures.

History

The need for semantic data models was first recognized by the U.S. Air Force in the mid-1970s as a result of the Integrated Computer Aided Manufacturing (ICAM) Program. The objective of this program was to increase manufacturing productivity through the systematic application of computer technology. The ICAM Program identified a need for better analysis and communication techniques for people involved in improving

manufacturing productivity. As a result, the ICAM Program developed a series of techniques known as the IDEF (ICAM Definition) Methods which included the following:

- IDEF0 used to produce a "function model" which is a structured representation of the activities or processes within the environment or system.
- IDEF1 used to produce an "information model" which represents the structure and semantics of information within the environment or system.
- IDEF2 used to produce a "dynamics model"

The initial approach to IDEF information modeling (IDEF1) was published by the ICAM program in 1981, based on current research and industry needs. The theoretical roots for this approach stemmed from the early work of Edgar F. Codd on relational theory and Peter Chen on the entity-relationship model. The initial IDEF1 technique was based on the work of Dr. R.R. Brown and Mr. T.L. Ramey of Hughes Aircraft and Mr. D.S. Coleman of D. Appleton Company (DACOM), with critical review and influence by Charles Bachman, Peter Chen, Dr. M.A. Melkanoff, and Dr. G.M. Nijssen.

In 1983, the U.S. Air Force initiated the Integrated Information Support System (I2S2) project under the ICAM program. The objective of this project was to provide the enabling technology to logically and physically integrate a network of heterogeneous computer hardware and software. As a result of this project, and industry experience, the need for an enhanced technique for information modeling was recognized.

From the point of view of the contract administrators of the Air Force IDEF program, IDEF1X was a result of the ICAM IISS-6201 project and was further extended by the IISS-6202 project. To satisfy the data modeling enhancement requirements that were identified in the IISS-6202 project, a sub-contractor, DACOM, obtained a license to the Logical Database Design Technique (LDDT) and its supporting software (ADAM). From the point of view of the technical content of the modeling technique, IDEF1X is a renaming of LDDT.

Logical Database Design Technique

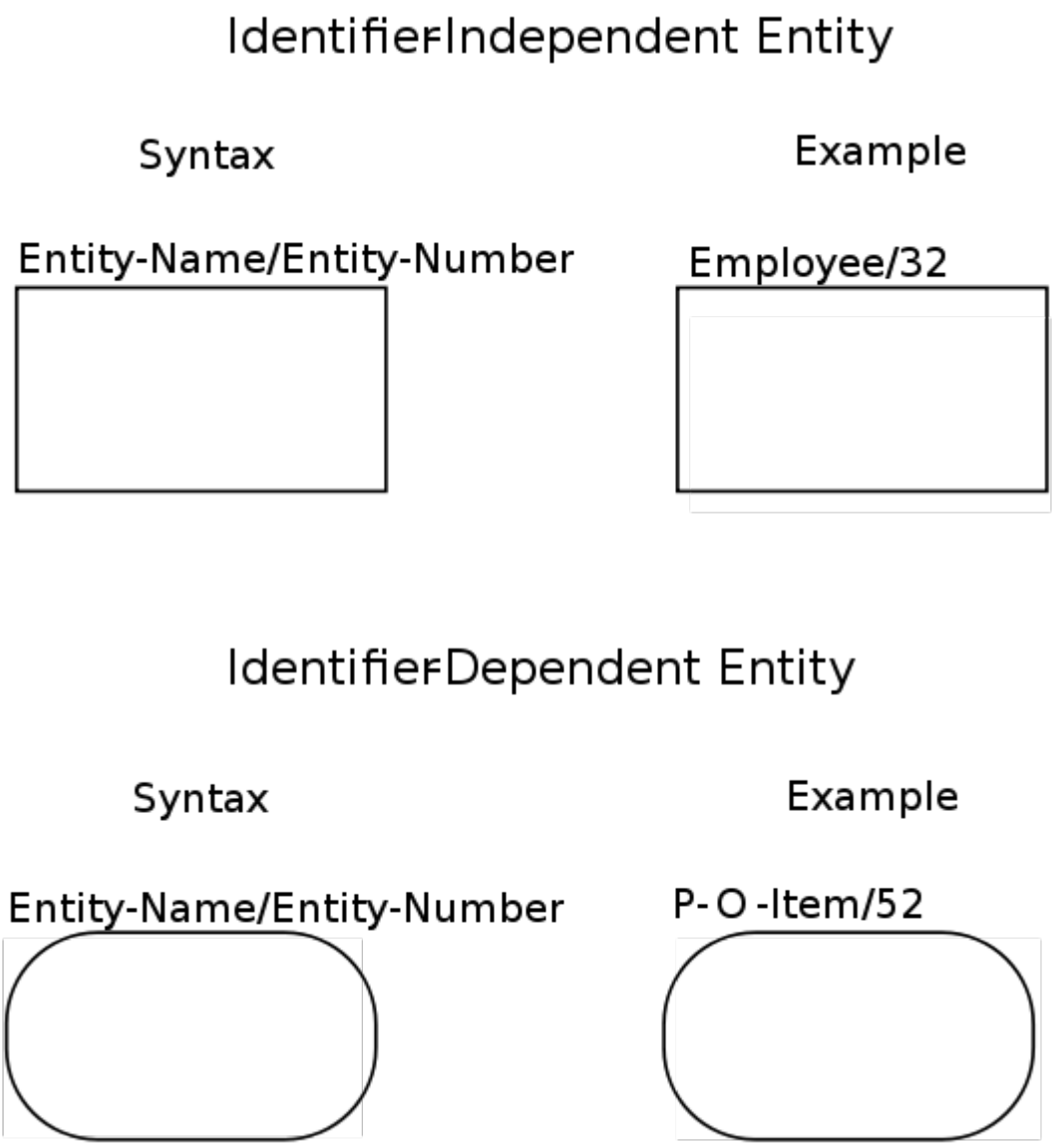
LDDT had been developed in 1982 by Robert G. Brown of The Database Design Group entirely outside the IDEF program and with no knowledge of IDEF1. Nevertheless, the central goal of IDEF1 and LDDT was the same: to produce a database neutral model of the persistent information needed by an enterprise by modeling the real-world entities involved. LDDT combined elements of the relational data model, the E-R model, and data generalization in a way specifically intended to support data modeling and the transformation of the data models into database designs.

LDDT included multiple levels of model, the modeling of generalization/specialization, and the explicit representation of relationships by primary and foreign keys, supported by a well defined role naming facility. The primary keys and unambiguously role-named foreign keys expressed sometimes subtle uniqueness and referential integrity constraints

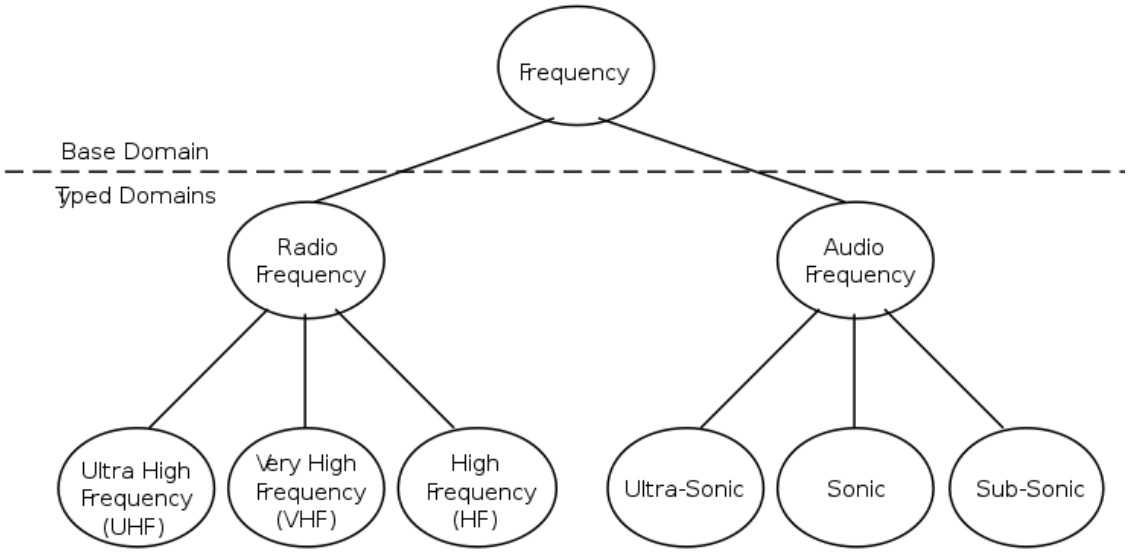
that needed to be known and honored by whatever type of database was ultimately designed. Whether the database design used the integrity constraint based keys of the LDDT model as database access keys or indexes was an entirely separate decision. The precision and completeness of the LDDT models was an important factor in enabling the relatively smooth transformation of the models into database designs. Early LDDT models were transformed into database designs for IBM's hierarchical database, IMS. Later models were transformed into database designs for Cullinet's network database, IDMS, and many varieties of relational database.

The graphic syntax of LDDT differed from that of IDEF1 and, more importantly, LDDT contained interrelated modeling concepts not present in IDEF1. Therefore, instead of extending IDEF1, Mary E. Loomis of DACOM wrote a concise summary of the syntax and semantics of a substantial subset of LDDT, using terminology compatible with IDEF1 wherever possible. DACOM labeled the result IDEF1X and supplied it to the ICAM program, which published it in 1985. (IEEE 1998, p. iii) (Bruce 1992, p. xii)

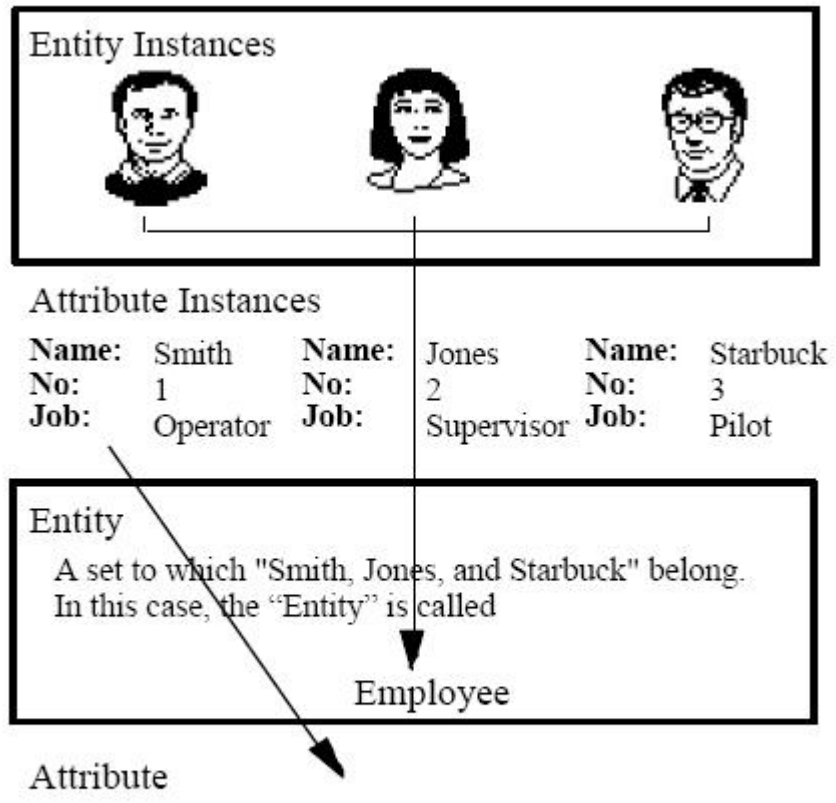
IDEF1X Building blocks



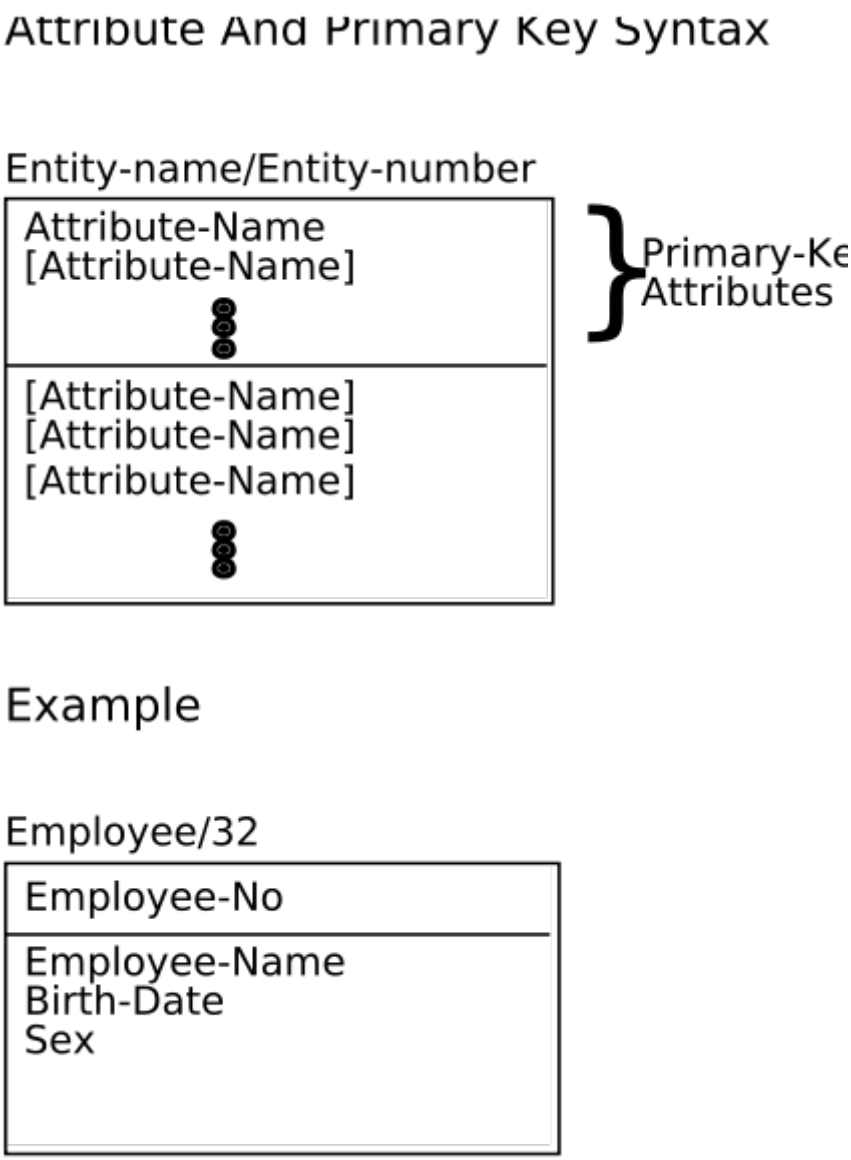
Entity Syntax



Domain Hierarchy



Attribute example



Primary Key Syntax

Entities

The representation of a set of real or abstract things (people, objects, places, events, ideas, combination of things, etc.) that are recognized as the same type because they share the same characteristics and can participate in the same relationships.

Domains

A named set of data values (fixed, or possibly infinite in number) all of the same data type, upon which the actual value for an attribute instance is drawn. Every attribute must be defined on exactly one underlying domain. Multiple attributes may be based on the same underlying domain.

Attributes

A property or characteristic that is common to some or all of the instances of an entity. An attribute represents the use of a domain in the context of an entity.

Keys

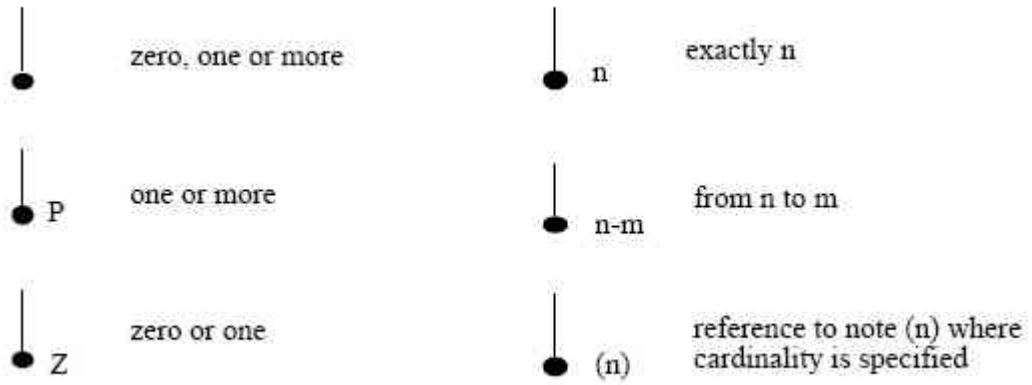
An attribute, or combination of attributes, of an entity whose values uniquely identify each entity instance.

Primary Keys

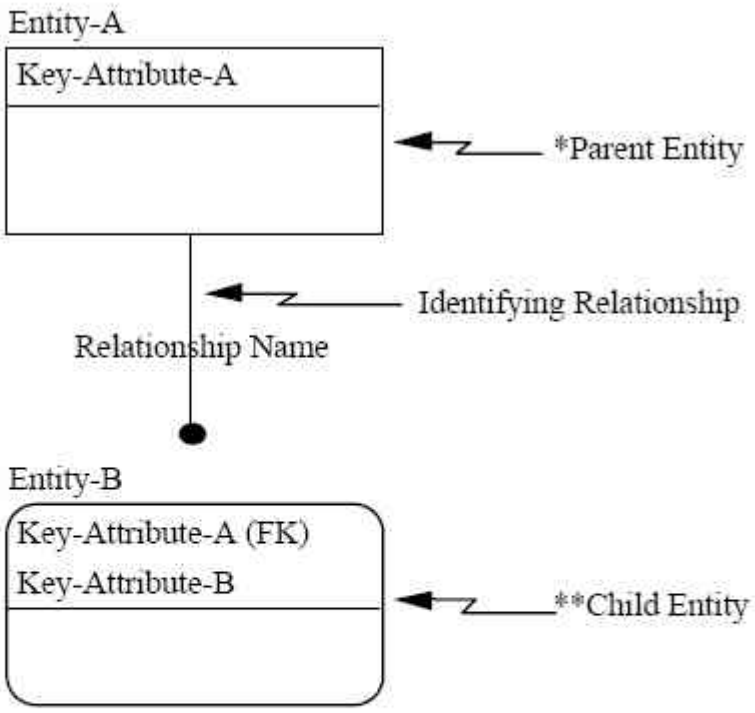
The candidate key selected as the unique identifier of an entity.

Foreign Keys

An attribute, or combination of attributes of a child or category entity instance whose values match those in the primary key of a related parent or generic entity instance. A foreign key results from the migration of the parent or generic entities primary key through a specific connection or categorization relationship.



