# BP*win*®

## Methods Guide

# Contents

## Chapter 1: Introducing Activity and Process Modeling

## Chapter 2: The IDEF3 Process Description Capture Method

# Chapter 3: The IDEF0 Function Modeling Method

# Chapter 4: Data Flow Diagramming

# Chapter 5: Activity-Based Cost and Performance Metrics

# Chapter 6: Integrating Activity and Data Models

# Glossary

# Index

# Introducing Activity and Process Modeling

## About the BPwin Methods Guide

The BPwin Methods Guide provides a brief introduction to activity and process modeling as well as an introduction to the three activity modeling techniques supported by BPwin—the IDEF3 process description capture method, the IDEF0 function modeling method, and data flow diagramming (DFD).

Activity modeling helps us understand the relationships among important activities within a system. The activity modeling techniques described in this book all use a simple box and arrow syntax. With minimal training, business professionals can learn to read and critique activity models.

Activity models, like any form of communication, take patience and skill to create. Experience is invaluable, and early activity analysis projects and expectations should be modest. However, the result of persistence is a more effective way for you and your colleagues to communicate about something that is vital, yet intangible—action!

## Intended Audience

This book is primarily intended for those responsible for developing programs and activities to support business strategy, including those who develop computer systems to automate these activities, and for those responsible for the successful performance of these activities.

# Activity and Process Models

Your company, like *every* company, transforms raw materials into finished goods through a complicated mass of interrelated business functions or *activities* and business *processes*. To a large extent, your company's success or failure depends on your ability to identify, design and execute appropriate activities better than your competition can. Therefore, activities are at the heart of your organization.

Textual descriptions of activities and processes are usually long and complicated, making them difficult to understand. But without a clear understanding of your company's activities, you can't effectively design new activities, or understand existing ones. What is needed is a technique to document activities in a precise format, clarifying and organizing important information while eliminating the superfluous. In this way, activities can be effectively analyzed, designed, and implemented.

Activity and process models (or just activity models) filter and organize information through their syntax and by a rigorous model development process. Activity models can be viewed as an extended form of punctuation that combines with a well-defined publishing format to filter and organize conversational language.

An activity model looks at a *system* as a collection of activities in which each activity transforms some object or collection of objects. Activity models emphasize activities by representing them with special graphic shapes, such as a box. Activities are usually labeled with a verb or a verb and a direct object that represents what the activity accomplishes. An example activity is shown in the following figure.

Activity Box Border

Run Video Store

Activity Name
(Verb + Direct Object)
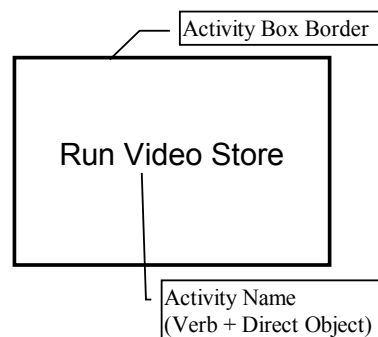
Figure 1-1. An activity is represented by a labeled box.

The interface between an activity and its environment, including other activities, is graphically depicted using arrows. The concept of activity remains relatively the same among the various modeling techniques we will discuss, but the meaning of the arrows, and the meaning of their connections with the boxes, changes from one modeling technique to another.
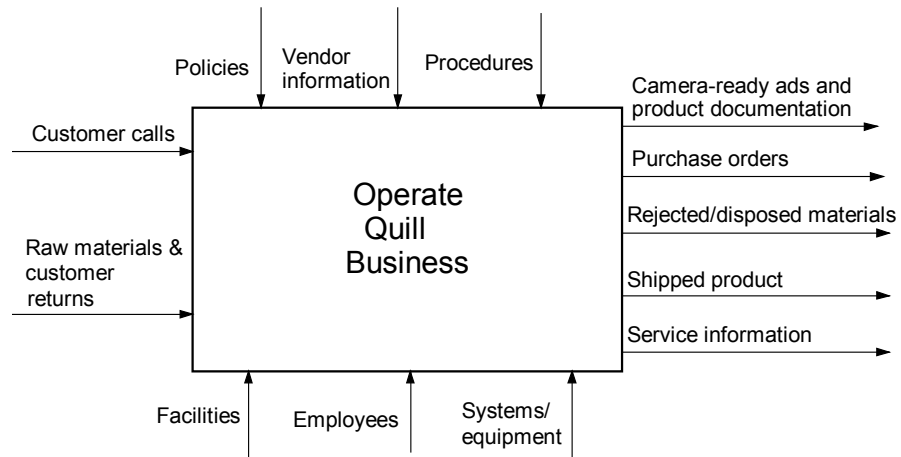
Figure 1-2. Arrows represent the activity's interface with its environment.

Like slides used for presentations, activity model diagrams control the level of detail being exposed at once. The goal of subdividing a model into diagrams is to gradually expose the details of a subject, striking a balance between *clarity*—ensuring that a given diagram can be understood by its intended audience—and *scope*—ensuring that a given diagram contains all relevant information.

The activity box shown in the previous figure represents the system's border, or boundary. If we take a closer look at this box—actually look *inside* it—we would see other boxes. These boxes, in turn, may have their own *child* boxes. The following figure represents the *hierarchical* relationship of an activity in a graphical display format.
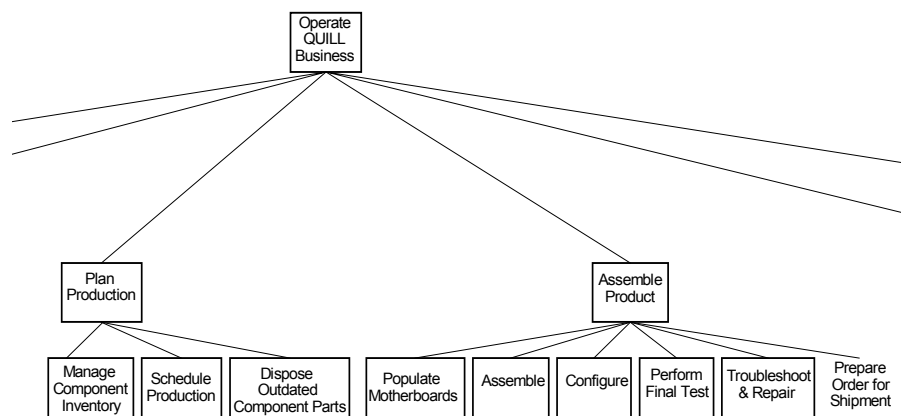


Figure 1-3. Part of an activity hierarchy in a graphical format.

A more abbreviated hierarchy display format is illustrated in the following table.

| Activity Hierarchy |
|---|
| Operate Quill Business |
|     Plan Production |
|         Manage Component Inventory |
|         Schedule Production |
|         Dispose of Outdated Component Parts |
|     Assemble Product |
|         Populate Motherboards |
|         Assemble |
|         Configure |
|         Perform Final Test |
|         Troubleshoot & Repair |
|         Prepare Order for Shipment |

Table 1-1. A partial activity hierarchy in outline form.

# Activity Modeling Techniques

This book describes three activity modeling techniques—called the IDEF0 function modeling method, the IDEF3 process description capture method, and data flow diagramming (DFD).

IDEF0, originally called the structured analysis and design technique (SADT®), was developed by SofTech, Inc., in the late 1960s as an engineering discipline for the development of relatively complex systems of machines and people. The U.S. Air Force adopted a large subset of SADT (naming this subset IDEF0 in the process) as part of its Integrated Computer-Aided Manufacturing (ICAM) program during the late 1970s, and the technique quickly became the U.S. Department of Defense's (DoD) standard activity modeling technique.

In 1993, the IDEF Users Group (now the Society of Enterprise Engineering, or SEE), in cooperation with the National Institutes of Standards and Technology (NIST), undertook an effort to create a documented standard for IDEF0 to be used by all civilian and military branches of the U.S. government and its allies. This standard is published as Federal Information Processing Standards (FiPS) Publication 183.

Somewhat independently, but using many of the same principles, DFD techniques became popular for the structured design (and later, structured analysis) of software development projects. Data flow diagrams are similar to IDEF0 models in many respects and can be used for detailed design modeling as a follow-up to the development of IDEF0 analysis models.

IDEF3 was developed specifically for a project sponsored by the U.S. Air Force's Armstrong Laboratories. It is a technique for capturing process descriptions from subject experts and for designing process models where it is important to understand activity sequence and concurrency. Although IDEF3 has not achieved the same status as FiPS (and no known FiPS effort for IDEF3 exists), the technique has gained wide acceptance within the DoD for process modeling as an adjunct to IDEF0 activity modeling.

## Structured Analysis and Design

Structured analysis and design is the name given to a class of somewhat formal methods (processes) used to analyze and design systems. Although it is possible to develop activity models outside of a structured analysis project, it is important to understand the context in which activity models are normally built. In addition to activity models, structured analysis includes information models, which model data, and state-transition diagrams (STD), which model the time-dependent behavior of a system.

A basic tenet of structured analysis and design is that a thorough analysis of the current system is necessary to design an optimum solution. A significant amount of data is collected and analyzed as part of the process. Traditionally, information is gathered by an analyst formally interviewing *domain experts*, people who are knowledgeable about the system of interest. Over time, some domain experts have learned to build activity models, and new activity modeling techniques (such as IDEF3) have been developed to facilitate the capture of information from domain experts. However, the analyst's central role in a structured analysis and design project remains largely intact.

Often, models are first developed to document the current situation (known as AS-IS models). These models are later used to aid in the development of the design models (known as TO-BE models). They are also used as the benchmark against which the TO-BE models are compared, to ensure that the proposed design is really an improvement.

In addition to guiding the design of the proposed system, TO-BE models are used for capacity planning, budgeting, and resource acquisition. The models are also used to derive the project implementation plan, which specifies how the system is transformed from the AS-IS situation to the TO-BE situation.

The benefits of developing AS-IS models need to be weighed against the cost and time to develop them. Business literature is replete with examples of systems being built that solve the wrong problem, an issue that AS-IS modeling can help clarify. On the other hand, if the problem is generally well understood (as is often the case when building computer systems), then the cost-effectiveness of AS-IS modeling is less apparent.

# Activity Modeling and Business Systems Engineering

Activity modeling is a critical component of business systems engineering because a company's strategy is implemented by activities. It is the success or failure of a company to perform these activities that ultimately determines success or failure of a particular strategy.

Corporate strategy can be thought of as a compass, steering the company through turbulent markets with a long-term broad view of the company's intentions for where it wants to be. Strategy is implemented by activities, so activity models serve as a company's road map, helping the company navigate the local terrain to get where it wants to be.

Briefly, corporate strategy covers four general areas:

- Market segments the company chooses to serve.
- Products and services the company will offer to these market segments.
- Distribution and marketing channels the company will use to reach these market segments.
- Activities and processes the company will use to execute the strategy.

Deciding which market segments the company intends to serve is the most significant decision, because it focuses and constrains the other issues. The final list of exact markets will certainly evolve during strategy development, but significant changes in the target market can invalidate other important decisions.

The general strategy for serving these markets has three dimensions: cost leadership, differentiated value, and market scope. A company can strive to be the cost leader in the target segment, offering the lowest price and making a profit through manufacturing efficiencies. A second approach is to completely ignore costs and offer the highest value to the target market. It is also possible to balance the two approaches. A company can also exclusively focus on serving a small niche market, target several niches with focused strategies, or cover a broad market with a common strategy.

If a company cannot reach a target market directly, then it develops channels, or intermediaries, that can better reach the target segment, for example, by being geographically closer. Different kinds of resellers are usually required to reach different market segments. Companies also use various channels to communicate their marketing messages.

The company must identify and design the activities that will be used to fulfill the strategy. Activity models play an integral role in this crucial step of strategy development. Activities must be designed to meet the company's objectives, and costs must be weighed against perceived value. Often, discoveries made while designing activities and processes, such as the activity's costs, may result in changes to other strategic decisions.

For example, even though customers in a particular target segment may need extensive training to use a particular product, the customers may not be willing to pay enough for the company to justify the expense of developing an appropriate training program. This can have a tremendous impact on the company's strategy for that market segment.

Strategy development and implementation are themselves processes that can be analyzed and designed. The more efficiently a company can revitalize its strategy to capitalize on new opportunities and counter new competitive threats, the greater its chance of sustaining a long-term competitive advantage.

If a company neglects to continuously reexamine and revitalize its competitive strategy, it will one day find itself at a competitive disadvantage. Such was the case for many American companies in the late 1980s, when management gurus Michael Hammer and James Champy, authors of the book *Reengineering the Corporation*, began to expound on the need for companies to reinvent themselves, to discard current business practices and develop new ones. Business Process Reengineering is a strategy development and implementation process in which the existing strategy is discarded, and activity modeling can be utilized as described in the previous paragraphs.

The increasing reliance on computing technology to support business activities has also increased the need to communicate information about activities among an increasingly diverse audience of business and computer systems professionals. The failure of the computer systems profession to consistently deliver correctly working software on time and within budget (commonly referred to as the *software crisis*) has been largely due to a combination of four root causes:

- Failure to meet stated system requirements.

- Inadequate or incorrect system design.

- Inadequate system performance.

- Failure to properly address the human/system interface.

A thorough understanding of the activities and processes the computer system is expected to support is crucial to successful software development. As computer systems continue to permeate society, the cost of such failures will increase. It is no wonder that a top priority on the agenda of company executives is to better align information technology with business needs.

# The IDEF3 Process Description Capture Method

## Overview of IDEF3

IDEF3 is a process description capture method whose primary goal is to provide a structured method by which a domain expert can describe a situation as an ordered sequence of events, as well as describe any participating objects.

IDEF3 is a technique well-suited to collecting data as part of structured analysis and design. Unlike some process modeling techniques, IDEF3 does not discourage the capture of incomplete or inconsistent system descriptions through rigid syntax or semantics. In addition, the model author (analyst) is not forced to intersperse his or her own assumptions with the domain expert's in the interest of filling gaps in the process descriptions. Figure 2-1 is an example of an IDEF3 process description.
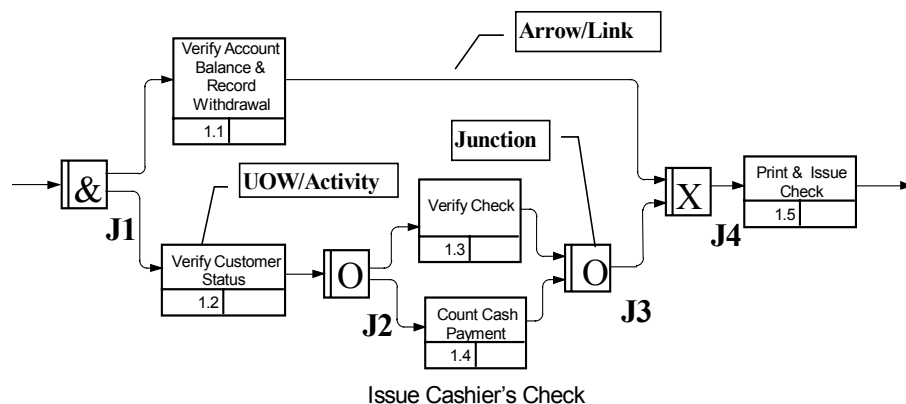


Figure 2-1. An IDEF3 process description.

IDEF3 can also be used as a process design method. IDEF3 modeling complements IDEF0 modeling and has gained increasing acceptance as a viable way to build design models that will be further analyzed via simulation. Simulation is commonly used to judge the performance of a system currently under design and is discussed in Chapter 5.

# IDEF3 Model Syntax and Semantics

## IDEF3 Models

The business scenariois the basic organizing structure of an IDEF3 model, which outlines a sequence of activities or process within a given setting. Because a scenario describes the purpose and scope of a model, it is important that the main activity be appropriately named with a verb, gerund (verb ending in *ing*), or verb + direct object, known as a *verb phrase*. For example, Process Customer Order, Implement New Design, and Develop Customer Response Profile are all examples of scenario names.

The viewpoint for the model should be documented. Normally, this will be the role or job title of the person providing the description.

It is also important to understand the model's purpose—what questions the model is trying to answer; scope—what is included in and excluded from the model; and audience—for whom this model is built.

A description of the objects, syntax, and semantics that define the business process in an IDEF3 model follows.

## Diagram

As in every activity modeling technique described in this book, the diagram is the basic organizing unit of an IDEF3 model. The organization of the diagrams in an IDEF3 model is more important if the model is to be published or read by others, as would be the case for most design models. In this case, the modeler would need to take care when determining what information is included in a particular diagram, in order to ensure that the diagram is comprehensive, and clear to the reader.

## Unit of Work (UOW)/Activity

As in all activity modeling techniques, the activity, also called unit of work (UOW), is the central component of the model. An IDEF3 diagram depicts an activity as a box with square corners. In an IDEF3 diagram, the activity is identified with a verb or verb phrase (verb plus direct object), and a unique number identifier.

The noun part of the verb phrase, that is, the direct object of the verb, usually describes the major input to the activity (such as Gather Data), the major output of the activity (such as Write Book), or the name of the system (Run Video Store). Sometimes, the noun is changed during modeling because a different noun is found to be the more accepted version, or the verb might be replaced with something more precise. When an activity is first created, it is given a unique number. Even if the activity is subsequently deleted, the number is not reused.

In an IDEF3 diagram, the activity number is usually preceded by the number of its parent activity (Figure 2-2).
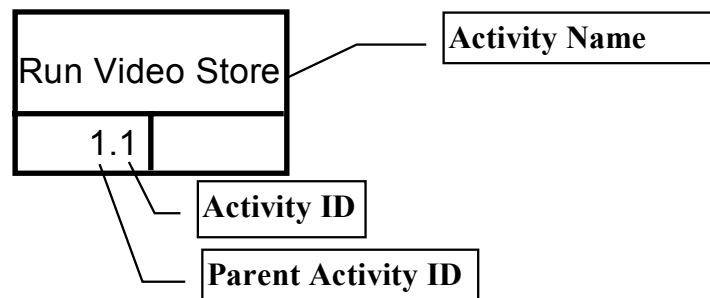
Figure 2-2. Basic IDEF3 activity and name numbering.

## Links

Links denote significant constraining relationships among activities. All links in IDEF3 are unidirectional, and although an arrow may originate or terminate at any side of an activity box, IDEF3 diagrams are generally organized from left to right so that links normally originate from the right side and terminate at the left side of activity boxes. Table 2-1 illustrates the three types of links.
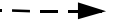
| Graphic | Link Name | Purpose |
|---|---|---|
| ⟶ | Temporal Precedence | The source activity must complete before the destination activity can begin. |
| ⟶▶ | Object Flow | The output of the source activity is input to the destination activity. This implies that the source activity must complete before the destination activity can begin. |
| · – – ▶ | Relational | The constraining relationship between the source and destination activities must be user-defined for each instance of a relational link. |

Table 2-1. Link types.

### Temporal Precedence Link

As the name implies, temporal precedence links, also known as precedence links, denote that the source activity must complete before the destination activity can begin. A precedence link appears in IDEF3 diagrams as an arrow, with the arrowhead pointing from the source activity to the destination (triggered) activity. The link should be labeled so that the reader can understand why the link exists. In many cases, the completion of one activity enables or triggers the activation of another, as shown in Figure 2-3. In this example, management must approve the project team's recommendations before they can be implemented.
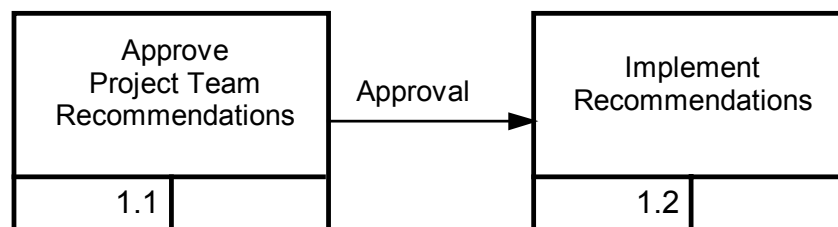


Figure 2-3. Precedence link between activities 1.1 and 1.2.

