CA Gen

Analysis Guide

Release 8.5

Second Edition

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be disclosed by you or used for any purpose other than as may be permitted in (i) a separate agreement between you and CA governing your use of the CA software to which the Documentation relates; or (ii) a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2014 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

CA Technologies Product References

This document references the following CA Technologies products:

- CA Gen
- COOL: Gen

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Documentation Changes

The following documentation update have been made since the last release of this documentation:

- COBOL Data Type (see page 56) - Updated the topic to address the STAR issue: 21793682.

Contents

Chapter 1: Introduction

The Analysis Phase

Analysis is a unique development phase that provides information and insight into the nature and needs of your business. The analysis phase helps you solve immediate business problems by defining what is needed in a single system or a set of integrated information systems. When the scope of a project includes more than one information system, analysis starts with a wide view of the business, and then focuses on details that lead to developing the set of systems your business requires.

The analysis phase consists of a cycle of information gathering and model building as you analyze the activities and data that is used in your business. The CA Gen Analysis Toolset is designed to encourage this cycle.

Conscientiously performing the tasks in the analysis phase ensures the success of your development effort. The tasks and procedures in the analysis phase include:

- Defining and refining the activities that are performed by the business (named business functions and business processes)
- Identifying the things that the business uses (named entity types)
- Identifying how the business activities and the things a business uses interact

You do not need to perform a complete and exhaustive analysis of your business before beginning other development activities. The analysis phase often overlaps the other phases in the application development life cycle.

Goals of Analysis

The goals of the analysis phase are to:

- Identify the set of business activities and data your information systems must support to meet the business objectives of the development project. Project stakeholders must agree that this list is complete.
- Analyze the current situation. Pay special attention to the current systems and organizations that affect the development project.
- Identify and plan a potential solution for development and implementation of a prioritized sequence of business systems. Typically, several business systems are defined to support a single business area.

Result of Analysis

The result of analysis is a model of the data and activities that support the identified business requirements. The model includes a detailed analysis of the parts that are selected for immediate implementation. Developers use the detailed analysis to design and construct the information systems. As a result, this model forms the foundation of all subsequent development activities.

Business Area

A subset of the architecture on which a development project is based is named a business area. A business area is a set of high-level activities and data. The business area is regarded pragmatically as the scope for development work, rather than as having some permanent significance for the enterprise.

CA Gen Analysis Tools

CA Gen provides a set of tools that you can use beginning in the analysis phase. These tools are listed in the following table.

CA Gen Tool	Use
Data Modeling, Data Model Browser, Data Model List, and Component Model	Building an Entity Relationship Diagram (ERD). These tools can also be used to represent data in current data stores and packages.
Component Model	Viewing each Subject Area individually.
Activity Hierarchy Diagram	Building an Activity Hierarchy Diagram to represent business activities hierarchically.
Activity Dependency Diagram	Building a Process Dependency Diagram to record events and dependencies between business activities.
Entity Life-cycle Diagram	Defining and summarizing interactions between processes and entity types, organizational units, roles, and locations.
Matrix	Clustering processes and entity types to help identify groups of functions, subject areas, or business systems.

Chapter 2: Preparing for Analysis

Follow these steps:

1. Identify the initial requirements, including preparing for analysis activities in a development project and determining whether an Information Strategy Plan exists.
2. Determine the scope of the project.
3. Establish an analysis team.
4. Schedule analysis activities.
5. Prepare a quality plan.
6. Plan the analysis environment.

More information:

Analyzing Data (see page 27)
Analyzing Activities (see page 85)

Identify Initial Requirements

Every development project begin with a statement of the objectives it achieve. These clearly stated objectives form the requirements of the project. The development or re-engineering of a system must support these business objectives.

The initial scope of any project depends largely on whether an Information Strategy Plan already exists and is up to date. It may be useful to identify the main activity and data objects to confirm the scope and help to plan the analysis work even when planners have already established the scope of the development project.

It is common to begin analysis without the benefit of an information architecture that is derived from a strategic plan. The resulting systems must meet business needs, whether they are based on an enterprise-wide information architecture or not. Although analysts, planners, and coordinators may not achieve the full benefits of establishing a corporate information strategy, they can still reap significant rewards from a thorough analysis of the needs of part of the business. Their efforts additionally contribute to developing that architecture.

You can either perform or review some of the work that is typically done during planning to prepare for the analysis phase. This initial scoping, also named an Initial Needs Survey or Mini-ISP, ensures that the scope of analysis is set correctly.

The scaled-down versions of planning activities prove helpful. The analysis team can establish an initial set of subject areas and entity types, business functions, relevant current systems, and parts of the organization that is involved in this analysis effort.

In practice, management expectations often determine the scope of a project. A thorough analysis team outlines the information architecture early in the project. Creating this outline ensures that you have identified possible interactions with other areas, and that the scope does not encompass too much of the business.

A typical development project last from three to nine months. You can expect analysis to occupy about one quarter of this time.

More information:

Analyzing Interactions (see page 121)

Define Project Scope

Defining the scope of a project is a critical task of analysis. By defining the project scope, you identify the elements in the business that must be examined or changed.

Correct scoping is critical to the success of any development project. Focus on the fewest possible objects that satisfy the business objectives of the development project, with a minimum of objects that immediately influence the project.

The project scope includes:

- Business activities to be supported

  These are modeled in the information architecture as a set of business functions and processes.

- Data that are needed by those activities

  These are modeled in the information architecture as a set of subject areas and entity types.

- Business systems and data stores to be developed
- Current business systems and data stores to be modified, reused, interfaced, or replaced
- Packaged solutions to be evaluated to determine their suitability to support some or all of the requirements
- Organizational elements that are involved in executing the business activities

  Example: organizational units, roles, skills, and locations

- Availability of alternative information technology to support business requirements

  Example: client/server techniques for distributing processes and data

- Other development efforts within the business that include developing or improving business processes, information systems, or information technology

Review the project scope throughout analysis to ensure its ongoing validity.

You can use cluster analysis to define and confirm the scope of activities and data based on the use of data by activity. This technique already have been used in the ISP to define the initial project scope. Defining clusters, groups of activities, and data minimizes interactions with other activities and data outside the scope. You can identify potential coordination requirements by examining data that activities and other projects use.

Business activities and data that is defined using CA Gen form the basis of the starter analysis model for the project.

Establish the Analysis Team

As in any project, success relies heavily on assembling an effective project organization.

In analysis, that organization include the following roles:

- Sponsor

  This is a senior executive within the business who is seeking an improvement.

  The sponsor:

  - Exercises authority and responsibility over the project
  - Ensures the availability of work force and resources as required
  - Eliminates political obstacles by championing the project to management

  By providing aid, the sponsor facilitates the timely and orderly completion of analysis. Do not expect the sponsor to contribute much time to analysis.

- Analysis team

  This is the full-time staff performing the analysis. The size of the team is sufficient to address the project scope.

  Typical participants are:

  - Project manager

    The project manager understands the business and possesses good communication and presentation skills.

  - Joint working group of information services and user staff

    This group includes several business analysts and members of the user community who are familiar with the business. They work together to formulate requirements and agree on the necessary information systems to be developed.

  - Analysis expert (possibly a consultant)

    Team member who either creates or refines a subset of the information architecture that is initially developed during planning or develop a starter model that is based on the initially identified requirements. This individual is proficient in the techniques that are used in analysis.

  - Stakeholders

    People who have a vested interest in the system being developed because their departments interact with the system.

  - Planners

    Team members who establish the scope and boundaries of the project either as part of an Information Strategy Plan, or by a short definition of initial requirements that satisfy a specific business need. Planners use the analysis model to define the system or systems to be developed and to plan for transition to the new systems.

  - Other specialists

    These individuals provide expertise at various times during the analysis; for example, facilitators to run facilitated sessions and specialists who are familiar with current systems.

  - Project steering group

    This group is established to provide management direction for the project.

- User team

  This is a reference group of users who can supplement the knowledge of the analysts and review analysis results.

More information:

Analyzing Data (see page 27)
Analyzing Activities (see page 85)
Analyzing Interactions (see page 121)
Analyzing Current Systems (see page 175)
Collecting and Validating Information (see page 21)

Schedule Analysis Activities

The project manager and the analysis team establish a project schedule that is based on the project scope. The task structure and management of the project depends on:

- Development life cycle route chosen
- Variations in the life-cycle route tasks or checkpoints, as dictated by the project objectives and scope
- Whether the project is intended to develop a stand-alone system or systems that are part of a set of systems within a Business Systems Architecture
- Whether Rapid Application Prototyping (RAP) techniques are used
- Whether current systems and data are analyzed
- Whether packaged solutions are evaluated
- Whether the distribution of procedures or data is among the technical possibilities

It is important to distinguish between technical facilities that are currently installed and those that is implemented in time for the transition of the systems to be developed. For example, this constrain the choice of distributed processing and use of client/server techniques. Where a system is to use new technical facilities, the project schedule allow for coordination with the necessary technical development work.

Review the analysis techniques to be used and develop a schedule using either automated or manual techniques.

Decide about the alternative courses of action available during analysis. Your choices affect the content of the task list and the project duration. These choices include:

- What facilitated sessions and more interviews are required for information gathering.
- Level of detail that is required for data analysis; for instance, complete attribute detail is deferred until after decisions about system scope.
- Level of detail that is required for activity analysis.
- Not all activities fully subdivided to decide about the scope of the system. The business objectives for the project suggest which activities must be investigated thoroughly.

- Requirements for, and timing of, current systems analysis.
- The analysis is performed to reuse or to replace current data or systems.
- The volume of work, or the availability of resources, require work to be started in parallel with activity and data analysis. Work can also be postponed until after decisions about the scope of the system have been made.
- Requirements for defining expected effects for all elementary processes. The expected effects are described in Analyzing Activities (see page 85).
- Level of detail define process logic. Some business rules are defined during analysis, but many can be deferred until the design phase when detailed procedure definitions are created.
- Requirements for formal and informal coordination with other development projects and the timing of this coordination.

Prepare a Quality Plan

Preparation of a quality plan for the analysis depends on the quality standards of the organization. The quality plan that is produced at this stage typically covers the whole development project.

A typical quality plan includes:

- Risks that are associated with any departure from quality standards
- Standards for analysis results, for coordination and confirmation processes, and for measuring the satisfaction of users with their involvement
- Quality roles and responsibilities, including technical reviews, internal audit, and so forth
- Quality checkpoints

Plan the Analysis Environment

In addition to choosing the right people, it is important to establish the right environment for analysis. The analysis environment include:

- Project software including CA Gen, project management tools, and documentation aids
- Computer hardware such as workstations, printers, and communications
- Meeting rooms with space for conducting facilitated sessions and interviews
- Other equipment such as filing cabinets, white boards, bulletin boards, and flip charts

- Project room
- If possible, an area that can serve as an informal meeting place during analysis
- Often, the walls of such a room is covered with diagrams for the business model and lists of issues to facilitate team discussion.

Results of Preparing for Analysis

The results of preparing for analysis are:

- Defined analysis scope in terms of business activities, data that is needed by activities, business systems to be developed, current business systems to be analyzed or reused, and the organizational units involved
- Starter model for analysis containing an initial view of activities and data to be analyzed
- Established analysis team
- Analysis plan including tasks, resource levels, and a schedule
- Quality plan defining risks, standards, quality roles, and checkpoints
- Established analysis environment (equipment, software, and location)

Chapter 3: Collecting and Validating Information

Collecting information involves project members and users working together to plan the project and to agree on and record facts about the business, its direction, information needs, activities, data, use of systems, and information technology. The team uses this data during analysis to identify processes, dependencies and information views, entity types, relationships, and attributes. This process also captures user assessments about the quality of, and problems with, current information systems.