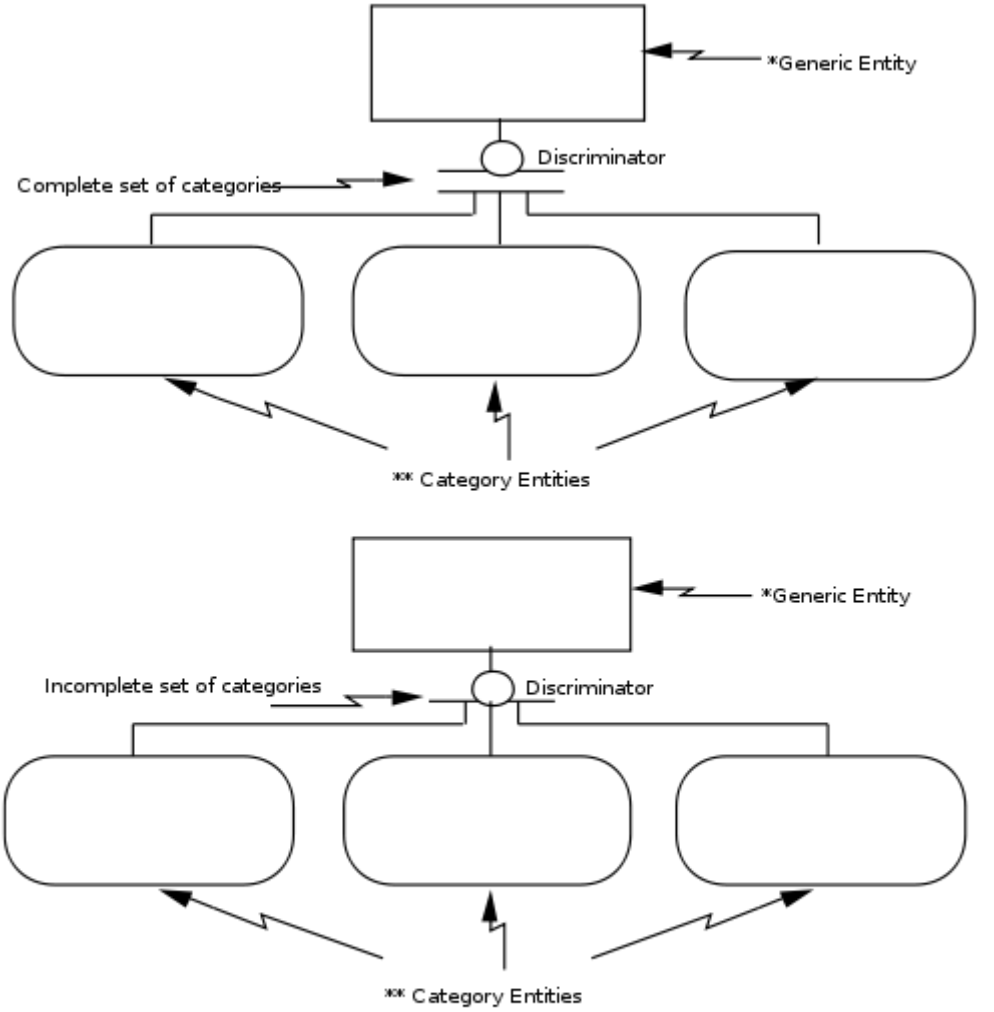
Relationship Cardinality Syntax



* The Parent Entity in an Identifying Relationship may be an Identifier-Independent Entity (as shown) or an Identifier-Dependent Entity depending upon other relationships.

** The Child Entity in an Identifying Relationship is always an Identifier-Dependent Entity.

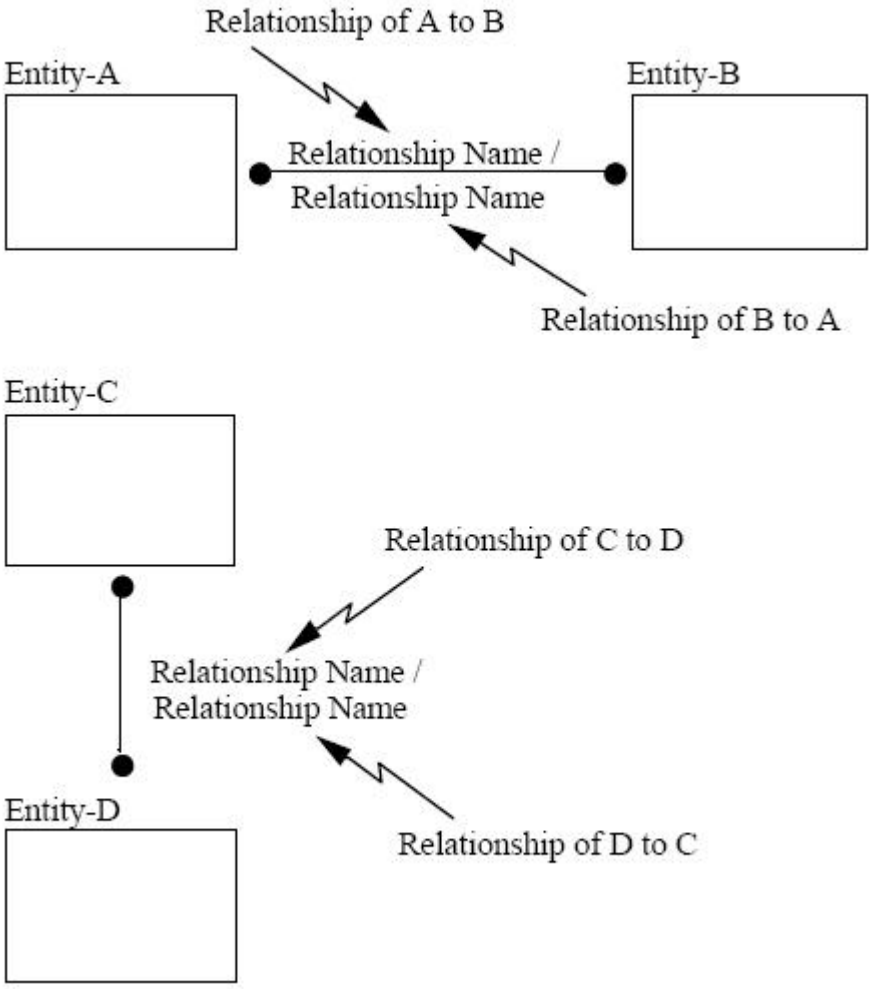
Identifying Relationship Syntax



* The Generic Entity may be an Identifier-Independent Entity (as shown) or a Identifier-Dependent Entity depending upon other relationships.

** Category Entities will always be Identifier-Dependent Entities.

Categorization Relationship Syntax



Non-Specific Relationship Syntax

Relationships

An association between two entities or between instances of the same entity.

Connection Relationships

The number of entity instances that can be associated with each other in a relationship.

Categorization Relationships

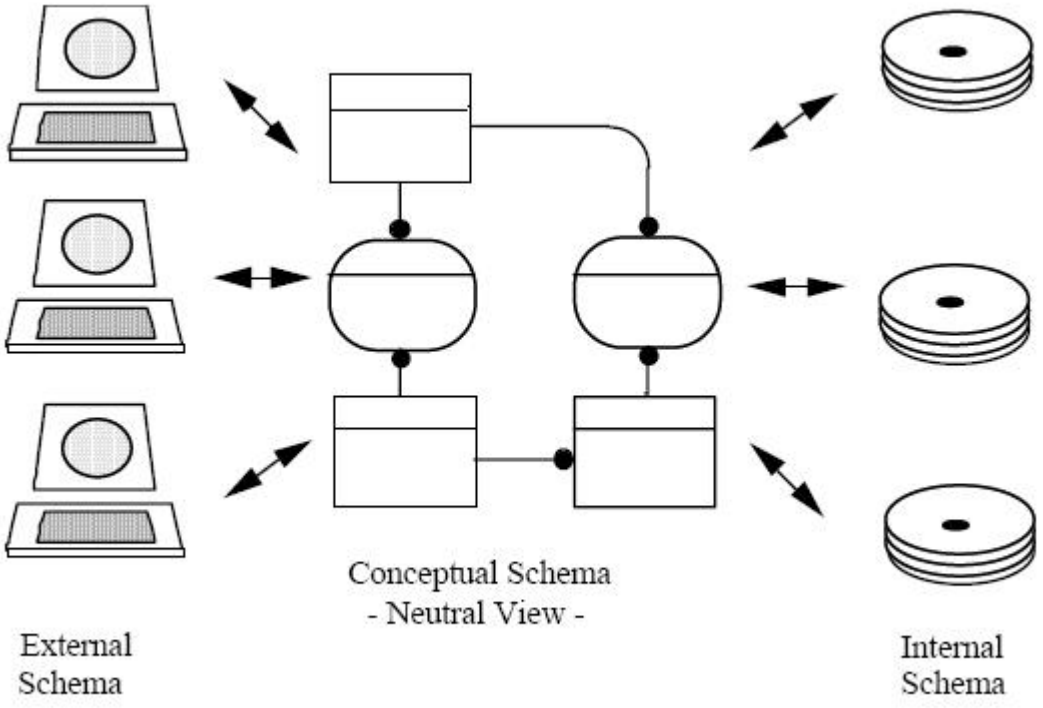
A relationship in which instances of both entities represent the same real or abstract thing. One entity (generic entity) represents the complete set of things, the other (category entity) represents a sub-type or sub-classification of those things. The category entity may have one or more characteristics, or a relationship with instances of another entity not shared by all generic entity instances. Each instance of the category entity is simultaneously an instance of the generic entity.

Non-Specific Relationships

A relationship in which an instance of either entity can be related to a number of instances of the other.

IDEF1X Topics

The Three Schema Approach



The three schema approach .

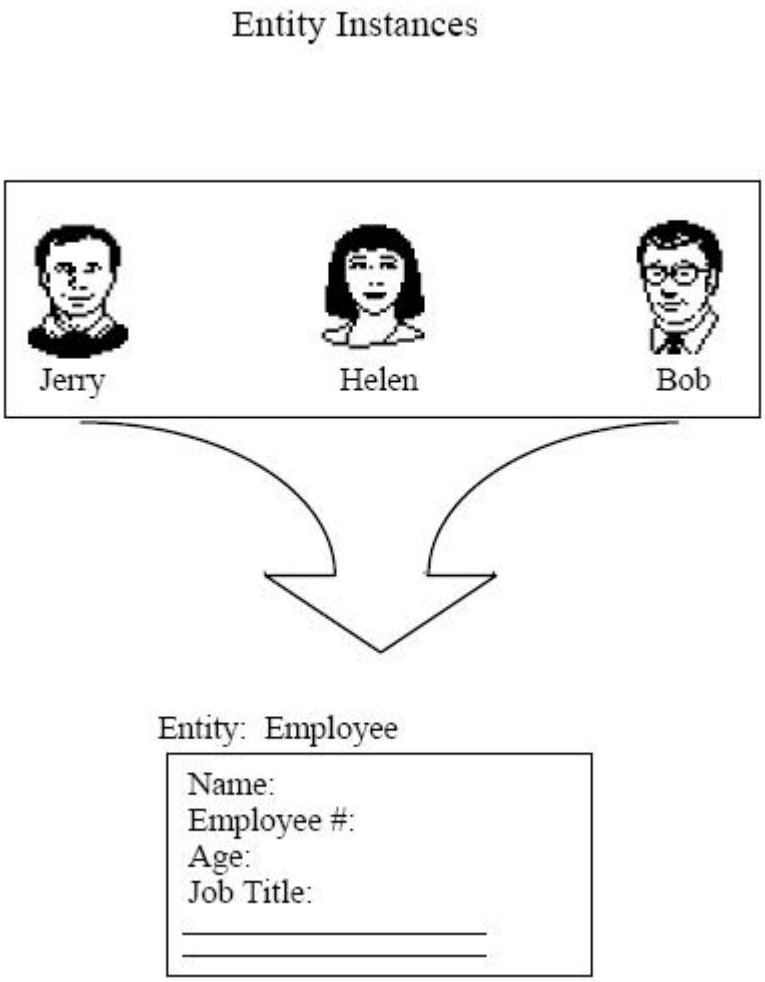
The three-schema approach in software engineering is an approach to building information systems and systems information management, that promotes the conceptual model as the key to achieving data integration.

A schema is a model, usually depicted by a diagram and sometimes accompanied by a language description. The three-schema approach has three types of schemas:

- External schema for user views
- Conceptual schema integrates external schemata
- Internal schema that defines physical storage structures

At the center, the conceptual schema defines the ontology of the concepts as the users think of them and talk about them. The physical schema describes the internal formats of the data stored in the database, and the external schema defines the view of the data presented to the application programs. The framework attempted to permit multiple data models to be used for external schemata.

Modeling Guidelines



Synthesizing an Entity in Phase One – Entity Definition

The modeling process can be divided into five stages of model developing.

Phase Zero - Project Initiation

The objectives of the project initiation phase include:

- Project definition — a general statement of what has to be done, why, and how it will get done.
- Source material — a plan for the acquisition of source material, including indexing and filing.
- Author conventions — a fundamental declaration of the conventions (optional methods) by which the author chooses to make and manage the model.

Phase One – Entity Definition

The objective of this phase is to identify and define the entities that fall within the problem domain being modeled. The first step in this process is the identification of entities.

Phase Two – Relationship Definition

The objective of Phase Two is to identify and define the basic relationships between entities. At this stage of modeling, some relationships may be non-specific and will require additional refinement in subsequent phases. The primary outputs from Phase Two are:

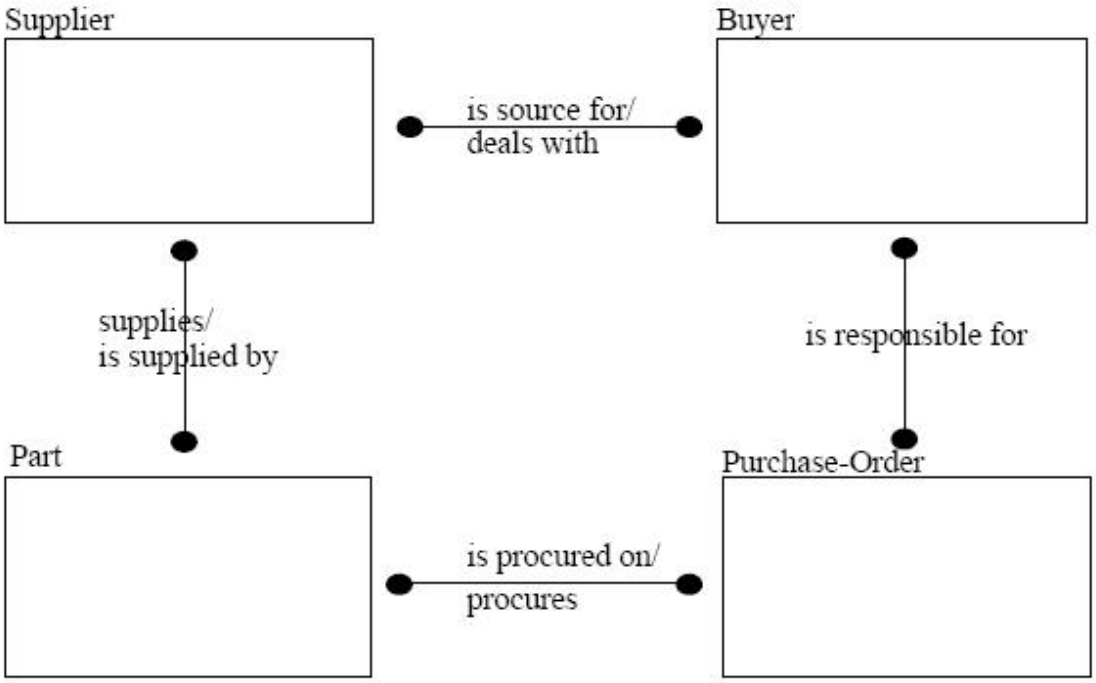
- Relationship matrix
- Relationship definitions
- Entity-level diagrams

Entity-Relationship Matrix Example

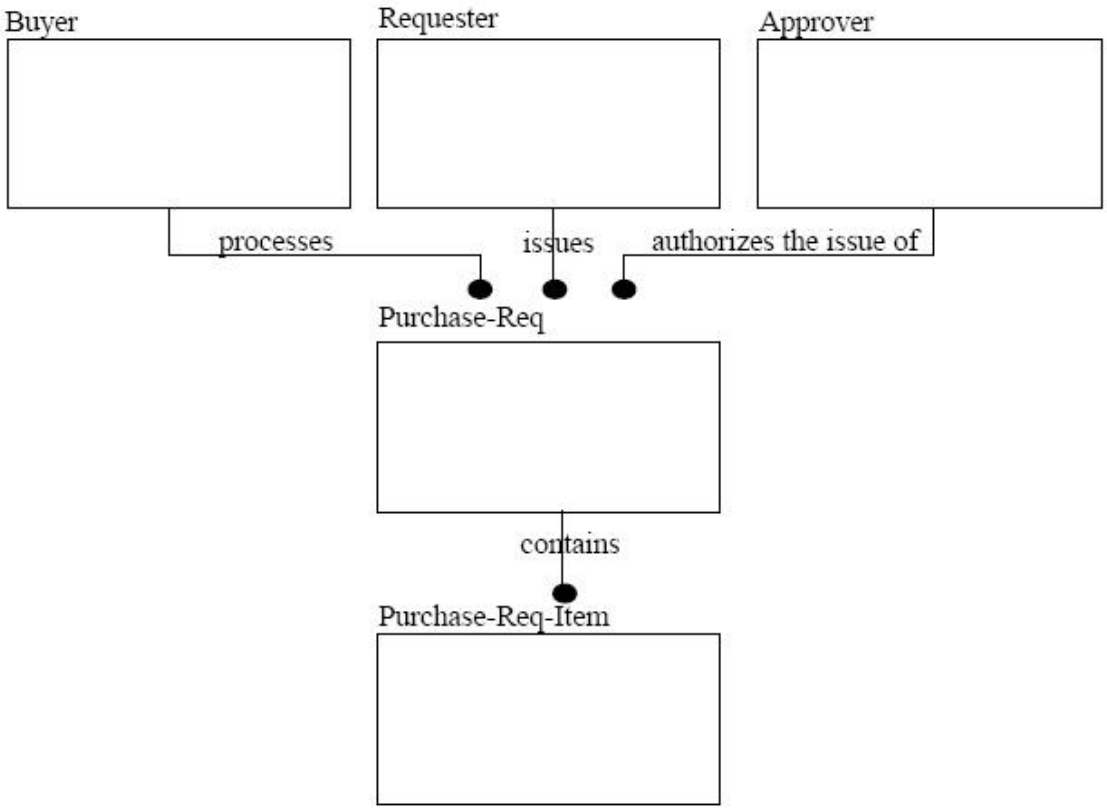
	Buyer	Requester	Approver	Purchase Requisition	Purchase Req. Item
Buyer		X		X	
Requester	X			X	
Approver				X	
Purchase Requisition	X	X	X		X
Purchase Req. Item				X	

An Entity-Relationship Matrix only reflects that a relationship of some kind may exist.