## Object Flow Link

One of the most common reasons for a temporal precedence link between two activities is that some object, produced by the source activity, is required by the destination activity. This object flow link is distinguished from the generic temporal precedence link by its double arrowhead. These links should be named to clearly identify the object that flows along this link. Object flow links have the same temporal semantics as the precedence link; that is, the activity from which the object flow link originates must complete before the activity to which the object flow link points can start as shown in Figure 2-4. In this example, the assembled part is the object that is produced by the source activity. It must be assembled before it can be painted.
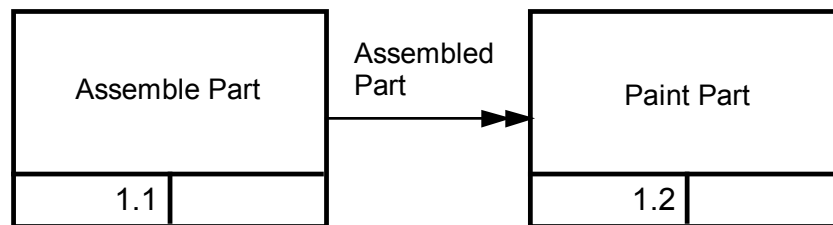


Figure 2-4. Object flow link between activities 1.1 and 1.2.

## Relational Link

Relational links denote relationships that do not imply temporal precedence or object flow. The meaning of each relational link must be defined, because the relational link imposes no constraints of its own.

Relational links can also be used to denote relationships between parallel activities. Figure 2-5 illustrates part of the process of starting a water-cooled saw (such as a tile or brick saw) and the special relationship between the activities Start Blade Motor and Start Water Pump. An arrow label can be used to describe the nature of the relationship, and a more thorough description could be captured as supplemental text.
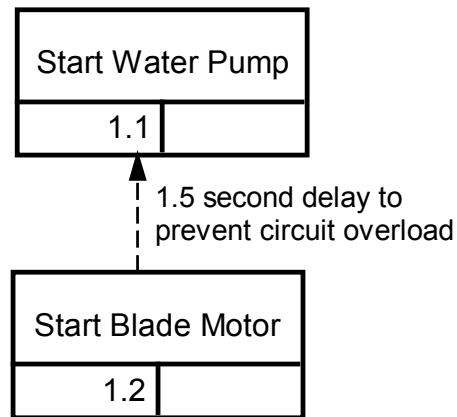
Figure 2-5. Relational link.

A common use of the relational link is to describe special cases of the precedence link, that is, alternate temporal relationships between activities. Figure 2-6a is a typical temporal precedence link. The vertical bars in Figure 2-6b shows the relationship between the start and end times for the two activities in Figure 2-6a implied by a temporal precedence link. In this example, as time moves from left to right, the start of Implement Recommendations occurs AFTER the completion of Approve Project Team Recommendations.
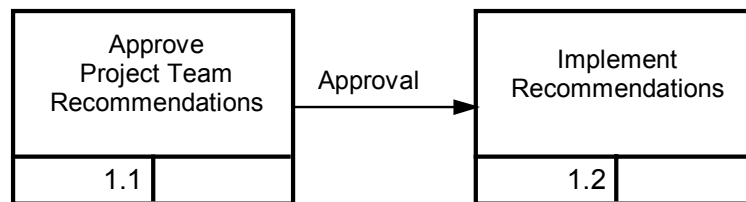


Figure 2-6a. Temporal precedence link.
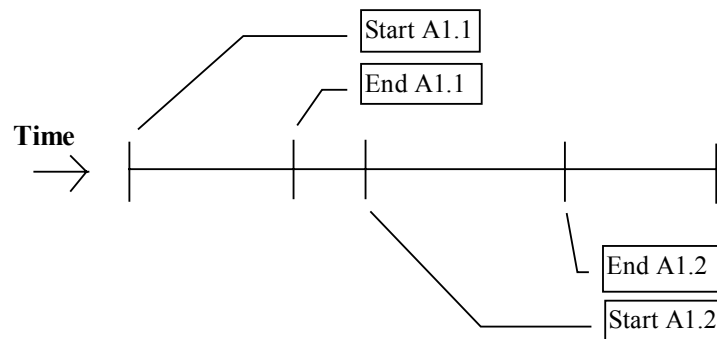


Figure 2-6b. Activity timing of the activities in Figure 2.6a.

An alternative temporal constraint between the two activities in Figure 2-7a is shown in Figure 2-7b. In this example, the project team starts to Implement Recommendations before Approve Project Team Recommendations ends. A relational link is shown in Figure 2-7a.
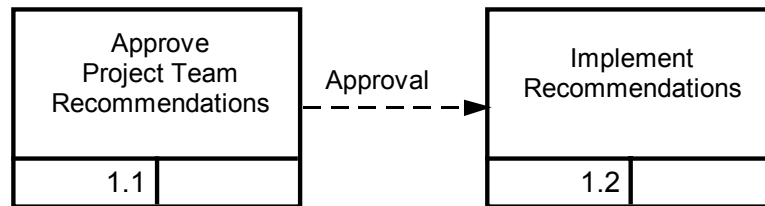


Figure 2-7a. A relational link.
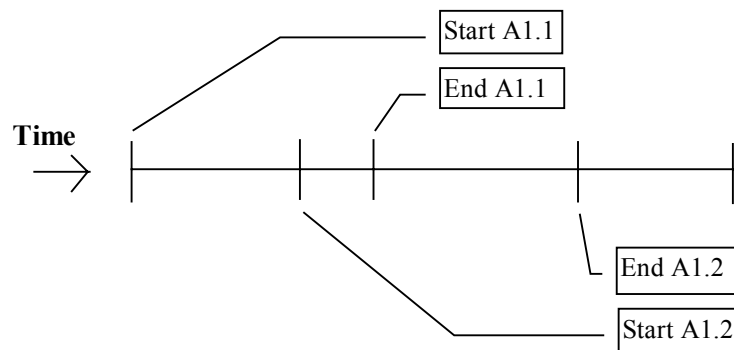


Figure 2-7b. One possible temporal constraint denoted by the relational link in Figure 2-7a.

It is important to clearly document the temporal constraint between two activities connected by a relational link. Consider another possible temporal constraint, shown in Figure 2-8. Applied to the example shown in Figure 2-7, the project team would start to Implement Recommendations after the activity Approve Project Team Recommendations begins, but Implement Recommendations would finish *before* Approve Project Team Recommendations finishes.
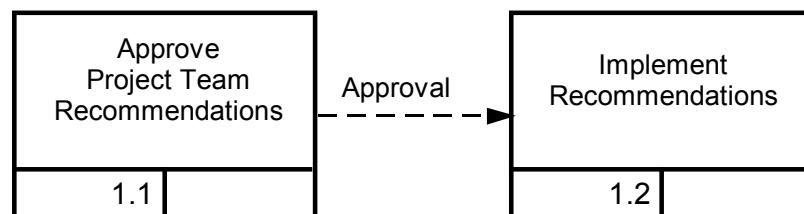


Figure 2-8a. A relational link.
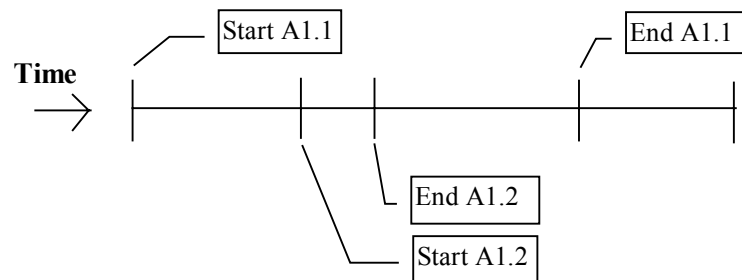
Figure 2-8b. Another possible temporal constraint denoted by the relational link in Figure 2-8a.

The temporal constraints shown in Figure 2-7 and 2-8 are both plausible, and the correct interpretation must be documented. It is important to emphasize that *correct* in this case means the interpretation that accurately reflects the situation being documented, and not the interpretation that in the analyst's view would lead to a more effective process.

## Junctions

The completion of one activity may enable several other activities to start, or an activity may have to wait for several activities to finish before it can start. Junctions either distribute or consolidate process flow and are used to describe process *branching*.

■ *Fan-out junctions* distribute process flow. The completion of one activity causes the activation of other activities.

■ *Fan-in junctions* consolidate process flow. The completion of one or more activities causes the activation of a single activity.

Table 2-2 summarizes the three types of junctions.

| Graphic | Name | Function | Activation Rules |
|---|---|---|---|
| & | AND Junction | Fan-Out | Every destination activity connected to the AND junction is always activated. |
| | | Fan-In | Every source activity connected to the AND junction must always complete. |
| X | Exclusive-OR Junction | Fan-Out | One and only one destination activity connected to the Exclusive-OR Junction is activated. |
| | | Fan-In | One and only one source activity connected to the Exclusive-OR junction must complete. |
| O | OR Junction | Fan-Out | One or more destination activities connected to the OR Junction are activated. |
| | | Fan-In | One or more source activities connected to the OR junction must complete. |

Table 2-2. Junction types.

Fan-in and fan-out junctions are shown in Figure 2-9.
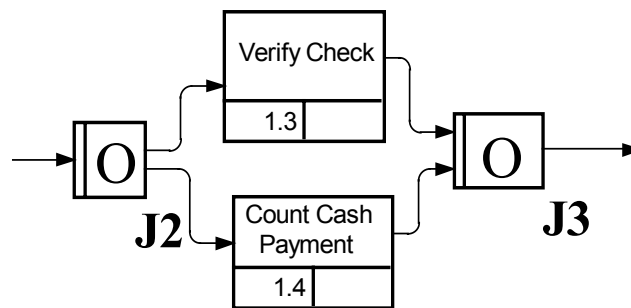


Figure 2-9. Fan-out (J2) and fan-in (F3) junctions in an IDEF3 diagram.

## AND Junction

The AND junction will always activate every destination activity to which it connects. All activities that connect to an AND fan-in junction must complete before the next activity can begin. In Figure 2-10, when Detect Fire completes, Sound Alarm, Notify Fire Department, AND Activate Fire Suppression are activated. When and only when all three of these activities complete, Log Fire Event is activated.
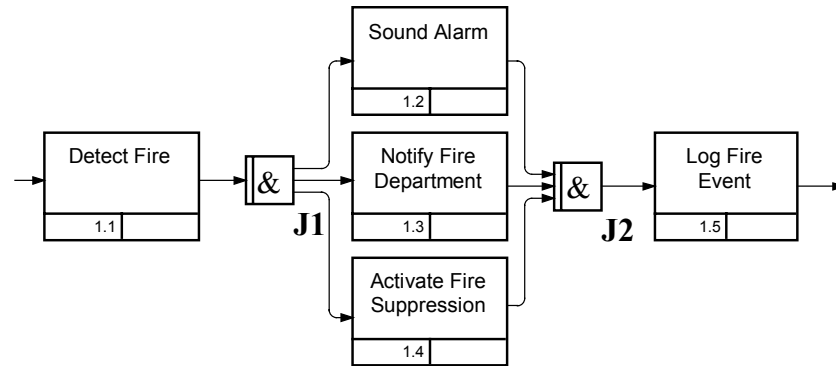


Figure 2-10. AND junctions.

## Exclusive-OR Junction

Regardless of the number of activities attached to an Exclusive-OR fan-in or fan-out junction, only one will be activated at any one time, and therefore, only one will complete before any activity following an Exclusive-OR fan-in junction can begin.

If the junction activation rules are known, then they should be captured either in the junction's elaboration (description), in a junction referent, or by labeling the arrows that emanate from the fan-out junction, as illustrated in Figure 2-11.

In Figure 2-11, an Exclusive-OR junction is used to show that Process Course Audit and Process Credit Course will never be activated at the same time. One and only one of these two activities will be activated by Check Student Status because a student can either take a course for credit or audit a course— but never both.
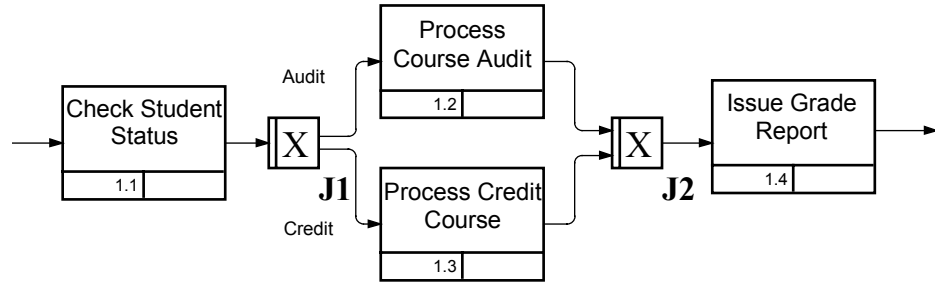
Figure 2-11. Exclusive-OR junctions.

## OR Junction

The OR junction captures activation combinations that cannot be described by AND (all) and Exclusive-OR (one and only one) junctions. Like the relational link, the OR junction is primarily user-defined. In Figure 2-12, the OR junction J2 can activate Verify Check and/or Count Cash Payment. Verify Check will be activated if the customer hands the teller a check, Count Cash Payment will be activated if the customer hands the teller cash, and both will be activated if the customer hands the teller both cash and a check.
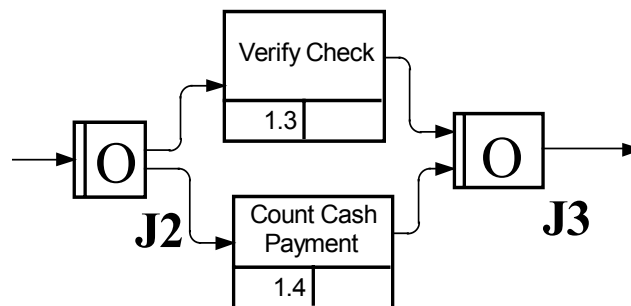
Figure 2-12. OR junctions.

## Synchronous and Asynchronous Junctions

In the AND and OR junction examples, we did not discuss the relationship between the beginning and ending of the activities activated by the fan-out junctions. In these examples, the activities were *asynchronous*—they did not have to begin or end at the same time. However, there are occasions when the start and/or end times of parallel activities must be *synchronous*—they must occur at the same time. Synchronous junctions are used to model this behavior. Table 2-3 shows the proper interpretations for synchronous junctions.

| Graphic | Type | Fan | Description |
|---------|------|-----|-------------|
| &#124;&amp;&#124; | AND | Fan-Out | All of the activities that fan out of the junction will begin together. |
| | AND | Fan-In | All of the activities that fan in to the junction will end together. |
| &#124;O&#124; | OR | Fan-Out | One or more of the activities that fan out of the junction will begin together. |
| | OR | Fan-In | One or more of the activities that fan in to the junction will end together. |
| &#124;X&#124; | Exclusive-OR | Fan-Out | Since one and only one activity connected to an Exclusive-OR fan-out junction is activated, synchronicity with other activities is impossible. |
| | Exclusive-OR | Fan-In | Since one and only one activity connected to an Exclusive-OR fan-in junction completes, synchronicity with other activities is impossible. |

Table 2-3. Synchronous junctions.

A synchronous junction is denoted by the two vertical bars inside the junction box, as opposed to the single vertical bar indicative of an asynchronous junction.

In many kinds of competitive races, the starter's gun must sound, the clock must start, and the participants must begin racing at the same time. Otherwise, the race is not considered to be a fair contest.

Figure 2-14 illustrates this example using a synchronous AND junction.
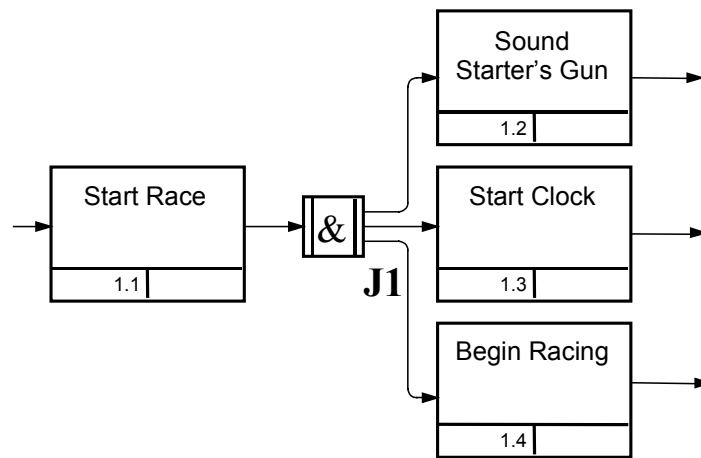


Figure 2-14. A synchronous junction.

If a junction is synchronous, then the modeler should note the tolerance in the timing constraint—that is, how closely to one another the activities have to begin or end. Also, it is not required that a synchronous fan-out junction be paired with a matching synchronous fan-in junction. Certainly, activities can begin together but not end together, such as in the race example, where the main attraction is the decidedly asynchronous finish. It is also possible for activities to begin asynchronously and end synchronously.

### Junction Pairs

On a diagram, junctions should be paired, that is, every fan-out junction has a paired fan-in junction. However, it is not required that the junction type be the same. In Figure 2-15, an AND fan-out junction is matched with an OR fan-in junction. The AND junction (J1) is interpreted to mean the same as in Figure 2-10. When Detect Fire completes, Sound Alarm, Notify Fire Department, AND Activate Fire Suppression are activated. The OR junction (J2) is interpreted as follows: When Sound Alarm and/or Notify Fire Department and/or Activate Fire Suppression complete, Log Fire Event is activated.
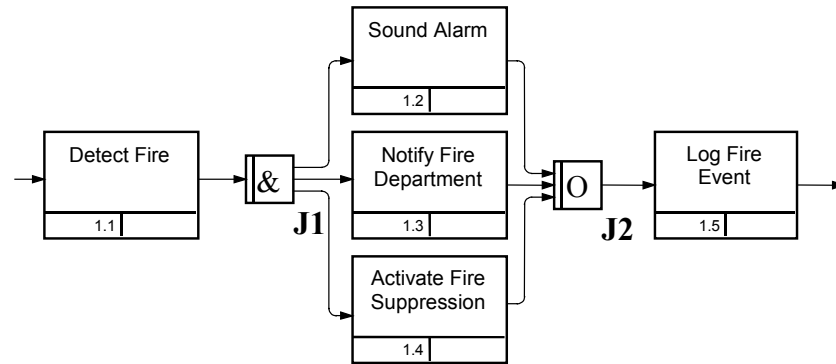
Figure 2-15. An example of combining two types of junctions—an AND fan-out junction with an OR fan-in junction.

## Junction Combinations

Junctions can be combined to create more complex branching rules (Figure 2-16). Junction combinations must be used judiciously, with a clear understanding of the purpose of the document being the ultimate guideline for deciding whether a particular junction combination will clarify or merely clutter a diagram. Complicated junction structures can be nested inside activity boxes.
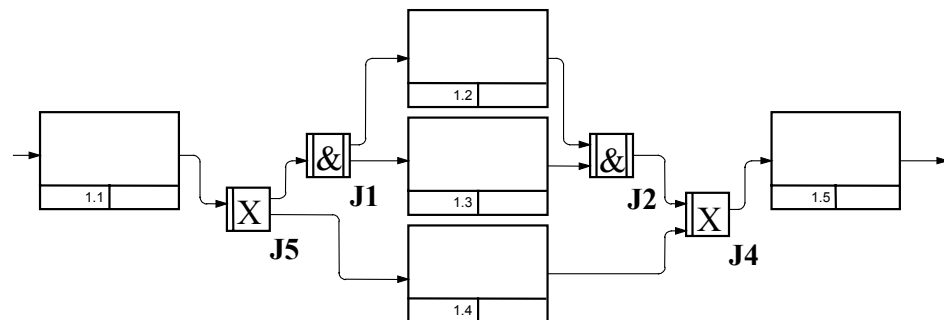


Figure 2-16. An IDEF3 diagram showing a junction combination.

## Referents

Referents are special symbols that refer to other parts of a process description. They are added to a diagram to direct the reader's attention to something important.

| Referent Type | Purpose |
| --- | --- |
| OBJECT | To describe the participation of an important object in an activity. |
| GOTO | To implement looping (repeating a sequence of activities), possibly on the same diagram but not necessarily so. If the activities are all on the same diagram, looping can also be depicted by drawing an arrow back to the starting activity. A GOTO referent can also refer to a junction. |
| UOB (unit of behavior) | To include another instance of an activity without looping. For example, if the activity Count Cash occurs several times within a process, the first occurrence of Count Cash can be created as an activity and subsequent occurrences drawn as UOB referents. The use of this referent type is normally not required when using automated tools. |
| NOTE | To document any important but general information that relates to some graphic on the diagram. In this regard, NOTE referents serve as an alternative to recording text notes directly on a diagram. |
| ELAB (elaboration) | To elaborate on a graphic, or describe it in more detail. Elaboration referents are commonly used to describe the branching logic of a junction. |

Table 2-4. Referent types.

A referent is depicted as a box (much like an activity). The referent name usually includes the referent type (such as object, GOTO) and an identifier. Figure 2-17 illustrates an Object referent.