Techniques for collecting and validating business information are:

- Facilitated sessions
- Structured interviewing

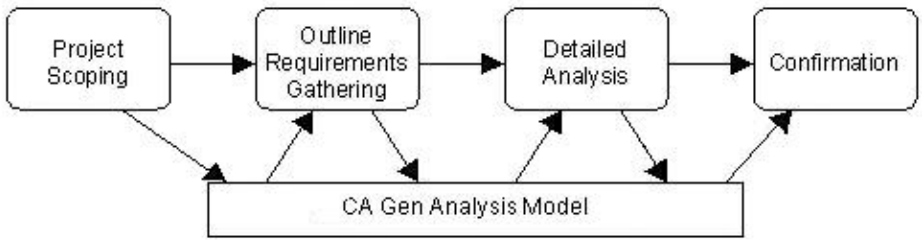
More information:

Analyzing Current Systems (see page 175)

Using Facilitated Sessions in Analysis

Facilitated sessions (also known as user workshops) provide an environment for people from the business community (users) and the information systems community (systems professionals) to collect, review, and agree on information quickly and interactively.

In analysis, the results of facilitated sessions help to produce a business model that feeds directly into design, as shown in the following illustration:



The primary uses for facilitated sessions during analysis are:

- Project scoping
- Requirements definition

- Activities and data definition
- Analysis model confirmation

Project Scoping Sessions

This type of facilitated session establishes the requirements and justification for a system project definition and scope in terms of business activities and data.

Suggested Participants

The suggested participants for the project scoping sessions are:

- Executive sponsor and user management participants who have an understanding of the overall business requirements and the power to influence them
- Project manager and team members

Inputs

The inputs that are required for the project scoping sessions are:

- Project definition
- Activity Hierarchy Diagram if available from the ISP
- Activity Dependency Diagram if available from the ISP
- Entity Relationship Diagram if available from the ISP
- Enterprise Organization Hierarchy Diagrams
- Function or Location Matrix if available from the ISP
- Results of any previous Information Strategy Plan such as data and activity diagrams and lists if they are up to date.

Suggested Agenda

The suggested agenda items for the project scoping sessions are:

- Opening presentation
- Review architecture
- Select function
- Resolve implementation issues
- Review requirements

Expected Results

The expected results of the project scoping sessions are:

- Revised activity hierarchy diagram
- High-level activity dependency diagrams
- Revised Entity Relationship Diagram or Data List
- Revised Organization Hierarchy Diagram
- Process Type or Entity Type Matrix
- (Optional) Preliminary event list
- System scope description and definitions
- System benefits definition
- Implementation issues list
- Open issues list

Requirements Sessions

One or more of these facilitated sessions establish the detailed functions that the system supports. These sessions are named Joint Requirements Planning in Rapid Application Development.

Hold these sessions before management commits to building the system. The facilitator guides the participants through a planned set of steps and activities to produce the requirements document.

System development participants translate the requirements into structured specifications during and immediately after the session. In particular, they document decisions dictating the direction of the system design activities. These decisions provide a blueprint for detailed analysis sessions once the executive sponsor gives approval.

Suggested Participants

The suggested participants for the requirements sessions are:

- End users who have an understanding of business requirements
- Project manager and team members

Activities and Data Definition Sessions

One or more of these sessions deliver the detailed definition of activities and data each system support. Conducting these sessions establishes the detailed analysis and structure of a given system. This builds on the results of the requirements planning session, sometimes involving the same participants.

Suggested Participants

The suggested participants for the activities and data definition sessions are:

- End users of systems who have the time and knowledge to define or reach agreement on analysis results
- Project manager and team members

Inputs

System scope definition, requires the following inputs:

- Activity hierarchy diagram
- Entity relationship diagram or data list
- Activity dependency diagram
- Process type or entity type Matrix

Suggested Agenda

The suggested agenda items for the activities and data definition sessions are:

- Develop an open issues list
- Detail the entity types
- Detail the activities

Expected Results

The expected results of the activities and data definition sessions are:

- Detailed entity relationship diagram or data list
- Detailed activity hierarchy diagram
- Detailed activity dependency diagrams
- (Optional) Entity Life-cycle Diagrams
- (Optional) Event response list
- (Optional) Process/Role Matrix

- (Optional) Business algorithm list
- Open issues list

Model Confirmation Sessions

This type of facilitated session provides confirmation by users of the analysis model as the project team develops the model. The analysis model is the basis for system design.

Suggested Participants

The suggested participants for the model confirmation sessions are:

- End users of systems
- Project manager and team members

Using Structured Interviewing in Analysis

The structured interviews with top executives and managers can be conducted in addition to the planning facilitated sessions and the review of available information sources. These interviews are useful in preparing for the facilitated sessions or for supplementing facilitated session results.

Results of Information Collection

Facilitated sessions and structured interviews produce these results:

- Project and planning results:
  - System scope description
  - System scope definitions
  - System benefits definition
  - Implementation issues list
- Analysis model elements:
  - Activity Hierarchy Diagram
  - Activity Dependency Diagrams
  - Entity Relationship Diagram/Data List
  - Entity Life-cycle Diagrams
  - Process/Entity Type Matrix
  - Process/Role Matrix

  - Business algorithm list
  - Organization Hierarchy Diagram
- Analysis working documents:
  - Open issues and actions list
  - User contact register
  - Document register

Chapter 4: Analyzing Data

Data analysis involves classifying, structuring, and defining data in a model that can faithfully represent all the real world occurrences of interest to the business and the relationships between them.

Compare this with activity analysis, in which the type process is used to represent all possible occurrences (or executions) of the same activity.

The business model that is developed during analysis integrates three equally important aspects of business requirements:

- Data aspects of the model, which describes things of interest to the business and the relationships between them.
- Activity aspect of the model, which records things the business does, or should do.
- Interaction aspect of the model, which details how things the business does (activities) affect things of interest to the business (data).

This aspect serves to confirm, and if necessary modify, the analysis model. It also provides a detailed basis for system design.

Data modeling is performed early in analysis, to set, or clarify the development scope, and then revisited in more depth until definitions are agreed on for all the data that support the business requirements.

When analysis is complete, the resulting data elements of the model depict in detail what information is used in the business.

The modeling techniques in this chapter are associated with top-down data modeling. Some of these techniques can also be used in reverse engineering to help plan system re-engineering, interfacing between new and current systems, and evaluating and implementing packaged applications.

The reverse engineering use of data modeling techniques is described in the chapter Analyzing Current Systems (see page 175).

Data and activities can be analyzed together.

More information:

Analyzing Activities (see page 85)
Analyzing Interactions (see page 121)
Building the Analysis Model (see page 201)

Basic Data Modeling Concepts

Data modeling is not an exact science. A business model is based largely on the pragmatic understanding of the business by the business practitioners. So, if two analysts study the same part of the business independently, it is likely that they develop models with differences.

In practice, data analysis proceeds in parallel with activity analysis to produce a business model where data supports the relevant business activities. This is described further in the chapter Analyzing Activities (see page 85). Some representations of the underlying business reality are better than others, but successful data modeling depicts the business environment in a thorough and workable way that is generally understood and agreed on by all the business organizations involved. This means that group activities involving business staff in defining and confirming the model are vital to analysis.

It is necessary to achieve consensus among the group of business practitioners participating in the development project and often to coordinate with results achieved by other projects. In practice, it may not be possible or desirable to achieve perfect consistency at all times and throughout the whole organization. The business objectives of the project have to be taken into account. An early return on investment takes a higher priority than a demand for perfect consistency.

Bear in mind that data modeling is not database design. It does, provide the information that produce a database design. In fact, CA Gen transforms this automatically. The emphasis during analysis is on modeling business data, not on designing optimal data structures. Initially therefore, the model omits implementation details. Later, during system development, or sometimes in parallel with analysis if performance is already identified as an issue, the designer take steps to optimize the physical database implementation using all the entity volume and process frequency and entity type usage details that are collected in the business model. These volume and frequency details are collected as properties of entity types and elementary processes. Volume, frequency, and usage later is analyzed further, during distribution analysis, which is described in the chapter Analyzing Interactions (see page 121).

The primary goal of data modeling is to depict accurately these fundamental elements of business information:

- Things with which the business deals
- Associations that exist between those things
- Detailed characteristics each thing possess

The following table summarizes the terms used for each concept at each level:

Concept	Type or Class	Single Occurrence or Instance
Thing	Entity Type	Entity

Concept	Type or Class	Single Occurrence or Instance
Association between things	Relationship	Pairing
Characteristic of a thing	Attribute	Attribute value

The following two levels of terminology are used for each of the three fundamental elements:

- Type or class level—A type or class is a description of some collection of fundamental elements that share similar traits.
- Occurrence or instance level—An occurrence or instance is a single instance of one of those elements. Data modeling is essentially a classification activity.

Those who are familiar with object-oriented concepts recognize the similarity to the distinction between object classes and objects or instances of object classes.

Data Modeling Terminology

The data modeling terms are as follows:

- Entity and entity type
- Relationship and pairing
- Attribute and attribute value
- Subject area
- Entity subtype and partitioning

Entity and Entity Type

An entity is a fundamental thing of relevance to the enterprise about which data is kept. An entity is an occurrence of an entity type.

An entity type is a description of all the entities to which a common definition and common relationships and attributes apply. An entity type describes a collection of entities.

For example, consider a company that sells two kinds of products: ball bearings and steel tubes. Three customers buy these products. In this example, CUSTOMER and PRODUCT are both entity types, while the specific instances of CUSTOMER and PRODUCT are entities.

This following table shows the entity types and entities for this example:

Entity Type	Entities
CUSTOMER	James Molt
	Cindy Mowinski
	Stuart DeVries
PRODUCT	Ball bearing 5mm alloy steel
	Steel tube 25mm dia. x 2m 16 gauge
NUMBERED PRODUCT	Steel tube 25mm dia. x 2m 16 gauge, serial 9305111
	Steel tube 25mm dia. x 2m 16 gauge, serial 9305112

For some entities, there are many individual occurrences physically present at one time. For example, there are many Ball bearings in inventory. For some of these entities, the business distinguish between individual occurrences and trace each of these occurrences. For example, steel tubes are installed in a nuclear reactor and distinguished with a serial number or some other identifying mark. Since this requires more information, steel tubes with serial numbers are seen as occurrences of the entity type NUMBERED PRODUCT.

Relationship and Pairing

From the example in Entity and Entity Type, clearly a customer buys products. That is, an entity belonging to the entity type CUSTOMER can buy an entity belonging to the entity type PRODUCT. The reason for associating CUSTOMERs and PRODUCTs (that is, buys) is named a relationship. A relationship is a reason of relevance to the enterprise for associating entities (that is, occurrences) of one or two entity types.

A pairing is an occurrence of an association between two entities for the reason that is described by a relationship. So, if James Molt (a CUSTOMER) buys a ball bearing (a PRODUCT), the entities James Molt, and Ball bearings are said to be paired based on the relationship buys. In other words, a pairing is an occurrence of a relationship between two entities, while a relationship is a type of association between one or two entity types.

In this example, the entities are of two entity types, but pairings are also found between entities of the same type.

More information:

Building the Analysis Model (see page 201)

Attribute and Attribute Value

An enterprise knows certain characteristics about each of the entities with which it deals. Typically, analysts know the same kinds of characteristics about each entity belonging to the same entity type.

For example, CUSTOMER characteristics that are of interest to the business are:

- Name
- Address
- Phone Number
- Credit Rating

Each of these possible characteristics of entities of the entity type CUSTOMER is named an attribute. An attribute is a descriptor whose values are associated with individual entities of a specific entity type.

A set of attributes describes each entity type. An attribute can belong only to an entity type. It cannot therefore describe a relationship and must not have attributes of its own.

Each entity has one attribute value for each of its attributes. An attribute value is a descriptor whose values are associated with individual entities of a specific entity type.

For example, one entity of the entity type CUSTOMER has James Molt as an attribute value for its Name attribute.

The multi-valued attributes can be modeled.

More information:

Building the Analysis Model (see page 201)

Subject Area

A subject area is a way to group together entity types in the same general area of interest.

For example, business activities within the scope of analysis deal with SALES and INVENTORIES. The names of subject areas are plural nouns by convention.

A subject area itself be composed of other, smaller subject areas that contain entity types. A subject area that contains only entity types is named a primitive subject area.

In the following table, CUSTOMERS, ORDERS, and INVENTORIES are primitive subject areas.

Subject Areas	Entity Types
SALES	
CUSTOMERS	CUSTOMER, DELIVERY POINT
ORDERS	ORDER, ORDER ITEM
	SALESPERSON
INVENTORIES	PRODUCT, STOCK, WAREHOUSE

At some point in analysis, a subject area contain both entity types and other subject areas. Try to resolve this unbalanced situation.