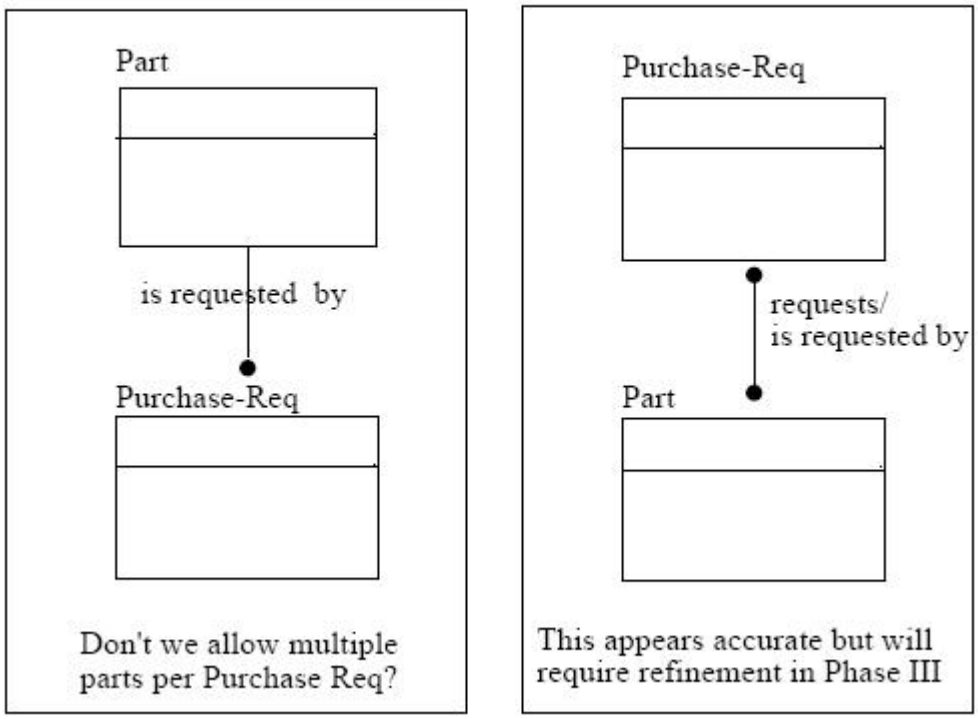
Entity Relationship Matrix



Entity Level Diagram



Entity Level Diagram Example



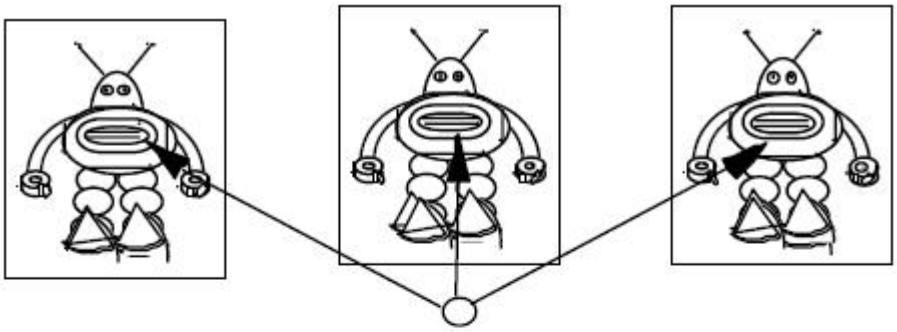
Reference Diagram

Phase Three - Key Definitions

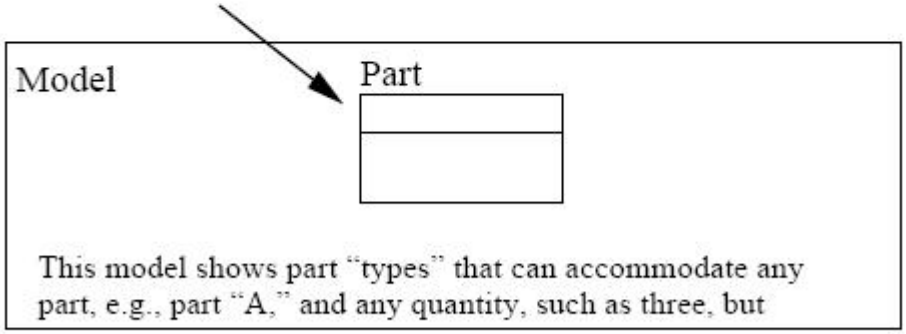
The objectives of Phase Three are to:

- Refine the non-specific relationships from Phase Two.
- Define key attributes for each entity.
- Migrate primary keys to establish foreign keys.
- Validate relationships and keys

Real World

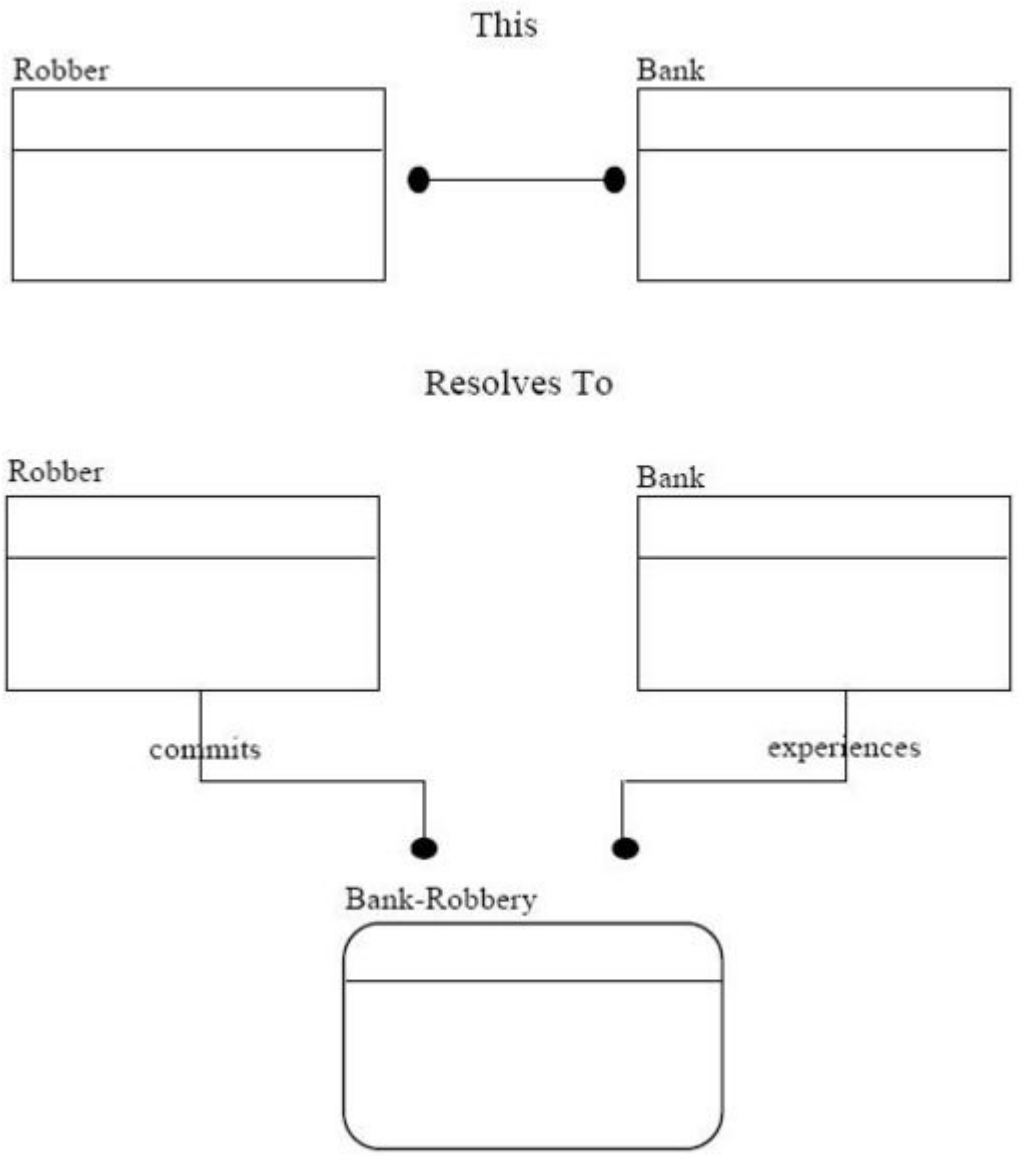


These three parts on the manufactured robot are of the same type. Let's call them part "A." There is no way that we can really tell one part "A" from any other part "A." That is, part "A's"are not uniquely identifiable This situation is modeled as follows:

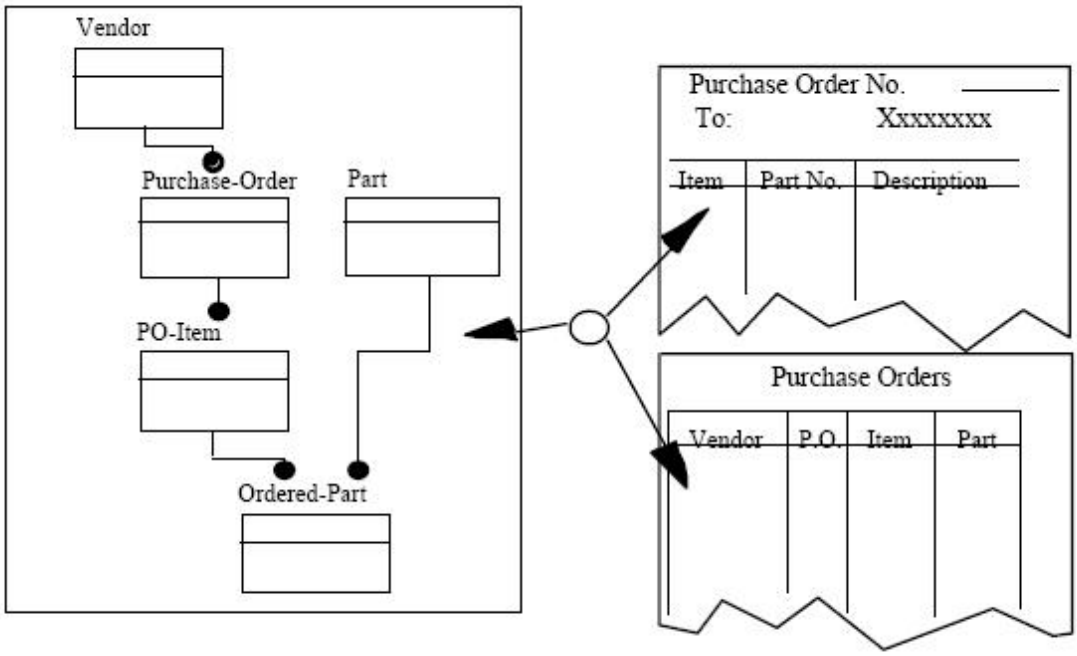


Problem What would happen to the model if serialized control of parts became a requirement?

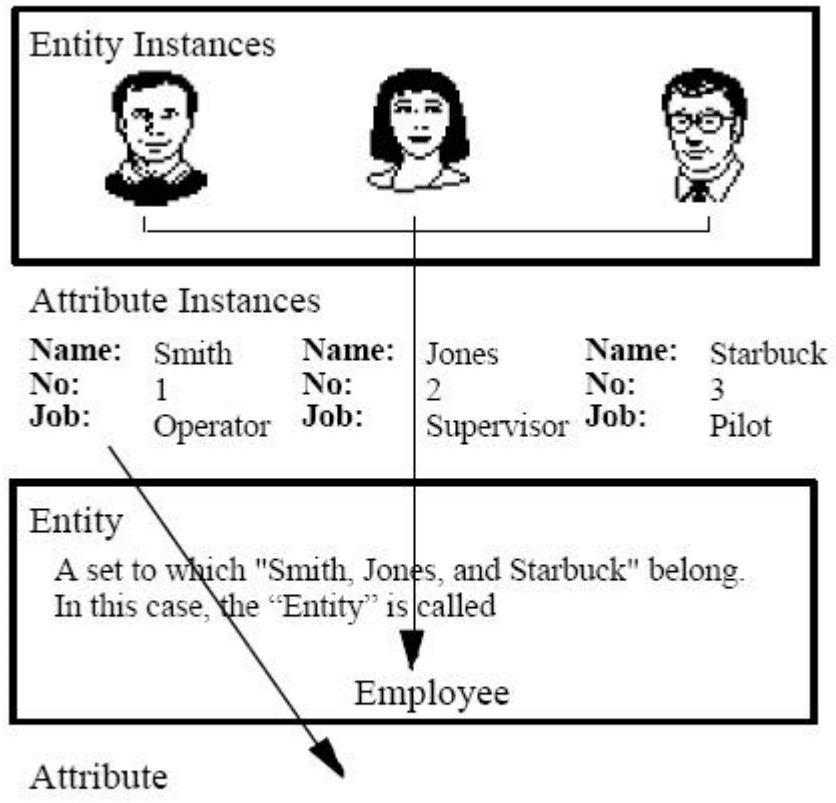
Example Reference Diagram



Non-Specific Relationship Refinement

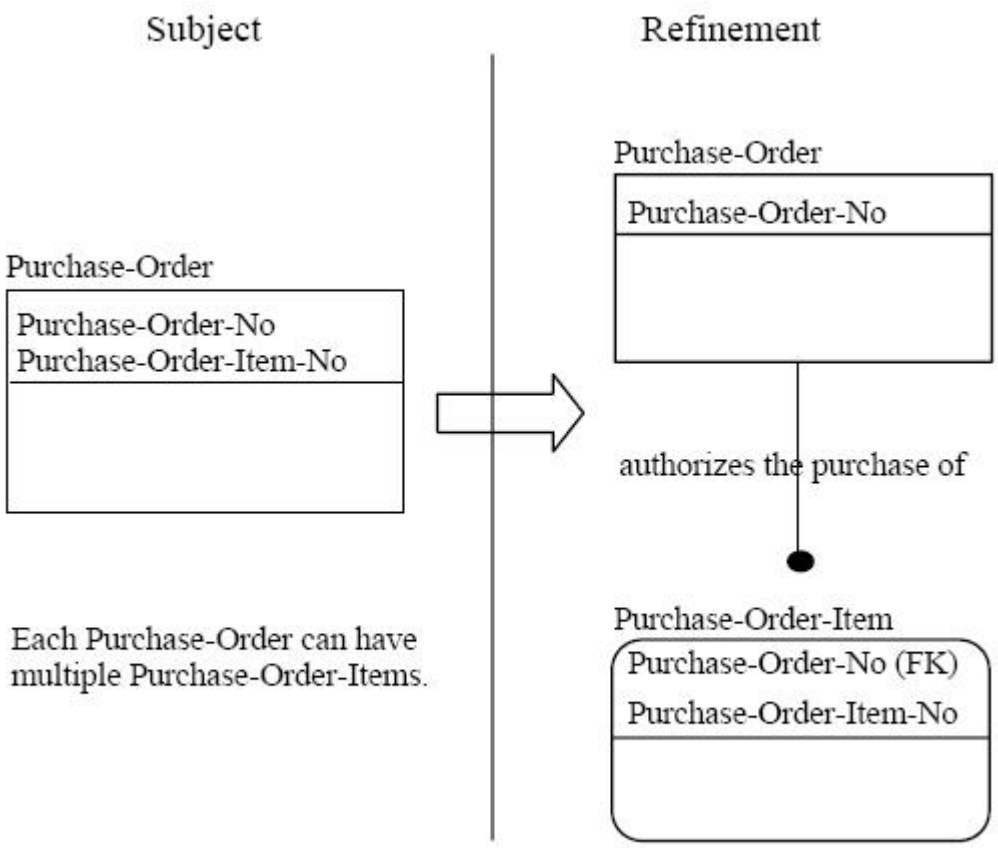


Scope of a Function View

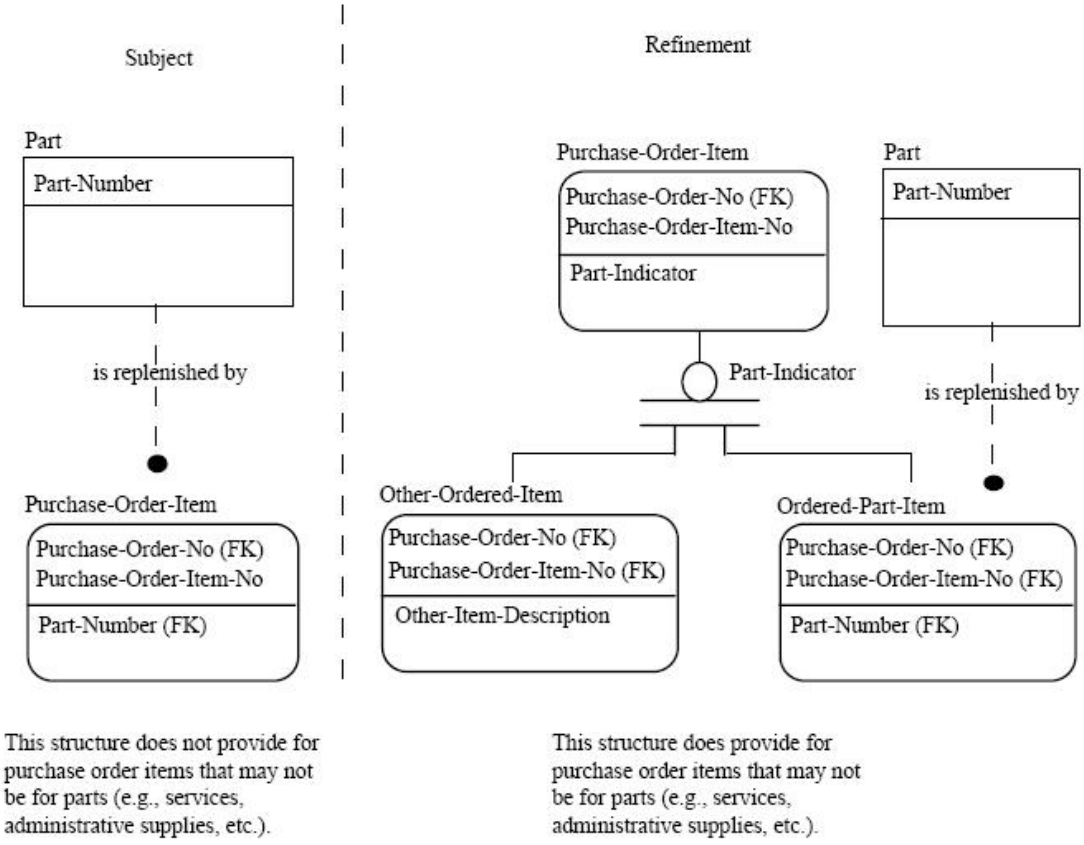


The "items" that commonly describe an Entity, e.g., Employee. In this case, the Attributes "Name, No., and Job" commonly describe each Employee.