OBJECT/Pilot
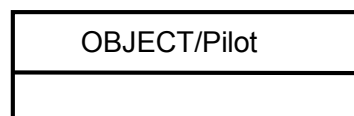
Figure 2-17. Object referent.

Figure 2-18 illustrates an important object/activity relationship.
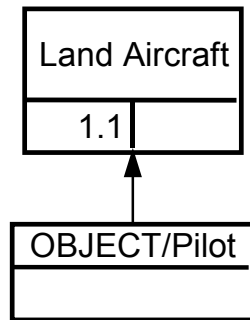


Figure 2-18. Object referent to an activity.

## Activity Decomposition

IDEF3 activities may be decomposed to express more detail. The IDEF3 method allows an activity to be decomposed *multiple* times—that is, have multiple children. This allows a single model to document alternative process flows.

To properly track activities in a multiple decomposition model, the numbering scheme must be extended to include the decomposition number as well as the activity ID and parent activity ID. This is illustrated in Figure 2-19.
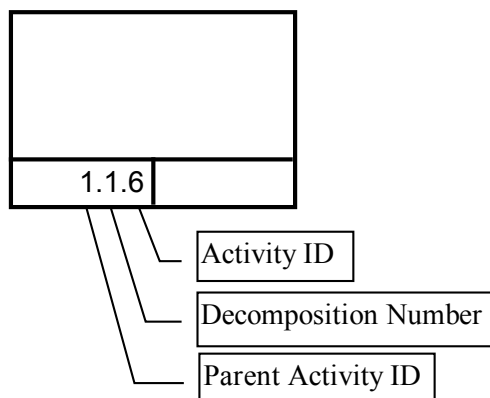


Figure 2-19. Activity identifier that includes the decomposition number.

# Building an IDEF3 Process Description Capture

In this section, we will describe the process of building an IDEF3 diagram from a text-based process description. We will assume that the process involves a modeler, or author (usually an analyst), and one or more subject experts from whom we will derive the process description.

## Defining Scenario, Scope, and Viewpoint

Before the subject experts are asked to prepare a process description, the scenario of interest and the scope of the model should be documented, so that the experts can understand the required depth and breadth of the description. In addition, if the viewpoint is different from that of the experts' normal relation to the system, this must be explicitly and carefully explained.

It is possible that the experts may not be able to produce an acceptable description without the modeler guiding the experts through a formal interview. The modeler should prepare a set of questions in much the same way a reporter prepares for an interview.

## Defining Activities and Objects

The subject experts will typically provide a textual narrative that describes the scenario of interest. In addition, written documentation may already exist to aid in understanding the current process. Regardless of whether the information is written or verbal, it is analyzed and separated into parts of speech in order to identify a candidate list of activities (verbs and verb phrases) that constitute the process and objects (nouns and noun phrases) that participate in the process. Although the process sounds straight forward, there are certain pitfalls, such as adjectives that can be verb forms as well (at least in English). An example of this is "test procedure," where it is unclear whether *test* is an adjective describing <u>procedure</u> or a verb.

In some cases it is also possible to create the graphic model in the presence of the subject experts. The graphic model can also be designed after the information gathering session so that the details of the diagram formatting do not distract the participants.

Because many IDEF3 models may be developed at the same time by separate teams, IDEF3 supports a simple scheme for allocating activity numbers across all models. Different modelers, each assigned a different range of activity numbers, can work independently. In Table 2-5, activity IDs are allocated in large blocks to each modeler. In this example, Tom exhausted his original supply of numbers and was given a second allocation.

| Modeler | IDEF3 Number Range |
|---------|-------------------|
| Tom | 1-999 |
| Lynn | 1000-1999 |
| Dan | 2000-2999 |
| Tom | 3000-3999 |

Table 2-5. Asignment of number ranges to IDEF3 modelers.

## Sequencing and Concurrency

If the diagram is being created after the interview, the modeler will need to make some decisions with respect to the diagram's hierarchy, for example, how much detail will be included on a single diagram page. If activity sequencing and concurrency are not clear, the experts can be interviewed again (perhaps having a copy of the incomplete diagram for reference) to fill in missing information. It is important, however, to distinguish between implied concurrency (concurrency that is implied by the absence of links) and explicit concurrency (concurrency clearly stated in the expert's description).

## Activity, Junction, and Object Elaboration Documents

IDEF3 allows for information to be captured in a variety of ways. For example, complicated junction logic can be described graphically using a combination of junctions. That same information can also be captured in an elaboration referent, or even as part of the junction's definition. This allows the modeler to capture information in the form most convenient at the time (if the diagram is being built in the presence of the expert). However, it is important that the models be reorganized, if necessary, to make them suitable for presentations. The choice of presentation format often has a drastic impact on the organization of a model, since elaborate junction combinations take considerable space on a diagram, and using hierarchies of junctions complicates activity placement.

IDEF3 can also be used to build design models and can be employed in this regard as a follow-up activity to IDEF0 and data flow modeling. For a description of the process of building an IDEF3 design model, see "Creating IDEF3 Models to Illustrate IDEF0 Activations," in Chapter 3.

# The IDEF0 Function Modeling Method

## Overview of IDEF0

IDEF0 activity modeling is a technique for analyzing whole systems as a set of interrelated activities or functions. This purely functional orientation is important: the functions (verbs) of a system are analyzed independently of the object(s) that perform them. The idea is that the packaging of functions within systems can be performed as part of a new design process but is irrelevant and possibly even misleading as part of analysis. A purely functional perspective allows for a clear separation of the issues of meaning from issues of implementation.
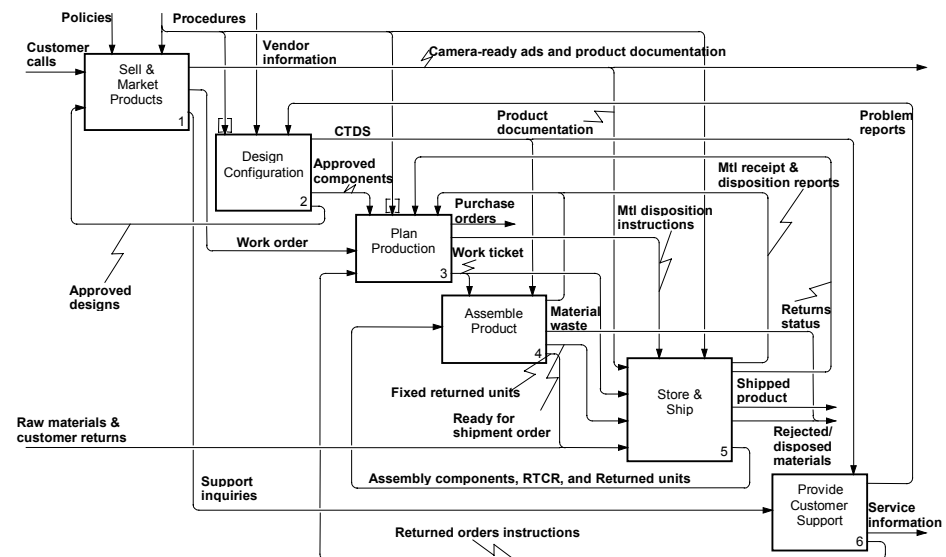


Figure 3-1. A typical IDEF0 diagram.

IDEF0 is best suited as an analysis and logical design technique. As such, it is generally performed in the early phases of a project, perhaps being preceded by IDEF3 modeling for data collection and AS-IS process modeling. An analysis using IDEF0 modeling can feed a design process using IDEF3 models and data flow diagrams.

# IDEF0 Model Syntax and Semantics

## An IDEF0 Model

IDEF0 combines a limited graphical notation (there are only two symbols—boxes and arrows) with a rigorous and well-defined process by which the models are authored to improve the quality of the completed model.

The IDEF0 methodology is similar in some respects to that used for publishing books, and often a set of printed IDEF0 models are organized into a binder with a table of contents, a glossary, and other "book-like" matter. In IDEF0, this is known as a *kit*.

The first step in building an IDEF0 model is to identify the *purpose* of the model—the set of questions the model is intended to answer. These questions should be enumerated as part of the model documentation, much like the preface of a book describes its purpose. A statement of purpose, which summarizes the questions, is often written.

The model's *scope* consists of the breadth and depth of the detail and is similar to information usually included in a book's preface. It is not sufficient to merely list the set of questions to be answered. The reader of the model, and so too, those who are to actually author the model, need to understand the amount of detail they should expect for each answer.

In addition, the intended *audience* needs to be identified. Often, the intended audience will have a great impact on the level of detail that can and/or should be included in a model. Consider what information the audience already knows about the subject, the background or technical information that they might need to understand the subject, and the language and style that is most appropriate.

*Viewpoint* is the perspective from which the model will view the system. The viewpoint is chosen to encompass the chosen scope and fulfill the purpose. Once selected, the viewpoint must remain consistent throughout a single model. If necessary, other models should be created to model the system from different viewpoints. Some viewpoints are customer, supplier, store owner, and editor.

## Activities

An activity, sometimes called a function, processes or transforms inputs into outputs. Because IDEF0 models a system as a set of hierarchical (nested) activities, the first activity to be defined is the activity that describes the system itself—the *context* activity. This is drawn as a box, and is given a name.

Activity names in IDEF0 generally consist of a single, active verb plus a common noun that clarifies the objective of the activity from the viewpoint of the model. An adjective may be used to further qualify the noun. It is important that the activity name accurately reflect the system as it is observed from the specified viewpoint for the model.

An activity box is shown in Figure 3-2. The activity is properly labeled with a verb + direct object. The word *manufacturing* functions as an adjective to qualify the noun *business*.
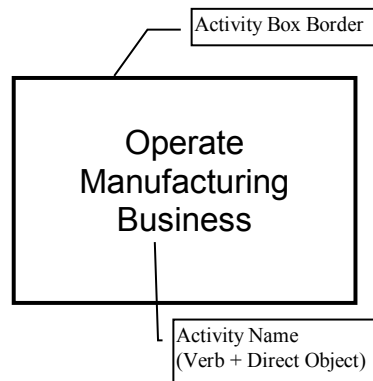


Figure 3-2. An IDEF0 activity box.

Earlier, we noted that IDEF0 models a system as a set of *hierarchical* (nested) *activities*. An activity can be *decomposed* (broken down) into its constituent activities (Table 3-1). Since IDEF0 defines a system as a box, we can view decomposition as sticking our head inside the box and observing that it consists of other boxes. Decomposition is commonly referred to as top-down modeling, but this is a misnomer. Functional decomposition can be more correctly viewed as outside-in modeling, in which we peel away at the layers of a system much like we peel away the layers of an onion in order to view the detail that lies within.

**Operate QUILL Business**

Sell & Market Products
Manage Advertising
    Take Orders
        Answer Phone Calls
        Provide General Information
        Answer Fax Pricing Requests
        Record Order Information
        Check Credit
        Provide Order Follow-up
    Provide Pricing Information
    Develop Documentation
    Prepare Work Ticket

Design Configuration
    Identify Vendors & Components
    Develop Specifications
    Specify Components
    Test Configuration
    Approve Vendors

Plan Production
    Order Assembly Components
    Issue Work Ticket
    Manage Component Inventory
    Schedule Production
    Dispose Outdated Component Parts

Table 3-1. Part of an activity hierarchy in node index format.

## Boundary and Interface (Arrows)

To be useful, the description of an activity must at a minimum also include a description of the objects the activity creates as output, as well as those objects the activity consumes or transforms.

In IDEF0, we also model *controls* and *mechanisms*. Controls are the objects that govern the manner in which inputs are transformed yet are not themselves transformed by the activity. Mechanisms are those objects that actually perform the transformation of inputs to outputs yet are not themselves transformed by the activity.

ICOM is an acronym for the categories of information that are captured on IDEF0 diagrams. It represents four types of arrows.

I=   Input: something consumed in the process.

C=   Control: a constraint on the operation of the process.

O=   Output: something resulting from the process.

M=  Mechanism: something which is used to perform the process, but is not itself consumed.

Figure 3-3 illustrates the four arrow types, showing the specific box side to which they must connect.
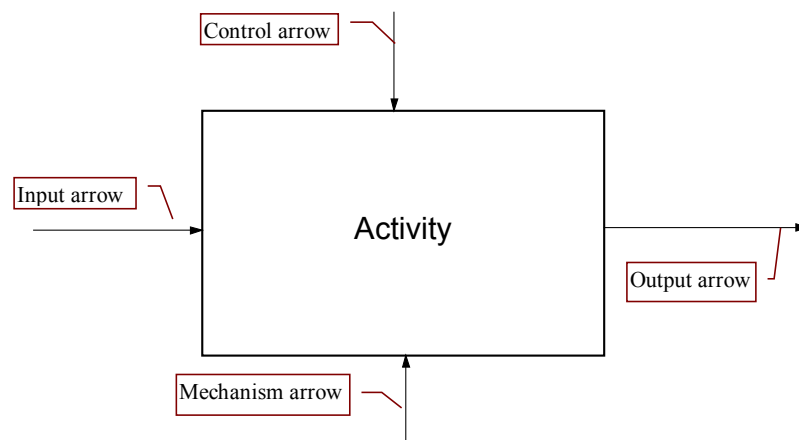
Figure 3-3. Each arrow type connects to one specific side of an IDEF0 activity.

Whereas activities are always verbs or verb phrases, arrows are always nouns. The arrows can represent people, places, things, concepts, or events. An arrow is represented on IDEF0 diagrams by lines with an arrowhead at one end. The line is labeled with the name of the arrows. As is the case with activities, an arrow name is not enough to guarantee that the model will be understood by its readers. A textual definition of each arrow is critical to developing an accurate and useful model.

## Input Arrows

*Inputs* represent material or information that is consumed or transformed by the activity in order to produce the outputs. Input arrows always enter the left side of an IDEF0 activity box. *Input arrows are optional*, since some activities do not transform or change anything. An example of an activity without an input might be Make Executive Decision, where several factors are analyzed to produce a decision, but none of the factors is transformed or consumed by the decision.

## Control Arrows

Controls govern or regulate how, when, and if an activity is performed and which outputs are produced. Since controls govern the conduct of an activity to ensure the creation of the desired output, e*ach activity must have at least one control arrow.* Controls *always* enter the top of an IDEF0 activity box.

Controls are often in the form of rules, regulations, policies, procedures, or standards. They influence the activity without actually being transformed or consumed. There will be times when an activity's purpose is to change a rule, regulation, policy, procedure, or standard. In this case, it would be expected that the arrow containing that information would be an input.

A control is a special type of an input to an activity. If it is unclear whether an arrow should be modeled as an input or a control; choose control until the ambiguity can be resolved. Integrating data and process models (Chapter 6) is a valuable technique for resolving ambiguities.

## Output Arrows

Outputs are the material or information produced by the activity. *Each activity must have at least one output arrow*. An activity that does not produce a definable output should not be modeled (or, at a minimum, should be a candidate for elimination).

In a non-manufacturing environment, outputs are often data that was processed in some form by the activity. It is important to use modifiers in front of the arrow labels to indicate how the output data is different from the input data. For example, the activity Admit Patients may have *patient data set* as both an input and an output. Properly labeled, the input arrow could be labeled *raw patient data*, and the output arrow could be labeled *verified patient data*.

## Mechanism Arrows

Mechanisms are those resources that perform the activity. Mechanisms could be the important people, machinery, and/or equipment that provide and channel the energy needed to perform the activity. A mechanism arrow may be omitted from an activity if it is determined that it is not needed to fulfill the model's purpose.

## Arrow Interface Combinations

There are five basic arrow interface combinations: output-input, output-control, output-mechanism, output-control feedback, and output-input feedback.

An output-input arrow represents an activity with precedence over another activity. In Figure 3-4, Procure Resource must precede Transform Resource.
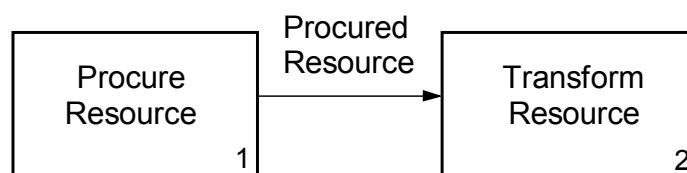


Figure 3-4. An output-input arrow connection.

An output-control represents activity dominance, in which one activity is dominant over another because it controls how that activity transforms inputs into outputs. In Figure 3-5, the approved plan guides the implementation of the recommendations. The recommendations themselves are not changed by the implementation, hence Approved Plan is depicted as a control arrow of Implement Recommendations.
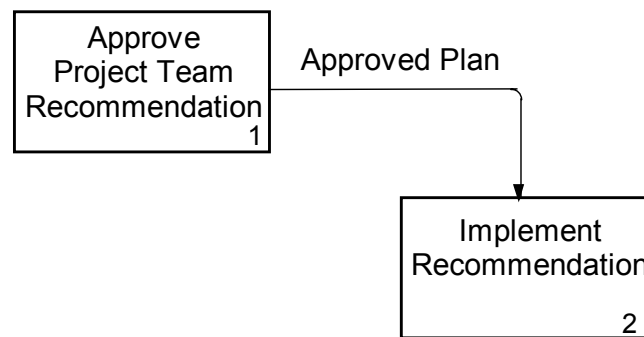
Figure 3-5. An output-control arrow connection.

Output-mechanism arrows are less common and represent a situation in which the output of one activity becomes the means to perform another. In Figure 3-6, a jig, a device used to temporarily hold something in place while it is being manipulated, must be fashioned in order to Build Part.
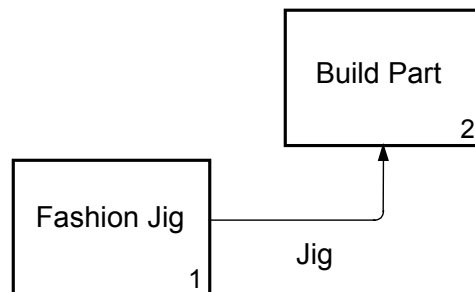
Figure 3-6. An output-mechanism arrow connection.

Control and input feedback represent cases where boxes of normally lesser dominance provide feedback to boxes of greater dominance. Figure 3-7 is an example of output-control feedback—the output of Evaluate Project Performance, which is the project performance evaluation. The project performance evaluation feeds back to the activity, Develop Project Plan. The intent is for the lessons learned on this project to be applied to better management of future projects. Project Performance Evaluation is a control arrow to Develop Project Plan because the evaluation is not transformed by the activity Develop Project Plan.
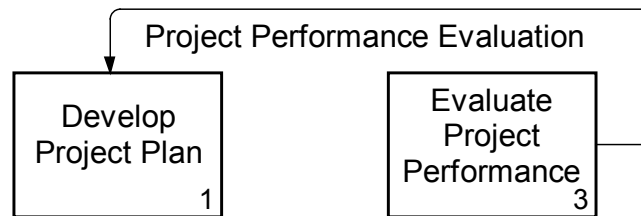
Figure 3-7. Output-control feedback.

Output-input feedback is commonly used to describe rework cycles. Figure 3-8 is an example of such an output-input feedback.
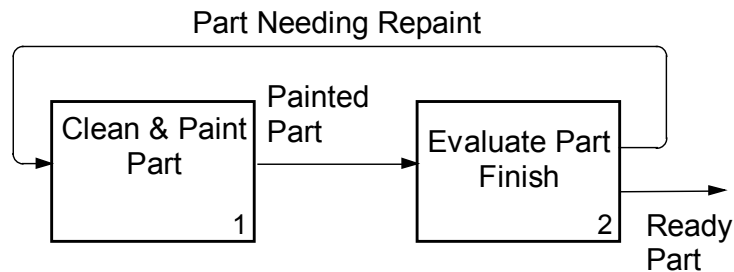
Figure 3-8. Output-input feedback.

Another use for output-input feedback is to describe cases where scrap materials can be reused along with raw materials. An example of the latter is the bottle-making process, where bottles that break during manufacture are simply remelted and fashioned into other bottles.