For example in the table, the SALES subject area includes the subject areas CUSTOMERS, which includes the entity types CUSTOMER and DELIVERY POINT, and ORDERS, which includes the entity types ORDER, ORDER ITEM, and SALESPERSON. SALESPERSON is included in another subject area within SALES, or does it belong in some other subject area such as HUMAN RESOURCES?

Decomposing subject areas is discussed further in the chapter Building the Analysis Model (see page 201).

In analysis, you can use subject areas in two ways:

- Progressively decompose data starting with subject areas; then subdivide them into more focused subject areas, based on the highest-level business functions within the project scope. As each entity type is discovered, it is added to one of the subject areas. Data analysis proceeds in parallel with activity analysis, and the data and activity results are kept in step at each level of analysis.

  Note: For more information, see Building the Analysis Model (see page 201).

- Begin analysis without subject areas and later assign each of the entity types that are discovered to one of a set of subject areas. This may be done based on common usage of entity types by processes or because entity types are closely related.

  Note: For more information, see Analyzing Interactions (see page 121).

In either case, the use of subject areas permits project teams to compare what they have each discovered about a subject area, for example, many parts of the business uses PRODUCTS.

A project can be given a CA Gen starter model that already contains the set of subject areas of the enterprise and the relevant entity types that have already been defined, as part of an Information Strategy Plan or previous system development projects.

An Information Strategy Plan exists, a few subject areas are defined and used as a high-level structure for the set of entity types that are identified during planning.

A complete Information Architecture, developed progressively by many development projects, include hundreds of entity types, so subject areas are a convenient means of dealing with related entity types and coordinating the needs of separate business systems for the same types of data. For this reason, even without the existence of an Information Strategy Plan, development coordinators may have provided a set of subject areas as a starting point for the analysis work of each development project.

Note: For more information about the examples of Entity Relationship Diagrams structured in subject areas, see Drawing Entity Relationship Diagrams (see page 79).

Entity Subtype and Partitioning

You find that entities of a single entity type have different or more attributes and relationships. This situation requires the definition of entity subtypes and partitioning.

An entity subtype is a description of entities of the same type that is more restrictive than that of the entity type and to which more common relationships and attributes apply.

Partitioning is a basis for subdividing the entities of one type into subtypes.

For example, the following table shows the partitioning of the entity type CUSTOMER.

Partitioning	Entity Subtypes
NATIONALITY	
	DOMESTIC CUSTOMER
	FOREIGN CUSTOMER
INCORPORATION	
	EXTERNAL
	INTERNAL

Although all CUSTOMERs share a set of relationships and attributes, some CUSTOMERs (FOREIGN ones) have relationships and attributes beyond those in the common set, and others (DOMESTIC ones) have different relationships and attributes in addition to the common set. This can be seen as specializing the set of CUSTOMER entities for a specific purpose. The entities are partitioned in this example based on the nationality of a customer.

It is possible to partition customer entities in many ways, for example, whether the customer is external to the enterprise or some other division internal to the corporation.

You can define these mutually exclusive groups of entities of the same type as entity subtypes belonging to a partitioning.

Defining Entity Types, Relationships, and Attributes

Entity types and their relationships are defined first. They are depicted graphically as an Entity Relationship Diagram (ERD) and described in supporting documentation. Most diagrams that are used to build business models deal with entity types, not occurrences; but sometimes it is useful to discuss entity occurrences to illustrate a real world example.

Later in analysis, full definitions of all their properties, and of all their attributes, are produced. This detailed definition is done only for those entity types that support the planned business systems.

Finding Different Kinds of Entity Types

An entity is a thing of fundamental relevance to the enterprise about which data is kept. The definition of an entity suggest that entities can only be tangible objects, such as CUSTOMER and PRODUCT. This is not the case. The intangible objects are also of interest to the business.

Conceptual entity types, such as COST CENTER and LEDGER ENTRY, are used to capture data about less tangible concepts of interest to the business.

Active entity types, such as LECTURE ATTENDANCE and EQUIPMENT BREAKDOWN, are used to represent information about activities that take place.

Active and conceptual entity types are more difficult to identify than tangible entity types but are no less important in determining all the entity types of interest to the business. A business model does not need to include entity types in all three categories, but remember that there are entity types other than those representing a set of purely physical things.

You discover entity types by identifying the things acted on by business activities by an examination of current data and by considering subject areas and other entity types.

Note: For more information, see Building the Analysis Model (see page 201).

Each primitive subject area can be expected to have a subject, or central entity type; for example, CUSTOMERS has CUSTOMER as its central entity type.

Entities are dependent or independent:

- Dependent entities
- Order items cannot exist independently of an order, and an order in turn depends on the existence of a customer.
- Dependent entity types further describe an entity type or associate entities together.
- DELIVERY POINT adds further detail to a customer and therefore be described as characteristic.
- ORDER ITEM associates ORDER and PRODUCT and so named associative.
- Independent entities
- Entities that exist externally to the enterprise are independent (CUSTOMER, for example).

The distinction between these categories of entity types are made explicitly and is not stored in a CA Gen model.

Defining Entity Types

You specify a number of details about entity types, most of which supplement the graphical representation of the Entity Relationship Diagram, or which is defined using the Data Model List, or the Data Model Browser.

CA Gen records this information about an entity type:

- Name
- Description
- Properties
- Expected number of occurrences
- Expected growth rate
- Identifiers
- Aliases
- Relationships (see Defining Relationships (see page 38))
- Mutually exclusive relationship memberships (see Additional Data Modeling Topics (see page 67))
- Attributes
- Partitioning
- Action blocks (see Analyzing Interactions (see page 121))

Entity Type Name

The entity type name is a short classification of its entities. Take the form of a singular noun (such as CUSTOMER, not CUSTOMERS). In CA Gen, an entity type name must not exceed 32 characters.

It must be unique within the enterprise. For example, you cannot have two entity types that are named DELIVERY POINT representing two things. To build a sensible model, entity type names must be unique, and CA Gen enforces this. If more than one entity type has the same name within the enterprise, seek agreement on which entity type bear the name. Record the name as an alias in the description of the other entity type.

You discover that an entity type has multiple names, or synonyms. For example, an entity type called CUSTOMER by one department that is known as CLIENT by another. In such cases, seek user consensus on the most commonly used alternative as the name and record the remaining synonyms as aliases.

More information:

Entity Type Aliases (see page 37)

Entity Type Description

The entity type description is a block of text that explains the entities of the entity type. A description never includes a layout of the detailed data that is recorded for each entity type.

You can specify the following kinds of descriptive information about an entity type:

- Precise and concise definition, preferably in terminology that is commonly used in the business
- This include:
  - Indication of exactly what the entity is (named the scope)
  - How it is distinguished from entities of other entity types (named the qualification).
- The qualification is used to exclude things that are of no interest to the business.
- The names of relationships and other entity types probably appear in the definition, but you should not merely respecify the Entity Relationship Diagram in words. For example, the definition of the entity type customer is:

  An individual or organization (scope) that purchases (qualifying relationship) products (qualifying entity type) marketed by (qualifying relationship) a division (qualifying entity type).

- Examples of the entity type, if desired
- Any integrity constraints that are associated with entities of the type being described.

  Note: Integrity constraints are discussed in Additional Data Modeling Topics (see page 67).

Entity Type Properties

The properties of entity types describe the estimated total number of entities that the enterprise expects to be interested in, and the anticipated increase or decrease in that number over time.

This statistical information is captured during analysis simply because analysts are in an excellent position to provide it. This information later influence the technical design of the database.

Entity Identifiers

An identifier is a way of uniquely identifying an entity of a particular entity type.

For example, assume that the entity type CUSTOMER has an attribute that is called Number. If no two customers have the same value for their Number attribute, the value of Number can be used to uniquely identify one and only one CUSTOMER entity.

In a consistent model, each entity type must have at least one identifier.

Each entity must be distinguishable from all other entities that are based on some combination of values and pairings.

More information:

Additional Data Modeling Topics (see page 67)

Entity Type Aliases

Each entity type is known by a number of different names, depending on who in the business is using it, and why.

For example, CUSTOMER is known variously as CLIENT, CUST, COOB (Customer Of Our Business), or CONSUMER. Each of these synonyms is recorded as an alias of the entity type.

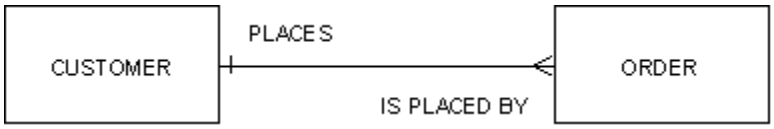
Each alias is identified as an acronym (like COOB), an abbreviation (like CUST), or neither (like CLIENT and CONSUMER).

CA Gen can also record a long name of up to 50 characters for the alias of an entity type. The long name is typically used only to provide compatibility with data dictionary products that support names longer than the 32-character standard.

Defining Relationships

Each relationship consists of two memberships, one for each participation of an entity type in the relationship.

For an instance, consider a company in which customers place orders. The following illustration is part of an Entity Relationship Diagram showing that customers place orders.

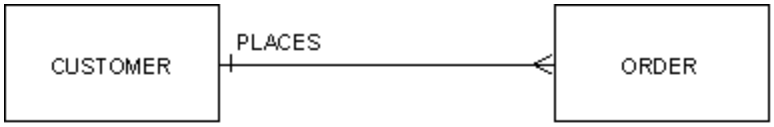


The rectangles represent entity types, and the line between them represents the relationship. The other symbols are introduced in Relationship Cardinality (see page 40).

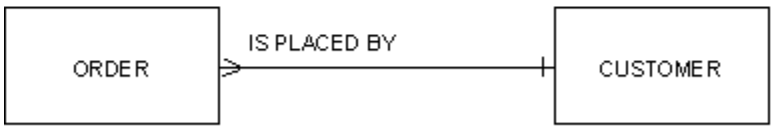
The relationship that is shown between CUSTOMER and ORDER consists of two separate relationship memberships:

- CUSTOMER places ORDER
- ORDER is placed by CUSTOMER

Each membership has its own set of characteristics. The following illustration depicts the division of this relationship into its two memberships.



One of a pair of relationship memberships



The other of a pair of relationship memberships

Each membership describes an aspect of the entire relationship. The relationship is fully described by the properties of its two relationship memberships.

In CA Gen, each element of a relationship membership definition contributes to building a sentence that describes that membership except for the description.

In addition, some of the definition appears in the graphical representation of the Entity Relationship Diagram.

The details that are captured for each relationship membership include:

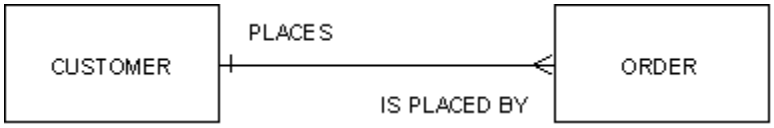
- Name
- Cardinality
- Optionality
- Percentage of entities participating in pairings that are based on this relationship for optional relationship memberships only
- Expected the number of pairings per entity type for relationship memberships with a cardinality of many
- Transferability
- Associate is modifying or referencing
- Description

Relationship Name

The name of a relationship membership describes the reason for joining two entities of the entity types participating in the relationship.

The name is a verb (either active or passive) that connects one entity type (the subject of a sentence) to another (the object of a sentence).

For example, places and is placed by are the relationship membership names in the following illustration.



The use of is a matter of style. Some analysts omit is to give shorter relationship names that are more likely to display fully on diagrams. Others prefer to use is placed by, not only placed by. Using is in this example not only makes for a better sentence here, but it also reads well in the syntax of action diagrams.

With simply the relationship membership names in place, the relationship can be described as follows:

- Customer places order
- Order is placed by customer

CA Gen makes an arbitrary distinction between relationship memberships, which are based on the order in which the analyst chooses the entity types participating in the relationship. The relationship membership that is associated with the entity type chosen first is named the source membership, while the other is named the destination membership. This distinction merely provides a means for selecting individual memberships when using CA Gen; it does not imply a direction to a relationship.