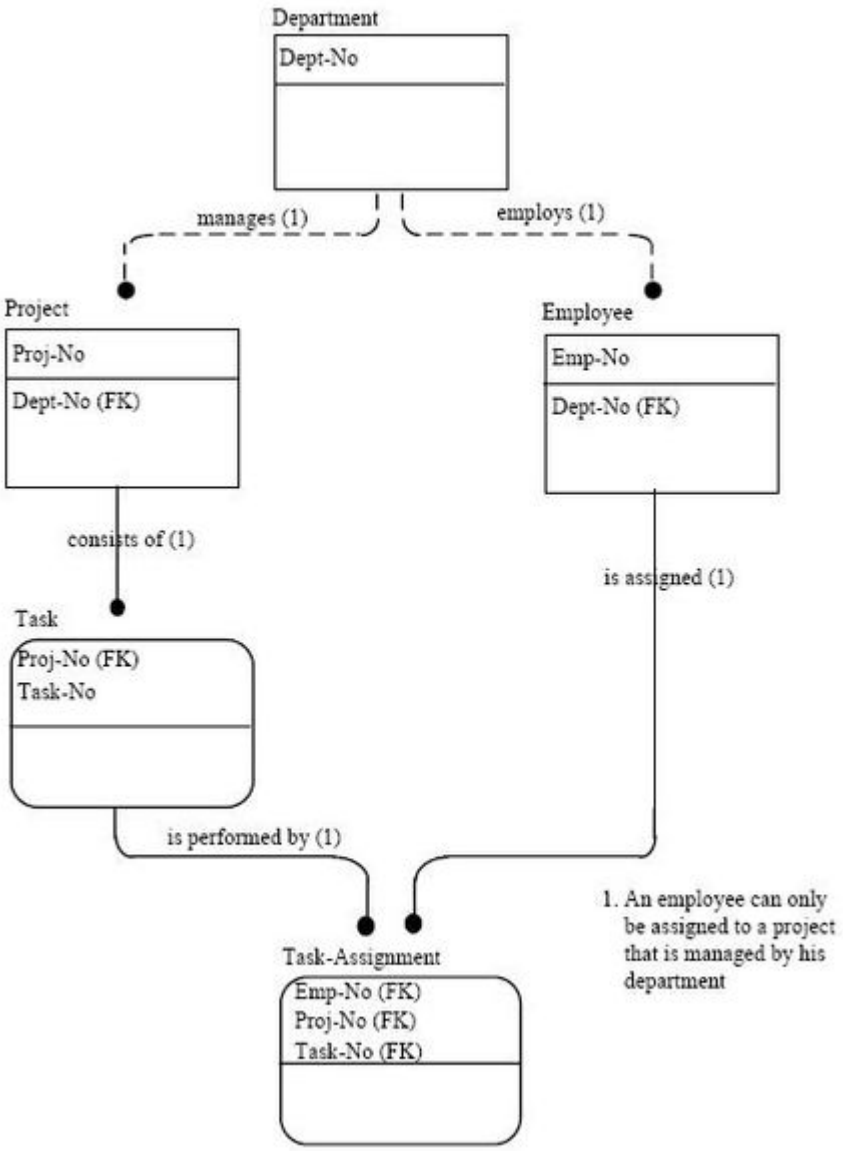
Attribute Examples



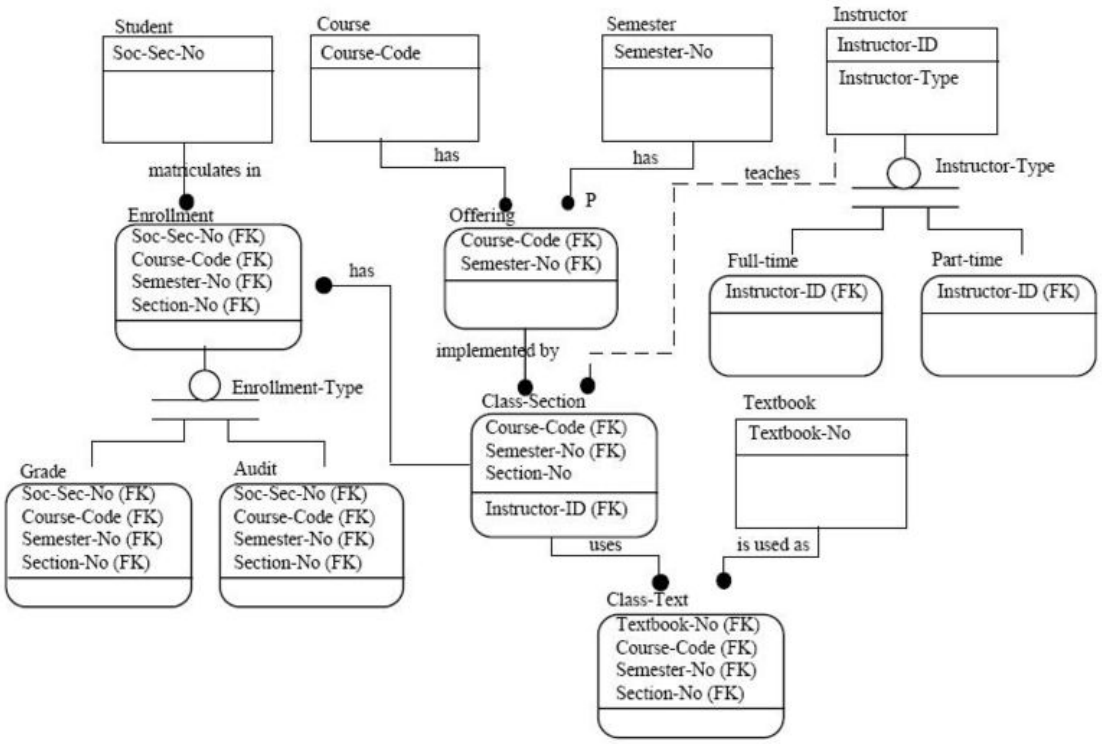
No-Repeat Rule Refinement



Rule Refinement.jpg



Path Assertions

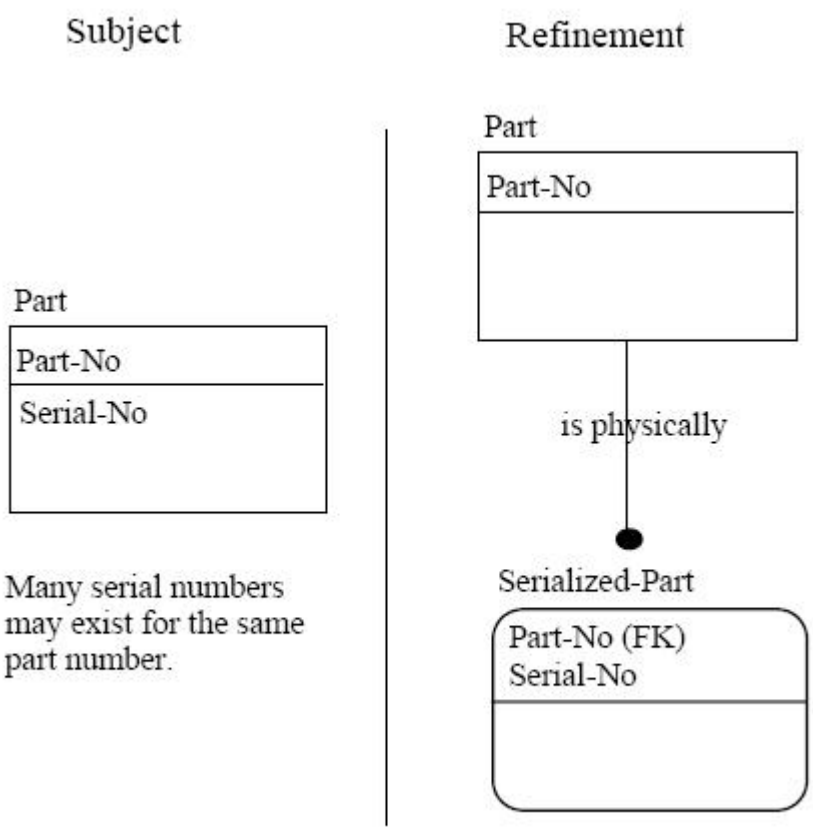


Example of Phase Three Function View Diagram

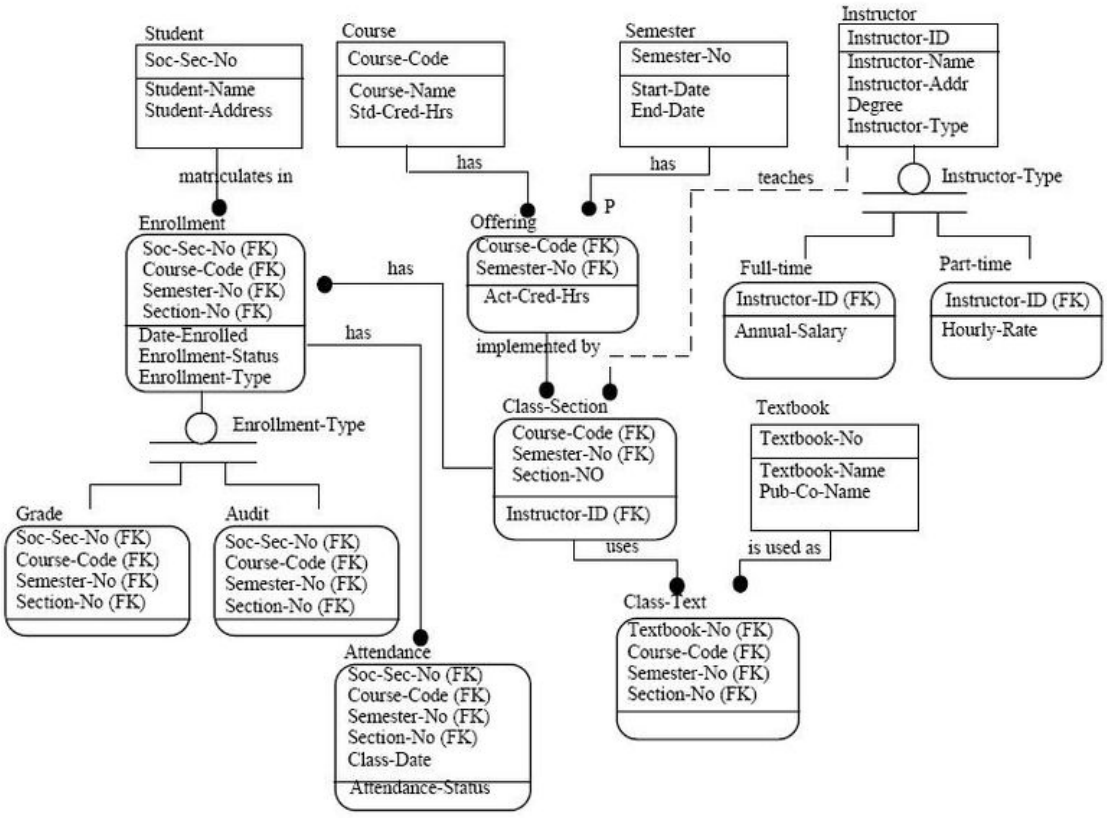
Phase Four - Attribute Definition

Phase Four is the final stage of model developing. The objectives of this plan are to:

- Develop an attribute pool
- Establish attribute ownership
- Define nonkey attributes
- Validate and refine the data structure

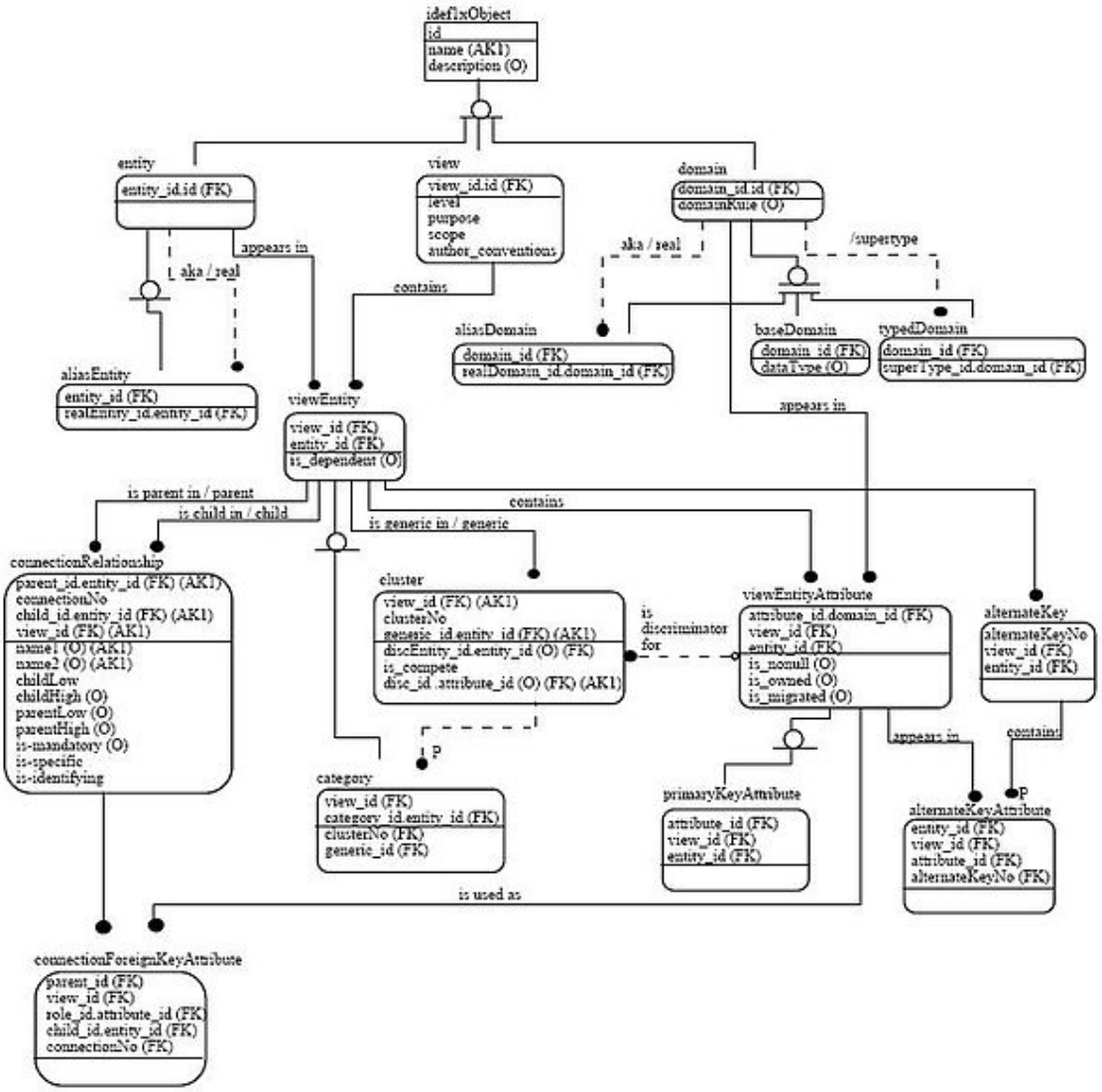


Applying the No Repeat Rule



Example of Phase Four Function

IDEF1X Meta Model



Meta Model of IDEF1X.

IDEF1X can be used to model IDEF1X itself. Such meta models can be used for various purposes, such as repository design, tool design, or in order to specify the set of valid IDEF1X models. Depending on the purpose, somewhat different models result. There is no “one right model”. For example, a model for a tool that supports building models incrementally must allow incomplete or even inconsistent models. The meta model for formalization emphasizes alignment with the concepts of the formalization. Incomplete or inconsistent models are not provided for. There are two important limitations on meta models. First, they specify syntax, not semantics. Second, a meta model must be supplemented with constraints in natural or formal language. The formal theory of IDEF1X provides both the semantics and a means to precisely express the necessary constraints.

A meta model for IDEF1X is given here. The name of the view is mm. The domain hierarchy and constraints are also given. The constraints are expressed as sentences in the formal theory of the meta model. The meta model informally defines the set of valid IDEF1X models in the usual way. The meta model also formally defines the set of valid IDEF1X models in the following way. The meta model, as an IDEF1X model, has a corresponding formal theory. The semantics of the theory are defined in the standard way. That is, an interpretaion of a theory consists of a domain of individuals and a set of assignments:

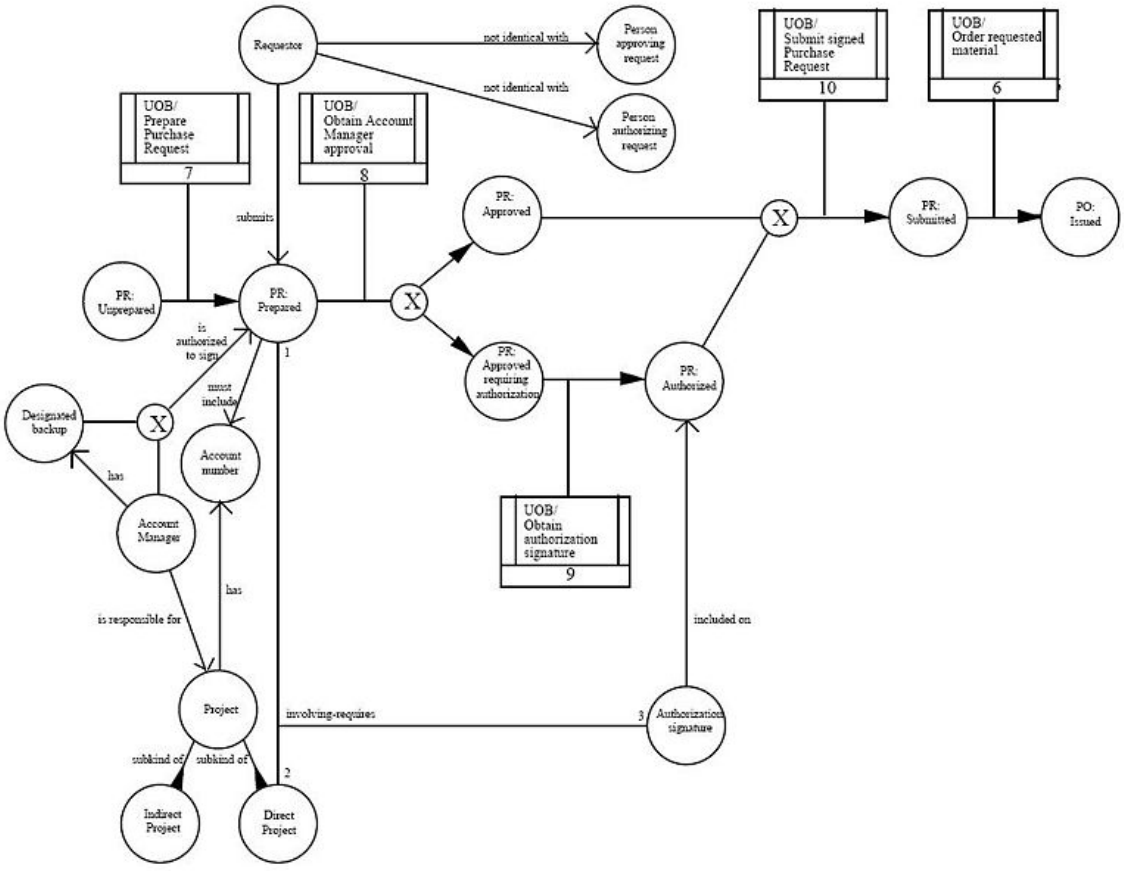
> To each constant in the theory, an individual in the domain is assigned.
> To each n-ary function symbol in the theory, an n-ary function over the domain is assigned.
> To each n-ary predicate symbol in the theory, an n-ary relation over the domain is assigned.

In the intended interpretation, the domain of individuals consists of views, such as production; entities, such as part and vendor; domains, such as qty_on_hand; connection relationships; category clusters; and so on. If every axiom in the theory is true in the interpretation, then the interpretation is called a model for the theory. Every model for the IDEF1X theory corresponding to the IDEF1X meta model and its constraints is a valid IDEF1X model.

Chapter 5

IDEF3 & IDEF5

IDEF3



Example of an Enhanced Transition Schematic, modelled with IDEF3.

IDEF3, officially named a *Integrated DEFinition for Process Description Capture Method*, is a business process modelling method complementary to IDEF0. The IDEF3 method is a scenario-driven process flow description capture method intended to capture the knowledge about how a particular system works.

The IDEF3 method provides modes to represent both

- Process Flow Descriptions to capture the relationships between actions within the context of a specific scenario, and
- Object State Transition to capture the description of the allowable states and conditions.

This method is part of the IDEF family of modeling languages in the field of systems and software engineering.

Overview

One of the primary mechanisms used for descriptions of the world is relating a story in terms of an ordered sequence of events or activities. The IDEF3 Process Description Capture Method was created to capture descriptions of sequences of activities, which is considered the common mechanisms to describe a situation or process. The primary goal of IDEF3 is to provide a structured method by which a domain expert can express knowledge about the operation of a particular system or organization. Knowledge acquisition is enabled by direct capture of assertions about real-world processes and events in a form that is most natural for capture. IDEF3 supports this kind of knowledge acquisition by providing a reliable and wellstructured approach for process knowledge acquisition, and an expressively, yet easy-to-use, language for information capture and expression.

Motives for the development of IDEF3 were the need:

- to speed up the process of business systems modeling,
- to provides mechanisms to describe this data life cycle information,
- to supported project management techniques by an automated tool,
- to provide the concepts, syntax, and procedures for building system requirements descriptions, and
- to work well both independently and jointly with other methods which address different areas of concentration (e.g., the IDEF0 Function Modeling method) as a complementary addition to the IDEF method family.

History

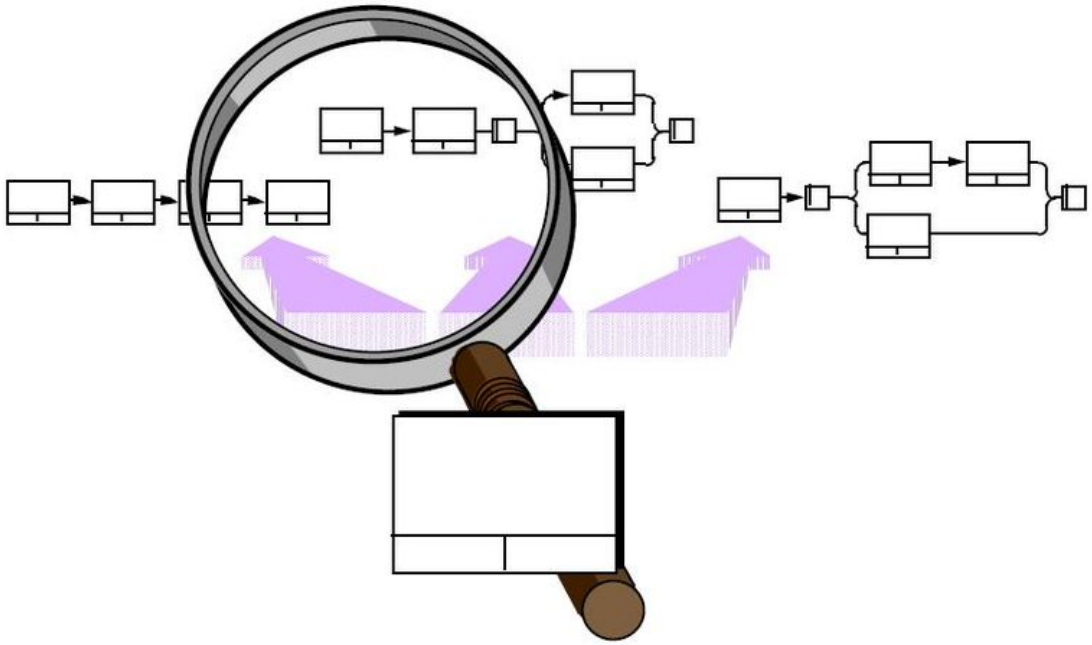
The original IDEFs were developed since the mid-1970s for the purpose of enhancing communication among people who needed to decide how their existing systems were to be integrated. IDEF0 was designed to allow a graceful expansion of the description of a systems' functions through the process of function decomposition and categorization of

the relations between functions (i.e., in terms of the Input, Output, Control, and Mechanism classification). IDEF1 was designed to allow the description of the information that an organization deems important to manage in order to accomplish its objectives.