## Branching and Joining

The outputs of an activity may be used by more than one other activity. In fact, IDEF0 is most valuable as a tool to help visualize activity interdependencies within a system. IDEF0 arrows may branch (split) and join (merge). Figure 3-9 is an example diagram showing only branching and joining arrows.
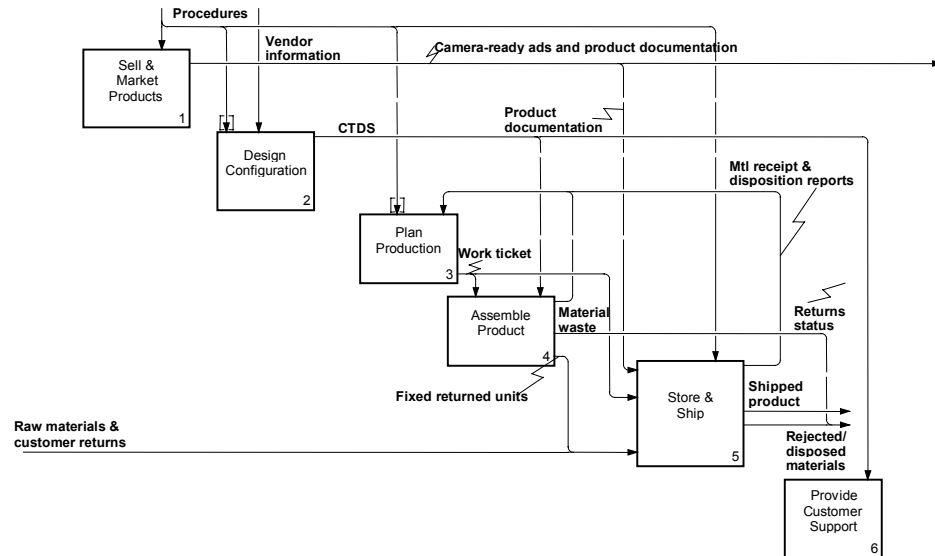


Figure 3-9. An IDEF0 diagram showing only arrows that branch or join.

The meaning of a split or joined arrow is made clear by the judicious labeling of the arrow segments that join and split. In Figure 3-10, the control arrow Policies & Procedures is an example of an arrow that splits but is not renamed. The Policies & Procedures arrow splits and is a control to both activity 2 and activity 3, which means that both activities use Policies & Procedures to guide their transformation of inputs to outputs.
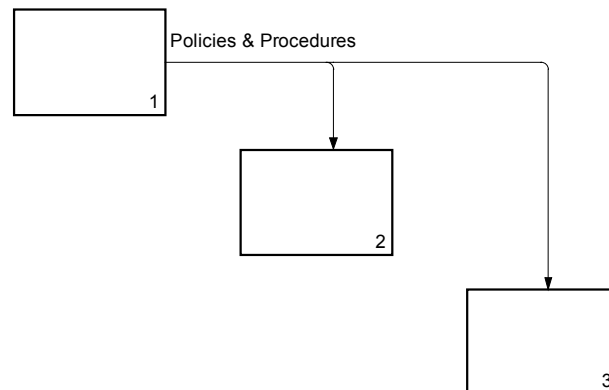


Figure 3-10. Policies & Procedures are created by activity 1 and used by activity 2 and activity 3.

In Figure 3-11, one of the arrow branches is renamed to show that only a subset of Policies & Procedures is used by activity 2. If an arrow is split and a branch is renamed, the new name must reflect a subset of the original arrow. Arrow splitting represents data decomposition (just as activity hierarchy represents function decomposition).
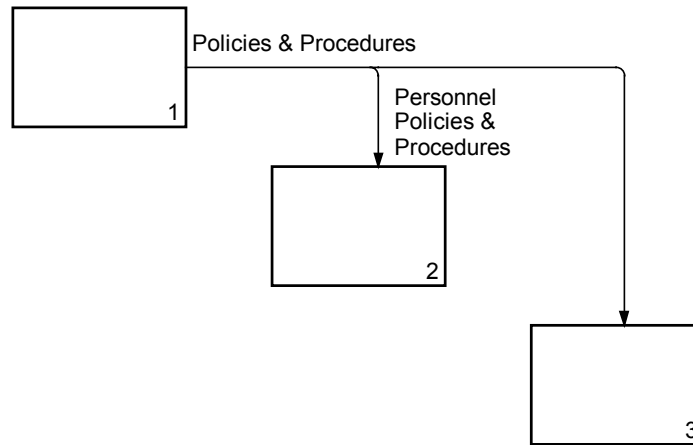
Figure 3-11. A split arrow that is renamed on only one segment.

Figure 3-12 is another example of data decomposition. The squiggles seen in this example are often used in IDEF0 diagrams to clearly link an arrow label and the line.
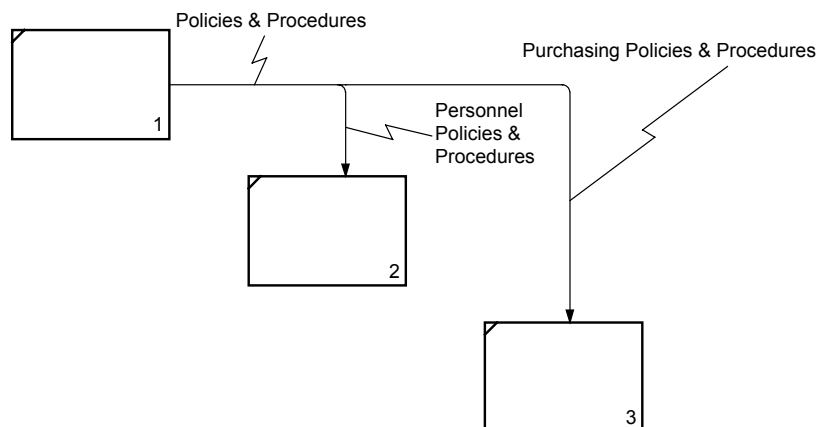
Figure 3-12. An arrow branching into two distinctly labeled segments.

In Figure 3-13, it is implied that the output of activities 1 & 2 is also labeled Rejected Materials.
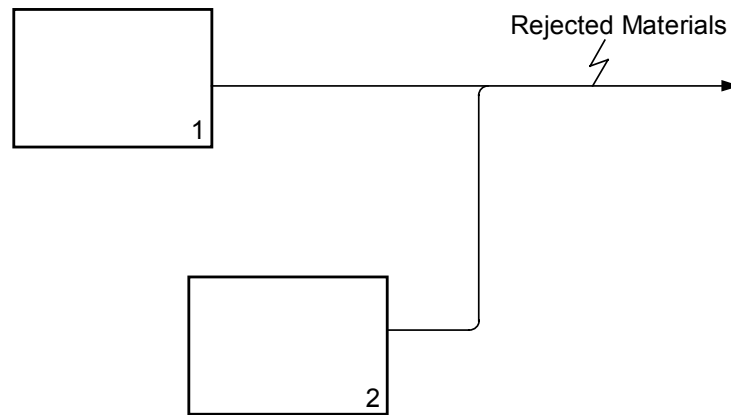
Figure 3-13. An example of merging arrows with only one name.

In Figure 3-14, each arrow segment is labeled to clearly identify the individual components and the bundled result.
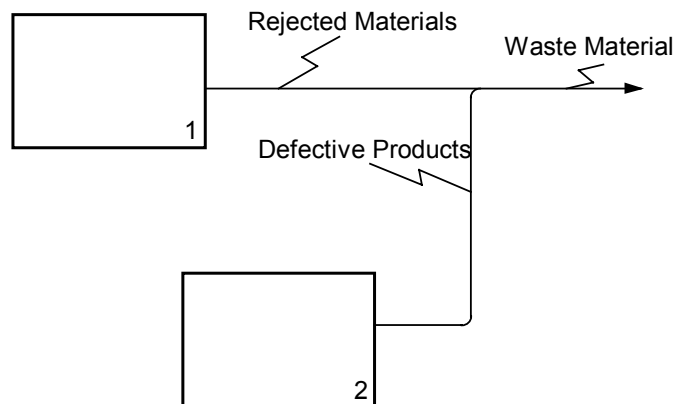
Figure 3-14. Proper labeling of different merging arrows.

**Call Arrows**

Call arrows are used to reference other models or diagrams within a model that may help the reader better understand the current model, or perform the same function. For example, in Figure 3-15, "Test & Calibration" is the name of another model that documents testing and calibration. Rather than being duplicated, the model is simply referenced, that is, *called*. A call arrow can refer to another diagram (more correctly, the diagram's parent activity) within the same model, and can also refer to a specific child activity in another model. This is a way in which functional duplication (similar functions operating on different data) can be properly documented in the model.

Test & Calibration
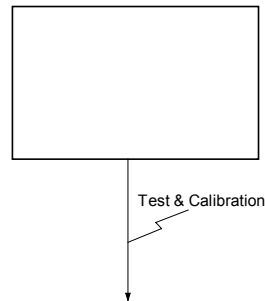
Figure 3-15. A call arrow.

## Tunnels

*Arrow bundling* is used to control the level of detail in a diagram. In the case of an arrow that does not warrant appearing on the parent diagram (it is not significant enough), but does not belong bundled with other arrows, the *tunnel* is used to indicate that the arrow exits or enters the system. In Figure 3-16, for example, MRP System is an important mechanism on this diagram, but is probably not used anywhere else in the model. Tunneling is used as an alternative to cluttering up parent diagrams by including this mechanism arrow.
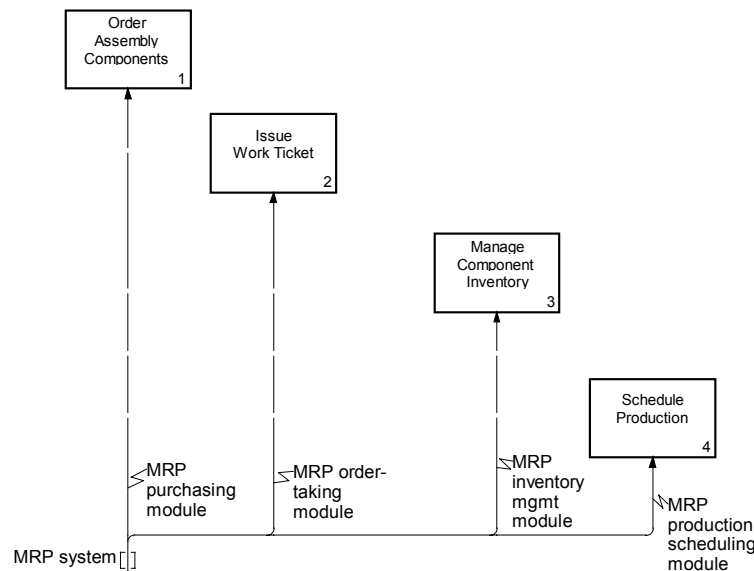


Figure 3-16. An example tunnel on the mechanism arrow MRP System.

In other cases, a tunnel can be used on an arrow leading to or from a parent activity. This indicates that the arrow's relationship with the activity's children is undefined. In Figure 3-17, the tunnels indicate that the Procedures arrow is not detailed in the decomposition of Design Configuration or Plan Production.
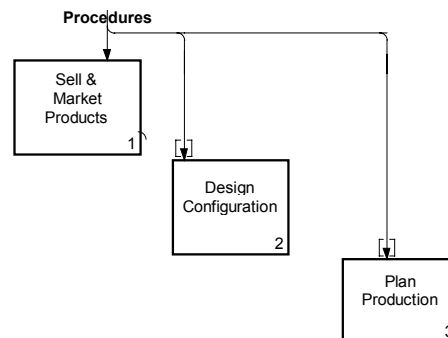


Figure 3-17. The arrow Procedures is tunneled at Design Configuration and Plan Production.

## Activities and Activations

Typically, we do not drill and lathe and route and punch every time we perform a machining operation. We generally perform a subset of these activities. For example, wood is drilled differently than metal, because wood is generally fashioned with screws and metal is fitted with bolts or rivets. A screw hole is usually countersunk, so that the screw head is embedded in the wood. Bolts are seldom countersunk. We could decompose the model to create separate activities for drilling, bolting, etc. We could instead choose to create separate IDEF3 models for each of these activities. This is particularly useful if we wanted to evaluate drill, bolt, and other activities using simulation.

As a simpler alternative to the above two approaches, an *activation* table can describe the various input, control, output and mechanism combinations for each activation of an activity. An activation is a unique configuration of input and control values, and resource requirements. Table 3-2 is such an activation table. Each activation is given a name unique within the activity, and the values of the various arrows are listed. The combination of arrow values should be unique for each activation, that is, two activations should not have the same values for all of their arrows.

| Activation Name | Arrow Name | Arrow Value |
| --- | --- | --- |
| Large amount of cash | Cash | >$500 |
| | Cash-counting machine* | 1 Required |
| Small amount of cash | Cash | <$500 |
| | Cash-counting machine | 0 Required |

Table 3-2. Activation table for the activity Count Cash.

The activation information in Table 3-2 tells us a little about the contents of the control arrow(s) for this activity. For example, we can imagine that the bank has a policy for counting cash that stipulates a counting machine is to be used for cash amounts greater than $500.

# Creating IDEF3 Models to Illustrate IDEF0 Activations

IDEF3 models can be built from an IDEF0 model to illustrate the activations of leaf-level activities (that is, non-decomposed activities—activities with no children). If the authors intend to reuse IDEF0 models in this way, then the activations of each activity's interface should be carefully documented. Table 3-3 describes different activations of the activity Process Account Deposit. As an alternative, IDEF3 models can be built and the activation models derived from these.

| Activation Name | Arrow Name | Arrow Value |
|---|---|---|
| Check-only | Cash in | =$0.00 |
| | Check | >$0.00 |
| | Deposited | = Check Total - Return |
| | Return | N/A |
| | Cash out | = Return |
| | Cash-counting machine | 0 Required |
| | Teller | 1 Required |
| Cash-only | Cash in | >$0.00 |
| | Check | =$0.00 |
| | Deposited | = Cash Total - Return |
| | Return | N/A |
| | Cash out | = Return |
| | Cash-counting machine | If cash in > $500, then 1 required |
| | | Else, 0 Required |
| | Teller | 1 Required |
| Cash & check | Cash in | >$0.00 |
| | Check | >$0.00 |
| | Deposited | = Check Total + Cash Total - Return |
| | Return | N/A |

| Activation Name | Arrow Name | Arrow Value |
|---|---|---|
| Cash & check | Cash out | = Return |
| | Cash-counting machine | If cash in > $500, then 1 required |
| | | Else, 0 Required |
| | Teller | 1 Required |

Table 3-3. Example activation table for the activity Process Account Deposit.

# Building an IDEF0 Model

In this section, we will define the model-building process in more detail.

## The Diagram

Figure 3-18 is a typical IDEF0 diagram shown with its border. The border consists of a well-defined header and footer. The border elements along the top (header) are used to track the model in progress. The border elements along the bottom display the diagram's identity and parentage.
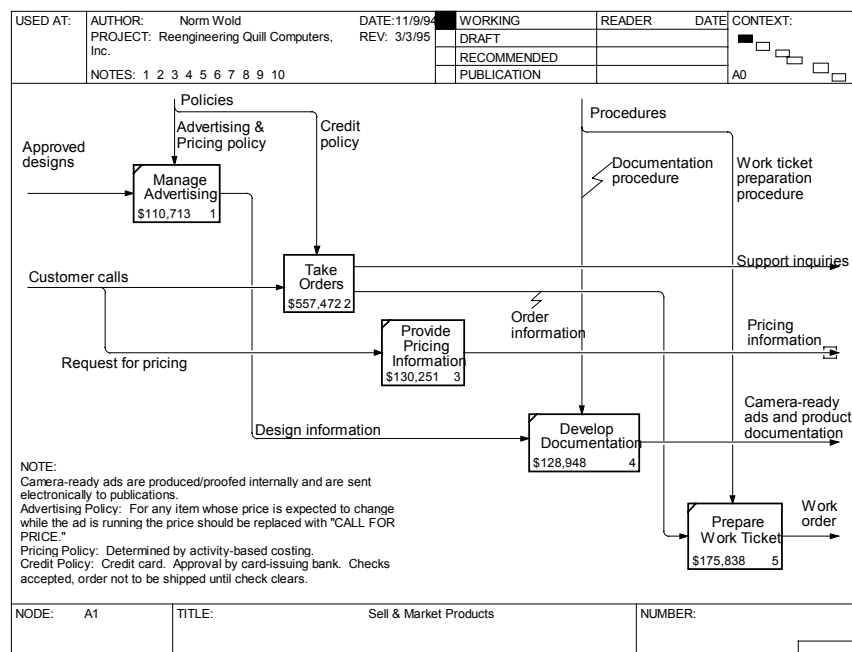


Figure 3-18. A typical IDEF0 diagram and border area.

The border elements along the top (header) are:

| Field | Purpose |
|---|---|
| Used At | Historically used to document all of the places where this diagram (more properly, the diagram's parent box) is called by a call arrow. |
| Author, Date, and Project | Tells who originally created the diagram, the date it was created, and the project title under which it was created. The project title was used to track diagrams within projects. Revision date is the date the diagram was last changed. |
| Notes 1 2 3 4 5 6 7 8 9 10 | When diagrams were edited by hand, readers would cross off a number each time they created a new note on the diagram. |
| Status | Status reflects the stage of approval for this diagram. This field is used to implement a formal publishing process with review and approval steps. |
| Working | A new diagram, a major change, or a new author for an existing diagram. |
| Draft | The diagram has reached some level of acceptance by the readers. It is ready to be evaluated by a review committee. |
| Recommended | The diagram and all its supporting text has been reviewed and approved. It is not expected to change. |
| Publication | The diagram is ready for final printing and publication. |
| Reader | The name(s) of the reader(s). |
| Date | The date of the reading. |
| Context | A sketch of the outlines of the activity boxes in the parent diagram, with the parent box of this diagram highlighted. The node number of the parent box is also listed. The context field of the context diagram (the topmost diagram) should read "TOP" (to indicate that there is no parent diagram in the model). |

The border elements along the bottom (footer) are:

| Field | Purpose |
| --- | --- |
| Node | The node number of this diagram is the same as the activity number of this diagram's parent activity box. |
| Title | This is the name of the parent activity box. |
| Number | Also called C-Number, the number is a unique identifier for THIS version of THIS diagram. Thus, any new version of a diagram will have a new C-Number. Normally, the C-Number consists of the author's initials (must be unique among all modelers on the project) and a sequential, unique identifier. For example, JDM001. C-Numbers are NEVER reused on a project. |
| Number | Upon publication, the C Number can be replaced with a standard page number. If a diagram replaces another, then the C-Number of the diagram being replaced is often included in parenthesis—for example, JDM002 (JDM001). This provides a complete history of revisions to all diagrams in the model. |

## The Author-Reader Cycle

Like the author/editor cycle used in book publishing, IDEF0 diagrams are typically reviewed and edited to validate their correctness and improve their quality.

When an author is ready for review, she will typically prepare a *kit* (folder) for each reviewer, who will review the model and make comments (notes) on the diagrams or related text. These commented diagrams are then returned to the author who makes the corrections. If there is disagreement, a commented diagram can be distributed to other readers for group consensus.

A formal review and approval mechanism is supported by the Status field and diagram versioning, and history is supported through the Number field. Usually, a librarian is assigned to large projects, and various levels of controls are implemented depending on the size of the project.

## Building a Model

No model should be constructed without a clear objective or purpose statement. Components of a purpose statement should answer the following questions:

Why is the given process being modeled?

What will this model show?

What can readers of the model do with it?

A purpose statement allows the modeling team to remain focused throughout the modeling effort. Without a purpose statement, modeling sessions may drift aimlessly.

An example of a purpose statement is: Identify the tasks of each shop worker and understand how the tasks relate in sufficient detail to develop a training manual.

Models are built to answer a set of questions. These questions should be developed early on and will serve as the basis for the model's purpose. Sample questions are:

- What are the foreman's tasks?
- What are the machinist's tasks?
- Who inspects completed products?
- Who inspects subassemblies?
- How do the assemblies move through the shop?
- Which tools are needed for which steps?

## Viewpoint

Although it is important to include individuals with differing viewpoints in a modeling session, a given model must still be constructed from a single specific viewpoint. Often, other viewpoints are documented briefly in attached *For Exposition Only (FEO)* diagrams. These diagrams are intended for presentation only.

The viewpoint must be chosen with care, based on the purpose. In the "machine shop" example given previously, only the foreman sees the interrelationships of the different tasks, so the model should be developed from the foreman's perspective.

It is important to maintain a single viewpoint throughout the model. The viewpoint should be a job title, department, or role (such as foreman or welder). As is the case for a purpose statement, a specific viewpoint is necessary to prevent drifting and continuous restructuring.

Models may need to be built from different perspectives in order to document all of the activities in detail.

## Scope

One of the primary benefits of constructing an activity model is the clarification of the *scope* of an entire system and its specific component activities. Although it is expected that the scope will be modified slightly during the modeling effort, it must be maintained to direct the modeling effort. As was the case with the purpose statement, without a defined scope it is difficult to know when the model is complete, as the scope of a model will tend to grow as the model grows.