Relationship Cardinality

The cardinality of a relationship membership determines the number of entities on one side of the relationship that is associated with a single entity on the other side.

The cardinality of each relationship membership is one of the following:

- One
- Many

Only three combinations of relationship membership cardinality can exist:

- One-to-one (1:1)

  Each entity participating in the relationship can be related to only one entity of the other entity type.

- One-to-many (1:M)

  A single entity of one entity type participating in the relationship can be related to one or more entities of the other type, but not conversely.

- Many-to-many (M:N)

  A single entity of either entity type participating in the relationship can be related to one or more entities of the other entity type.

On an Entity Relationship Diagram, at either end of the relationship line, the cardinality of the relationship membership is depicted as follows:

- A bar that crosses the line at a right angle indicates only one.
- A crow's foot at the end of the line indicates one or more.

For example, the following illustration depicts a 1:M relationship.

The illustration shows that each customer can place one or more orders, but that only one customer places each order.

The most common cardinality for relationships is 1:M. Although 1:1 and M:N relationships sometimes occur naturally, they often appear as the result of inaccurate analysis.

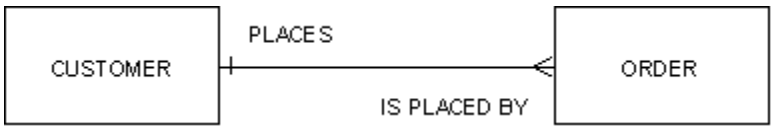
More information:

Building the Analysis Model (see page 201)

Relationship Optionality

It is important to define whether all entities must participate in a relationship membership, in which case, the relationship membership is mandatory or only some, in which case, it is optional.

For example, in the following illustration, neither relationship membership is indicated as being optional.



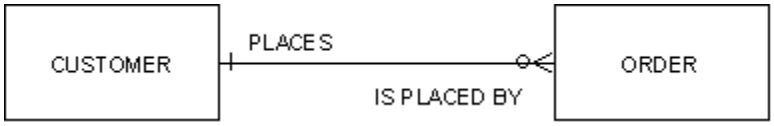
From this illustration, it is possible to infer the following:

- Each ORDER must be placed by a CUSTOMER
- Each CUSTOMER must place at least one ORDER

The first statement makes sense. It is unlikely that the business accept an order for its products without knowing and approving the identity of the customer placing it. Likewise, if the customer placing the order ceases to be a customer, it is reasonable that all the customer's outstanding orders cease to be treated as orders. If this is true, the relationship membership ORDER is placed by CUSTOMER is said to be mandatory.

The truth of the second statement is doubtful. Assume that the CUSTOMER entities include information about the customer's name, date of first becoming a customer, and credit rating. Further assume that a canceled order is of no further interest to the business, and so may be deleted. In this case, the business loses all knowledge of a customer whose only order is canceled. It is unlikely that this condition reflects the business reality. Rather, the business probably wants to retain knowledge of a customer whether the CUSTOMER has an active pairing with an ORDER. If this is true, the relationship membership CUSTOMER places ORDER is said to be optional.

Optional relationship memberships are drawn as a circle on the relationship line next to the cardinality symbol. The following illustration depicts CUSTOMER places ORDER as an optional relationship membership.



The sentences describing the relationship memberships can now be expanded to reflect optionality, as follows:

- Each CUSTOMER sometimes places one or more ORDERs
- Each ORDER always is placed by exactly one CUSTOMER

Some analysts prefer to consider optionality a special case of cardinality that permits zero pairings. The previous two sentences would then read:

- Each CUSTOMER places zero, one or more ORDERs
- Each ORDER is placed by exactly one CUSTOMER

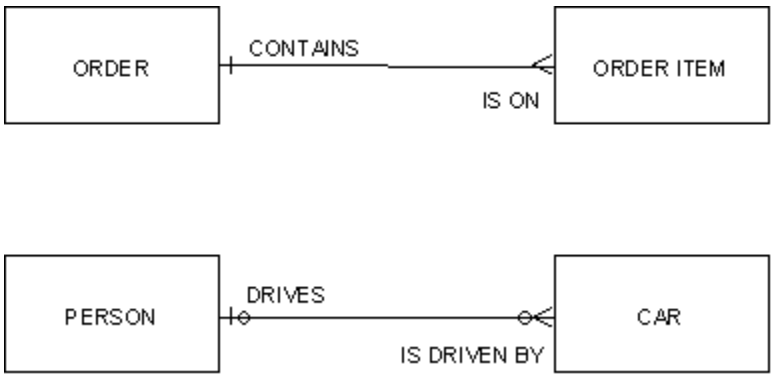
The distinction between optionality and zero cardinality, as the other alternative is named, is academic. This book and CA Gen express zero cardinality through the optionality of a relationship membership.

Entities whose pairings are all optional tend to be the more permanent ones in the business. A CUSTOMER can exist independently of the existence of entities of any other type, but an ORDER cannot exist without a related CUSTOMER.

The optionality of a relationship can be one of the following options:

- Fully mandatory—Both memberships are mandatory.
- Fully optional—Both memberships are optional.
- Partly optional—One membership is mandatory; one is optional.

The following illustration shows examples of both a fully mandatory relationship and a fully optional relationship.



An ORDER cannot exist without ORDER ITEMs, neither can an ORDER ITEM exist without an ORDER. On the other hand, a CAR can exist with no assigned driver, and a PERSON can exist without having a CAR to drive.

In practice, most relationships are partly optional. Some cardinality and optionality combinations are unusual and is viewed with suspicion.

Note: For more information, see the chapter Building the Analysis Model (see page 201).

Often, the existence of a pairing for an optional relationship membership depends on some combination of attribute values and existence of relationship memberships. This combination is named an optionality condition.

The following types of relationship optionality conditions:

- Required—A pairing must exist if the condition is true.
- Forbidden—A pairing may not exist if the condition is true.
- Unrestricted—The pairing can exist regardless of other values.

The unrestricted condition requires no explicit definition. For example, assume that the entity type PERSON has the attributes Job Title and Age. Assume also that if Job Title of a PERSON has a value of "Chauffeur," then that person must have a pairing with a CAR. This optionality condition for the relationship membership PERSON drives CAR can be represented as: Required when PERSON Job Title is equal to "Chauffeur."

Likewise, assume that if Age of a PERSON is less than the legal driving age (say, 16) then that PERSON cannot drive a CAR. This optionality condition can be represented as: Forbidden when PERSON Age is less than 16.

Note the implied constraint that a PERSON under 16 cannot be a Chauffeur.

Record these optionality conditions in the description of the relationship membership that is being affected. Later, during Interaction Analysis, these conditions can be incorporated into process specifications. See the chapter "Analyzing Interactions" for more information.

More information:

Analyzing Interactions (see page 121)

Relationship Pairing Percentage

For an optional relationship membership, the probability that at least one pairing is likely to exist for a given entity is useful during database design and for projecting the performance of the planned system. You can conveniently gather this information from the business.

For example, assume that 100 customers exist and 95 of them have an outstanding order. The probability of a CUSTOMER being paired with an ORDER based on the places relationship membership is therefore 0.95, or 95.

This can be specified in CA Gen, and by showing the percentage in parentheses, as follows:

- Each CUSTOMER sometimes (95%) places one or more ORDERs
- Each ORDER always is placed by exactly one CUSTOMER

The second membership is mandatory, so it remains unchanged.

Number of Relationship Pairings

Another piece of information that can be specified is the estimated number of pairings participated in by an entity, that is the single member in a many relationship memberships. This information is useful during performance analysis and database design.

In the following example, there is only one many relationship memberships: customer places one or more orders. The number of pairings of interest represents the potential number of orders to be placed by a single customer.

In CA Gen, you specify the following information for the number of pairings:

- Minimum number of pairings for the single entity, and whether that minimum is estimated or absolute.
- For optional memberships, the minimum is obviously zero. However, the value that is specified here presupposes the existence of at least one pairing. So, consider "the minimum number of pairings if any exist."

- Maximum number of pairings for the single entity, and whether that maximum is estimated or absolute.
- Average number of pairings for the single entity

For example, assume that experience teaches the business the following things:

- Each CUSTOMER who has any ORDERs outstanding typically has at least two.
- A CUSTOMER is never expected to have more than ten outstanding ORDERs outstanding.
- Most CUSTOMERs have three outstanding ORDERs.

These numbers are estimates, and a customer who wishes to place an 11 order will not be turned away based on a technicality. Given these assumptions, the sentences is modified as follows:

- Each CUSTOMER sometimes (95 percent) places at least two (estimated), at most 10 (estimated) and on average three ORDERs
- Each ORDER always is placed by exactly one CUSTOMER

The second membership remains the same because it is neither optional nor many.

A relationship membership in which the minimum, maximum, and average are equal, and for which the minimum and maximum are absolute, is said to have fixed cardinality.

There are cases where the number of pairings that are allowed varies depending on combinations of values. For example, a customer whose credit rating indicates that a questionable risk is allowed one outstanding order at a time. A customer with a more favorable rating is allowed to place many orders at one time. You can document such a rare situation by recording a cardinality condition in the description of the affected relationship membership.

Relationship Transferability

In some cases, it must be possible to move a pairing from one entity to another entity of the same entity type.

For example, two relationship memberships:

- Each PROJECT MANAGER sometimes manages exactly one PROJECT
- Each PROJECT always is managed by exactly one PROJECT MANAGER

This model represents a changeable situation. A PROJECT is always managed by a PROJECT MANAGER, but the identity of the PROJECT MANAGER change.

For example, Lisa Trueheart is been assigned the management responsibility instead of Ernest R. Steadman. As a result, the pairing “is managed by Ernest R. Steadman” must be transferred to Lisa Trueheart (another entity of type PROJECT MANAGER).

This illustrates a characteristic of relationship memberships that are called permanence of membership.

A relationship membership can be one of the following options:

- Permanent
  - Once a pairing is established, it relates the same two entities until the entities are disassociated.
  - A relationship membership whose pairings are permanent is named a fixed membership.
- Temporary
  - The pairing can move from entity to entity as needed.
  - A relationship membership whose pairings are temporary is named a transferable membership.

In the CUSTOMER/ORDER example, it is stated that after a customer places an order, the order at the customer's request be transferred to a different customer. Therefore, the membership ORDER is placed by CUSTOMER is transferable.

The enterprise never performs the converse (transferring a customer between orders).

To complete the sentences, these facts are appended:

- Each CUSTOMER sometimes (95 percent) places at least two (estimated), at most 10 (estimated) and on average three ORDERS and is not transferable
- Each ORDER always is placed by exactly one CUSTOMER and is transferable

Transferring a relationship causes knowledge of previous pairings to be lost. Ask whether the business needs to know which entities were formerly associated. If so, the membership should not be transferable, and the relationship cardinalities must also be reviewed.

For example, if the former managers of a project remain of interest, then a project is managed (over time) by many project managers.

More information:

Building the Analysis Model (see page 201)

Associate Is Modifying or Referencing

A property of each relationship membership, and defines whether procedure executions can simultaneously access an entity through that membership.

- Associate is modifying—Only one procedure execution at a time can access an entity through that membership, so allowing that entity to be updated or deleted.
- Associate is referencing—More than one procedure execution at a time can access an entity through that membership. This is safe providing no procedure execution allows that entity to be updated.

This property affects the behavior of the generated system, and probably let the system designer specify it and for system performance analysts to review. "Modifying" is the default in CA Gen model; this default is the safe setting for the integrity of the database.