The third IDEF (IDEF2) was originally intended as a user interface modeling method. However, since the Integrated Computer-Aided Manufacturing (ICAM) Program needed a simulation modeling tool, the resulting IDEF2 was a method for representing the time varying behavior of resources in a manufacturing system, providing a framework for specification of math model based simulations. It was the intent of the methodology program within ICAM to rectify this situation but limitation of funding did not allow this to happen. As a result, the lack of a method which would support the structuring of descriptions of the user view of a system has been a major shortcoming of the IDEF system. The basic problem from a methodology point of view is the need to distinguish between a description of what a system (existing or proposed) is supposed to do and a representative simulation model that will predict what a system will do. The latter was the focus of IDEF2, the former is the focus of IDEF3.

IDEF3 basic concepts

Descriptions and models



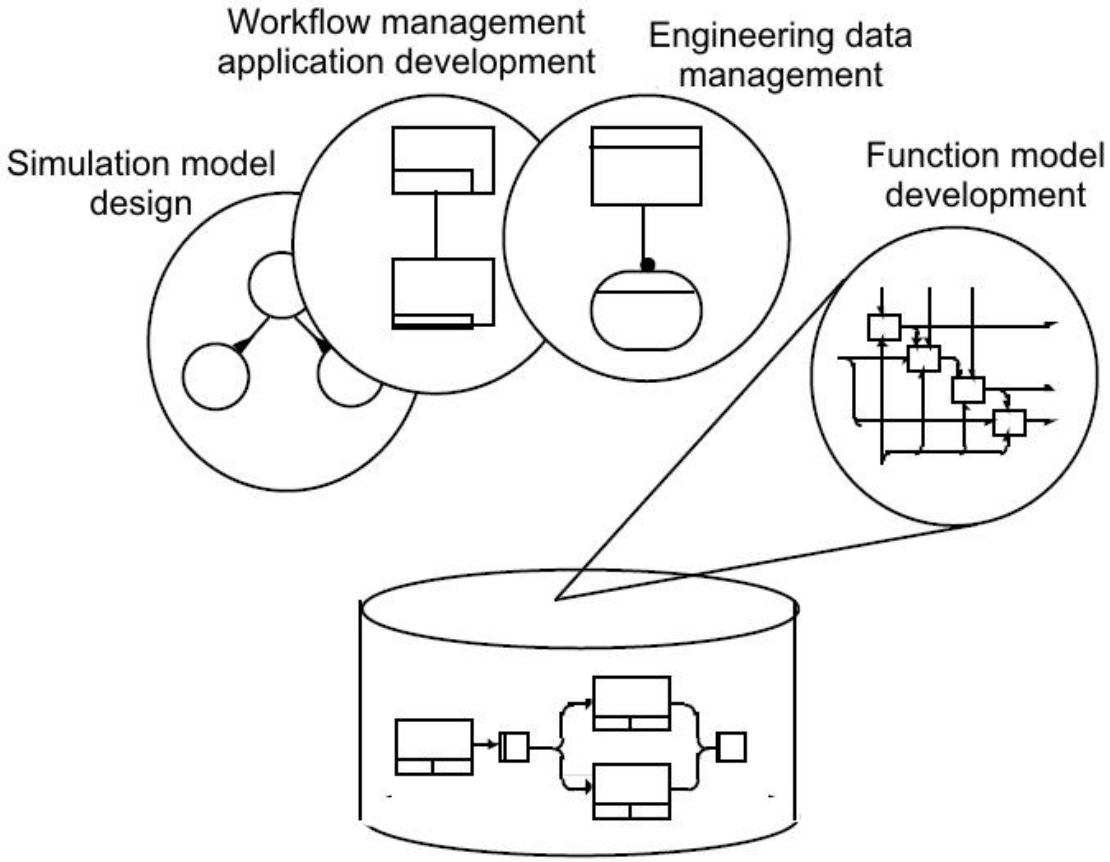
IDEF3 want to offer alternative descriptions of the same process from multiple viewpoints on the process.

The distinction between descriptions and models, though subtle, is an important one in IDEF3, and both have a precise technical meaning.

- The term description is used as a reserved technical term to mean records of empirical observations; that is, descriptions record knowledge that originates in or is based on observations or experience.
- The term model is used to mean an idealization of an entity or state of affairs. That is, a model constitutes an idealized system of objects, properties, and relations that is designed to imitate, in certain relevant respects, the character of a given real-world system. Frictionless planes, perfectly rigid bodies, the assumption of point mass, and so forth are representative examples of models.

The power of a model comes from its ability to simplify the real-world system it represents and to predict certain facts about that system by virtue of corresponding facts within the model. Thus, a model is a designed system in its own right. Models are idealized systems known to be incorrect but assumed to be close enough to provide reliable predictors for the predefined areas of interest within a domain. A description, on the other hand, is a recording of facts or beliefs about something within the realm of an individual's knowledge or experience. Such descriptions are generally incomplete; that is, the person giving a description may omit facts that he or she believes are irrelevant, or which were forgotten in the course of describing the system. Descriptions may also be inconsistent with respect to how others have observed situations within the domain. IDEF3 accommodates these possibilities by providing specific features enabling the capture and organization of alternative descriptions of the same scenario or process.

Description capture



IDEF focus on Description Capture, that enables reuse.

Modeling necessitates taking additional steps beyond description capture to resolve conflicting or inconsistent views. This, in turn, generally requires modelers to select or create a single viewpoint and introduce artificial modeling approximations to fill in gaps where no direct knowledge or experience is available. Unlike models, descriptions are not constrained by idealized, testable conditions that must be satisfied, short of simple accuracy.

The purpose of description capture may be simply to record and communicate process knowledge or to identify inconsistencies in the way people understand how key processes actually operate. By using a description capture method users need not learn and apply conventions forcing them to produce executable models (e.g., conventions ensuring accuracy, internal consistency, logical coherence, non-redundancy, completeness). Forcing users to model requires them to adopt a model design perspective and risk producing models that do not accurately capture their empirical knowledge of the domain.

Scenarios

The notion of a scenario or story is used as the basic organizing structure for IDEF3 Process Descriptions. A scenario can be thought of as a recurring situation, a set of situations that describe a typical class of problems addressed by an organization or system, or the setting within which a process occurs. Scenarios establish the focus and boundary conditions of a description. Using scenarios in this way exploits the tendency of humans to describe what they know in terms of an ordered sequence of activities within the context of a given scenario or situation. Scenarios also provide a convenient vehicle to organize collections of process-centered knowledge.

Process-Centered Views

IDEF3 Process Schematics are the primary means for capturing, managing, and displaying process-centered knowledge. These schematics provide a graphical medium that helps domain experts and analysts from different application areas communicate knowledgeabout processes. This includes knowledge about events and activities, the objects that participate in those occurrences, and the constraining relations that govern the behavior of an occurrence.

Object-Centered Views

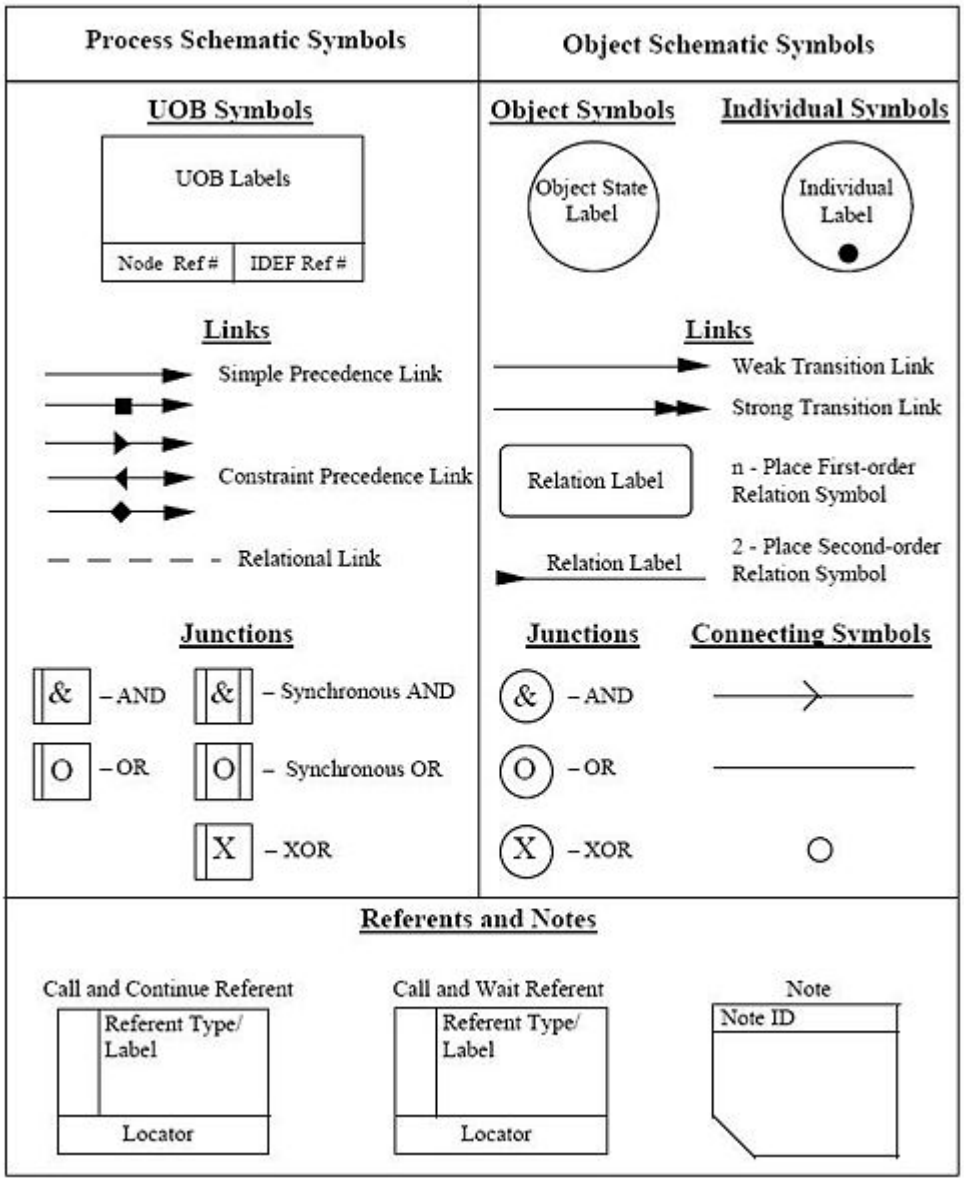
IDEF3 Object Schematics capture, manage, and display object-centered descriptions of a process—that is, information about how objects of various kinds are transformed into other kinds of things through a process, how objects of a given kind change states through a process, or context-setting information about important relations among objects in a process.

IDEF3 process description language

IDEF3 descriptions are developed from two different perspectives: process-centered and object-centered. Because these approaches are not mutually exclusive, IDEF3 allows cross-referencing between them to represent complex process descriptions.

Process Schematics

Process schematics tend to be the most familiar and broadly used component of the IDEF3 method. These schematics provide a visualization mechanism for processcentered descriptions of a scenario. The graphical elements that comprise process schematics include Unit of Behavior (UOB) boxes, precedence links, junctions, referents, and notes. The building blocks here are:



Symbols Used for IDEF3 Process Description Schematics.

- Unit of Behavior (UOB) boxes
- Links : Links are the glue that connect UOB boxes to form representations of dynamic processes.
- Simple Precedence Links : Precedence links express temporal precedence relations between instances of one UOB and those of another.
- Activation Plots : Activation plots are used to represent activations.
- Dashed Links : Dashed links carry no predefined semantics.
- Link Numbers : All links have an elaboration and unique link numbers.
- Activation Semantics for Nonbranching Process Schematics.
- Junctions : Junctions in IDEF3 provide a mechanism to specify the logic of process branching.

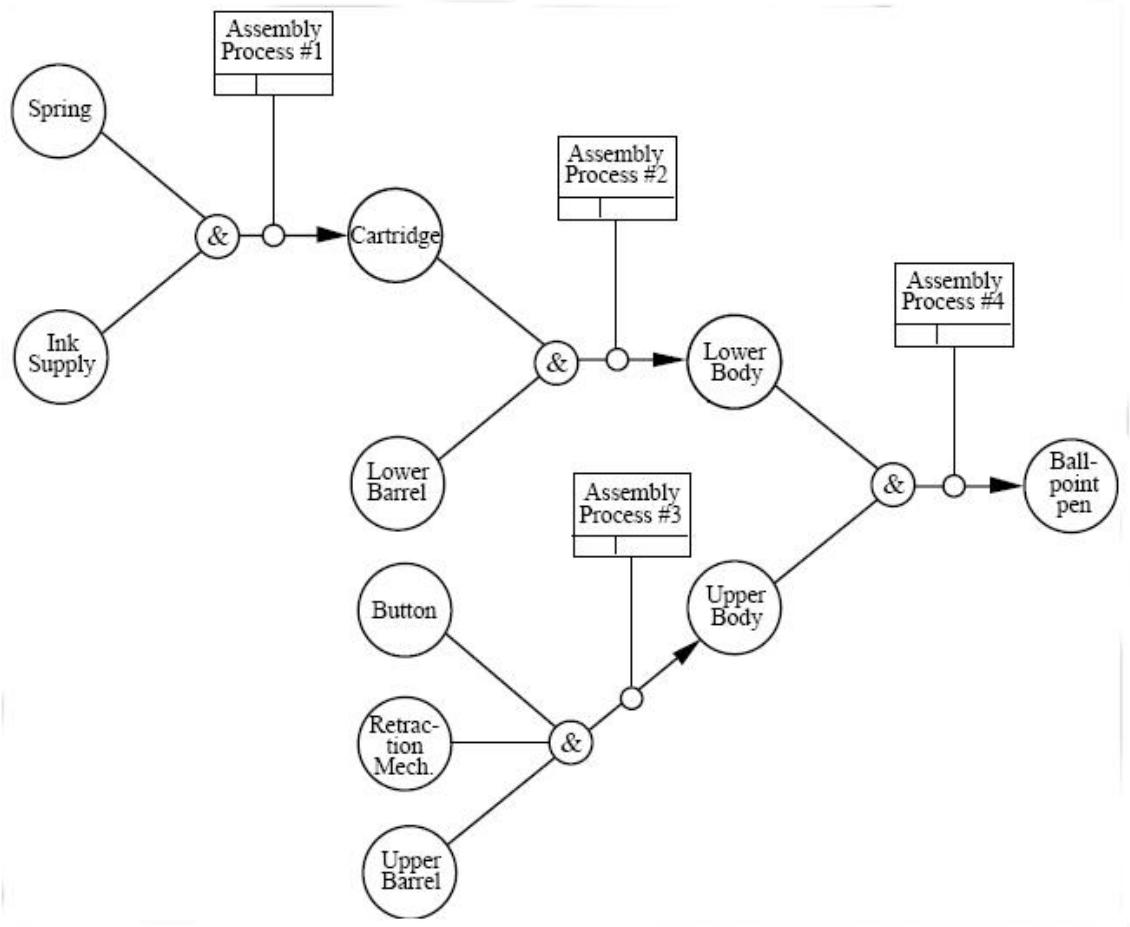
- UOB Decompositions : Elaborations capture and structure detailed knowledge about processes.
- UOB Reference Numbering Scheme : A UOB box number is assigned to each UOB box in an IDEF3 Process Description.
- Partial Descriptions : UOB boxes are joined together by links. Because of the description capture focus of IDEF3, it is possible to conceive of UOBs without links to other parts of an IDEF3 schematic.
- Referents : Referents enhance understanding, provide additional meaning, and simplify the construction (i.e., minimize clutter) of both process schematics and object schematics.

Object Schematics

IDEF offers a series of building blocks to express detailed object-centered process information; that is, information about how objects of various kinds are transformed into other kinds of things through a process, or how objects of a given kind change states through a process.

- Objects : An object of a certain kind, like a chassis, will be represented simply by a circle containing an appropriate label.
- Object States : A certain kind of object being in a certain state will be represented by a circle with a label that captures the kind itself and a corresponding state, representing thereby the type, or class, of objects that are in that state.
- Object schematics : The construction of complex representations built from kind symbols and object state symbols.
- Transition Schematics : The first and most basic construct is the basic state transition schematic or simply, transition schematic.

IDEF5



Example of an IDEF5 Composition Schematic for a Ballpoint Pen.

IDEF5 (*Integrated Definition for Ontology Description Capture Method*) is a software engineering method to develop and maintain usable, accurate, domain ontologies. This standard is part of the IDEF family of modeling languages in the field of software engineering.

Overview

In the field of computer science ontologies are used to capture the concept and objects in a specific domain, along with associated relationships and meanings. In addition, ontology capture helps coordinate projects by standardizing terminology and creates opportunities for information reuse. The lDEF5 Ontology Capture Method has been

developed to reliably construct ontologies in a way that closely reflects human understanding of the specific domain.

In the IDEF5 method, an ontology is constructed by capturing the content of certain assertions about real-world objects, their properties, and their interrelationships and representing that content in an intuitive and natural form. The IDEF5 method has three main components:

- A graphical language to support conceptual ontology analysis
- A structured text language for detailed ontology characterization, and
- A systematic procedure that provides guidelines for effective ontology capture.

IDEF5 Topics

Ontology

In IDEF5 the meaning of the term "ontology" is characterized to include a catalog of terms used in a domain, the rules governing how those terms can be combined to make valid statements about situations in that domain, and the “sanctioned inferences” that can be made when such statements are used in that domain. In every domain, there are phenomena that the humans in that domain discriminate as (conceptual or physical) objects, associations, and situations. Through various language mechanisms, we associate definite descriptors (e.g., names, noun phrases, etc.) to that phenomena.

Central Concepts of Ontology

The construction of ontologies for human engineered systems is the focus of the IDEF5. In the context of such systems, the nature of ontological knowledge involves several modifications to the more traditional conception. The first of these modifications has to do with the notion of a kind. Historically, a kind is an objective category of objects that are bound together by a common nature, a set of properties shared by all and only the members of the kind.

While there is an attempt to divide the world at its joints in the construction of enterprise ontologies, those divisions are not determined by the natures of things in the enterprise so much as the roles those things are to play in the enterprise from some perspective or other. Because those roles might be filled in any of a number of ways by objects that differ in various ways, and because legitimate perspectives on a domain can vary widely, it is too restrictive to require that the instances of each identifiable kind in an enterprise share a common nature, let alone that the properties constituting that nature be essential to their bearers. Consequently, enterprise ontologies require a more flexible notion of kind.

Ontology development process

Ontology development requires extensive iterations, discussions, reviews, and introspection. Knowledge extraction is usually a discovery process and requires considerable introspection. It requires a process that incorporates both significant expert involvement as well as the dynamics of a group effort. Given the open-ended nature of ontological analyses, it is not prudent to adopt a "cookbook" approach to ontology development. In brief, the IDEF5 ontology development process consists of the following five activities:

1. *Organizing and Scoping*: This activity involves establishing the purpose, viewpoint, and context for the ontology development project and assigning roles to the team members.
2. *Data Collection*: This activity involves acquiring the raw data needed for ontology development.
3. *Data Analysis*: This activity involves analyzing the data to facilitate ontology extraction.
4. *Initial Ontology Development*: This activity involves developing a preliminary ontology from the acquired data.
5. *Ontology Refinement and Validation*: This activity involves refining and validating the ontology to complete the development process.

Although the above activities are listed sequentially, there is a significant amount of overlap and iteration between the activities.