The scope has two components: *breadth* and *depth*. The breadth of the model defines the lateral borders of the effort. Depth defines the level of detail of activity decomposition.

To facilitate the definition of an accurate scope, many IDEF0 modeling efforts spend considerable time developing the model's context diagram. A diagram may even be developed that represents one level above the context diagram to validate the larger system within which this system resides. It is well worth the extra effort since the context diagram becomes the anchor and central point of reference for the entire modeling effort because all changes to the context diagram cascade down to the decomposition diagrams.

When the scope statement is defined, material that will not be included in the model is clarified as well.

## Naming the System (Activity)

The recommended order of operations is first to define the model's purpose, then to decide the model's viewpoint, and finally, to identify the model's scope. The activity name that appears in the context diagram summarizes the scope statement. It is the highest-level activity in the model.

The context activity must be assigned an active verb phrase that is consistent with the model's scope statement. For this reason, broad verb phrases such as Manage Customer Service or Process Applications are often used as labels. It is expected that the model's scope will fluctuate slightly during the first few days of modeling. Purpose and viewpoint should be rigid from the start. To illustrate issues from different viewpoints, FEO diagrams can always be developed.

It is important to always begin with the definition of the purpose, viewpoint, and scope of the model. The rigid definition of these terms is critical to the entire modeling effort and thus worth a significant investment of time. Throughout the modeling project, the model's purpose, viewpoint, and scope will be continually referenced to guide the project.

## Defining the Major ICOMs

With respect to order of construction of the arrows in IDEF0 diagrams, it is often easier to start from the outputs, then move to the inputs, followed by the mechanisms and controls. Each activity exists for a specific function, and that function often has readily identifiable outputs. Should the outputs of a given activity be difficult to ascertain, it may be a sign that an opportunity exists to improve the business process.

### Defining Outputs

As the outputs are being identified, it is important to note that the model captures *if ever* scenarios. That is, if it is ever possible for a given situation to exist in the business, it must be modeled to show that possibility. Many new modelers forget to model as outputs the negative results of an activity. For example, the activity Administer Driver's Test will certainly produce an output of licensed drivers; however, we would expect that there would be another output arrow for failed applicants. Negative results are often used as feedback arrows and must be considered for every activity. Likewise, it is important to include questionable arrows in a diagram and let the business experts decide whether they should be included in the model.

## Defining Inputs

After the outputs are created, inputs should be considered. Inputs are specifically transformed or consumed by the activity to produce the outputs. In the manufacturing industry it is easy to note how raw material inputs are transformed and/or consumed into drastically different outputs. However, in the information industry, an input of data may initially appear not to be transformed or consumed at all. It is very rare that an input and an output arrow will be labeled exactly the same. Generally, this indicates that this activity is adding little value to the business, or the output was improperly labeled. The solution is to use adjectives to modify the nouns in the arrow labels to indicate the transformation that took place during the activity. For example, an input could be labeled "raw patient data," and the corresponding output could be labeled "verified patient data." The adjectives *raw* and *verified* modify *patient data* to clarify the transformation.

## Defining Mechanisms

After the outputs and inputs have been created, it is time to consider the mechanisms, or resources, applied to the activity. Mechanisms include people, machinery, computer systems, and so on. For example, an activity, Make Part, will often require some kind of machinery, such as a drill. In the Administer Driver's Test, a proctor administers the test. In the hospital example, some member of the clerical or nursing staff will verify the data, often with the help of another mechanism—computer equipment.

## Defining Controls

Finally, the controls that regulate the activity are added. Controls often are in the form of rules, regulations, policies, procedures, or standards. All activities in IDEF0 are required to have at least one control. Uncertain cases of input versus control status should default to control. Remember, controls are still a form of input to an activity.

When the context diagram appears to be complete and stable, ask the following questions:

- Does the diagram summarize the business activity to be modeled?

- Is the context diagram consistent with the scope, viewpoint, and purpose statements?

- Are the arrows at an appropriate level of detail for the activity? (As a guideline, you should limit the number of arrows to six per type.)

- Does the model have work group consensus?

## Numbering Activities and Diagrams

All IDEF0 activities are numbered. A prefix of any length can be used, but in almost all models A is used. Following the prefix is a number. The root activity is almost always numbered A0.

The prefix is repeated for each activity. The numbers are used to represent how detailed the activity is. The A0 activity is decomposed into A1, A2, A3, and so on. A1 is decomposed into A11, A12, A13, and so on. A11 is decomposed into A111, A112, A113, and so on. At each level of decomposition, another sequential digit is added. The only exception to this format is the first level, where A0 is decomposed not into A01, A02, and so on, but into A1, A2, and so on.

## Relationship of Diagram to Parent Activity (Boundary and Hierarchy)

An activity is decomposed if it is necessary to document it in more detail. When decomposing an activity, think about its life cycle. This will provide a list of candidate child activities. For example, the node "make cookies" can be thought to have the life cycle of "gather materials," "prepare batter," "bake dough," and so on.

In IDEF0 function modeling, it is important to realize that the border of a child diagram is the border of the parent activity. There is no difference. This has an important ramification. All work is performed in the leaf (lowest-level) activities. Unlike hierarchy as defined in structured programming, the higher-level activities are *not* controllers of the child activities. The children *are* the parent, only shown in more detail. Activities performed by the CEO can show up next to activities performed by line workers.

ICOM codes are placed at the end of *border arrows* in child diagrams to indicate where the corresponding arrow is on the parent diagram (Figure 3-19). They serve as a consistency check and can be helpful when the order of arrows in the child differs from that in the parent. An ICOM code consists of a letter, *I*, *C*, *O*, or *M*, and a number indicating the top-down or left-right placement of this arrow on the parent activity box, such as I1, C1, O1, and M1.
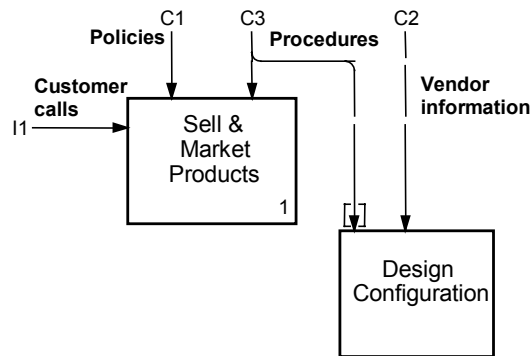
Figure 3-19. ICOM codes appear at the diagram's border.

## Breadth-first Versus Depth-first Modeling

Models can be built either *breadth-first*, in which each diagram is detailed as much as possible before decomposing, or *depth-first*, in which the activity hierarchy is first identified and then the arrows are created to connect them. Of course, both techniques may be employed during the construction of a single model, and an activity hierarchy will sometimes change slightly as the arrows are being drawn because drawing the arrows identifies some new insights into the structure.

## Knowing When to Stop

The diagram's purpose should indicate the kinds of questions that the IDEF0 model will answer. When these questions can be answered by the model, then the model has fulfilled its purpose and is considered complete. When developing the first-level decomposition, ensure that the activities appearing on the diagram are within the model's scope. Before decomposing an activity, ascertain whether the activity is covered in sufficient depth to fulfill the model's purpose. In addition, IDEF0 should only proceed to the point where precedence (output-input) arrows dominate the diagram. If necessary, IDEF3 models can then be used to model the detailed processes.

## Other IDEF0 Diagrams

In addition to the context and decomposition diagrams, other IDEF0 diagrams can assist in the development and presentation of the model.

### Node Trees

The node tree is an overview diagram of the entire model. Figure 3-20 is an example of a portion of the node tree for Operate QUILL Business. Normally, the top node corresponds to the context diagram activity, and the entire model hierarchy is developed below. However, you can designate any activity as the top node, with its children composing the rest of the tree. Due to the highly iterative nature of activity modeling in general, you should expect the node tree to be reworked a number of times before a stable version is created. Remember that the purpose, scope, and viewpoint statements of the context diagram are available as anchors. Without having those statements as a reference, the node tree could never reach a stable state. Viewing your model as a node tree helps you concentrate on the functional decomposition of the model, without regard for the flows that connect them.
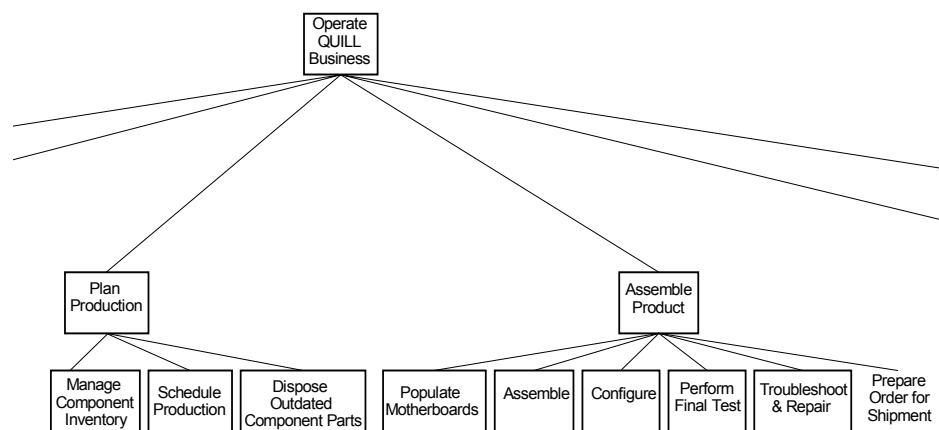


Figure 3-20. Part of a graphical node tree.

## FEO Diagrams

For Exposition Only (FEO) diagrams (pronounced *fee-oh*) are often included in models to illustrate other viewpoints or details not explicitly supported by the traditional IDEF0 modeling syntax. FEOs allow any and all of the rules and guidelines of IDEF0 to be broken for the sake of highlighting an important point or an area of the model. Of course, if a FEO diagram is created just to show a different viewpoint of the model, it would make sense to still follow the rules and conventions of IDEF0.

One use for a FEO diagram is to isolate an activity from its siblings, by creating a diagram with a single activity and the arrows which connect to its border, much like a context diagram. This can be useful in situations in which the facilitator is trying to collect information about the interface (arrows) of the function and the decomposition diagram is too cluttered to easily accommodate changes.
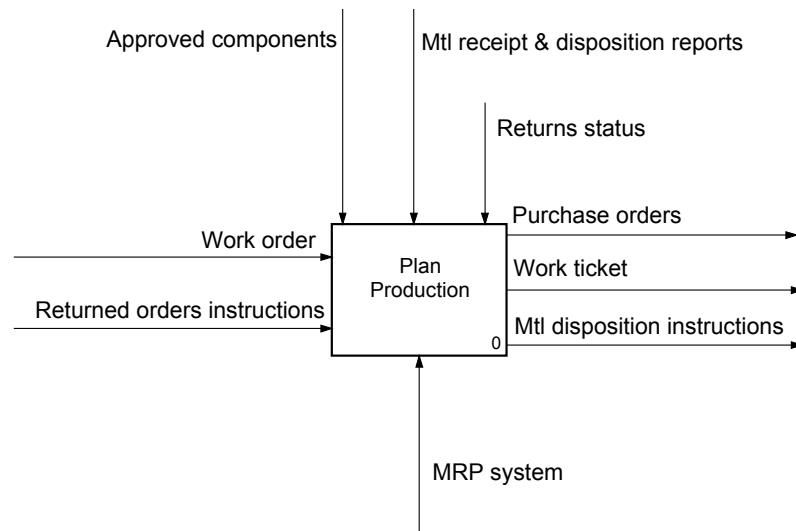


Figure 3-21. A FEO diagram that isolates a single activiy and the arrows connected to it.

Other common FEO diagrams are:

■  Alternate context diagrams for a child activity. This type of FEO diagram can be used to explore the potential impact of changes in the environment (context) on an activity.

■  A copy of a diagram that includes all of the activities, but only the arrows that connect to a chosen activity. This type of FEO diagram highlights the chosen activity's interaction with the other activities in the diagram.

■  A copy of a diagram that includes all of the activities, but only the arrows that directly represent the parent activity's main output. This type of FEO diagram highlights how the main input(s) transform into the main output(s).

■ Different viewpoints one level deep. This type of FEO diagram can be used to explore how important stakeholders view the system being modeled.

■ A copy of a diagram that includes some or all of the activities, and only those arrows needed to highlight a particular point. In this chapter, Figures 3-9, 3-16, 3-17, and 3-19 are all examples of this type of FEO diagram, because certain arrows and activities were removed to clarify the particular point being made.

# Data Flow Diagramming

## Overview

Like IDEF0, data flow diagrams (DFDs) model systems as a network of activities connected to one another by pipelines of objects. Data flow diagrams also model holding tanks called *data stores*, and *external entities* which represent interfaces with objects outside the bounds of the system being modeled. Figure 4-1 is a typical data flow diagram.
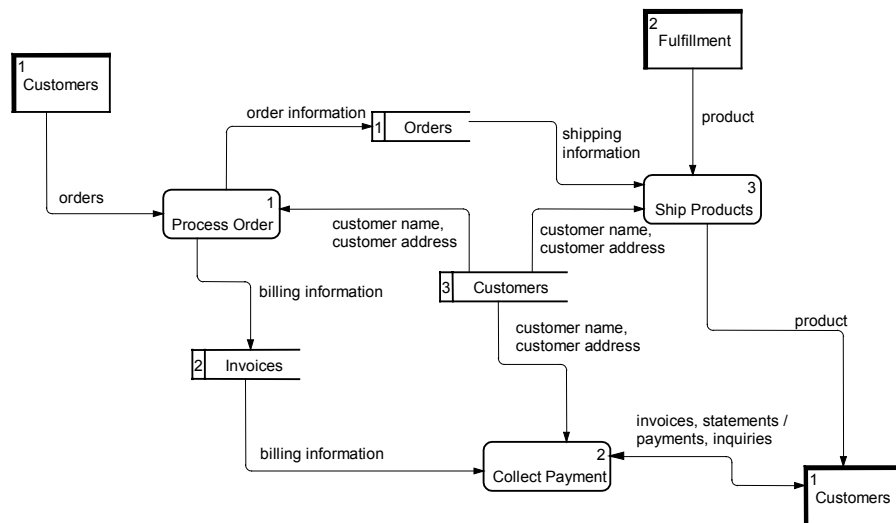


Figure 4-1. A typical data flow diagram.

In contrast to IDEF0 arrows, which represent constraining relationships, arrows in DFDs show how objects (including data) actually flow, or move, from one activity to another. This flow representation, combined with data stores and external entities, gives DFD models more resemblance to some of the physical characteristics of a system—that is, issues of object *movement* (data flows), object *storage* (data stores), and object *procurement* and *distribution* (external entities).

Data flow diagramming is mostly associated with the development of software applications because it originated for that purpose. The particular box-shape being used in the examples is that adopted by Chris Gane and Trish Sarson, authors of the Gane and Sarson DFD method. Activities are represented as boxes with *rounded* corners. The discussion equally applies to the Yourdon/DeMarco DFD method in which circles (also called bubbles) represent activities. The Yourdon/DeMarco DFD method was authored by Edward Yourdon and Tom DeMarco.

# DFD Model Syntax and Semantics

In contrast to IDEF0, which views systems as interconnecting activities, data flow diagramming views systems as nouns. The context data flow diagram often consists of an activity box and external entities. The activity box is usually labeled with the name of the system; for example, Quill Computer Business System. Figure 4-2 is a typical data flow context diagram.

The addition of external entities does not alter the fundamental requirement that a model needs to be built from a single viewpoint and must have a well-defined purpose and scope, as defined in Chapter 3.
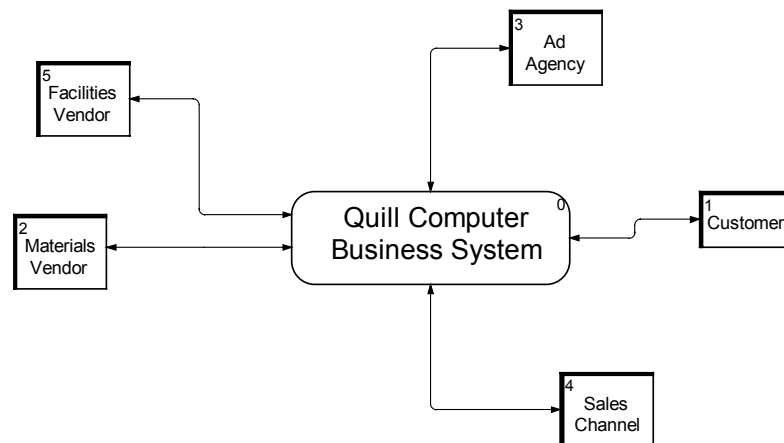
Figure 4-2. A typical data flow context diagram.

## Activity

A DFD activity represents a function that processes or transforms inputs to outputs. Although generally drawn as rounded-corner boxes, activities in DFDs are synonymous with activities in IDEF0 and IDEF3. Like IDEF3 activities, DFD activities have inputs and outputs, but do not support controls or mechanisms as arrows, as in IDEF0. In some implementations of Gane and Sarson DFDs, IDEF0 mechanisms are modeled as resources and depicted in the bottom of the box (Figure 4-3).
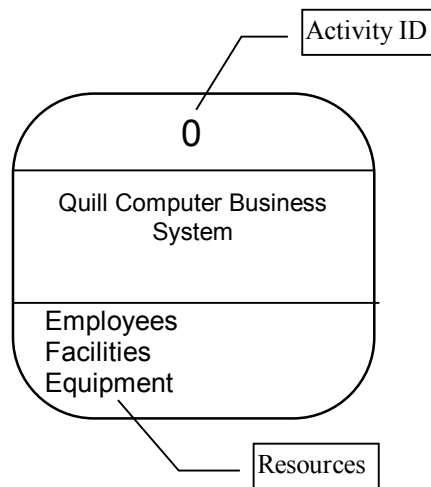
Figure 4-3. An example of Gane and Sarson data flow diagramming.

## External Entities

External entities provide inputs into the system, and/or receive the outputs. An external entity can both provide inputs (act as a supplier) and receive outputs (act as a customer). External entities are depicted as shadowed boxes and usually appear at the edges of a diagram. A single external entity (such as Customer) can appear multiple times in a single diagram. This is often used to reduce the clutter of long lines cutting across a diagram.
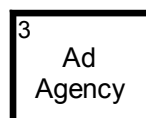
Figure 4-4. An example of an external entity.

## Arrows (Data Flow)

Arrows describe the movement (flow) of objects from one part of the system to another. Because DFD activity box sides do not have a dedicated function (as in IDEF0, where each box side has a specific meaning), arrows can originate from any part of the activity. DFDs also use a double-headed arrow to indicate a coordinated command-response dialogue between two activities, between an activity and an external entity, and between external entities. For example, Figure 4-5 illustrates a double-headed arrow that represents a coordinated interchange between Quill Computer Business System and Customer.
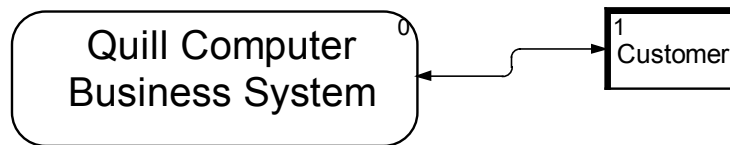


Figure 4-5. A two-headed arrow connecting an activity and an external entity.

## Data Store

Whereas flows represent objects in motion, data stores represent objects at rest. In a material-handling system, data stores are places where in-process work is inventoried throughout the factory, such as queues. In a data processing system, data stores represent any mechanism by which data is held for subsequent processing. Figure 4-6 illustrates a typical data store.
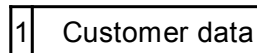


Figure 4-6. An example of a data store.

## Branching and Joining

A DFD arrow may be split (branched), and arrow segments may be relabeled to show the decomposition of the data being carried along the flow. Figure 4-7 shows an example of the customer information arrow splitting into three separate arrows.
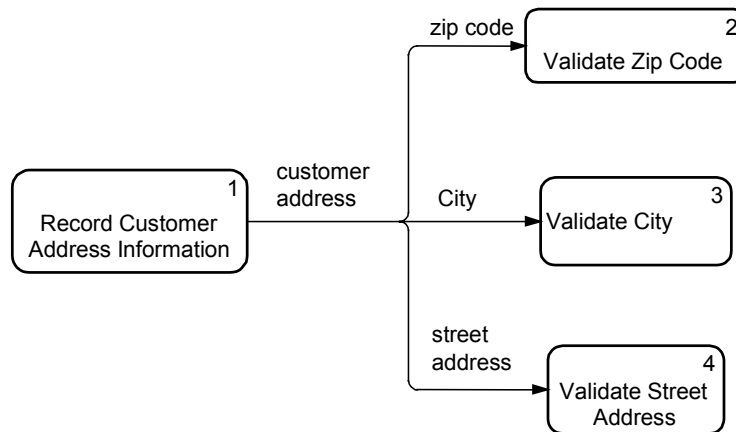
Figure 4-7. A split arrow that shows data decomposition.