Relationship Description

In general, the sort of information that is captured as part of the description reflect the following types of integrity conditions for relationship memberships:

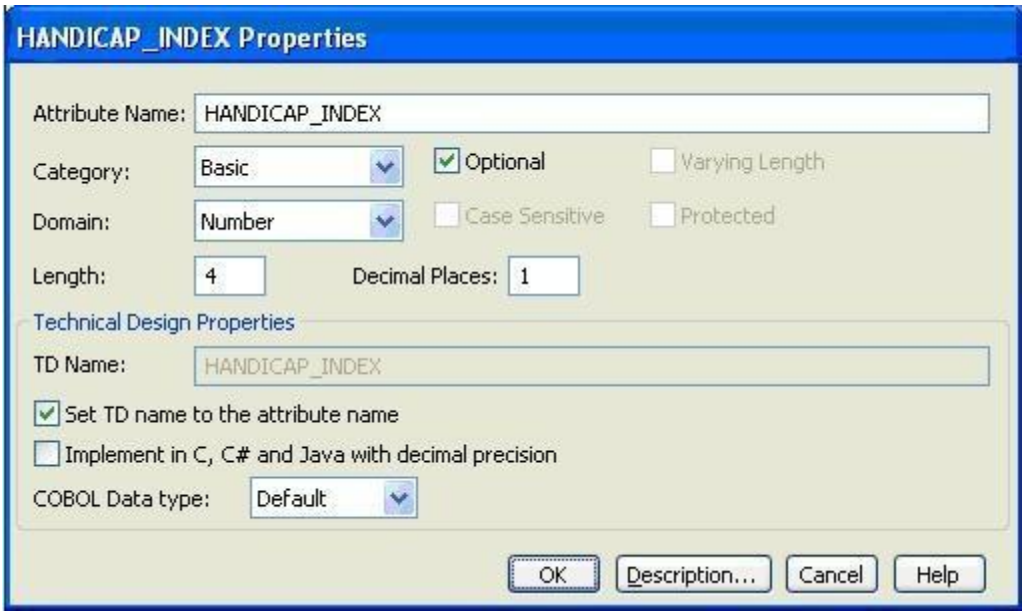
- Cardinality conditions
- Optionality conditions

Defining Entity Type Attributes

For each attribute of an entity type, you can record the following information:

- Name
- Description
- Optionality
- Source category (basic, designed, derived, auto number)
- Domain (number, text, date, time)
- Length
- Number of decimal places (numbers only)
- Case sensitivity (text attributes only)
- Permitted values
- Default value or algorithm
- Derivation algorithm (derived attributes only)
- Aliases

Following illustration shows an example to define the properties of an attribute HANDICAP_INDEX:



Attribute Name

The attribute name is a noun or noun-phrase describing the purpose or content of the attribute, preferably using terminology prevalent in the business.

The following lists the attributes of the entity type CUSTOMER:

- Number
- Name
- Address
- Phone Number
- Status
- Credit Rating
- Date Known
- Outstanding Orders Value

Note: None of the attribute names shown here include the name of the entity type. Since each attribute is associated with exactly one entity type, repeating the entity type name in the attribute name is redundant. CA Gen automatically adds the entity type name where necessary.

Attribute Description

The description of an attribute include any textual information that the business or analyst believes to be useful.

There are two optional elements of an attribute description:

- Attribute's definition—Elaborates on the information about the attribute's role available from its name
- Any optionality conditions

Attribute Optionality

The optionality of an attribute indicates whether each entity of the entity type that is described by the attribute must have a known value for the attribute under consideration.

There are two options for attributes:

- Mandatory—Attributes for which each entity must have a value
- Optional—Attributes for which each entity does not need to have a value

Consider the attributes of the entity type CUSTOMER:

- Number—Each customer is always assigned a Number when the business is made aware of the customer.
- Name—Each customer with which the business deals must have a Name.
- Address
  - A customer have more than one Address.
  - A customer’s Address may not yet be known to the enterprise.
  - The customer announce a change of address, and the business may need to know the value of a former address. If so, Address is multi-valued, this is discussed later in Additional Data Modeling Topics, or clearly defined as being the current address only.
- Phone Number
  - A customer may have no telephone, or that the Phone Number is not yet known to the enterprise. If the business accepts customers with no telephones, then some customer entities exist with no value for their Phone Number attribute.
  - A customer may have more than one Phone Number.
- Status—Each customer is always assigned a Status when the business is made aware of the customer.

- Credit Rating—Some customers may not require credit, preferring instead to pay cash for all purchases. In such a case, the attribute Credit Rating would not have a value.
- Date Known—The date on which the business became aware of the customer always be present.
- Outstanding Orders Value—The Outstanding Orders Value for a customer always has some value, even if it is zero.

Of the attributes for the entity type CUSTOMER, five are mandatory and three are optional as summarized in the following table.

Attribute	Optionality
Number	Mandatory
Name	Mandatory
Address	Optional
Phone Number	Optional
Status	Mandatory
Credit Rating	Optional
Date Known	Mandatory
Outstanding Orders Value	Mandatory

As with relationship memberships, the optionality of an attribute can sometimes be affected by a combination of values of other relationships and attributes. In this case, the combination can be represented as an optionality condition.

There are three types of attribution optionality conditions:

- Required—A pairing must exist if the condition is true.
- Forbidden—A pairing may not exist if the condition is true.
- Unrestricted—The pairing can exist regardless of other values.

For example, assume that the attribute Status can have an attribute value of Cash Only and that only CUSTOMERs with that Status are allowed to have no Credit Rating. The following optionality condition would apply to Credit Rating: Required when Status is not equal to Cash Only.

Forbidden conditions are not nearly as common as Required ones, but they are possible. Consider the case where another possible value of Status is Highly Credit Worthy. Assume that the business wants to avoid offending such CUSTOMERs with queries about their credit worthiness and to forbid the inclusion of a value for Credit Rating for them. Further assume that the previous Cash Only optionality condition is true.

The new optionality conditions for Credit Rating are:

- Required when Status is not equal to Cash Only and Status is not equal to Highly Credit Worthy
- Forbidden when Status is equal to Highly Credit Worthy

When you find that the same integrity condition determines the existence of several optional attributes and relationship memberships, it is worth considering defining a subtype or another entity type and moving the affected attributes and relationships so that they describe only entities of that subtype.

Using CA Gen, optionality conditions for an attribute is specified as part of its description.

More information:

Defining Partitioning (see page 64)

Attribute Source Category

An attribute fall into one of four source categories:

- Basic—The values of the attribute are intrinsic to entities of the entity type being described and cannot be deduced from the attribute values of other attributes or the existence of relationships.
- Derived—The values of the attribute are always deduced or calculated from the values of other attributes or the existence of relationships.

  Note the word always. For example, the value of the attribute ORDER ITEM Price may have been initially derived using the then current PRODUCT Price. However, it is not repeatedly re-evaluated in the same way whenever PRODUCT Price is modified. This attribute value is not treated as derived.
- Designed—The enterprise invents the attribute to overcome some sort of business constraint or to simplify a system operation. The attribute value is derived in some way, such as a numerically increased serial number.
- Auto Number—The DBMS of the application generates the value of an auto number attribute automatically. An auto number attribute must be a whole number with no decimal places.

CA Gen treats designed attributes as if they were basic.

CA Gen treats derived attributes differently. Values are never merely set to some value; they must be derived using a calculation that is called a derivation algorithm that is assigned to the attribute as part of its definition.

Note: For more information, see Derivation Algorithm (see page 58).

As an example, the likely source categories of attributes for the entity type CUSTOMER are listed in the following table.

Attribute	Source Category
Number	Designed or Auto Number
Name	Basic
Address	Basic
Phone Number	Basic
Status	Designed or Derived
Credit Rating	Basic, Designed Or Derived
Date Known	Basic
Outstanding Orders Value	Derived

Status and Credit Rating are considered to be either designed or derived, under differing conditions.

Credit Rating could be basic if its value is determined outside the organization, for example, by a credit agency.

Credit Rating could be derived if it is given a value by the business as the result of a specified calculation, or if it is subject to tightly specified rules. However, if it is the result of a less formally specified judgment, it is designed.

Outstanding Orders Value for a particular CUSTOMER entity is assumed to be the sum of the Amounts (an attribute) of all ORDERs (an entity type), which are placed by (a relationship membership) that CUSTOMER. So, its value depends wholly on the values of other attributes and relationship memberships. If the customer places a new order, causing a new pairing that is based on places/is placed by, the value of Outstanding Orders Value is likely to change. If the Amount of any order that is placed by the customer is modified, the value of Outstanding Orders Value change. Outstanding Orders Value is therefore a derived attribute.

Attribute Domain

The term domain refers to the collection of possible values for an attribute or set of attributes. Put another way, the domain of the attribute must include the set of all values that can actually be assigned to that attribute.

An attribute domain is:

- Primitive—A primitive domain is a collection of values that constrain the values of attributes or of parts of complex or user domains.
- CA Gen requires each attribute to be assigned to one of the following primitive domains:
  - Text—Each value is a string of characters.
  - Mixed Text—Each value is a combination of a double-byte character set and a single-byte character set.
  - DBCS Text—Each value is a double-byte character set.
  - Number—Each value is a number (positive or negative, integer, or real).
  - Date—Each value represents a date (including century).
  - Time—Each value represents an hour, minute, and second.
  - Time Stamp—Not relevant in analysis; it is dealt with during design.
  - GUI Object—Each value represents a pointer of an object linking and embedding (OLE) object. This option is available only with the workset attribute.
  - BLOB—Each value represents a variable sized Binary Large Object.
- Complex—A complex domain contains elements each of which has values that are constrained by a primitive domain.

  Some analysts consider the domains Date and Time to be complex domains that are based on the primitive domain Number. Dates and times are so universally important and uniquely configured that they are regarded here as primitive domains.
- User-defined—The first four primitive domains (Text, Number, Date, and Time) can be used as the basis for forming a hierarchy of other domains, named user-defined domains.

  Note: For more information, see Additional Data Modeling Topics (see page 67).

The values of attributes from different primitive domains cannot usually be compared or used together in calculations. For example, it makes no sense to compare CUSTOMER Number with CUSTOMER Address. Makes no sense to multiply ORDER Quantity by Current Date. However, be required to add a number of days to a date.

The following table shows the primitive domains for the attributes of the entity type CUSTOMER.

Attribute	Primitive Domain
Number	Text

Attribute	Primitive Domain
Name	Text
Address	Text
Phone Number	Text *
Status	Text
Credit Rating	Text
Date Known	Date
Outstanding Orders Value	Number

*Assume that Phone Number includes parentheses and hyphens for readability.

Attribute Length

The length of an attribute indicates the maximum number of characters or digits for each of its values. Considered to be a subset of the domain of an attribute, because the number of characters or digits restricts the possible set of values for the attribute.

Using CA Gen, the maximum length of an attribute varies based on the primitive domain to which it belongs.

The following table lists the possible lengths of attributes that belong to the primitive domains.

Domain	Possible Attribute Length
Text (fixed)	1 - 255
Text (varying)	1 - 4094
DBCS Text (fixed)	1 - 127
DBCS Text (varying)	1 – 2047
Mixed Text (fixed)	1 - 255
Mixed Text (varying)	1 – 4094
Number	1 - 18
Date	8
Time	6
Timestamp	20
BLOB	1 byte to 2 gigabytes

Note: No range of lengths is available for attributes belonging to the domains Date, Time, and Timestamp because the formats of these attributes are fixed.

Each Date includes four digits for the year, two for the month, and two for the day.

Each Time includes two digits for the hour, two for the minute, and two for the second.

CA Gen automatically sets the length property of Date and Time attributes and does not allow them to be modified. Specify the length of an attribute before recording its permitted values.

Attribute Number of Decimal Places

This property is available only for attributes belonging to the Number domain. It reflects the number of digits of the attribute's total length, which fall to the right of the decimal point.

The following table illustrates the relationship between the length and number of decimal places.

Format of Number	Length	Decimal Places
9999	4	0
999.9	4	1
99.99	4	2
9.999	4	3
.9999	4	4

Attribute Case Sensitivity

For text attributes, you can use the case-sensitive option to specify whether the attribute is to contain both uppercase and lowercase characters or only uppercase characters. If both uppercase and lowercase characters are needed, this option must be specified. This option is because the current CA Gen default is "Not case sensitive."

In the generated system, this default causes all characters in any views of the attribute to be translated to uppercase before any operations are performed on them, regardless of how the characters were originally entered into the system.

Attribute Varying Length

For text attributes, the varying length option allows you to specify that the length of the text attribute can vary when stored in the database. The database only stores the actual length of the text string.

For example, assume that varying length is specified for the Name attribute of the CUSTOMER entity type and the length of the attribute is 30 characters. In the implemented system, if the value of CUSTOMER Name is only 20 characters, the database only stores 20 characters.