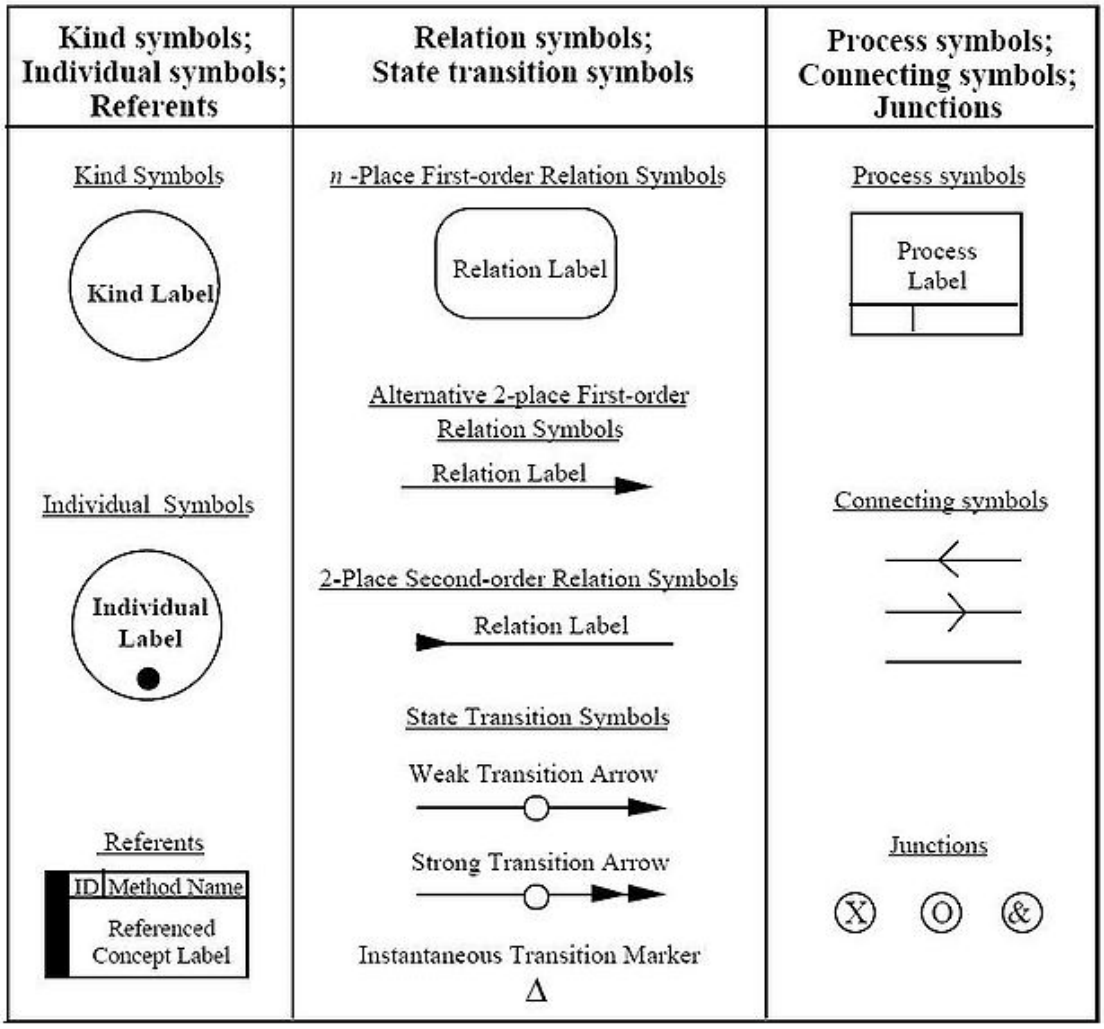
Ontological Analysis

Ontological analysis is accomplished by examining the vocabulary that is used to discuss the characteristic objects and processes that compose the domain, developing rigorous definitions of the basic terms in that vocabulary, and characterizing the logical connections among those terms. The product of this analysis, an ontology, is a domain vocabulary complete with a set of precise definitions, or axioms, that constrain the meanings of the terms sufficiently to enable consistent interpretation of the data that use that vocabulary.

IDEF5 Building blocks

Definitions

Some of the key terms in IDEF5 and the basic IDEF5 Schematic Language Symbols, see figure.:



Kind

Informally, a group of individuals that share some set of distinguished characteristics. More formally, kinds are properties typically expressed by common nouns such as ‘employee’, ‘machine’, and ‘lathe’.

Individual

The most logically basic kind of real world object. Prominent examples include human persons, concrete physical objects, and certain abstract objects such as programs. Unlike objects of higher logical orders such as properties and relations, individuals essentially are not multiply instantiable. Individuals are also known as first-order objects.

Referent

A construct in the IDEF5 elaboration language used to refer to a kind, object, property, relation, or process kind in another ontology or an IDEF model.

Relation

An abstract, general association or connection that holds between two or more objects. Like properties, relations are multiply instantiable. The objects among which a relation holds in a particular instance are known as its arguments.

State

A property, generally indicated by an adjective rather than a common noun, that is characteristic of objects of a certain kind at a certain point within a process. For example, water can be in frozen, liquid, or gaseous states.

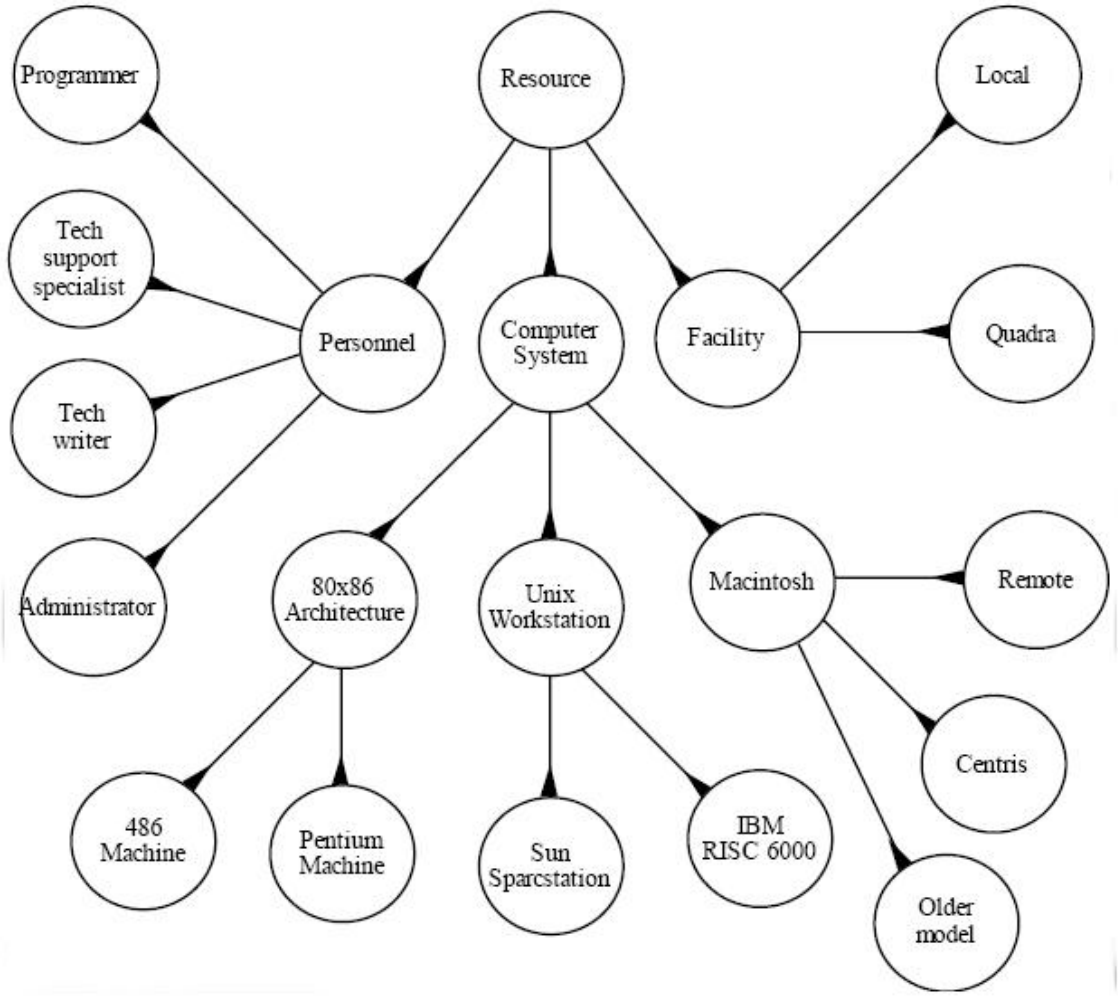
Process

A real world event or state of affairs involving one or more individuals over some (possibly instantaneous) interval of time. Typically, a process involves some sort of change in the properties of one or more of the individuals within the process. Because of the ambiguity in the term “process”, sometimes referred to as process instance.

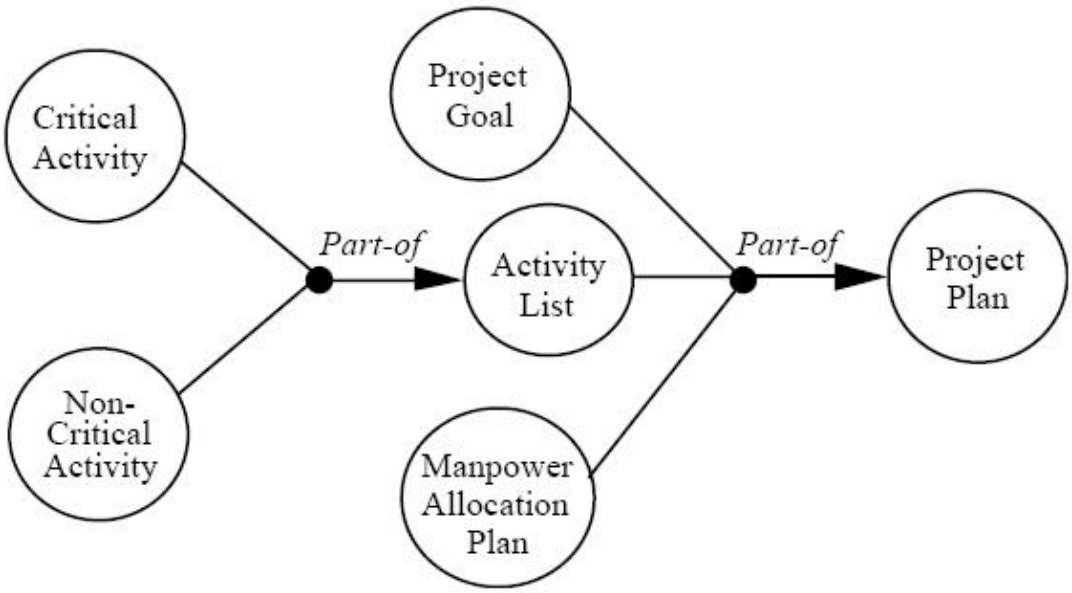
Diagram types

Various diagram types, or schematics, can be constructed in the IDEF5 Schematic Language. The purpose of these schematics, like that of any representation, is to represent information visually. Thus, semantic rules must be provided for interpreting every possible schematic. These rules are provided by outlining the rules for interpreting the most basic constructs of the language, then applying them recursively to more complex constructs. There are four primary schematic types derived from the basic IDEF5 Schematic Language which can be used to capture ontology information directly in a form that is intuitive to the domain expert.

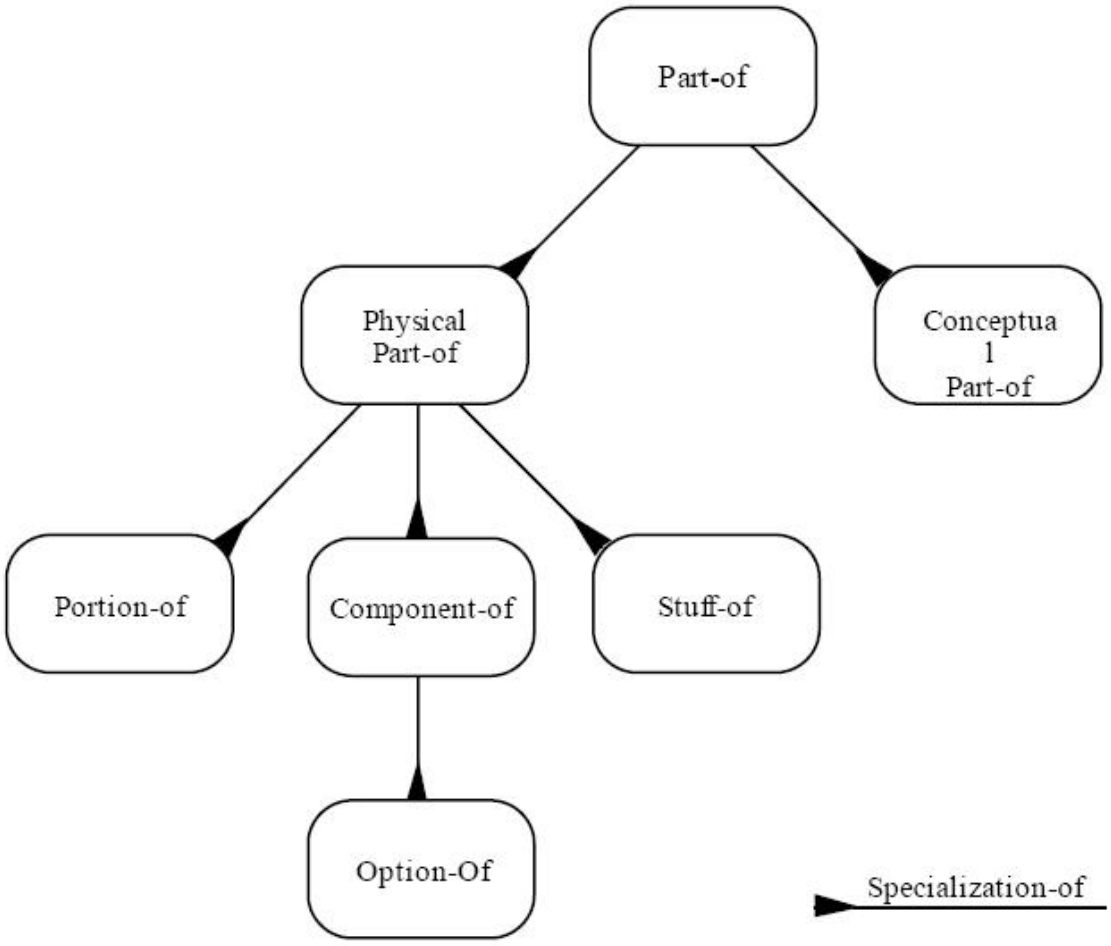
- *Classification Schematics* : Classification schematics provide mechanisms for humans to organize knowledge into logical taxonomies. Of particular merit are two types of classification: description subsumption and natural kind classification.
- *Composition Schematics* : Composition schematics serve as mechanisms to represent graphically the "part-of" relation that is so common among components of an ontology.
- *Relation Schematics* : Relation schematics allow ontology developers to visualize and understand relations among kinds in a domain, and can also be used to capture and display relations between first-order relations.
- *Object State Schematics* : Because there is no clean division between information about kinds and states and information about processes, the IDEF5 schematic language enables modelers to express fairly detailed object-centered process information (i.e., information about kinds of objects and the various states they can be in relative to certain processes). Diagrams built from these constructs are known as Object-State Schematics.



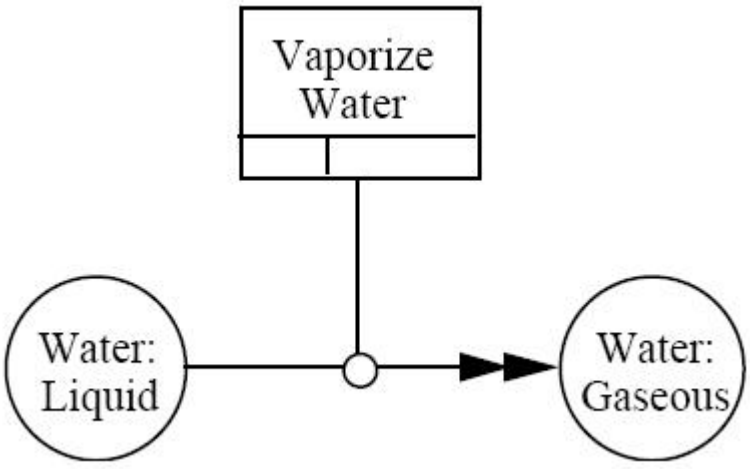
Classification Schematics



Composition Schematics



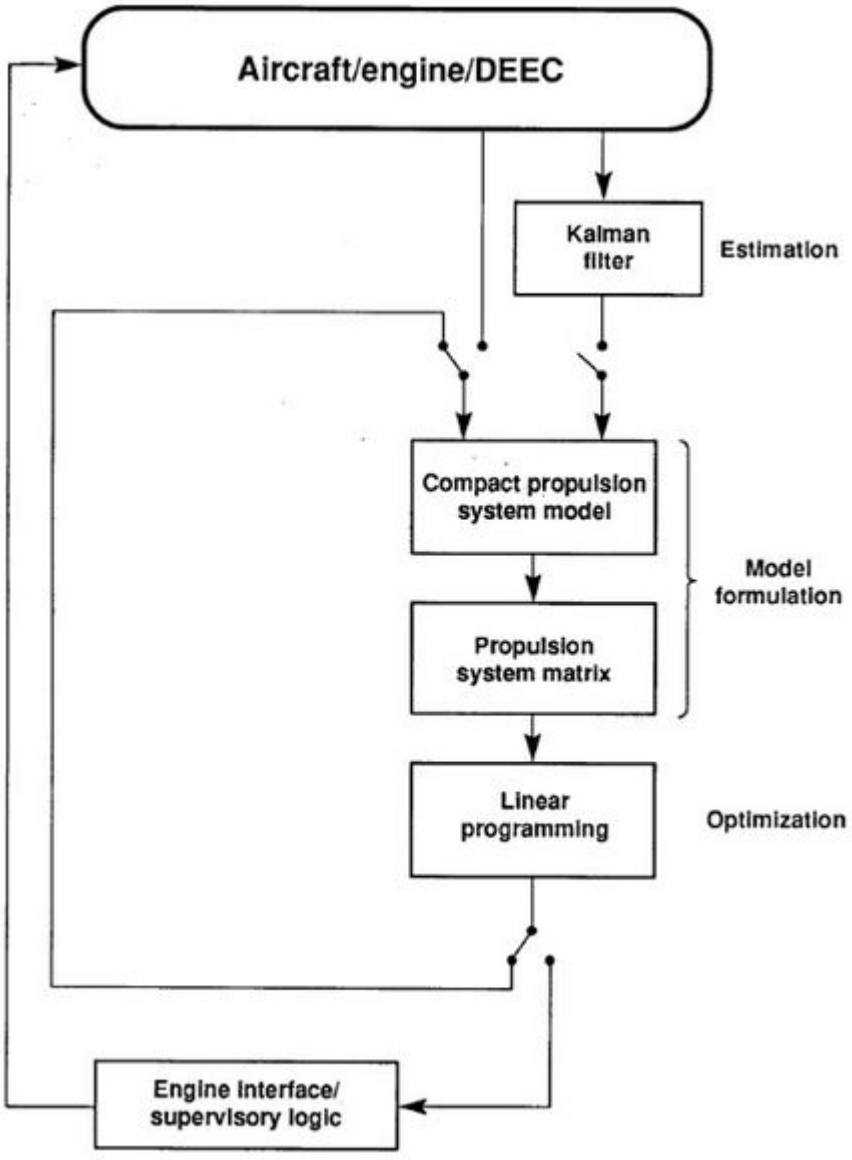
Relation Schematics



Object State Schematics

Chapter 6

Control Flow Diagram



Example of a so called "performance seeking control flow diagram".

A **control flow diagram** (**CFD**) is a diagram to describe the control flow of a business process, process or program.

Control flow diagrams were developed in the 1950s, and are widely used in multiple engineering disciplines. They are one of the classic business process modeling methodologies, along with flow charts, data flow diagrams, functional flow block diagram, Gantt charts, PERT diagrams, and IDEF.

Overview

A control flow diagram can consist of a subdivision to show sequential steps, with if-then-else conditions, repetition, and/or case conditions. Suitably annotated geometrical figures are used to represent operations, data, or equipment, and arrows are used to indicate the sequential flow from one to another.

There are several types of control flow diagrams, for example:

- Change control flow diagram, used in project management
- Configuration decision control flow diagram, used in configuration management
- Process control flow diagram, used in process management
- Quality control flow diagram, used in quality control.

In software and systems development control flow diagrams can be used in control flow analysis, data flow analysis, algorithm analysis, and simulation. Control and data flow analysis are most applicable for real time and data driven systems. These flow analyses transform logic and data requirements text into graphic flows which are easier to analyze than the text. PERT, state transition, and transaction diagrams are examples of control flow diagrams.

Types of Control Flow Diagrams

Process Control Flow Diagram

A flow diagram can be developed for the process control system for each critical activity. Process control is normally a closed cycle in which a sensor provides information to a process control software application through a communications system. The application determines if the sensor information is within the predetermined (or calculated) data parameters and constraints. The results of this comparison are fed to an actuator, which controls the critical component. This feedback may control the component electronically or may indicate the need for a manual action.