Arrows may also merge (join) to form aggregate objects. Figure 4-8 shows an example.
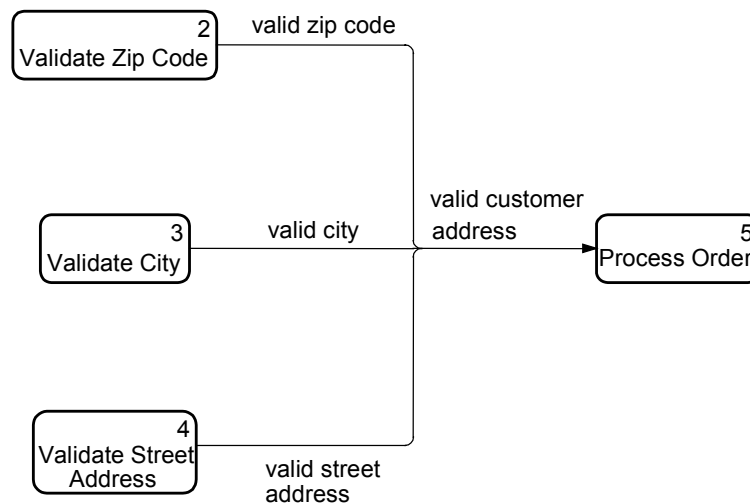
Figure 4-8. An example of arrows merging to form an aggregate object.

# Building a DFD

DFDs can be built using a traditional structured analysis and design approach similar to that described for IDEF0. A current physical model is first built to model the actual system that the user is currently using. Next, a current logical model is created to model the essential requirements of the current system. After this, a new logical model is created to model the essential requirements of the proposed system. Finally, a new physical model is created to model the implementation.

An alternate approach that has gained increasing popularity in software design is called *event partitioning*, in which several DFDs are built to model a system. First, a *logical model* is built to model the system as a collection of *activities* and document *what* the system should do.

Next, the *environment model* describes the system as an object that responds to events from external entities. This environment model usually consists of a statement of the system's purpose, a single context diagram, and an event list. The context diagram consists of a single box that represents the entire system, and the external entities with which this system will interact, that is, its environment.

Finally, the *behavior model* is created to model how the system will handle all of the events. This model begins as a single diagram, with one box representing each response to an event identified in the environment model. Data Stores are added to model data that must be remembered between events. Flows are added to connect the other elements, and the diagram is checked for consistency with the environment model.

A clean-up process is usually needed to reformat the model for presentations. Activity aggregation is used to create simplified parent diagrams, and decomposition is performed to improve clarity.

## Object Numbering

In a DFD, each activity number can include a prefix, parent diagram number, and object number. Figure 4-9 shows an example. The object number uniquely identifies the activity on a diagram. The parent diagram number and object number together uniquely identify each activity in your model.

**Diagram Number**

**Prefix**          **Object Number**

$$A \quad 12 \quad 5$$
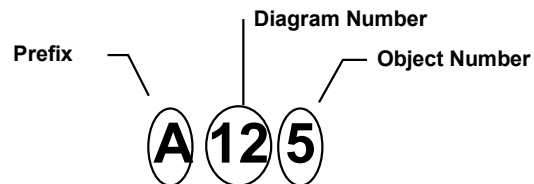
Figure 4-9. Components of a DFD activity number.

Unique numbers are assigned to each data store or external reference name, regardless of the location of the object in the diagram.

Each data store number can include the prefix D and a unique store number (Figure 4-10).

**Prefix**          **Object Number**

$$D \quad 5$$
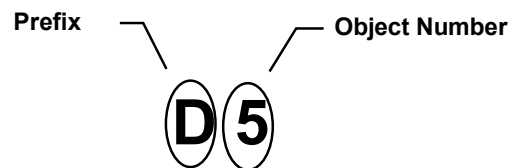
Figure 4-10. DFD data store numbering.

Similarly, each external reference can include the prefix E and a unique external entity number (Figure 4-11).

**Prefix**          **Object Number**

$$E \quad 5$$
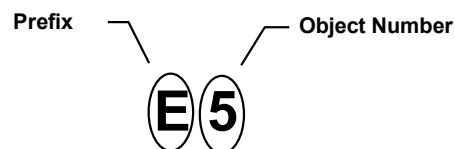
Figure 4-11. DFD external entity numbering.

# Activity-Based Cost and Performance Metrics

## Overview

Activity models can serve as the foundation for other modeling. For example, activity-based cost models relate the costs of building products to the activities used to create them and the material with which they were built.

In addition, activity models can serve as a foundation for building *simulation* models, which are used to explore time-varying aspects of a system. For example, a simulation model may be built to forecast the projected throughput rate of a new system, that is, the quantity of an object produced by the system over some period of time.

## Activity-Based Costing

Activity-based costing (ABC) is a technique for capturing and analyzing activity costs. ABC captures the costs of resources (such as materials, labor), allocates these costs to various activities, and then allocates activities to various system outputs, called cost objects. Compared to traditional cost accounting, which systematically under-costs low-volume products and over-costs high-volume ones, ABC provides a more exact calculation of the cost to produce a specific product based on the cost to perform all of the activities involved in creating it.

## An Activity-Based Cost Model

The design of an ABC system will, because it is activity-based, mirror a company's operation. Because an ABC system can be expensive to implement, it is important to focus on the significant costs a company incurs.

In Table 5-1, the resource (mechanism) costs are recorded per activity and product. The activity Drill, for example, requires 2 resources—Drill Press and Drill Press Operator. Product A requires 4 units of each resource and Product B requires 6 units.

| Activity Name | Resource | | Product A | | Product B | |
|---|---|---|---|---|---|---|
| | Name | Cost($) | Qty | Total ($) | Qty | Total ($) |
| Operations | | | | | | |
| Machining | | | | | | |
| Drill | Drill Press | $06.75 | 4 | $27.00 | 6 | $40.50 |
| | Drill Press Operator | $02.50 | 4 | $10.00 | 6 | $15.00 |
| Lathe | Lathe | $01.20 | 5 | $06.00 | 5 | $06.00 |
| | Lathe Operator | $02.50 | 5 | $12.50 | 5 | $12.50 |
| Route | Router | $12.45 | 3 | $37.35 | 9 | $112.05 |
| | Router Operator | $10.50 | 3 | $31.50 | 9 | $94.50 |
| Punch | Punch Press | $01.00 | 1 | $01.00 | 2 | $02.00 |
| | Punch Press Operator | $50.00 | 1 | $00.50 | 2 | $01.00 |
| Assembly | | | | | | |
| Rivet | Rivet Tool | $00.50 | 9 | $04.50 | 9 | $04.50 |
| | Rivet Tool Operator | $02.50 | 9 | $22.50 | 9 | $22.50 |
| Weld | Arc Welder | $14.85 | 1 | $14.85 | 2 | $29.70 |
| | Arc Welder Operator | $09.50 | 1 | $09.50 | 2 | $19.00 |

| Activity Name | Resource | | Product A | | Product B | |
|---|---|---|---|---|---|---|
| | Name | Cost($) | Qty | Total ($) | Qty | Total ($) |
| Bolt | Wrench | $00.01 | 19 | $00.19 | 19 | $00.19 |
| | Crewman | $01.50 | 19 | $28.50 | 12 | $28.50 |
| Screw | Screwdriver | $00.01 | 12 | $00.12 | 12 | $00.12 |
| | Crewman | $01.50 | 12 | $18.00 | 12 | $18.00 |
| Nail | Nail Gun | .$10.00 | 0 | $00.00 | 1 | $00.10 |
| | Crewman | $01.50 | 0 | $00.00 | 8 | $18.00 |
| Glue | Glue Gun | $00.10 | 6 | $00.60 | 8 | $00.80 |
| | Glue Man | $01.50 | 6 | $09.00 | 8 | $12.00 |
| Total Cost per Product | | | | $233.61 | | $436.96 |

Table 5-1. Activities, resources, and costs for two products.

In Table 5-1, the formula that links a resource to a product is:

```
Resource Cost x Quantity = Resource Cost for Product
```

These formulas are known as *cost drivers*. Products are often referred to as *cost objects*, and major organizational units, such as departments, are commonly referred to as *cost centers*

If the cost of raw materials (inputs) represents a significant percentage of the product's total cost, another table similar to Table 5-1 could be created to track the costs of raw materials for each product by activity.

Likewise, the cost to utilize what IDEF0 considers a control is required if the cost to utilize it represents a significant percentage of the activity's cost. Another table similar to Table 5-1 could be created.

Finally, it may be important to understand the cost of an activity's failure if this represents a significant cost, either in terms of destroyed work (scrap) or the need for rework. These costs need to be correctly allocated to the products. Another table similar to Table 5-1 could be created.

In addition to creating separate tables, the information could also be captured by adding rows to Table 5-1, intermingling resources, inputs, controls and outputs.

In a hierarchical activity model, costs are only applied to leaf-level activities (that is, activities that have not been further decomposed)—the costs of their parents being the sum of the costs of these leaf-level activities.

# Simulation

Simulation is a modeling technique used to study changes in the system that occur as a function of time. As a simulation advances with time, pertinent statistics are gathered about the simulated system in much the same way as is performed in real life.

When activity models are simulated, statistically relevant changes are associated with events. Simulation is performed, in effect, by jumping forward in time from one event to the next. This type of simulation is commonly known as *discrete event simulation*.

Simulation is usually associated with *operations research*, a decision science that attempts to determine the optimum (best) course of action with limited resources. However, the objectives and constraints of many real-world systems are difficult to express either quantitatively or mathematically. Alternatively, simulation models the system under study as a collection of elemental modules linked by well-defined logical relationships rather than as a (usually complex) mathematical formula. Compared to mathematical models, simulation models usually provide greater flexibility in defining objectives and constraints.

Simulation modeling does have two notable drawbacks. First, minute details could have significant impact on the outcome, thus necessitating a rather expensive detailed modeling effort, and second, simulations can execute for considerable periods of time even on high-performance computer systems.

The relationship between simulation models and activity models is synergistic: activity models can be translated into a skeleton (incomplete) simulation model, and simulation models greatly increase our understanding of the performance of some system, perhaps resulting in changes being made to the original activity models.

A description of the major components of a simulation model follows.

## Sources and Destinations

*Sources* model the arrival of inputs and are similar in concept to external entities in data flow diagrams. The inter-arrival rate (time between arrivals) is recorded as a mathematical expression, usually a statistical distribution function such as *normal (mean, standard deviation)*.

*Destinations* are also like external entities, but in simulation, their role is generally that of an information-gathering device. Destinations capture information related to the journey of an object through the system, which can include the total time to transform the object from input to output, the total cost to produce the output, and so on.

## Queues

Simulation shares another construct with DFDs—the data store, known in simulation as a *queue*. A queue is a way to model the characteristics of one or more objects waiting to be processed. Activity cycle time or *duration* is the time it takes an activity to execute. Activity duration may vary from one iteration to the next, causing fluctuations in output. For example, consider two activities that normally all take about the same length of time. Sometimes, an earlier activity executes faster than the later activity, which may cause partially finished goods to pile up in the queue.

Queue behavior must be encoded in the simulation. For example, a queue can function as a stack—the last item placed on the stack is the first to be retrieved. This is known as last-in-first-out (LIFO). Alternatively, a waiting line generally operates on a first-in-first-out (FIFO) basis. A queue can also operate randomly, or according to some other formula.

When IDEF0 and IDEF3 models are simulated, a queue for each input and control is assumed.

## Facilities

Facilities model the activities of the system. The term *facility* reflects the manufacturing roots of simulation in which an activity is performed at some work station (usually a dedicated piece of machinery) and the materials moved from one station to the next in assembly-line fashion. The processing time is recorded as a mathematical (usually statistical) expression, and rules for the use of resources (mechanisms) are specified.

## Simulation Example

In this example, we will model a bank lobby. We are trying to understand how to best allocate bank tellers to perform a variety of transactions (each requiring the teller to perform a different set of activities). Five teller transactions are handled at this bank, outlined as follows:

Deposit into Customer Account

1. If a check is being deposited, verify check, then record deposit.

2. If cash is being deposited, count the cash, record till deposit, and deposit the cash in till.

3. If a large amount of cash is being deposited, use the cash counter.

4. If cash is to be returned to the customer, withdraw money from the till, count it, and give it to customer.

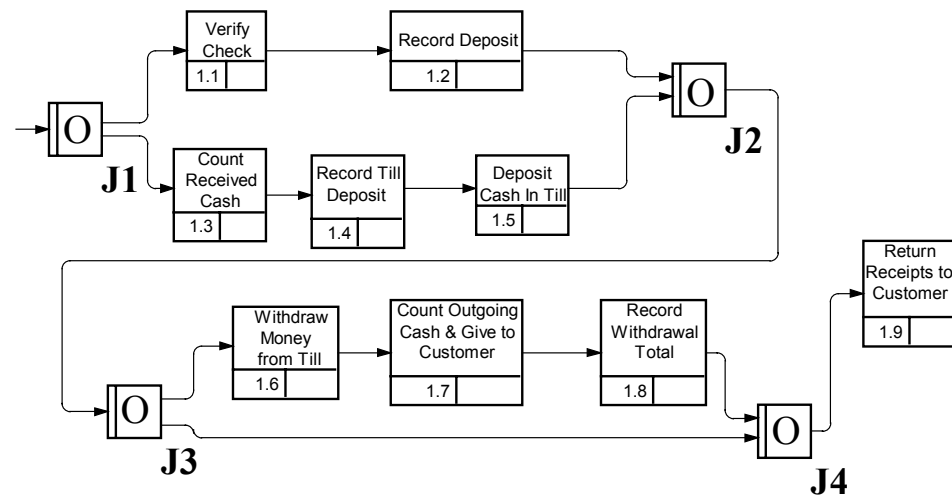5. Record the withdrawal total.

6. Return receipts to the customer.



Figure 5-1. Deposit into customer account.

Withdrawal from Customer Account

1. Verify account balance.

2. Withdraw money from the till, count it, and give it to the customer.
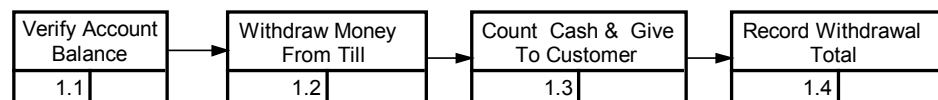
3. Record the withdrawal total.



Figure 5-2. Withdrawal from customer account.

Transfer among Customer Accounts

1. Verify source account balance.

2. Record withdrawal.

3. Record deposit.

4. Return receipt to the customer.
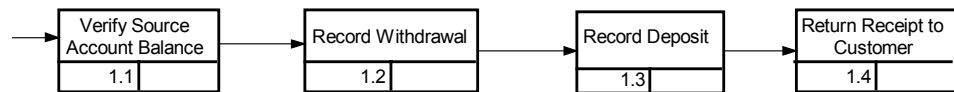


Figure 5-3. Transfer among customer accounts.

Issue Cashier's Check

1. If withdrawal, verify account balance. Record withdrawal.

2. If a cash transaction, verify customer status, verify checks, and count cash.

3. Print and issue check.

In Figure 5-4, we can see that there are two general transaction types based on how the user pays in exchange for a cashier's check: the user will either withdraw money from an account or pay with cash. As shown by the OR junction, a user can pay with cash, check, or both. If there are significant differences between the three types of cash transactions in the amount of time (and cost) to perform them, then it is important to model all cases for the simulation to be accurate. Likewise, if transaction times for business accounts varied considerably from personal accounts, this also would need to be incorporated into the model.



Figure 5-4. Issue cashier's check.

Create New Accounts

Create new accounts is not explicitly modeled in this example. Rather, a formula is used. It is important to keep in mind that it is not always necessary to model all activities to the same level of detail—only create a model in sufficient detail to fulfill the model's purpose.
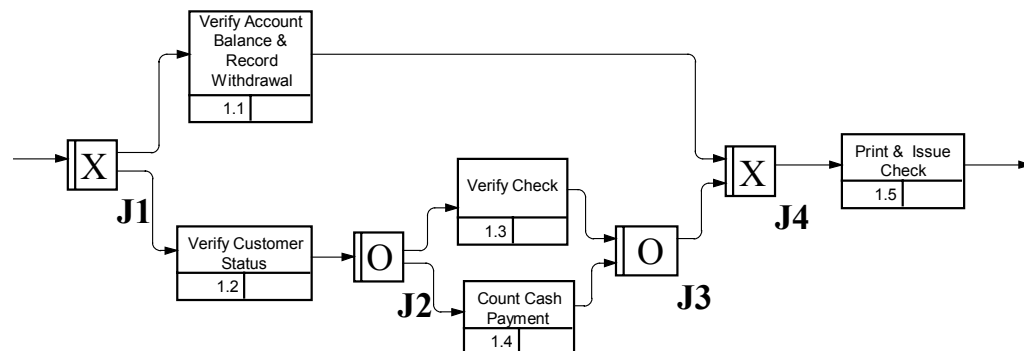
The important simulation variables are shown below in table format. Table 5-2 describes the resources (IDEF0 mechanisms) needed to perform the activities.

| Resource Name | Number Available | Cost per Minute | MTBF | Downtime |
|---|---|---|---|---|
| Teller | 3 | $0.20 | 0 | 0 |
| Cash Counting Machine | 1 | $0.05 | Normal (6000,25) | Normal (16,3) |

MTBF = mean time between failure.

Table 5-2. Resources.

In the table, MTBF is used to record situations where something is periodically unavailable due to failure. Downtime measures how long the resource is typically out of commission. If we needed a more accurate model of the availability of individual tellers, such as would be required to simulate several days worth of activity, then a resource calendar can be used to document the availability schedule.

In statistics, certain formulas have been developed to describe the probability of some phenomenon. For example, on average, how many customers enter a particular bank lobby between 12:00 and 1:00 p.m. on Friday? The average, called the *mean,* is represented on the three distributions in Figure 5-5 by the peak of each curve. The margin of error, that is, how close together are the values of all the samples, is called the standard deviation, and is represented by the width of each curve. There are many probability curves other than the normal distribution shown in Figure 5-5 that together describe a wide variety of situations. Graphically, the shapes of these other probability curves will vary substantially from that shown in Figure 5-5.
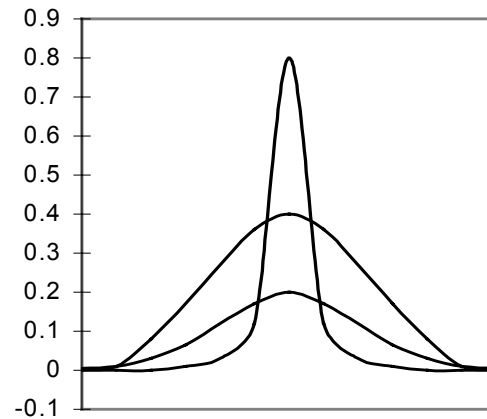
Figure 5-5. Three normal probability distributions with the same mean value but different standard deviations.

These probability formulas are at the heart of simulation. In addition to representing the range of execution times of activities, statistical probability formulas are also used to represent the range of time between arrivals, and the availability of resources. When the simulation engine is run, the software that actually executes the simulation continuously utilizes these formulas to determine activity execution times, arrival rates, and resource availability.