Technical Design Name

You can select to set the TD name to be the same as the attribute name. This is the default condition.

Also enter a different name to be used while transforming or retransforming the local data model into the selected Technical Design.

Implementing with Decimal Precision

COBOL store and manipulate numbers with precision. The native C, C#, or Java data types have limitations in either the number of digits or their precision. For example, most numbers with decimals are generated as a double which has a limit of 15 digits of precision, and loses precision when used in some mathematical operations.

CA Gen generate an alternative data type that can achieve precision of 18 digits. CA Gen does not generate these alternative data types by default. Rather, review your needs and determine if numeric attributes are generated in this alternative form.

Using the alternative data type consumes more CPU resources, as it uses a software library to perform the mathematical operations, whereas, the double data type uses the hardware instructions to perform the same mathematical operations.

C Language DBMS Considerations with Decimal Precision

For C generated code, CA Gen implements all attributes that are marked for decimal precision as a character string. Some DBMSs pass the string directly to the database and store it with high precision. However, other DBMSs require a double data type to store and retrieve numeric data. Therefore, limit the number of digits and precision that can be safely stored and retrieved.

COBOL Data Type

Four COBOL data types are available as attribute properties for technical design. These data types are for attributes in the number domain.

- COMP—binary data items
- COMP1—internal floating-point items with single precision
- COMP2—internal floating-point items with double precision
- COMP3—internal decimal items

You can use these data types to specify the COBOL internal format to allow:

- Compatibility with externally provided data
- Better performance by allowing the data to be stored internally in the most efficient form (either storage or processing time)

Note: The COMP COBOL data types are not generated for Procedure Step import and export views. They are generated for Procedure Step local views. For action blocks, the COMP COBOL data types are generated for import, export, and local views. If the number of decimal places is defined for any numeric attribute, specified COMP COBOL data type has no effect on generation.

You can specify that attribute state flags not be defined on a view-by-view basis. This permits exact matching of the external storage format. This eliminates the movement of data that is required in the code that is generated for USE statements to match between the external action blocks view without attribute state flags and the action block view with them.

Attribute Permitted Values

You can use permitted values to further restrict the domain of certain kinds of attributes. The set of permitted values for an attribute exhaustively describes the potential values of the attribute.

For example, there are seven specified values for Day (of week). If you specify no permitted values for an attribute, CA Gen considers any value consistent with the attribute's primitive domain and length to be a legal value.

You specify permitted values for any attribute that is either basic or designed, and either text or a number. Auto Number and BLOB attributes may not have permitted values, a default value, or an algorithm.

The permitted values for text attributes are restricted to discrete values.

The permitted values for number attributes include ranges and discrete values.

Each permitted value is given its own description in CA Gen.

Attribute Default Value or Algorithm

For each attribute that contain permitted values (basic or designed, text, or number), you specify either a default value or default algorithm, but not both.

- The default value is a single value for the attribute that must be among its permitted values.
- When creating an entity, CA Gen automatically assigns default values to all attributes for which default values were specified, unless another value is explicitly set. For example, imagine that customer Status has a default value of "Cash Only." In such a case, whenever a customer is created, Status is automatically set to "Cash Only" unless it is explicitly overridden.
- The default algorithm is a calculation whose result is used to initialize an attribute value.
- When the entity that contains the attribute is created, the attribute is assigned its value that is based on the result of the default algorithm. For example, suppose that CUSTOMER Numbers are assigned sequentially. When a new CUSTOMER is created, its Number attribute receives the value of the previously created CUSTOMER Number plus one. The rule that states this is a default algorithm. For example: (CUSTOMER Number = previous CUSTOMER Number + 1).
- The specification of default algorithms requires the use of the Action Diagramming Tool. Details on building a default algorithm are, therefore, covered in the chapter Analyzing Interactions (see page 121).

Derivation Algorithm

You specify derivation algorithms only for derived attributes.

Every time the value of a derived attribute is requested, the derivation algorithm executes, so the value change over time. For example, EMPLOYEE Age is derived from the Date of today and EMPLOYEE Birth Date, whenever the business knows how old the employee is. If the business knows the value that was originally calculated (for example, EMPLOYEE Age When Hired) a default algorithm is used, not a derivation algorithm.

There are some performance considerations that are associated with derived attributes. In practice, designers must decide whether a derived attribute is calculated or stored; this decision is a performance-oriented design decision.

If calculated, the value of the derived attribute is recalculated every time that it is requested.

If stored, the value of the derived attribute is recalculated only when one of the attributes and relationships on which it depends is modified.

The distinction between calculation and storage is relevant during analysis if performance is a critical issue. This calls for performance analysis in parallel with analyzing the business requirements; otherwise it is deferred until design.

Derivation algorithms are specified during procedure design using the Action Diagramming Tool.

If rules for deriving the value are critical to the business, then the algorithms are specified during analysis. See the chapter "Analyzing Interactions."

More information:

Analyzing Interactions (see page 121)

Attribute Aliases

As with entity types, each attribute is known by a number of different names. For example, the attribute Number of the entity type CUSTOMER is known variously as NUM, NO, CN (Customer Number), or Numeric Identifier. Each synonym is recorded as an alias of the attribute.

Each alias is identified as an acronym (such as CN), an abbreviation (NUM or no), or neither (such as Numeric Identifier). Additionally, a long name of up to 50 characters can be recorded as an alias of the attribute. The long name is typically used only to provide compatibility with data dictionary products that support names longer than the 32-character CA Gen standard.

Defining Entity Subtypes and Partitioning

You find that entities belonging to the same entity type have differing characteristics. This situation requires the definition of entity subtypes and partitioning.

For example, consider this expanded attribute list for the entity type CUSTOMER:

- Number
- Name
- Address
- Phone Number
- Status
- Credit Rating
- Outstanding Orders Value
- Nationality (Foreign Or Domestic)

- Country Code
- Import License Number
- Currency
- Tax ID Number
- State of Incorporation
- Type (Commercial Or Government)
- Government Agency Name
- Interested Politician Name

Given this set of attributes, assume the following attributes to be true:

- The attributes: Country Code, Import License Number, and Currency have values only for FOREIGN CUSTOMERs.
- The attributes: Tax ID Number and State of Incorporation have values only for DOMESTIC CUSTOMERs.
- The attributes: Government Agency Name and Interested Politician Name are valid for either FOREIGN CUSTOMERs or DOMESTIC CUSTOMERs, but only when the CUSTOMER is a GOVERNMENT CUSTOMER, not a COMMERCIAL CUSTOMER.

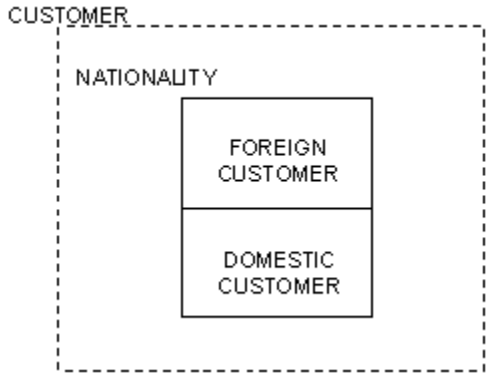
You could record these facts as a set of integrity constraints on CUSTOMER. However, you would handle this situation by defining types.

Defining Entity Subtypes

The list of extended attributes for the entity type CUSTOMER in Defining Entity Subtypes and Partitioning (see page 59) shows that entities of the type CUSTOMER fall into two mutually exclusive groups: FOREIGN and DOMESTIC.

Note: For more information, see Entity Subtype and Partitioning (see page 33).

The following illustration shows an Entity Relationship Diagram in which the entity type CUSTOMER has been separated into the entity subtypes FOREIGN CUSTOMER and DOMESTIC CUSTOMER.



The entity types FOREIGN CUSTOMER and DOMESTIC CUSTOMER are subtypes of the entity type CUSTOMER. In the illustration, the dashed box indicates a partitioning of the entity type CUSTOMER into subtypes. Another way of saying this is that the entity type CUSTOMER is a supertype to FOREIGN CUSTOMER and DOMESTIC CUSTOMER.

The name appearing inside the dashed box, NATIONALITY, is the name of the classifying attribute of the partitioning. Classifying the attributes are discussed later in this section.

Relationships and attributes particular to an entity subtype are mutually exclusive with relationships and attributes particular to other subtypes in the same partitioning. For example, a FOREIGN CUSTOMER may not have values for Tax ID Number and State of Incorporation, while a DOMESTIC CUSTOMER may not have values for Country Code or Import License Number.

Subtypes are said to inherit the relationships, attributes, and integrity conditions of their supertypes. For example, every FOREIGN CUSTOMER can have values for all relationships and attributes that are defined for CUSTOMER, and those defined specifically for FOREIGN CUSTOMER. Since each CUSTOMER must have a value for the attribute Number (it is a mandatory attribute), each FOREIGN CUSTOMER must also have a Number. Similarly, because each CUSTOMER place ORDERs, each FOREIGN CUSTOMER also place ORDERs.

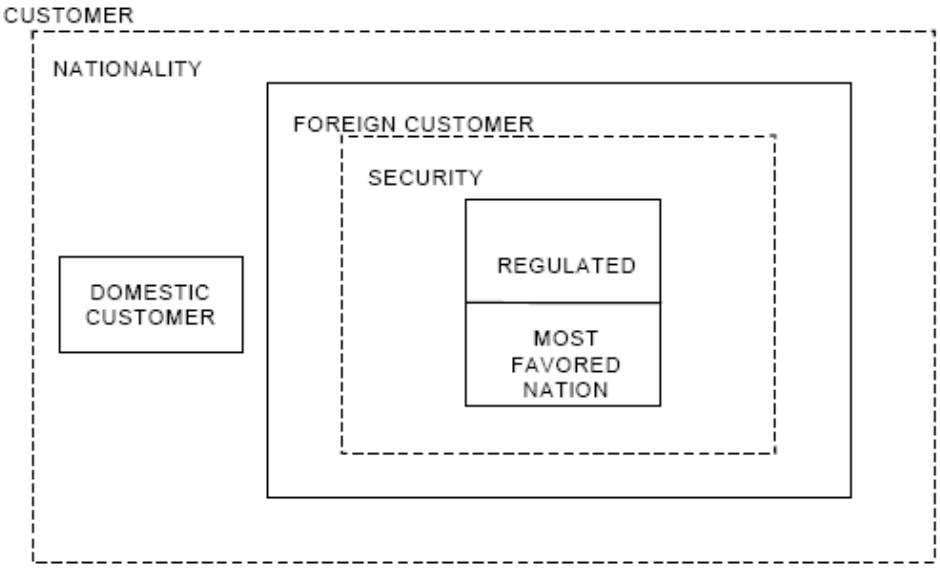
The optionality of relationships and attributes of a subtype depends, on whether a given entity belongs to that subtype. In the previous illustration, for example, assume that Country Code is defined as a mandatory attribute of the subtype FOREIGN CUSTOMER. Here, every FOREIGN CUSTOMER must have an attribute value for Country Code. A DOMESTIC CUSTOMER cannot have a value for Country Code.

Partitioning

The Entity Subtypes show that the entity type CUSTOMER partitioned into two subtypes along a single partitioning. However, there is no limit to the number of subtypes that can be defined in a partitioning.

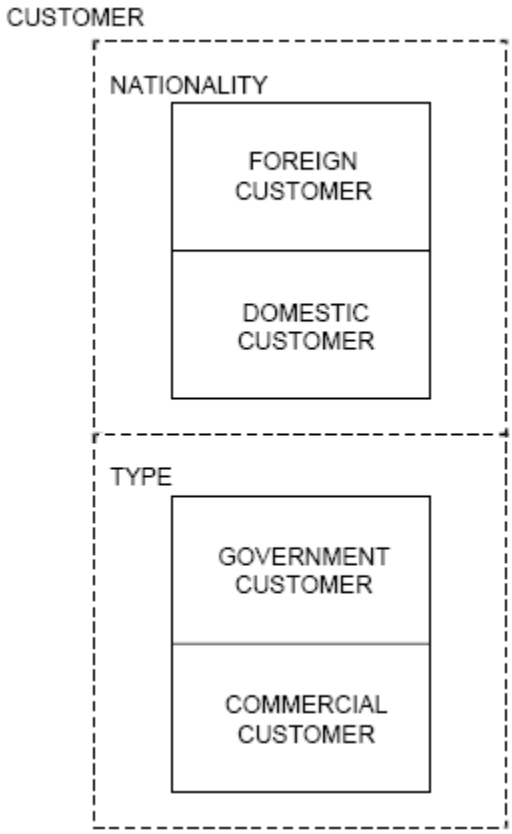
Note: For more information, see Entity Subtype and Partitioning (see page 33).