This closed-cycle process has many checks and balances to ensure that it stays safe. The investigation of how the process control can be subverted is likely to be extensive because all or part of the process control may be oral instructions to an individual monitoring the process. It may be fully computer controlled and automated, or it may be a hybrid in which only the sensor is automated and the action requires manual

intervention. Further, some process control systems may use prior generations of hardware and software, while others are state of the art.

Performance seeking control flow diagram

The figure presents an example of a performance seeking control flow diagram of the algorithm. The control law consists of estimation, modeling, and optimization processes. In the Kalman filter estimator, the inputs, outputs, and residuals were recorded. At the compact propulsion system modeling stage, all the estimated inlet and engine parameters were recorded.

In addition to temperatures, pressures, and control positions, such estimated parameters as stall margins, thrust, and drag components were recorded. In the optimization phase, the operating condition constraints, optimal solution, and linear programming health status condition codes were recorded. Finally, the actual commands that were sent to the engine through the DEEC were recorded. dfd(data float diagam)is network manen ment system

Chapter 7

Policy Analysis

Policy analysis is "determining which of various alternative policies will most achieve a given set of goals in light of the relations between the policies and the goals". However, policy analysis can be divided into two major fields. Analysis **of** policy is analytical and descriptive—i.e., it attempts to explain policies and their development. Analysis **for** policy is prescriptive—i.e., it is involved with formulating policies and proposals (e.g., to improve social welfare). The area of interest and the purpose of analysis determines what type of analysis is conducted. A combination of policy analysis together with program evaluation would be defined as Policy studies.

Policy Analysis is frequently deployed in the public sector, but is equally applicable to other kinds of organizations. Policy analysis has its roots in systems analysis as instituted by United States Secretary of Defense Robert McNamara during the Vietnam War.

Approaches

Although various approaches to policy analysis exist, three general approaches can be distinguished: the analycentric, the policy process, and the meta-policy approach.

The **analycentric** approach focuses on individual problems and its solutions; its scope is the micro-scale and its problem interpretation is usually of a technical nature. The primary aim is to identify the most effective and efficient solution in technical and economic terms (e.g. the most efficient allocation of resources).

The **policy process** approach puts its focal point onto political processes and involved stakeholders; its scope is the meso-scale and its problem interpretation is usually of a political nature. It aims at determining what processes and means are used and tries to explain the role and influence of stakeholders within the policy process. By changing the relative power and influence of certain groups (e.g., enhancing public participation and consultation), solutions to problems may be identified.

The **meta-policy approach** is a systems and context approach; i.e., its scope is the macro-scale and its problem interpretation is usually of a structural nature. It aims at explaining the contextual factors of the policy process; i.e., what are the political, economic and socio-cultural factors influencing it. As problems may result because of structural factors (e.g., a certain economic system or political institution), solutions may entail changing the structure itself.

Methodology

Policy analysis is methodologically diverse using both qualitative methods and quantitative methods, including case studies, survey research, statistical analysis, and model building among others. One common methodology is to define the problem and evaluation criteria; identify all alternatives; evaluate them; and recommend the best policy agenda per favor.

Models

Many models exist to analyze the creation and application of public policy. Analysts use these models to identify important aspects of policy, as well as explain and predict policy and its consequences.

Some models are:

Institutional model

Public policy is determined by political institutions, which give policy legitimacy. Government universally applies policy to all citizens of society and monopolizes the use of force in applying policy. The legislature, executive and judicial branches of government are examples of institutions that give policy legitimacy.

Process model

Policy creation is a process following these steps:

- Identification of a problem and demand for government action.
- Formulation of policy proposals by various parties (e.g., congressional committees, think tanks, interest groups).
- Selection and enactment of policy; this is known as **Policy Legitimation**.
- Implementation of the chosen policy.
- Evaluation of policy.

This model, however, has been criticized for being overly linear and simplistic. In reality, stages of the policy process may overlap or never happen. Also, this model fails to take the multiple actors attempting the process itself as well as each other, and the complexity this entails.

Rational model

The rational model of decision-making is a process for making logically sound decisions in policy making in the public sector, although the model is also widely used in private corporations. Herbert Simon, the father of rational models, describes rationality as "*a style of behavior that is appropriate to the achievement of given goals, within the limits imposed by given conditions and constraints*". It is important to note the model makes a series of assumptions in order for it to work, such as:

- The model must be applied in a system that is stable,
- The government is a rational and unitary actor and that its actions are perceived as rational choices,
- The policy problem is unambiguous,
- There are no limitations of time or cost.

Indeed, some of the assumptions identified above are also pin pointed out in a study written by the historian H.A. Drake, as he states:

In its purest form, the Rational Actor approach presumes that such a figure [as Constantine] has complete freedom of action to achieve goals that he or she has articulated through a careful process of rational analysis involving full and objective study of all pertinent information and alternatives. At the same time, it presumes that this central actor is so fully in control of the apparatus of government that a decision once made is as good as implemented. There are no staffs on which to rely, no constituencies to placate, no generals or governors to cajole. By attributing all decision making to one central figure who is always fully in control and who acts only after carefully weighing all options, the Rational Actor method allows scholars to filter out extraneous details and focus attention on central issues.

Furthermore, as we have seen, in the context of policy rational models are intended to achieve maximum social gain. For this purpose, Simon identifies an outline of a step by step mode of analysis to achieve rational decisions. Ian Thomas describes Simon's steps as follows:

1. Intelligence gathering— data and potential problems and opportunities are identified, collected and analyzed.
2. Identifying problems
3. Assessing the consequences of all options
4. Relating consequences to values— with all decisions and policies there will be a set of values which will be more relevant (for example, economic feasibility and

environmental protection) and which can be expressed as a set of criteria, against which performance (or consequences) of each option can be judged.
5. Choosing the preferred option— given the full understanding of all the problems and opportunities, all the consequences and the criteria for judging options.

In similar lines, Wiktorowicz and Deber describe through their study on 'Regulating biotechnology: a rational-political model of policy development' the rational approach to policy development. The main steps involved in making a rational decision for these authors are the following:

1. The comprehensive organization and analysis of the information
2. The potential consequences of each option
3. The probability that each potential outcome would materialize
4. The value (or utility) placed on each potential outcome.

The approach of Wiktorowicz and Deber is similar to Simon and they assert that the rational model tends to deal with "the facts" (data, probabilities) in steps 1 to 3, leaving the issue of assessing values to the final step. According Wiktorowicz and Deber values are introduced in the final step of the rational model, where the utility of each policy option is assessed.

Many authors have attempted to interpret the above mentioned steps, amongst others, Patton and Sawicki who summarize the model as presented in the following figure:

File:Interpretation of the rational model of decision making.PNG

1. Defining the problem by analyzing the data and the information gathered.
2. Identifying the decision criteria that will be important in solving the problem. The decision maker must determine the relevant factors to take into account when making the decision.
3. A brief list of the possible alternatives must be generated; these could succeed to resolve the problem.
4. A critical analyses and evaluation of each criterion is brought through. For example strength and weakness tables of each alternative are drawn and used for comparative basis. The decision maker then weights the previously identified criteria in order to give the alternative policies a correct priority in the decision.
5. The decision-maker evaluates each alternative against the criteria and selects the preferred alternative.
6. The policy is brought through.

The model of rational decision-making has also proven to be very useful to several decision making processes in industries outside the public sphere. Nonetheless, many criticism of the model arise due to claim of the model being impractical and lying on unrealistic assumptions. . For instance, it is a difficult model to apply in the public sector

because social problems can be very complex, ill-defined and interdependent. The problem lies in the thinking procedure implied by the model which is linear and can face difficulties in extra ordinary problems or social problems which have no sequences of happenings. This latter argument can be best illustrated by the words of Thomas R. Dye, the president of the Lincoln Center for Public Service, who wrote in his book `Understanding Public Policy´ the following passage:

There is no better illustration of the dilemmas of rational policy making in America than in the field of health…the first obstacle to rationalism is defining the problem. Is our goal to have good health — that is, whether we live at all (infant mortality), how well we live (days lost to sickness), and how long we live (life spans and adult mortality)? Or is our goal to have good medical care — frequent visits to the doctor, wellequipped and accessible hospitals, and equal access to medical care by rich and poor alike?

The problems faced when using the rational model arise in practice because social and environmental values can be difficult to quantify and forge consensus around. Furthermore, the assumptions stated by Simon are never fully valid in a real world context.

However, as Thomas states the rational model provides a good perspective since in modern society rationality plays a central role and everything that is rational tends to be prized. Thus, it does not seem strange that "we ought to be trying for rational decision-making".

Decision Criteria for Policy Analysis — Step 2

As illustrated in Figure 1, rational policy analysis can be broken into 6 distinct stages of analysis. Step 2 highlights the need to understand which factors should be considered as part of the decision making process. At this part of the process, all the economic, social, and environmental factors that are important to the policy decision need to be identified and then expressed as policy decision criteria. For example, the decision criteria used in the analysis of environmental policy is often a mix of —

- Ecological impacts — such as biodiversity, water quality, air quality, habitat quality, species population, etc.
- Economic efficiency — commonly expressed as benefits and costs.
- Distributional equity — how policy impacts are distributed amongst different demographics. Factors that can affect the distribution of impacts include location, ethinicity, income, and occupation.
- Social/Cultural acceptability — the extent to which the policy action may be opposed by current social norms or cultural values.

- Operational practicality — the capacity required to actually operationalize the policy. For example,

- Legality — the potential for the policy to be implemented under current legislation versus the need to pass new legislation that accommodates the policy.

- Uncertainty — the degree to which the level of policy impacts can be known.

Some criteria, such as economic benefit, will be more easily measurable or definable, while others such as environmental quality will be harder to measure or express quantitatively. Ultimately though, the set of decision criteria needs to embody all of the policy goals, and overemphasising the more easily definable or measurable criteria, will have the undesirable impact of biasing the analysis towards a subset of the policy goals.

The process of identifying a suitably comprehensive decision criteria set is also vulnerable to being skewed by pressures arising at the political interface. For example, decision makers may tend to give "*more weight to policy impacts that are concentrated, tangible, certain, and immediate than to impacts that are diffuse, intangible, uncertain, and delayed*."^8. For example, with a cap-and-trade system for carbon emissions the net financial cost in the first five years of policy implementation is a far easier impact to conceptualise than the more diffuse and uncertain impact of a country's improved position to influence global negotiations on climate change action.

Decision Methods for Policy Analysis — Step 5

Displaying the impacts of policy alternatives can be done using a policy analysis matrix (PAM) such that shown in Table 1. As shown, a PAM provides a summary of the policy impacts for the various alternatives and examination of the matrix can reveal the tradeoffs associated with the different alternatives.

Table 1. Policy analysis matrix (PAM) for SO2 emissions control.

Once policy alternatives have been evaluated, the next step is to decide which policy alternative should be implemented. This is shown as step 5 in Figure 1. At one extreme, comparing the policy alternatives can be relatively simple if all the policy goals can be measured using a single metric and given equal weighting. In this case, the decision method is an exercise in benefit cost analysis (BCA).

At the other extreme, the numerous goals will require the policy impacts to be expressed using a variety of metrics that are not readily comparable. In such cases, the policy analyst may draw on the concept of utility to aggregate the various goals into a single score. With the utility concept, each impact is given a weighting such that 1 unit of each weighted impact is considered to be equally valuable (or desirable) with regards to the collective well-being.

Weimer and Vining also suggest that the "*go, no go*" rule can be a useful method for deciding amongst policy alternatives^8. Under this decision making regime, some or all policy impacts can be assigned thresholds which are used to eliminate at least some of the policy alternatives. In their example, one criterion "*is to minimize SO2 emissions*" and so a threshold might be a reduction SO2 emissions "of at least 8.0 million tons per year". As such, any policy alternative that does not meet this threshold can be removed from consideration. If only a single policy alternative satisfies all the impact thresholds then it is the one that is considered a "go" for each impact. Otherwise it might be that all but a few policy alternatives are eliminated and those that remain need to be more closely examined in terms of their trade-offs so that a decision can be made.

Case Study Example of Rational Policy Analysis Approach

To demonstrate the rational analysis process as described above, let's examine the policy paper "Stimulating the use of biofuels in the European Union: Implications for climate change policy" by Lisa Ryan where the substitution of fossil fuels with biofuels has been proposed in the European Union (EU) between 2005–2010 as part of a strategy to mitigate greenhouse gas emissions from road transport, increase security of energy supply and support development of rural communities.

Considering the steps of Patton and Sawicki model as in Figure 1 above, this paper only follows components 1 to 5 of the rationalist policy analysis model:

1. Defining The Problem – the report identifies transportation fuels pose two important challenges for the European Union (EU). First, under the provisions of the Kyoto Protocol to the Climate Change Convention, the EU has agreed to an absolute cap on greenhouse gas emissions; while, at the same time increased consumption of transportation fuels has resulted in a trend of increasing greenhouse gas emissions from this source. Second, the dependence upon oil imports from the politically volatile Middle East generates concern over price fluctuations and possible interruptions in supply. Alternative fuel sources need to be used & substituted in place of fossil fuels to mitigate GHG emissions in the EU.
2. Determine the Evaluation Criteria – this policy sets Environmental impacts/benefits (reduction of GHG's as a measure to reducing climate change effects) and Economical efficiency (the costs of converting to biofuels as alternative to fossil fuels & the costs of production of biofuels from its different potential sources)as its decision criteria. However, this paper does not exactly talk about the social impacts, this policy may have. It also does not compare the operational challenges involved between the different categories of biofuels considered.
3. Identifying Alternative Policies – The European Commission foresees that three alternative transport fuels: hydrogen, natural gas, and biofuels, will replace transport fossil fuels, each by 5% by 2020.
4. Evaluating Alternative Policies – Biofuels are an alternative motor vehicle fuel produced from biological material and are promoted as a transitional step until

more advanced technologies have matured. By modelling the efficiency of the biofuel options the authors compute the economic and environmental costs of each biofuel option as per the evaluation criteria mentioned above.

5. Select The Preferred Policy – The authors suggest that the overall best biofuel comes from the sugarcane in Brazil after comparing the economic & the environmental costs. The current cost of subsidising the price difference between European biofuels and fossil fuels per tonne of CO_2 emissions saved is calculated to be €229–2000. If the production of European biofuels for transport is to be encouraged, exemption from excise duties is the instrument that incurs the least transactions costs, as no separate administrative or collection system needs to be established. A number of entrepreneurs are producing biofuels at the lower margin of the costs specified here profitably, once an excise duty rebate is given. It is likely that growth in the volume of the business will engender both economies of scale and innovation that will reduce costs substantially.

Group model

The political system's role is to establish and enforce compromise between various, conflicting interests in society.

Elite model

Policy is a reflection of the interests of those individuals within a society that have the most power, rather than the demands of the mass.

Six-step model

1. Verify, define and detail the problem
2. Establish evaluation criteria
3. Identify alternative policies
4. Evaluate alternative policies
5. Display and distinguish among alternative policies
6. Monitor the implemented policy

Chapter 8

Problem Frames Approach

Problem Analysis or the **Problem Frames Approach** is an approach to software requirements analysis. It was developed by British software consultant Michael A. Jackson in the 1990s.

History

The Problem Frames Approach was first sketched by Jackson in his book *Software Requirements & Specifications* (1995) and in a number of articles in various journals devoted to software engineering. It has received its fullest description in his *Problem Frames: Analysing and Structuring Software Development Problems* (2001).

A session on problem frames was part of the 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ)] held in Klagenfurt/Velden, Austria in 2003. The First International Workshop on Applications and Advances in Problem Frames was held as part of ICSE'04 held in Edinburgh, Scotland. One outcome of that workshop was a 2005 special issue on problem frames in the *International Journal of Information and Software Technology*.

The Second International Workshop on Applications and Advances in Problem Frames was held as part of ICSE 2006 in Shanghai, China. The Third International Workshop on Applications and Advances in Problem Frames (IWAAPF) was held as part of ICSE 2008 in Leipzig, Germany. In 2010, the IWAAPF workshops were replaced by the International Workshop on Applications and Advances of Problem-Orientation (IWAAPO). IWAAPO broadens the focus of the workshops to include alternative and complementary approaches to software development that share an emphasis on problem analysis. IWAAPO-2010 was held as part of ICSE 2010 in Cape Town, South Africa.

Today research on the Problem Frames Approach is being conducted at a number of universities, most notably at the Open University in the United Kingdom as part of its

Relating Problem & Solution Structures research theme Interesting work is being done by Professors Lavazza and del Bianco on using problem frames with UML.

The ideas in the Problem Frames Approach have been generalized into the concepts of Problem-Oriented Development (POD) and Problem-Oriented Engineering (POE), of which Problem-Oriented Software Engineering (POSE) is a particular sub-category. The first International Workshop on Problem-Oriented Development was held in June 2009.

Overview

Fundamental philosophy

Problem analysis or the *Problem Frames approach* is an approach — a set of concepts — to be used when gathering requirements and creating specifications for computer software. Its basic philosophy is strikingly different from other software requirements methods in insisting that:

- The best way to approach requirements analysis is through a process of parallel — not hierarchical — decomposition of user requirements.
- User requirements are about relationships in the real world—the *application domain* -- not about the software system or even the interface with the software system.

It is more helpful ... to recognize that the solution is located in the computer and its software, and the problem is in the world outside. ... The computers can provide solutions to these problems because they are connected to the world outside.

The moral is clear: to study and analyse a problem you must focus on studying and analysing the problem world in some depth, and in your investigations you must be willing to travel some distance away from the computer. ... *[In a call forwarding problem...]* You need to describe what's there -- people and offices and holidays and moving office and delegating responsibility -- and what effects *[in the problem world]* you would like the system to achieve -- calls to A's number must reach A, and *[when B is on vacation, and C is temporarily working at D's desk]* calls to B's or C's number must reach C.

None of these appear in the interface with the computer.... They are all deeper into the world than that.

The approach uses three sets of conceptual tools.

Tools for describing specific problems

Concepts used for describing specific problems include: *phenomena* (of various kinds, including *events*), *problem context*, *problem domain*, *solution domain* (aka the *machine*), *shared phenomena* (which exist in *domain interfaces*), *domain requirements* (which exist

in the problem domains) and *specifications* (which exist at the problem domain:machine interface).

The graphical tools for describing problems are the *context diagram* and the *problem diagram*.

Tools for describing classes of problems (problem frames)

The Problem Frames Approach includes concepts for describing classes of problems. A recognized class of problems is called a *problem frame* (roughly analogous to a **design pattern**).

In a problem frame, domains are given general names and described in terms of their important characteristics. A domain, for example, may be classified as *causal* (reacts in a deterministic, predictable way to events) or *biddable* (can be bid, or asked, to respond to events, but cannot be expected always to react to events in any predictable, deterministic way). (A biddable domain usually consists of people.)

The graphical tool for representing a problem frame is a *frame diagram*. A frame diagram looks generally like a problem diagram except for a few minor differences—domains have general, rather than specific, names; and rectangles representing domains are annotated to indicate the type (causal or biddable) of the domain.

A list of recognized classes of problems (problem frames)

The first group of problem frames identified by Jackson included:

1. required behavior
2. commanded behavior
3. information display
4. simple workpieces
5. transformation

Subsequently, other researchers have described or proposed additional problem frames.

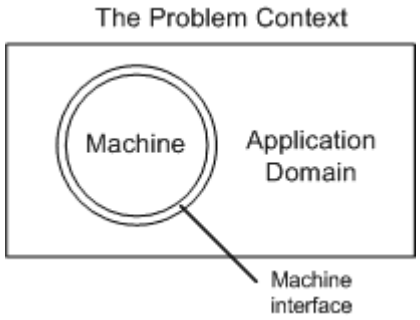
Describing Problems

The Problem Context

Problem analysis considers a software application to be a kind of software *Machine*. A software development project aims to change the problem context by creating a software Machine and adding it to the problem context, where it will bring about certain desired effects.

The particular portion of the problem context that is of interest in connection with a particular problem — the particular portion of the problem context that forms the context of the problem — is called the *application domain*.

After the software development project has been finished, and the software Machine has been inserted into the problem context, the problem context will contain both the application domain and the Machine. At that point, the situation will look like this:



The problem context contains the Machine and the application domain. The *machine interface* is where the Machine and the application domain meet and interact.