As Table 5-3 on the following page shows, the time to perform each activity is represented as a statistical probability formula. For example, the time to perform Verify Check and Record Deposit is distributed normally and takes .5 minute to perform, on average. This is the first number in the formula. It doesn't always take .5 minute to complete this activity; this is just the average time. We can also see that, except on rare occasions, this transaction is completed within the range of .4 minute and .6 minute. We calculate this by taking the second number in the formula—the standard deviation—and adding it to the mean to get .6, and subtracting it from the mean to get .4.

| Activity Name | Resource Name | Resource Quantity | Time |
|---|---|---|---|
| Deposit into Customer Account | | | |
| Verify Check and Record Deposit | Teller | 1 | Normal(.5,.1) |
| Count Small Cash Deposit | Teller | 1 | Normal(.5,.5) |
| Count Large Cash Deposit | Teller | 1 | Normal(1,.5) |
| | CCM | 1 | |
| Record Till Deposit and Deposit Cash in Till | Teller | 1 | Normal(1,.5) |
| Withdraw Money from Till | Teller | 1 | Normal(.3,.1) |
| Count Small Cash Return | Teller | 1 | Normal(.5,.5) |
| Count Large Cash Return | Teller | 1 | Normal(1,.5) |
| | CCM | 1 | |
| Give to Customer | Teller | 1 | Normal(.3,.1) |
| Record Withdrawal Total | Teller | 1 | Normal(.3,.1) |
| Return Receipt to Customer | Teller | 1 | Normal(.3,.1) |
| Withdrawal from Customer Account | | 1 | |
| Verify Account Balance | Teller | 1 | Normal(.5,.3) |
| Withdraw Money from Till | Teller | 1 | Normal(.3,.1) |
| Count Small Cash Return | Teller | 1 | Normal(.5,.5) |

| Activity Name | Resource Name | Resource Quantity | Time |
|---|---|---|---|
| Count Large Cash Return | Teller | 1 | Normal(1,.5) |
| | CCM | 1 | |
| Give to Customer | Teller | 1 | Normal(.3,.1) |
| Record Withdrawal Total | Teller | 1 | Normal(.3,.1) |
| Transfer among Customer Accounts | | | |
| Verify Source Account Balance | Teller | 1 | Normal(.5,.3) |
| Record Withdrawal | Teller | 1 | Normal(.3,.1) |
| Record Deposit | Teller | 1 | Normal(.3,.1) |
| Return Receipt to Customer | Teller | 1 | Normal(.3,.1) |
| Issue Cashier's Check | | | |
| Verify Account Balance | Teller | 1 | Normal(.5,.3) |
| Record Withdrawal | Teller | 1 | Normal(.3,.1) |
| Verify Customer Status | Teller | 1 | Normal(.3,.1) |
| Verify Checks | Teller | 1 | Normal(1,.5) |
| Count Cash | Teller | 1 | Normal(.5,.5) |
| Print and Issue Check | Teller | 1 | Normal(1,.5) |
| Open New Account | Teller | 1 | Normal(15,6) |

CCM = Cash-counting machine.

Table 5-3. Resource utilization over time.

Arrivals (sources) can be modeled in one of two ways: a separate arrival node for each customer type or one arrival node with the different activity types represented as probabilities. Each representation has advantages with respect to complexity and accuracy. We will represent arrivals as a single node with probabilities for each transaction type.

First, we define the arrival rate as Normal (1.2, 0.3). This means that some customer will arrive on the average of once every 1.2 minutes. Had the arrival rate varied over time, we could have created a table of arrival rates, each valid for a certain time period, similar to how we would have defined the availability of resources.

According to Table 5-4, 40% (.4) of all transactions are deposits. Of these, half are check-only deposits, 30% are small deposits, and 20% are large deposits. Similar values indicate the probabilities of the other activity types.

| General Activity Type | Special Type | Probability |
| --- | --- | --- |
| Deposit into Customer Account | | .40 |
| | Check Only | .50 |
| | Small Cash Deposit | .30 |
| | Large Cash Deposit | .20 |
| Withdraw from Customer Account | | .25 |
| | Small Cash Return | .75 |
| | Large Cash Return | .25 |
| Transfer among Customer Accounts | | .15 |
| Issue Cashier's Check | | .15 |
| | Withdrawal | .80 |
| | Cash | .20 |
| Create New Accounts | | .05 |

Table 5-4. Arrivals.

To define queue behavior, the modeler must make some choices about the physical implementation. For example, each General Activity Type could have its own queue, or there could only be one queue (or some in-between arrangement). We will assume one queue.

In Table 5-5, we assume that the queue has an unlimited capacity. However, it might be useful to put limits on the queue and see if any customers are lost because the queue is too long (that is, if the line looks too long, customers will not wait).

| Queue Name | Behavior | Capacity |
|---|---|---|
| Incoming Queue | FIFO | 0 |

Table 5-5. Queue behavior.

Finally, we need to identify what information we want to capture about each of the important components. The choice of output variables to monitor depends on the purpose for building the model. For our model, the most obvious measures are the wait times for customers in line and the queue length. However, simply recording the mean and standard deviations is not good enough. We need to see how the queue size and wait time vary over time.

The process that consumes queue 1 in Figure 5-6 is not keeping up; the queue gets longer and longer. The size of queue 2 varies within a range. If the range is not acceptable, that is, the queue is getting too long at certain times, further analysis is needed to understand why. Perhaps a better resource allocation scheme is required to keep up with peak periods. Queue 3 (and the activities that go along with it) is a candidate for elimination if the supplier and consumer can coordinate their schedules a little better.
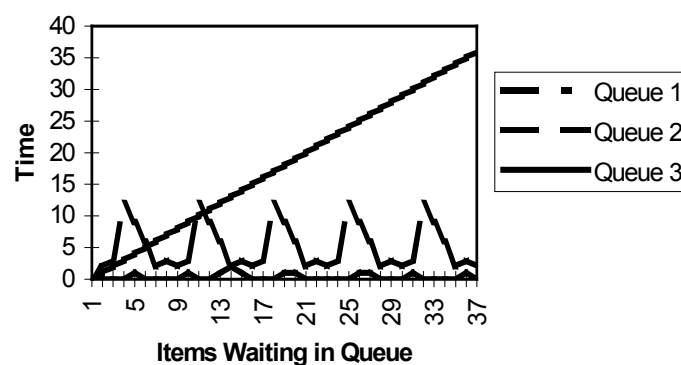


Figure 5-6. The sizes of three different queues over time.

In this case, resource costs are also important because we are trying to identify the smallest number of tellers and cash-counting machines that can handle the workload. We want to understand the amount of time that each of the resources was idle in order to calculate the idle cost (cost for the resource to do nothing), as well as the amount of time the teller was occupied performing each transaction type. We can also capture the amount of time a teller was waiting for a cash-counting machine. All of this information is captured by developing formulas in the simulation to compute the results.

## Managing Simulation Experiments

A simulation is not run just once. The same model may be simulated many times in order to obtain a more accurate representation of performance. In addition, numerous experiments are run on the same basic model, each time manipulating one variable (such as the number of cash-counting machines). In Table 5-6, we start off with the smallest number of tellers, run a simulation experiment, and then and increase the number of tellers by one. We then increase the number of cash-counting machines and run another simulation experiment to explore the effect. This process is repeated, increasing the number of tellers and/or the number of cash-counting machines, until we finally run an experiment with four of each.

| Experiment | Tellers | Cash-counting Machines |
| --- | --- | --- |
| 1 | 1 | 1 |
| 2 | 2 | 1 |
| 3 | 2 | 2 |
| 4 | 3 | 1 |
| 5 | 3 | 2 |
| 6 | 3 | 3 |
| 7 | 4 | 1 |
| 8 | 4 | 2 |
| 9 | 4 | 3 |
| 10 | 4 | 4 |

Table 5-6. Ten simulation experiment variables.

# Integrating Activity and Data Models

## Overview of Data Modeling

Just as activity models view a system as a collection of activities connected by objects, data models view a system as a collection of objects connected by relationships. Data and activity models can be integrated, which helps to integrate related activity models. Integration of models is a valuable mechanism for ensuring consistency (of name usage) and completeness among the integrated models. Just as there are several similar activity methods, there are several data modeling methods. The most popular are IDEF1X and Information Engineering (IE).

## Entities and Attributes

Data models use a box and line notation similar to activity models. The boxes are fairly easy to understand. The top of the box lists the name of the object (called an *entity*) about which we have some interest. An entity name is always a noun. An entity's *attributes* are listed inside the box. Attributes are the specific properties of the entity. Attribute names are also nouns and they further describe the entity.

Data is often described in terms of a series of rows that contain one or more named columns—much like a spreadsheet. Consider, for example, the entity CUSTOMER. Each piece of data is stored in a single field, or *attribute* (for example, name, city, and state are attribute names). Each instance of a customer is contained in a single *row* of data. The entire collection of rows is a table, or *entity* (for example, CUSTOMER is an entity name).

In Figure 6-1, we see that the entity name (CUSTOMER) is located at the top of the entity box. Although the entity CUSTOMER represents many customers, entity names are always singular. In addition, entity names are usually always written in all capital letters. The entity's attributes are listed inside the box. A horizontal line separates the attributes into two kinds; *primary key* attributes and attributes. Primary key attributes are listed above the line in what is called the *primary key area* of the entity box, while attributes are listed below the line in the *non-key area* of the entity box. The set of primary key attributes in a single entity must uniquely identify a single instance of the entity, for example, a single CUSTOMER. Sometimes, when it is not easy to find a good set of attributes to serve as a primary key, an attribute is added to serve solely as an identifier. Such a case is shown in Figure 6-1. The primary key attribute is called Customer ID. Attribute names are always singular and are usually written with an initial capital letter.

CUSTOMER

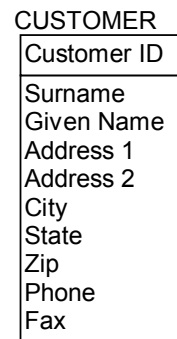| Customer ID |
|---|
| Surname |
| Given Name |
| Address 1 |
| Address 2 |
| City |
| State |
| Zip |
| Phone |
| Fax |

Figure 6-1. A data modeling entity.

## Relationships

Entities are associated with one another by *relationships*. For example, we might have an entity, called ORDER, that stores information about all of the orders placed by our customers. In Figure 6-2, we see the relationship verb phrase <places> associates CUSTOMER with ORDER. The relationship can be easily read as a sentence "CUSTOMER places ORDER." Relationship verb phrases are singular and usually written in all lowercase letters.

The relationship shown in Figure 6-2 is usually called a *parent-child* relationship. One parent, in this case CUSTOMER, places many ORDERs, just as a human parent can have many children. We also see that the attribute Customer ID appears in the ORDER entity. The primary key of the parent entity migrates along the relationship to the child entity. In this way, it is possible to identify the customer who placed the order. In the child entity, migrated attributes are collectively referred to as *foreign keys*.

## Cardinality

But a CUSTOMER places *how many* ORDERs? In this example, the answer is probably "as many as possible." *Cardinality* refers to the number of children and parents involved in a relationship. Is a customer really a customer if they have yet to place an order? If the answer is yes, then the places relationship can be read "one (CUSTOMER) places zero or more ORDERs." Figure 6-2 shows this relationship.
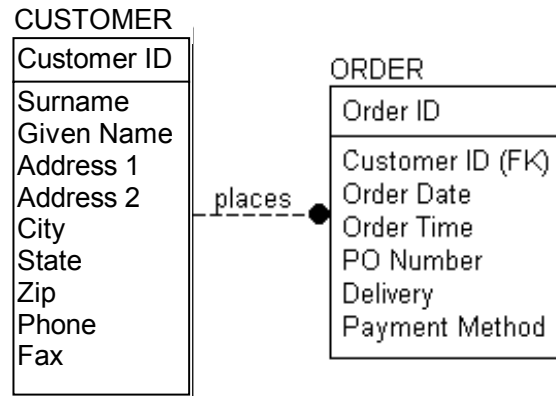
CUSTOMER

| Customer ID |
|---|
| Surname |
| Given Name |
| Address 1 |
| Address 2 |
| City |
| State |
| Zip |
| Phone |
| Fax |

places ●

ORDER

| Order ID |
|---|
| Customer ID (FK) |
| Order Date |
| Order Time |
| PO Number |
| Delivery |
| Payment Method |

Figure 6-2. One to zero or more cardinality.

If the answer is no, then the places relationship would be read "one (CUSTOMER) places one or more ORDERs." To distinguish this cardinality from the other, a capital "P" is placed next to the ball at the child end of the relationship. This is shown in Figure 6-3.

CUSTOMER

| Customer ID |
|---|
| Surname |
| Given Name |
| Address 1 |
| Address 2 |
| City |
| State |
| Zip |
| Phone |
| Fax |

places ●
P

ORDER

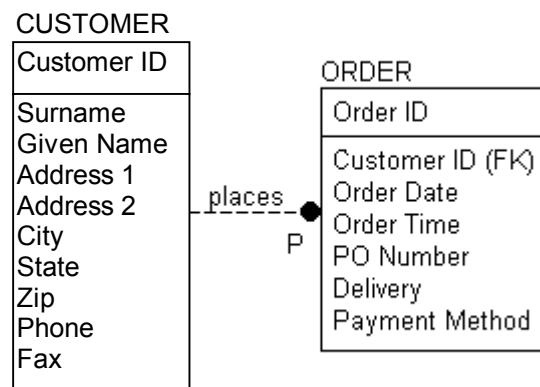| Order ID |
|---|
| Customer ID (FK) |
| Order Date |
| Order Time |
| PO Number |
| Delivery |
| Payment Method |

Figure 6-3. One to one or more cardinality.

The letter Z is shown if the relationship cardinality is one to zero or one. If the cardinality is an exact number, then that number is shown.

## Identifying and Non-Identifying Relationships

In Figure 6-2, Customer ID appears in the ORDER entity as just another attribute. Sometimes, however, it is necessary for the primary key attributes of the parent to be part of the child's primary key. Such a case is shown in Figure 6-4. The primary key of LINE ITEM is actually made up solely of foreign keys—Order ID from the ORDER entity and Product ID from the PRODUCT entity. The rounded shape of the LINE ITEM entity box highlights the fact that this entity depends on other entities for its primary key. The relationships between ORDER and LINE ITEM and PRODUCT and LINE ITEM are called *identifying* relationships, because the primary keys of these parent entities help to identify a particular instance of a child. Note also that these relationship lines are solid lines, to distinguish them from *non-identifying* relationships, which appear as dashed lines.
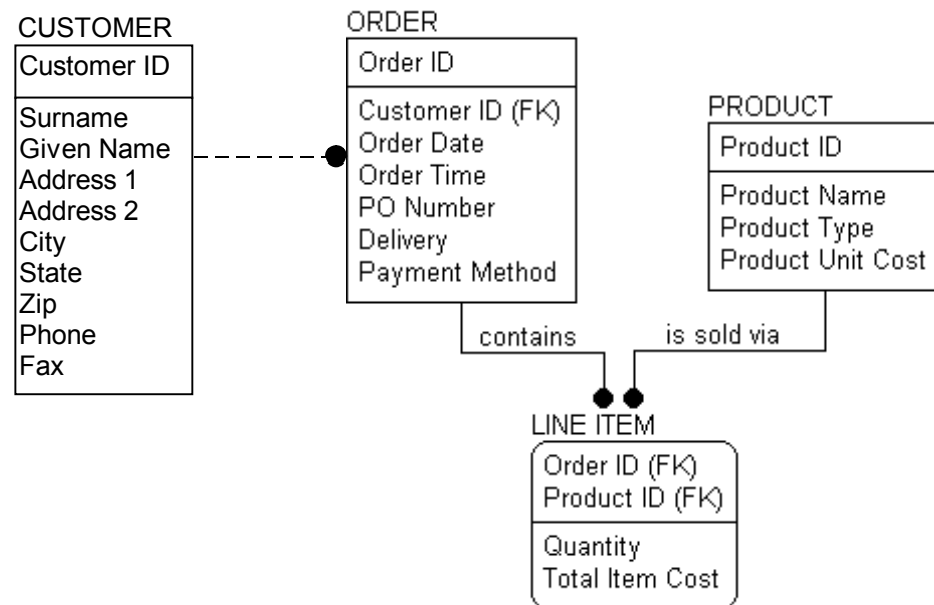


Figure 6-4. Identifying and non-identifying relationships.

## Optional Relationships

Identifying relationships are always mandatory, that is, each instance of the child entity must be associated with an instance of the parent entity. Otherwise, part or all of the child's primary key would be null (that is, have no value). In the case of non-identifying relationships, however, there may be times when it is unnecessary for each instance of the child to be associated with an instance of the parent. This kind of relationship is said to be *optional* and is denoted by a diamond at the parent end of the relationship.

An example of an optional non-identifying relationship is shown in Figure 6-5. Some products, because of their small size, are stored in bins, others are not. The relationship in Figure 6-5 is read "zero or one PRODUCT is stored in zero or one BIN." The first "zero or one" comes from the diamond at the parent end of the relationship, which designates this relationship as being optional. The second "zero or one" comes from the "Z" at the child end of the relationship.
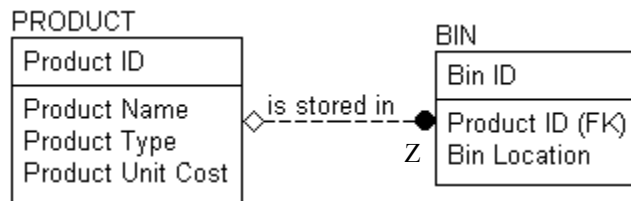


Figure 6-5. Optional relationship.

## Categories

In addition to parent-child relationships, data modeling supports *category* relationships, also known as subtype/supertype, or hierarchy relationships. Figure 6-6 illustrates a simple category. The entity SELLABLE ITEM has two subtypes—PRODUCT and SERVICE. Note that the primary key attributes of these entities are identical to each other but the non key attributes are different. This is precisely the situation for which categories are intended. The circle with the two lines underneath denotes a category relationship. Note that category relationships are not named. Instead, "is a" is the implied name and a category relationship can be read from the subtype to the supertype. For example, the relationship in Figure 6-6 is read "PRODUCT *is a* SELLABLE ITEM."
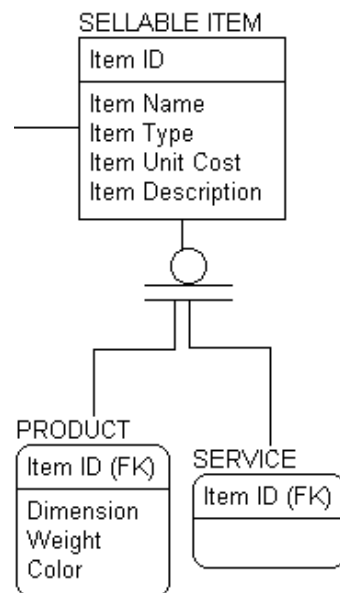


Figure 6-6. A complete category.

Actually, Figure 6-6 is an example of a *complete* category. All of the subtypes are known and shown in the diagram. In other situations, however, we either don't know about or don't need to include some of the subtypes. In this case, we have an *incomplete* category, which is denoted by a single line underneath the circle. How we know to which subtype a particular instance belongs is determined by an attribute known as the *discriminator*, which can be any attribute in the supertype.

# Data Usage

In an activity model, an arrow can map directly to a single attribute of an entity, an entity and some or all of its attributes, or multiple entities and some or all of their attributes. Because the same arrow name maps to the same set of attributes and entities, the interface between an activity and an arrow represents objects being used by, or created by that particular activity. In the case of those arrows that represent data, activities use or create specific instances of entities and attributes.

## Arrow Rules

Data cannot be used arbitrarily by an activity. Only certain operations (create, retrieve, update, or delete) are permitted for each of the four arrow types in an IDEF0 model. The rules are summarized as follows:

- Input arrows represent objects that the function transforms into output or consumes. For data to be transformed, it must always be retrieved. In an activity model, input entities must always be updated and/or deleted. In fact, this test can be used to determine if an arrow is actually an input or a control. Input data can never be created by the activity. However, the activity can create new instances of the same entities, which is documented on the output arrows.

- Control arrows represent the conditions required to produce correct output. For data to control an activity, it must be retrieved. Control data can never be created, updated, or deleted by the activity.

- Output arrows represent objects created and/or modified by the activity. If an output represents data that is being updated, it must contain data that is provided by a corresponding input arrow. Output arrows may also represent data that is being created but do not usually represent data that is being retrieved or deleted. It is possible, and sometimes desirable, to model an output arrow that represents deleted (disposed) data.

- IDEF0 mechanism arrows represent the means used to perform an activity and may not correspond with elements in the data model that is being developed. It may, however, correspond with elements in another data model that models the support activities of the organization.

It is possible that an arrow in the activity model was designated as an input but, upon integration with the data model, was discovered to be a control. The converse is also possible.

## Entity and Attribute Rules

There are also usage rules for entities and attributes, which are summarized in Table 6-1 for each operation. There are slight differences between the operations performed on entities and those performed on their attributes (Table 6-1). The acronym for the allowed operations on entities is CRUD and for attributes it is IRUN. These terms are often used instead of the term "data usage."

| Entities | Attributes | Differences |
|----------|-----------|-------------|
| Create | Insert | Only entities can be created; attribute values are inserted into their owner-entity. |
| Retrieve | Retrieve | No difference. |
| Update | Update | No difference. |
| Delete | Nullify | Only entity instances (entire rows, or records) can be deleted. Deleting an attribute actually consists of setting the attribute's value to NULL (that is, having no value). |

Table 6-1. Differences between operations performed on entities and attributes.