A subtype can also have partitioning of its own. For example, the subtype FOREIGN CUSTOMER have different relationships and attributes, depending on whether the CUSTOMER resides in a FAVORED NATION country, as shown in the following illustration.



Analysts divide each entity type into multiple partitionings, where each partitioning is a different reason for having subtypes. For example, the list of extended attributes for the entity type CUSTOMER and the integrity constraints that resulted in the first partitioning show the potential for a second partitioning: GOVERNMENT CUSTOMERs have relationships and attributes not shared by COMMERCIAL CUSTOMERs.

Since a GOVERNMENT CUSTOMER can be either A DOMESTIC CUSTOMER or a FOREIGN CUSTOMER (that is, GOVERNMENT CUSTOMERs are not mutually exclusive with DOMESTIC or FOREIGN CUSTOMERs), GOVERNMENT and COMMERCIAL CUSTOMERs are broken down into their own partitioning, as shown in the following illustration.



Subtypes within a partitioning are mutually exclusive with each other but not with subtypes in other partitionings. In this example there are at least four possible permutations for CUSTOMER:

- FOREIGN, GOVERNMENT
- DOMESTIC, GOVERNMENT
- FOREIGN, COMMERCIAL
- DOMESTIC, COMMERCIAL

Some situations require cross-partitioning restrictions to accurately reflect the needs of the business. For example, business rules dictate that the company does not deal with FOREIGN, GOVERNMENT CUSTOMERs. Record such a restriction as an integrity constraint of the entity type being partitioned, in this case, in the description of customer.

Defining Partitioning

You specify the following for each partitioning:

- Description
- Enumeration
- Classifying attribute
- Life-cycle partitioning
- Classifying values (one for each subtype)

Normalization

Normalization is a step-by-step technique that is used to ensure that the analyst has assigned each attribute to the proper entity type and defined sufficient entity types. The technique of normalization involves refining an entity type that is based on the interdependencies of its attributes.

The principles of normalization can be used throughout data analysis as each attribute is added to the model, and as a final confirmation of the model. Normalization can also be employed as needed for inspecting current system data and for defining entity types that data represents.

Each step of the technique results in the conformance of the entity type to a normal form. For the purposes of confirmation, it is sufficient to normalize to the third normal form (abbreviated 3NF).

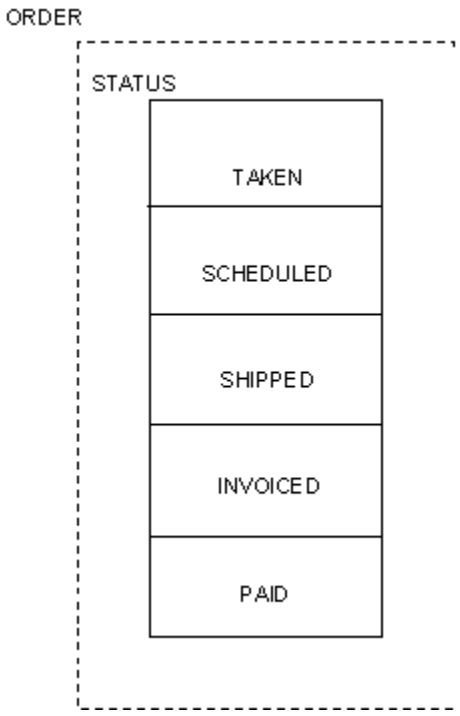
C. J. Date (An Introduction to Database Systems, Vol. I, Addison Wesley.) describes higher normal forms.

The following table shows the three normal forms of normalization with the steps that cause an entity type to conform to these forms.

Normal Form	Description
First Normal Form (1NF)	Multi-valued attributes are removed to form a separate entity type. Thus, no entity type in 1NF can have repeating groups (multi-valued attributes).
Second Normal Form (2NF)	Attributes that are not fully dependent on the identifiers of the group are removed to a separate entity type.
Third Normal Form (3NF)	Attributes that are dependent on attributes, other than the identifiers are removed to a separate entity type.

The steps of normalizing data are best performed by examining example values of data that is found in the business, or developed with the help of users to illustrate the combinations of data values that arise.

The following illustration shows a group of attribute values that are laid out as columns in a table. Separate line items of a possible entity type ORDER appear as lines in the table.

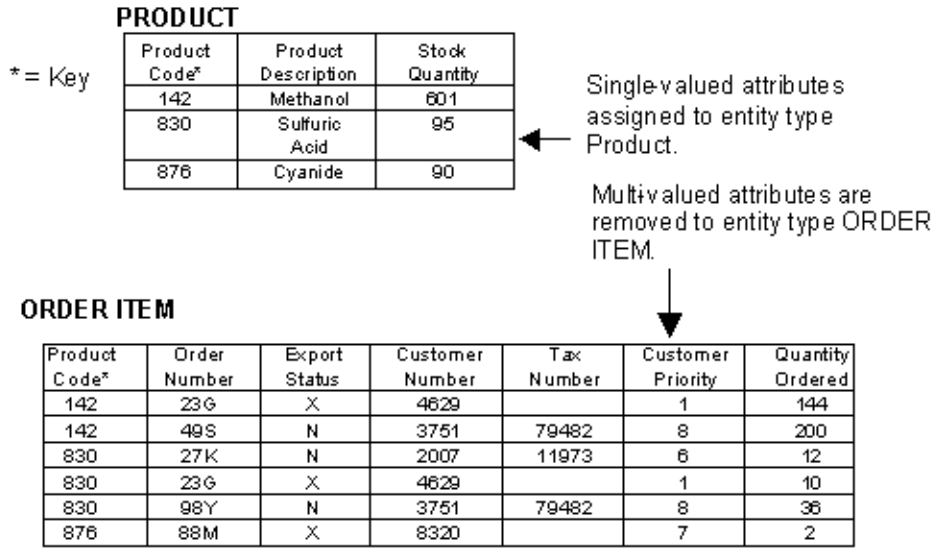


The next three illustrations show the normalization of an attribute group. Normalization involves dividing the original entity type into implied entity types. Normalization also involves assigning attributes to the correct entity types; later this is useful for data design. In this case the table eventually breaks down into four implied entity types: PRODUCT, ORDER ITEM, ORDER, and CUSTOMER.

The following illustration shows an attribute group in first normal form (1NF) with sample data values.

Product Code	Product Description	Stock Quantity	Order Number	Export Status	Customer Number	Tax Number	Customer Priority	Quantity Ordered
142	Methanol	601	23G 49G	X X	4629 3751	79482 79482	1 8	144 200
830	Sulfuric Acid	95	27K 23G 98Y	N X N	2007 4629 3751	11973 79482	6 1 8	12 10 36
876	Cyanide	90	88M	X	8320		7	2

In second normal form (2NF) one level of multivalued groups of attributes are removed to the ORDER ITEM entity type to form a separate entity type as shown in the following illustration.

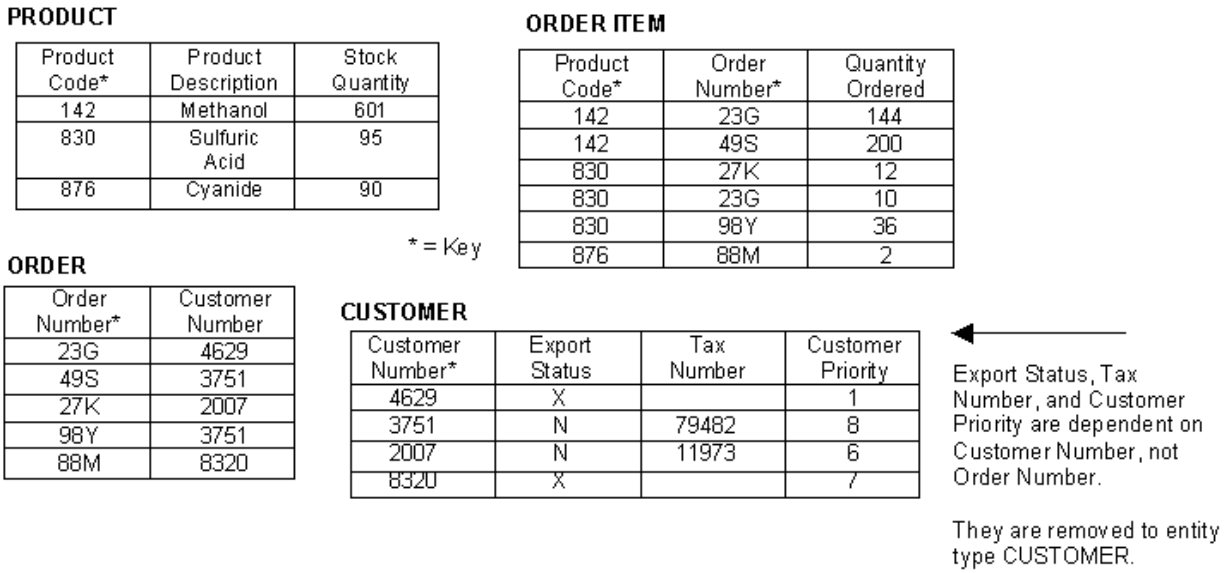


Product Code appears as the identifier of PRODUCT and as the foreign identifier of ORDER ITEM (1NF).

In third normal form (3NF), attributes that are not fully dependent on the identifier of the entity type are removed to additional entity types until no partial dependencies remain.

PRODUCT is already in 3NF because its attributes are dependent on the identifier Product Code. However, the attributes of ORDER ITEM are not fully dependent on Product Code. Removing these attributes to other entity types (ORDER then CUSTOMER) must be done to remove partial dependencies.

The following illustration shows the attribute group in third normal form (3NF).



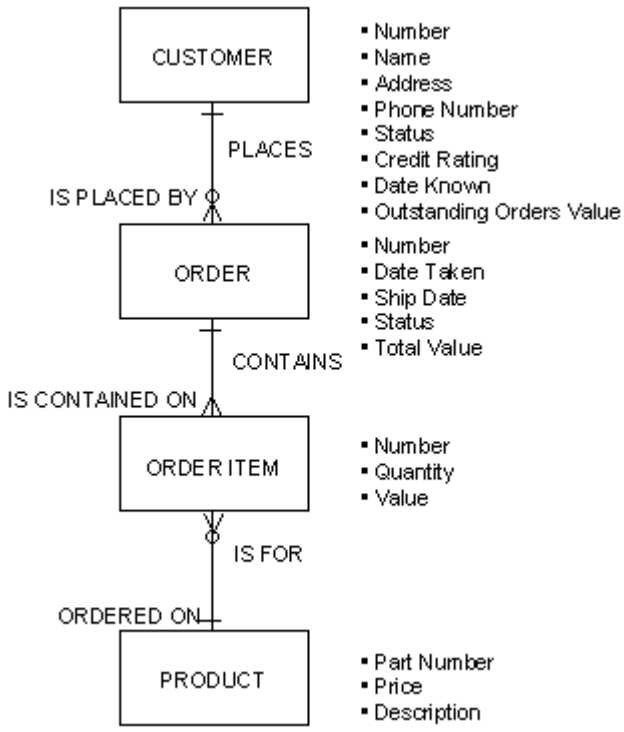
Additional Data Modeling Topics

More About Identifiers

An identifier of an entity type is a combination of attributes and relationships whose values uniquely identify an entity.

Each entity type must have at least one identifier, and many have only one, but an enterprise identify entities in more than one way. A salesperson use a name of a customer, while an accountant needs a number of a customer. CA Gen allows up to five separate identifiers to be specified for each entity type. Each identifier of an entity type must uniquely identify an entity.

The following illustration shows part of an Entity Relationship Diagram.