The same situation can be shown in a different kind of diagram, a *context diagram*, this way:

The Context Diagram

The problem analyst's first task is to truly understand the problem. That means understanding the context in which the problem is set. And that means drawing a *context diagram.*

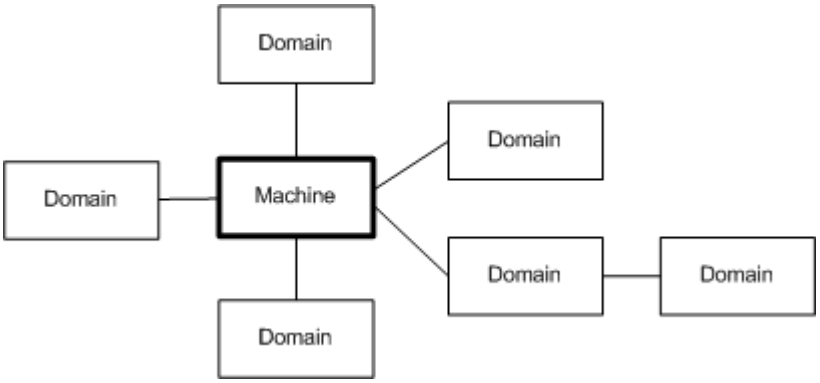
Here is Jackson's description of examining the problem context, in this case the context for a bridge to be built:

You're an engineer planning to build a bridge across a river. So you visit the site. Standing on one bank of the river, you look at the surrounding land, and at the river traffic. You feel how exposed the place is, and how hard the wind is blowing and how fast the river is running. You look at the bank and wonder what faults a geological survey will show up in the rocky terrain. You picture to yourself the bridge that you are going to build. (*Software Requirements & Specifications*: "The Problem Context")

An analyst trying to understand a software development problem must go through the same process as the bridge engineer. He starts by examining the various problem domains in the application domain. These domains form the context into which the planned

Machine must fit. Then he imagines how the Machine will fit into this context. And then he constructs a context diagram showing his vision of the problem context with the Machine installed in it.

The context diagram shows the various *problem domains* in the application domain, their connections, and the Machine and its connections to (some of) the problem domains. Here is what a context diagram looks like.



This diagram shows:

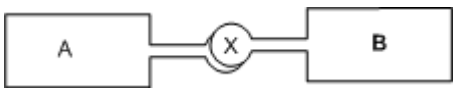
- the *Machine* to be built. The dark border helps to identify the box that represents the Machine.
- the *problem domains* that are relevant to the problem.
- the solid lines represent *domain interfaces* — areas where domains overlap and share phenomena in common.

A domain is simply a part of the world that we are interested in. It consists of *phenomena* — individuals, events, states of affairs, relationships, and behaviors.

A domain interface is an area where domains connect and communicate. Domain interfaces are not data flows or messages. An interface is a place where domains partially overlap, so that the phenomena in the interface are *shared phenomena* — they exist in both of the overlapping domains.

You can imagine domains as being like primitive one-celled organisms (like amoebas). They are able to extend parts of themselves into pseudopods. Imagine that two such organisms extend pseudopods toward each other in a sort of handshake, and that the cellular material in the area where they are shaking hands is mixing, so that it belongs to both of them. That's an interface.

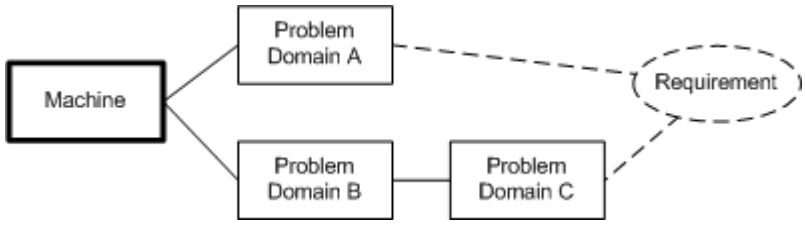
In the following diagram, X is the interface between domains A and B. Individuals that exist or events that occur in X, exist or occur in both A and B.



Shared individuals, states and events may look differently to the domains that share them. Consider for example an interface between a computer and a keyboard. When the keyboard domain sees an event *Keyboard operator presses the spacebar* the computer will see the same event as *Byte hex("20") appears in the input buffer.*

Problem Diagrams

The problem analyst's basic tool for describing a problem is a *problem diagram*. Here is a generic problem diagram.

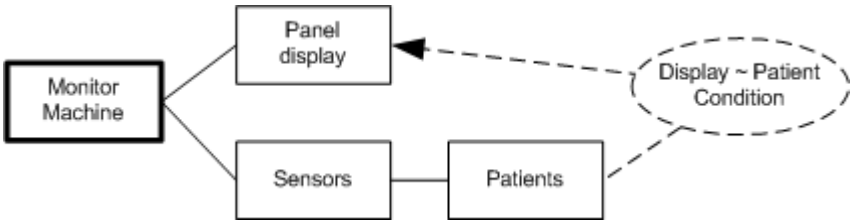


In addition to the kinds of things shown on a context diagram, a problem diagram shows:

- a dotted oval representing the *requirement* to bring about certain effects in the problem domains.
- dotted lines representing *requirement references* — references in the requirement to phenomena in the problem domains.

An interface that connects a problem domain to the Machine is called a *specification interface* and the phenomena in the specification interface are called *specification phenomena*. The goal of the requirements analyst is to develop a specification for the behavior that the Machine must exhibit at the Machine interface in order to satisfy the requirement.

Here is an example of a real, if simple, problem diagram.



This problem might be part of a computer system in a hospital. In the hospital, patients are connected to sensors that can detect and measure their temperature and blood pressure. The requirement is to construct a Machine that can display information about patient conditions on a panel in the nurses station.

The name of the requirement is "Display ~ Patient Condition". The tilde (~) indicates that the requirement is about a relationship or correspondence between the panel display and patient conditions. The arrowhead indicates that the requirement reference connected to

the Panel Display domain is also a requirement constraint. That means that the requirement contains some kind of stipulation that the Panel display must meet. In short, the requirement is that *The panel display must display information that matches and accurately reports the condition of the patients.*

Describing Classes of Problems

Problem Frames

A *problem frame* is a description of a recognizable class of problems, where the class of problems has a known solution. In a sense, problem frames are problem patterns.

Each problem frame has its own *frame diagram*. A frame diagram looks essentially like a problem diagram, but instead of showing specific domains and requirements, it shows types of domains and types of requirements; domains have general, rather than specific, names; and rectangles representing domains are annotated to indicate the type (causal or biddable) of the domain.

Variant frames

In *Problem Frames* Jackson discussed variants of the five basic problem frames that he had identified. A variant typically adds a domain to the problem context.

- a *description variant* introduces a description lexical domain
- an *operator variant* introduces an operator
- a *connection variant* introduces a connection domain between the machine and the central domain with which it interfaces
- a *control variant* introduces no new domain; it changes the control characteristics of interface phenomena

Problem Concerns

Jackson also discusses certain kinds of concerns that arise when working with problem frames.

Particular Concerns

- overrun
- initialization
- reliability
- identities
- completeness

Composition Concerns

- commensurable descriptions

- consistency
- precedence
- interference
- synchronization

Recognized Problem Frames

The first problem frames identified by Jackson included:

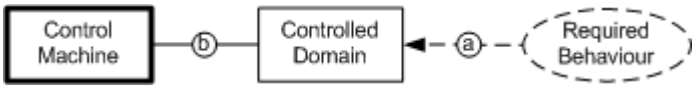
1. required behavior
2. commanded behavior
3. information display
4. simple workpieces
5. transformation

Subsequently, other researchers have described or proposed additional problem frames.

Required Behavior Problem Frame

The intuitive idea behind this problem frame is:

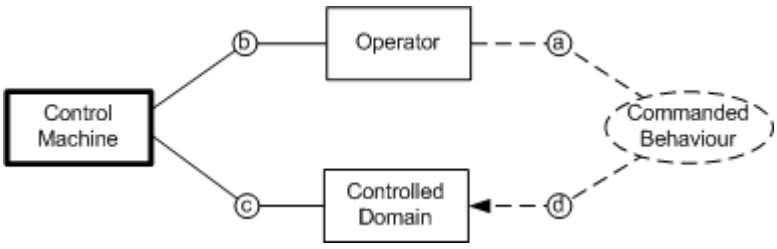
- *There is some part of the physical world whose behavior is to be controlled so that it satisfies certain conditions. The problem is to build a machine that will impose that control.*



Commanded Behavior Problem Frame

The intuitive idea behind this problem frame is:

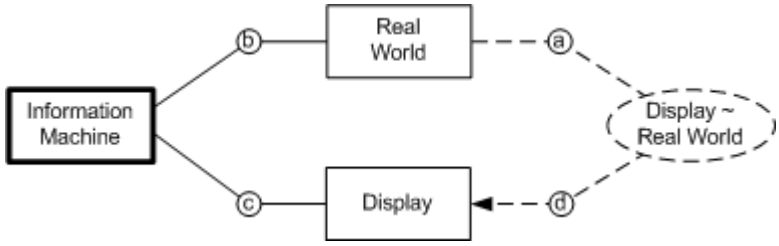
- *There is some part of the physical world whose behavior is to be controlled in accordance with commands issued by an operator. The problem is to build a machine that will accept the operator's commands and impose the control accordingly.*



Information Display Problem Frame

The intuitive idea behind this problem frame is:

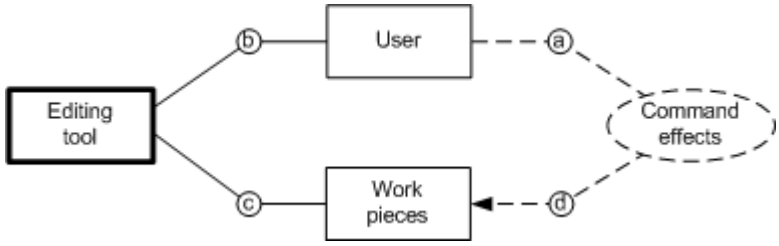
- *There is some part of the physical world about whose states and behavior certain information is continually needed. The problem is to build a machine that will obtain this information from the world and present it at the required place in the required form.*



Simple Workpieces Problem Frame

The intuitive idea behind this problem frame is:

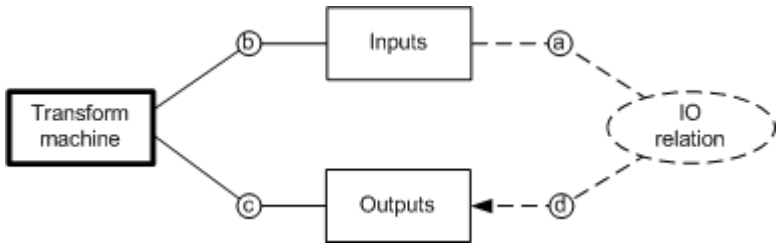
- *A tool is needed to allow a user to create and edit a certain class of computer-processible text or graphic objects, or similar structures, so that they can be subsequently copied, printed, analyzed or used in other ways. The problem is to build a machine that can act as this tool.*



Transformation Problem Frame

The intuitive idea behind this problem frame is:

- *There are some given computer-readable input files whose data must be transformed to give certain required output files. The output data must be in a particular format, and it must be derived from the input data according to certain rules. The problem is to build a machine that will produce the required outputs from the inputs.*



Problem Analysis and the Software Development Process

When problem analysis is incorporated into the software development process, the software development lifecycle starts with the problem analyst, who studies the situation and:

- creates a context diagram
- gathers a list of requirements and adds a requirements oval to the context diagram, creating a grand "all-in-one" problem diagram. (However, in many cases actually creating an all-in-one problem diagram may be impractical or unhelpful: there will be too many requirements references criss-crossing the diagram to make it very useful.)
- decomposes the all-in-one problem and problem diagram into simpler problems and simpler problem diagrams. These problems are *projections*, not subsets, of the all-in-one diagram.
- continues to decompose problems until each problem is simple enough that it can be seen to be an instance of a recognized problem frame. Each subproblem description includes a description of the specification interfaces for the machine to be built.

At this point, problem analysis — *problem decomposition* — is complete. The next step is to reverse the process and to build the desired software system though a process of *solution composition*.

The solution composition process is not yet well-understood, and is still very much a research topic. Extrapolating from hints in *Software Requirements & Specifications*, we can guess that the software development process would continue with the developers, who would:

- compose the multiple subproblem machine specifications into the specification for a single all-in-one machine: a specification for a software machine that satisfies all of the customer's requirements. This is a non-trivial activity — the composition process may very well raise *composition problems* that need to be solved.
- implement the all-in-one machine by going through the traditional code/test/deploy process.

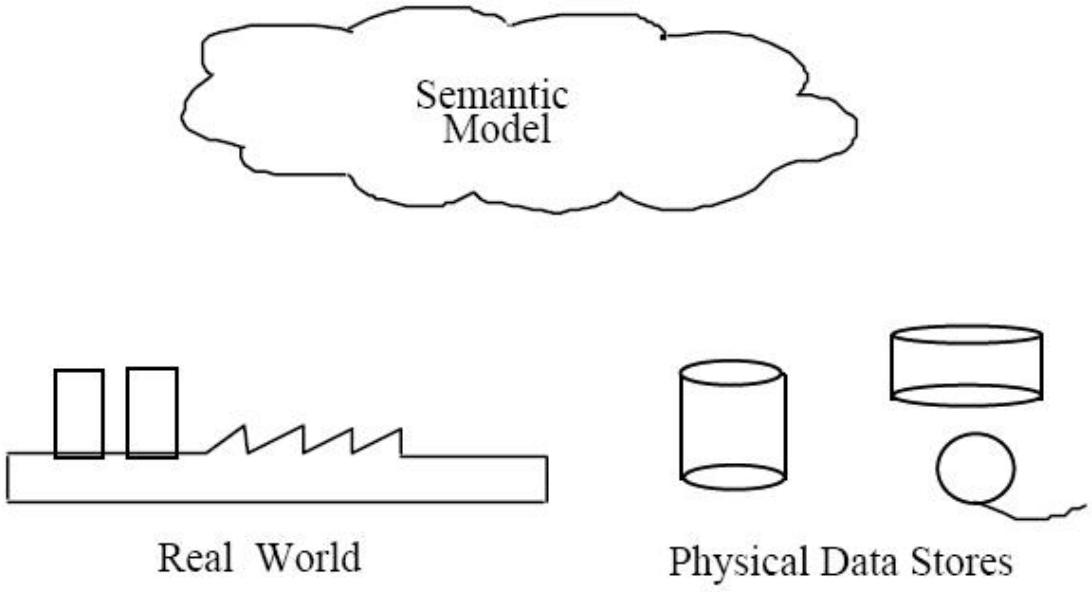
Similar Approaches

There are a few other software development ideas that are similar in some ways to problem analysis.

- The notion of a design pattern is similar to Jackson's notion of a problem frame. It differs in that a design pattern is used for recognizing and handling design issues (often design issues in specific object-oriented programming languages such as C++ or Java) rather than for recognizing and handling requirements issues. Furthermore one difference is that design patterns cover *solutions* while in problem frames *problems* are represented.
- *Aspect-Oriented Programming*, AOP (also known as Aspect-Oriented Software Development, AOSD) is similarly interested in parallel decomposition, which addresses what AOP proponents call *cross-cutting concerns* or *aspects*. AOP addresses concerns that are much closer to the design and code-generation phase than to the requirements analysis phase.
- Martin Fowler's book **Analysis Patterns** is very similar to problem analysis in its search for patterns. It doesn't really present a new requirements analysis method, however. And the notion of parallel decomposition — which is so important for problem analysis — is not a part of Fowler's analysis patterns.
- Jon G. Hall, Lucia Rapanotti, together with Jackson, have developed the Problem Oriented Software Engineering (POSE) framework which shares the Problem Frames foundations. Since, 2005, Hall and Rapanotti have extended POSE to Problem Oriented Engineering (POE), which provides a framework for engineering design, including a development process model and assurance-driven design, and may be scalable to projects that include many stake-holders and that combine diverse engineering disciplines such as software and education provision.

Chapter 9

Semantic Data Model



Semantic data models.

A **semantic data model** in software engineering has various meanings:

1. It is a conceptual data model in which semantic information is included. This means that the model describes the meaning of its instances. Such a semantic data model is an abstraction that defines how the stored symbols (the instance data) relate to the real world.
2. It is a conceptual data model that includes the capability to express information that enables parties to the information exchange to interpret meaning (semantics) from the instances, without the need to know the meta-model. Such semantic models are fact oriented (as opposed to object oriented). Facts are typically expressed by binary relations between data elements, whereas higher order relations are expressed as collections of binary relations. Typically binary

relations have the form of triples: Object-RelationType-Object. For example: the Eiffel Tower <is located in> Paris.

Typically the instance data of semantic data models explicitly include the kinds of relationships between the various data elements, such as <is located in>. To interpret the meaning of the facts from the instances it is required that the meaning of the kinds of relations (relation types) is known. Therefore, semantic data models typically standardise such relation types. This means that the second kind of semantic data models enable that the instances express facts that include their own meaning. The second kind of semantic data models are usually meant to create semantic databases. The ability to include meaning in semantic databases facilitates building distributed databases that enable applications to interpret the meaning from the content. This implies that semantic databases can be integrated when they use the same (standard) relation types. This also implies that in general they have a wider applicability than relational or object oriented databases.

Overview

The logical data structure of a database management system (DBMS), whether hierarchical, network, or relational, cannot totally satisfy the requirements for a conceptual definition of data, because it is limited in scope and biased toward the implementation strategy employed by the DBMS. Therefore, the need to define data from a conceptual view has led to the development of semantic data modeling techniques. That is, techniques to define the meaning of data within the context of its interrelationships with other data. As illustrated in the figure. The real world, in terms of resources, ideas, events, etc., are symbolically defined within physical data stores. A semantic data model is an abstraction which defines how the stored symbols relate to the real world. Thus, the model must be a true representation of the real world.

According to Klas and Schrefl (1995), the "overall goal of semantic data models is to capture more meaning of data by integrating relational concepts with more powerful abstraction concepts known from the Artificial Intelligence field. The idea is to provide high level modeling primitives as integral part of a data model in order to facilitate the representation of real world situations".

History

The need for semantic data models was first recognized by the U.S. Air Force in the mid-1970s as a result of the Integrated Computer-Aided Manufacturing (ICAM) Program. The objective of this program was to increase manufacturing productivity through the systematic application of computer technology. The ICAM Program identified a need for better analysis and communication techniques for people involved in improving manufacturing productivity. As a result, the ICAM Program developed a series of techniques known as the IDEF (ICAM Definition) Methods which included the following:

- IDEF0 used to produce a “function model” which is a structured representation of the activities or processes within the environment or system.
- IDEF1 used to produce an “information model” which represents the structure and semantics of information within the environment or system.
    - IDEF1X is a semantic data modeling technique. It is used to produce a graphical information model which represents the structure and semantics of information within an environment or system. Use of this standard permits the construction of semantic data models which may serve to support the management of data as a resource, the integration of information systems, and the building of computer databases.
- IDEF2 used to produce a “dynamics model” which represents the time varying behavioral characteristics of the environment or system.

During the 1990s the application of semantic modelling techniques resulted in the semantic data models of the second kind. An example of such is the semantic data model that is standardised as ISO 15926-2 (2002), which is further developed into the semantic modelling language Gellish (2005). The definition of the Gellish language is documented in the form of a semantic data model. Gellish itself is a semantic modelling language, that can be used to create other semantic models. Those semantic models can be stored in Gellish Databases, being semantic databases.

Applications

A semantic data model can be used to serve many purposes. Some key objectives include:

- Planning of Data Resources: A preliminary data model can be used to provide an overall view of the data required to run an enterprise. The model can then be analyzed to identify and scope projects to build shared data resources.
- Building of Shareable Databases: A fully developed model can be used to define an application independent view of data which can be validated by users and then transformed into a physical database design for any of the various DBMS technologies. In addition to generating databases which are consistent and shareable, development costs can be drastically reduced through data modeling.
- Evaluation of Vendor Software: Since a data model actually represents the infrastructure of an organization, vendor software can be evaluated against a company’s data model in order to identify possible inconsistencies between the infrastructure implied by the software and the way the company actually does business.
- Integration of Existing Databases: By defining the contents of existing databases with semantic data models, an integrated data definition can be derived. With the proper technology, the resulting conceptual schema can be used to control transaction processing in a distributed database environment. The U.S. Air Force Integrated Information Support System (I2S2) is an experimental development and demonstration of this type of technology applied to a heterogeneous DBMS environment.