### Create/Insert

When an entity instance is created, a new row is added to the database. In order for the database management system (DBMS) to accept it, the new record must meet certain requirements. Since data entities are often related, adding an instance to one entity may require that you add an instance to others. These additional entity instances must also satisfy all of the appropriate requirements. In some cases, you may need to insert instances in several entities to complete your transaction.

For example, if the entity being inserted is the generic parent in a complete subcategory, then an instance will have to be inserted in one of the category members—exactly which one being determined by the value of the category's discriminator. If the entity being inserted is the generic parent in an incomplete subcategory, then an instance *may* have to be inserted in one of the category members, depending on the value of the category's discriminator. If, when inserting an instance in a dependent child entity, the values of the foreign key attributes do *not* match the values in any instance of the parent entity, you must first insert an instance containing these values in the parent.

Some DBMSs, particularly relational DBMSs, automatically add instances to other entities for you in certain cases or disallow or reject the addition of a record. This information is sometimes documented in the information model and is known as *referential integrity*. You must ensure that the actions of the DBMS are appropriately specified for your particular business situation.

Special rules exist for attributes. Certain fields are mandatory and must have a value. At a minimum, this includes all attributes that are part of the entity's *primary key* (a set of attributes that uniquely identifies a single row in a table and is used to relate this row to other tables) as well as *foreign keys* (a set of attributes that relates this row to primary key attributes in another table) from mandatory relationships. In addition, since every attribute is of a certain data type, its values must be appropriate to that type. For instance, an attribute of type DATE must contain values that are valid dates.

Certain attributes are further restricted to specific values or a range of values (commonly referred to as the attribute's domain). For example, male and female constitute the domain for the attribute gender. Foreign key attributes are similarly restricted because their values must match values in the entity to which they relate.

## Retrieve

There are no special rules for retrieving data. It is common practice for modelers to mark an entity as being retrieved and to *not* mark the individual attributes of that entity as being retrieved. The underlying assumption is that unless a subset of an entity's attributes are specifically marked as being retrieved, *all* of the attributes of that entity are being retrieved. The benefit to this approach is that it shortens the task of assigning usage, and the disadvantage is that it may hide important usage information and thus hamper proper data usage analysis. It also increases the difficulty in determining whether the task of recording data usage is complete.

## Update

When an entity instance is updated, the record must meet the same requirements as for a new record. Updates involving the primary key are analogous to deleting a row and adding a new one (since one of the rules for a primary key is that a new key is, in fact, a new instance; this enables you to track an object through its life without losing relevant information).

Special rules exist for attributes. When an entity instance is updated, each attribute may be updated (that is, the value is changed to another), nullified (that is, the attribute no longer has a value), or not affected. The value of an updated attribute must be valid—that is, part of the attribute's domain. If an attribute is nullified, this must be permitted in the database schema, as described in the section on Create/Insert.

### Delete/Nullify

Just as adding a record may require the addition of other records, deleting a record may require the deletion of other records. Some DBMSs, particularly relational DBMSs, automatically delete instances in other entities or restrict deletion if it would invalidate referential integrity. Just as in the case of creating new records, you must ensure that the actions of the DBMS are appropriately specified for your particular business situation.

Special rules exist for attributes. When a row is deleted, all of its attributes' values are deleted and need not be individually specified. Otherwise, the value of individual attributes can be set to NULL, which literally means "no value."

### Presenting Data Usage Reports

Data usage reports are often created in matrix format in which the activity names are located in rows on the left side of the matrix, entity or attribute names in columns at the top of the matrix, and the data usage (the letters CRUD or IRUN) appearing at the appropriate intersection of the rows and columns. Such a report is referred to as an *affinity matrix*.

# The Integration Process

Before an activity model is built, it is important to understand the model's purpose, that is, why we are building this model. By and large, the purpose of a model is to answer a set of questions. If an activity model is to be integrated with a data model, then it is important to note this fact as part of the model's purpose. Documenting this fact will keep the issue in the minds of the modeling team and will probably prevent serious incongruities between the models.

The first step, if not already done, is to rationalize and standardize the names used in the models. For example, customer must always mean the same thing regardless of where it appears in any activity or data model. Because of the difficulty of this step, and the enormous consequences, coordination of the model building activities is essential.

Integration of models cannot be practically performed as an "after modeling" activity. Standardizing names across the various models should ideally occur as part of the modeling process. This is true regardless of the kinds of models being integrated. However, the modeling teams must make sure that the names used in a given activity model accurately represent the system from the chosen viewpoint. Cross-references can be developed to map homonyms (that is, two words spelled and pronounced alike but having different meanings) and other relationships. This is not to say that the models must be developed concurrently. Rather, the development of the models must be coordinated in order to reuse the results of previous efforts.

The second step is to associate entities and attributes from the data model with arrows in the activity models. If the activity models are in sufficient detail, then some of the arrows will be named the same as entities and, in a few cases perhaps, attributes. It is possible for most, if not all, of the arrows to be aggregates of several entities in the data model—particularly when the activity models are business models (as opposed to computer systems models). Arrow bundling, which is largely a construct for simplifying diagrams by reducing the number of parallel arrows, creates even larger aggregates.

Finally, the lowest-level activities are analyzed, and the data usage is assigned to the entities flowing in and out along the arrows.

Of course, it is not essential, or even practical, that all of the arrows are associated with entities and attributes before determining data usage. In fact, it is quite common for the process of assigning usage to identify missing entity/attribute/arrow associations in the activity model.

If data models and activity models are developed in parallel and kept relatively integrated, then these steps can be performed at each level in the activity hierarchy. In this way, there is less likelihood of finding significant errors at the "end" of the model-building process. Obviously, however, there may be an additional cost in doing so, which must be weighed against the benefits.

# Glossary

## A-0 context diagram

A diagram depicting the highest level activity in a model. The context diagram represents the boundary of the process under study, with respect to purpose, scope, and viewpoint.

## A0I3 top level diagram

A diagram depicting the highest level in a IDEF3 model. The top diagram represents the boundary of the workflow under study, with respect to objects, facts, description, and constraints.

## ABC

(activity-based costing) is a technique for capturing and analyzing activity costs. Work center costs are identified and then allocated to activities. Parent activities generally consolidate the costs of all related child activities.

## activity

A verb phrase that describes an action that processes or transforms data or resources (e.g., Process orders, Run Video Store). An IDEF0 model highlights inefficient activities (those that do not have controls and/or an outputs) and assists business process reengineering efforts, accordingly.
An IDEF3 activity, also called a unit of behavior or UOW, describes a process, action, decision, or other procedure performed in a system or business. A DFD activity depicts an action that processes or transforms data.

## affinity matrix

A table that helps in system analysis and design by listing 1) the arrows in a model, 2) the entities and attributes associated with arrows in a model, and 3) the IRUN or CRUD assignments for each arrow.

## arrow

An arrow on an IDEF0 diagram represents an input, control, output, or mechanism (ICOM) of an activity. In IDEF3, arrows represent order or precedence between activities (drawn with a solid line), relationships (drawn with a dashed line), or object flow (drawn with two heads on a solid line arrow). In a DFD, the arrow represents the flow of data between activities, data stores, and external references.

## arrow association

The CRUD or IRUN data for an entity and/or attribute that is assigned to an IDEF0 arrow.

## arrow segment

One of the portions of an arrow; head, middle, or tail of a branched or split arrow.

## arrow stub

A portion of an arrow drawn by BPwin to show the existence of an incomplete off-page reference or unresolved arrow.

## AS-IS model

A model that represents how the business or organization currently works.

## author

The name of the person, department, or business that entered the activity, arrow, datastore, or external reference information into the BPwin diagram or the original IDEF0, IDEF3, or DFD document.

## BPX

A BPwin file format used to export entity and attribute information to ERwin.

## border arrow

An arrow that runs between an activity and the diagram border.

## BPSimulator

A simulation tool in which you can reuse your BPwin models to explore the time varying (dynamic) interactions of your business process. Resource allocations and process flows can then be optimized to meet your target workloads.

## business function

A term used to describe what activities are performed by the business. IDEF0 provides support for modeling of business functions through notations that support activities and ICOMs.

## business process

A term used to describe how activities are performed by the business. IDEF3 provides support for modeling of business processes through notations that support the ideas of precedence and activity timing.

## CRUD

An acronym for the actions that can be performed on instances of a database entity (create, read, update, delete).

## C-number

A chronological creation number used to uniquely identify a diagram and trace its history. A c-number may be used as a Detail Reference Expression to specify a particular version of a diagram.

## call arrow

An arrow on an IDEF0 diagram that references a related model or an existing model in a model library. A call arrow is a type of mechanism arrow that exits the bottom of an activity box and runs to the bottom border of the diagram.

## child diagram

A diagram that shows two or more subprocesses within a single parent activity. A child diagram is also called a decomposition diagram.

## color palette

A set of colors defined in BP*win* and used to standardize the use of color in a set of related diagrams.

## Constraints (IDEF3)

The section of an IDEF3 elaboration document that lists the assertions about the limiting factors on the UOW process, such as conditions that must be met for the process to start, continue, or end. Constraints are typically facts about the UOW that bound it or govern its occurrence, by describing factors that impact the UOW before, during, or after its execution, or that must always or never be present for the execution to occur.

## context diagram

A diagram depicting the highest level activity in a model. The context diagram represents the boundary of the process under study, with respect to purpose, scope, and viewpoint.

## control arrow

An arrow that enters the top of an IDEF0 activity box that represents a constraint on the activity with respect to how, when, and/or if an activity is performed (e.g., rules, regulation, policies, procedures, standards). Data or objects modeled as controls are not transformed by the function. Control arrows are associated with the top side of an IDEF0 box.

## cost center

Any defined business unit, function, or condition (e.g., labor, materials, manufacturing, administration, or overhead) that contributes to the actual cost of completing an activity.

## child box

A box on a child diagram.

## context

The immediate environment in which a function (or set of functions on a diagram) operates.

## DFD

Data flow diagram. A graphical representation (flowchart) of a data processing system that describes the data in the system and the actions that process or transform it.

## data flow

An arrow that runs between symbols on a data flow diagram to indicate the flow of data. Each arrow must have both a source and a destination. Unlike IDEF0 arrows (ICOMs), DFD arrows can enter or exit any side of the activity.

**data flow diagramming**

A method that is used to represent the movement and processing of information within a business or organization. It is generally used for application analysis and design.

**data store**

A representation used in a DFD to show the flow of data to and from a database table and/or ERwin entity.

**decompose**

To break an activity into its composite steps or subprocesses.

**decomposition**

The partitioning of a modeled function into its component functions.

**decomposition diagram**

A diagram that shows two or more subprocesses (sibling activities) of a parent activity and their associated arrows. A decomposition diagram is also called a child decomposition.

**Description (IDEF3)**

The section of an IDEF3 elaboration document that contains a textual description of the UOW that is then used as a glossary entry for it. The description recounts the information in the object, fact, and constraint lists.

**diagram**

A single unit in an IDEF0, IDEF3, or DFD model. An IDEF0 model that presents the details of an activity. An IDEF0 model comprises four basic types of diagrams: context, decomposition, node tree, and for exposition only (FEO).

**duration**

The average length of time required to complete an activity.

**Easy ABC**

A popular activity-based costing software tool manufactured by ABC Technologies.

**Elab Referent (IDEF3)**

A referent that adds an elaboration with a junction, so that you can describe additional facts, constraints, and decision logic for that junction.

**entity**

A logical representation of a database table. An entity can be created in BPwin or the ERwin data modeling product.

**external reference**

A location, entity, person, or department that is a source or destination of data but is outside the scope of a data flow diagram (DFD). An external reference can be internal to a company, such as "Accounts Payable" or outside it, such as "Vendor" or "Bank." This is similar to the concept of an external entity found in the Gane and Sarson notation.

**FEO**

A "for exposition only" (FEO) diagram is used as a tool to explain part of a process, to show a different viewpoint or highlight other functional details that are not explicitly supported by IDEF0 syntax. FEOs can be annotated with additional explanatory text and are not required to follow IDEF0 rules.

**Facts (IDEF3)**

The section of an IDEF3 elaboration document that lists the assertions about the UOW or objects participating in the UOW, including object properties and relationships that are maintained between objects during the process. Facts about the UOW can also include UOW properties, including duration, frequency, or cost.

**fan-in**

A junction that depicts the joining or convergence of multiple processing paths into a single process.

**fan-out**

A junction that depicts the split or divergence of a process into multiple alternative processing paths.

**frequency**

The average number of times an activity occurs for each single occurrence of its parent activity.

**function**

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

### Go-To Referent (IDEF3)

A referent that allows a diagram to span multiple pages and supports looping between diagrams and loop back (return) to a UOW earlier in the diagram. Indicates the UOW or junction that initiates following the completion of the referenced process.

### ICOM

An acronym for the categories of information (input, output, control, or mechanism) captured in an IDEF0 model. ICOMs represent a variety of objects that impact an activity, from raw materials to finished products, consumable and non-consumable resources, machinery, and rules governing the activity. "Under control, an activity transforms inputs into outputs, using mechanisms."

### IDEF0

A modeling method that supports the graphical description of business functions as a set of interrelated activities and the information or resources required for each activity. The purpose of an IDEF0 model is for documentation and restructuring of the functions for better efficiency and effectiveness.

### IDEF1X

A modeling method that supports the graphical depiction of logical data constructs, including entities, attributes, and entity relationships.

### IDEF3

A process modeling method that supports the graphical description of what a system or business does. IDEF3 provides rules for development of both process flow network diagrams and object state transition network diagrams.

### IDL

A standard file format used for importing and exporting IDEF0 model data.

### IRUN

An acronym for the actions that can be performed on an attribute of a database entity (insert, read, update, nullify).

### input

An arrow that enters the left side of an IDEF0 activity box that represents material or information transformed or consumed by the process to produce output.

### interface

A shared boundary across which data or objects are passed; the connection between two or more model components for the purpose of passing data or objects from one to the other.

### internal arrow

An input, control or output arrow connected at both ends (source and use) to a box on a diagram.

### join

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

### junction

An IDEF3 construct that indicates process branching. Different junction types are available to indicate AND, OR, and EXCLUSIVE OR conditions, and further highlight the timing (synchronous or asynchronous start and/or completion) of associated activities.

### Junction Referent (IDEF3)

A referent that associates special constraint sets to junctions. That is, associate an elaboration with a junction that contains additional facts, constraints, or decision logic which describe how that junction works.

### kit

A complete set of the printed diagrams, supporting text, and a glossary for an IDEF0 model that is used as a means of gathering feedback on the model through reviewer comments. Also the top section of the diagram frame

### leaf corner

A line drawn diagonally across the top-left corner of an IDEF0 activity to indicate that it has no related decomposition diagrams.

### link

An IDEF3 arrow. A precedence link (solid line arrow) depicts a significant constraining relationship among activities (UOBs), such as the flow of objects or precedence between activities. A relational link (dotted line arrow) shows a relationship between activities. An object flow link (double-headed arrow) highlights the participation of an object in two activities.

**mechanism arrow**
An arrow that enters the bottom of an IDEF0 activity box that represents a person, machine, or other non-consumable resource used to perform the activity. Also includes the special case of Call Arrow.

**model**
A set of diagrams and any supporting text or glossary that is required to fully explain a business process and/or function. An IDEF0 model has a defined scope, purpose, and viewpoint and comprises a single.

A-0 context diagram and its related decomposition diagram(s).

**model dictionary**
A BPwin construct that stores information such as name, status, author, definition, note, source, color, and font for each IDEF0 and/or DFD object in the model.

**node number**
A diagram number that corresponds to the number of the diagram's parent activity.

**node tree diagram**
A hierarchical diagram of an IDEF0 model or part of a model that shows activities and the relationships (parent-child, sibling) between activities. A node tree provides an overview of the entire model, with the top node in the hierarchy corresponding to the context diagram activity and several levels of child decompositions of the context activity comprising the rest of the tree.

**Note Referent (IDEF3)**
Additional information associated with the workflow object.

**OSTN diagram**
Acronym for Object State Transition diagram used to model the state of an object through a specific process or workflow.

**Objects (IDEF3)**
The section of an IDEF3 elaboration document that lists physical objects that participate in the UOW process. Object descriptions for the UOW should define whether the object is an agent (does the process), is altered by the process, participates without causing the process or being transformed by it, or is created or destroyed during the process.

**Object Referent (IDEF3)**
A referent that emphasizes the participation of specific objects or relations in a UOW (e.g., video tape in the video store example).

**off-page reference**
A reference from one activity to a related activity in a different diagram in the model (e.g., between process number 2.4 and process number 5.3.3).

**output**
An arrow that exits from the right of an IDEF0 activity box. It represents material or information produced by the activity.

**parent activity**
An activity that is decomposed into a child diagram containing two or more subprocesses.

**purge**
A BPwin action that eliminates unused activity, datastore, external reference, or arrow names from the model dictionary.

**purpose**
An explanation of why the process is under study, what the model shows, and what readers can do with the model.

**referent**
an external reference that is used in IDEF3 to support looping, off-page references, constraints on junctions, or references between process flow diagrams and object state transition diagrams.

**scope**
The breadth (lateral borders) and depth (level of detail) of the subject the model covers.

**source**
The name of the person, department, or business that provided the information about an activity, arrow, datastore, or external reference information.

### square tunnel

A symbol ({bmc BMP00103.BMP}) used to represent an unresolved arrow. An unresolved arrow occurs when you create a border arrow in a child decomposition with no equivalent reference in the parent diagram, in the parent diagram with no equivalent reference in the child decomposition. It serves as a reminder to the modeler to either continue the arrow on a different diagram (resolve it) or leave the arrow off the related diagram (tunnel it).

### squiggle

A small jagged line that can be used to relate an arrow name to the arrow symbol on an IDEF0 diagram. It also may be used to associate a model note with a component of a diagram. It

### status

The current state of completeness of the IDEF0 model, diagram, activity, or arrow. Standard options include Working, Draft, Recommended, and Publication.

### subprocess

A process that occurs within the context of a larger process.

### TO-BE model

A model that represents how the process will work in the future.

### tunneled arrow

A symbol that indicates an intentionally truncated arrow (an arrow that appears in the parent diagram but not the child decomposition, or vice versa). An arrow can be truncated for clarity.

### UOW

An IDEF3 acronym for Unit of Work. A UOW (also called a unit of behavior) describes a process, action, decision, or other procedure performed in a system or business within a workflow model.

### UOW Referent (IDEF3)

A referent that refers to a previously defined UOW without duplicating its definition in order to indicate that another instance of a previously defined UOW occurs at a specific point in the process (without loop back).

### unit of measure

The unit used to calculate duration or frequency for an activity. Units of measure are specified for time and cost information.

### unresolved arrow

A border arrow in a child decomposition with no equivalent reference in the parent diagram, or a border arrow in the parent diagram with no equivalent reference in the child decomposition.

### user-defined property

A user-defined property (UDP) is created by the modeler to associate business-specific information with an activity or an arrow. BPwin supports various types of UDPs, including pull-down lists for storing information such as "organization" or "level of automation," command UDPs that attach multimedia references, and text lists for storing information such as "critical success factors."

### viewpoint

A job title, department, or role (e.g., Department Manager, Foreman, Machinist) that serves as the basis for developing and viewing the process model.

# Index

## R

Relationship, 6-2
    Cardinality, 6-3
    Category, 6-6
    Identifying, 6-4
    Mandatory, 6-5
    Non-identifying, 6-4
    Optional, 6-5
    Parent-child, 6-2
    Subtype/supertype, 6-6

Resource
    Acquisition, using models for, 1-5

Retrieve, 6-9

## S

SADT, 1-4

Scenario, 2-2, 2-17

Scope, 2-2, 2-17, 3-2, 3-21, 4-2

Simulation, 5-4
    Dataflow diagram, 5-5
    Destination, 5-5
    Discrete-event, 5-4
    Facility, 5-5
    Models, 2-1
    MTBF, 5-8
    Queue, 5-5, **5-13**
    Source, 5-5, 5-12
    Statistical mean, 5-8
    Statistical probability, 5-8
    Statistical standard deviation, 5-8

Structured analysis and design, 1-4, 4-6

Structured Analysis and Design, 1-5

## T

TO-BE models, 1-5

Tunnel, 3-14

## U

UOW, 2-3

Update, 6-9

## V

Viewpoint, 2-2, 2-17, 3-2, 3-21, 4-2