Here, a new entity type, Order Item, has been added. Although attribute names do not typically appear on Entity Relationship Diagrams, they are included in this example to help clarify the notion of identifiers.

Beginning with the entity type CUSTOMER, assume that the attribute Number is an identifier. In other words, no two CUSTOMERs have the same attribute value for Number. Further assume that while two customers can have the same name, and two customers can live at the same address, no two CUSTOMERs can have the same name and can live at the same address. The combination of Name and Address can therefore be an identifier.

So, the entity type CUSTOMER has two identifiers, as shown in the following table.

Identifier	Description
Identifier 1	Number
Identifier 2	Name Address

Next, consider the entity type ORDER. Assuming that no two ORDERs have the same values for Number, then Number is an identifier of order.

Likewise, the entity type PRODUCT has an identifier, which is an attribute that is called Part Number.

ORDER ITEM is slightly different. Assuming that the ORDER ITEM attribute named Number is unique only within an ORDER, no value of ORDER ITEM Number can be used to uniquely identify each of its occurrences. The following illustration, for example, shows two orders, each with order item Numbers of 001, 002 and 003.

Order Number 25607 Date, Fri., Sep. 30, 1999

No.	Part Number	Description	Quantity	Price	Total
001	5ZA96	Bolt	40	0.25	10.00
002	36B92	Nut	40	0.40	16.00
003	1A462	Bracket	20	1.00	20.00
Total	46.00				

Order Number 11572 Date, Fri., Aug. 19, 1999

No.	Part Number	Description	Quantity	Price	Total
001	1A462	Bracket	10	1.00	10.00
002	782B67	Clip	40	0.30	12.00
003	1A462	Retainer	20	0.40	8.00
004	27B566	Fastening	4	1.50	6.00
Total	36.00				

However, because the value of ORDER ITEM Number is unique for a given order, you can uniquely identify each ORDER ITEM entity by the combination of its pairing with a particular order and its Number, that is, by a combination of one relationship membership and one attribute: Identifier 1 is contained on order "order item Number."

If there is a business rule that each product can appear only once on an order (that is, no two ORDER ITEMs on any one ORDER reference the same PRODUCT), another identifier emerges that requires no attributes at all.

An ORDER ITEM can be uniquely identified by the combination of its pairing with a specific ORDER and a specific PRODUCT. The identifier is therefore composed entirely of relationship memberships. As a result, the set of identifiers for ORDER ITEM is:

- Identifier 1 is contained on order "order item Number"

- Identifier 2 is contained on order "is for product"

There are some necessary restrictions on the attributes and relationship memberships that participate in identifiers:

- Each entity must have a value for each element of every identifier, and these elements must therefore be mandatory.
- In the previous example, it is found that ORDER ITEMs exist for which the product cannot yet be identified. If this is so, then the relationship membership is for PRODUCT cannot be used to identify ORDER ITEMs.
- An attribute or relationship membership that participates in an identifier cannot be changed by a business operation. Therefore, the initial value of an attribute cannot be changed.

For example, if it is accepted that a CUSTOMER can change Address, then this should not be part of the identifier for CUSTOMER.

It also follows that an identifying relationship membership cannot be transferred. This will become significant in interaction analysis. For example, suppose that the business needs to be able to substitute a PRODUCT on an ORDER ITEM. Since the relationship membership is for PRODUCT must not be transferable, then either it should no longer form part of an identifier, or when altering the PRODUCT on an ORDER ITEM, one ORDER ITEM would need to be deleted, and another created and associated with the new PRODUCT.

Analysts familiar with database terminology might recognize that the concept of an identifier is similar to that of a key. Do not, however, be influenced by database concepts, facilities or design techniques when defining the identifiers used by business. There are important differences:

- In analysis all useful ways to identity entities may be defined, but in database design it is usually necessary to choose identifiers to define as keys.
- Database keys generally imply sequence, but an analyst need not specify the attributes and relationships that comprise an identifier in any particular order because an identifier ensures uniqueness, not sequence.
- A database management system may allow the specification of a primary key and multiple secondary indexes but generally only requires the primary key to be unique, whereas all identifiers must ensure uniqueness.
- In database design, a relationship membership may be represented by duplicating a data item and using it as a foreign key.

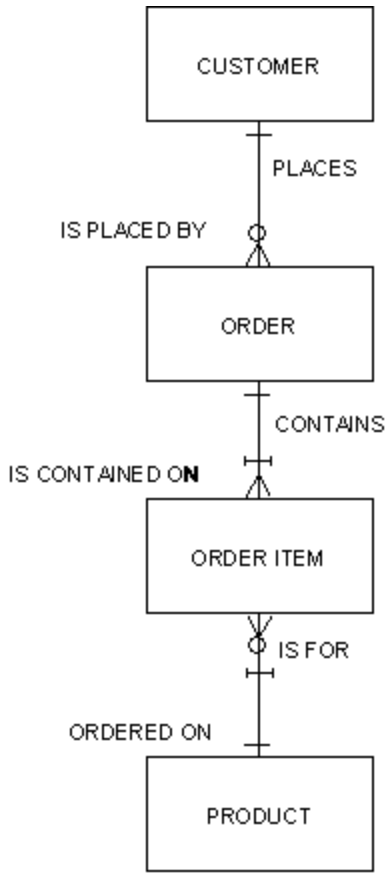
For example, instead of using is contained on ORDER as an identifier of ORDER ITEM, you could have added the identifier of ORDER (in this case, Number) as an attribute of ORDER ITEM, so that each ORDER ITEM contains the Number of the ORDER on which it appears. Since Number is not really an attribute of ORDER ITEM, but exists only to facilitate access to ORDER, it is called a foreign key.

A foreign key serves the same purpose as a relationship: to relate two entities. However, when using CA Gen in database design, a relationship membership should always be preferred as an identifier to using an attribute as a foreign key.

Explicitly including an attribute as a foreign key obscures the fact that entities of the two affected types can be related and requires that the value of the attribute be maintained explicitly by a business system to reflect the value of an attribute in the paired entity.

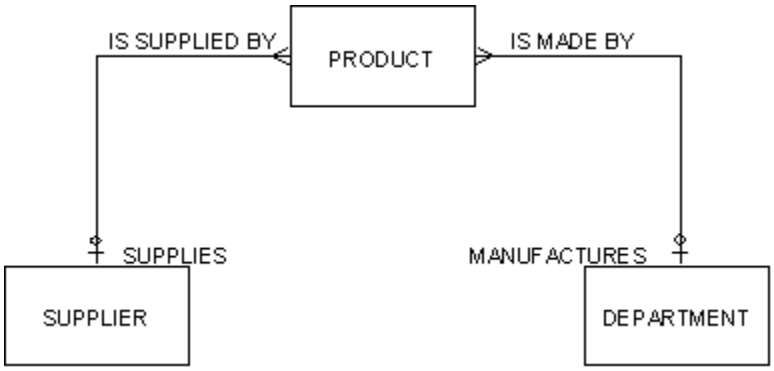
A relationship explicitly defines a possible pairing, so using relationships as identifiers provides a more accurate representation of the view of data for business, and allows the developer to postpone decisions on how to implement data until design is reached.

In CA Gen, relationship memberships that participate in identifiers are highlighted on the Entity Relationship Diagram with an "I" superimposed over the end of the relationship nearest to the entity type it identifies. The following illustration shows this convention assuming that "is contained on ORDER" and "is for PRODUCT" both participate in identifiers.



Mutually Exclusive Relationships

Consider the case where the PRODUCT entities are either manufactured by an internal manufacturing DEPARTMENT or purchased from a SUPPLIER. Assume that each PRODUCT can be acquired from only one source. In other words, each PRODUCT entity either is supplied by a SUPPLIER or is made by a DEPARTMENT, but never both. Here, the existence of a pairing that is based on one of the relationships precludes the existence of one based on the other. For a given entity type, relationship memberships whose occurrences preclude the existence of occurrences of other relationship memberships are said to be mutually exclusive of one another as shown in the following illustration.



More than two relationship memberships can be mutually exclusive. For example, PRODUCT could also be stocked by SUPPLIER, for shipment direct to customers. A relationship membership that is involved in a mutually exclusive set must be optional, and may not therefore be part of an identifier.

In CA Gen, mutual exclusivity of a set of relationship memberships is recorded when defining the entity type to which the relationship memberships belong. Analysts specify up to five sets of mutually exclusive relationship memberships for each entity type.

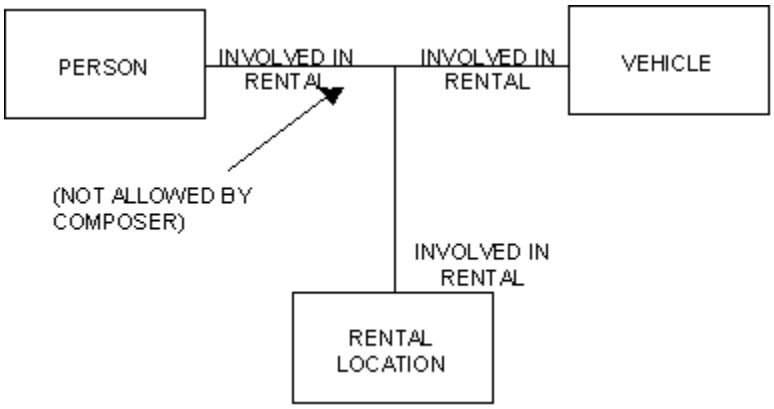
Alternative Relationship Representations

The way of handling relationships in CA Gen appears to be different from some traditional entity relationship modeling approaches.

In CA Gen, only entity types can have attributes. The relationship memberships are considered to be details of entity types; not objects in their own right. A relationship can therefore never have an attribute. Usually, if you feel the need to add an attribute to a relationship, you have discovered a new entity type (probably a conceptual or active one).

In CA Gen, a relationship has exactly two memberships; two and only two entities can participate in a single pairing. This style of relationship is named a binary relationship. Other modeling approaches allow the appearance of three or more entities in a single relationship occurrence. The term "pairing" makes no sense in this context. That style of model is said to support n-ary relationships. Using CA Gen, a relationship that appears to require more than two memberships is replaced by an entity type. An entity type that is introduced for this reason is named an associative entity type.

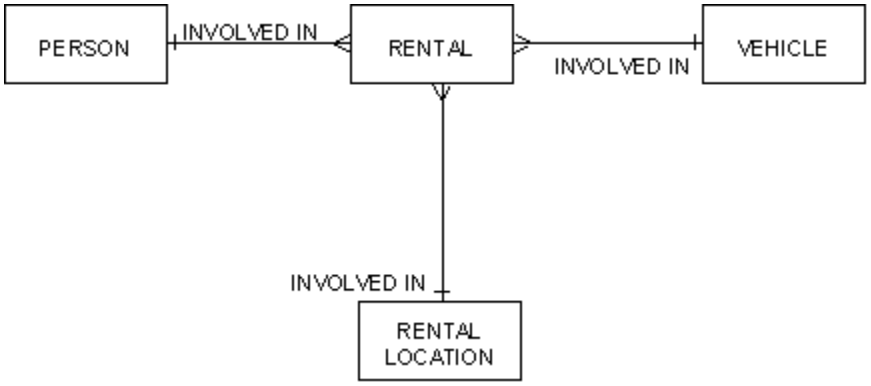
For example, consider the following illustration.



This illustration shows a three-membership relationship between PERSON, VEHICLE, and RENTAL LOCATION, which represents a leasing arrangement.

Because traditional modeling approaches allow only two memberships in a relationship, a fourth entity type, Rental, and some new relationships, must be added to model the business reality. In practice, you find further attributes of RENTAL, such as Start date, Start time, Rental period.

The resulting relationships, which are shown in the following illustration, are now all binary.



Although the approach of representing relationships (binary with no attributes) in CA Gen initially appears restrictive, it greatly simplifies the operations on relationships that are specified during interaction analysis. Because you can model the same set of real world instances using either style of modeling, and operations on model objects are much less complex, the binary/no-attribute style of relationship modeling is superior in the context of analysis.

General Integrity Constraints

In some cases, certain combinations of attribute values and relationship memberships can be invalid for entities of a given type. Conditions that determine the validity of entities of a given type are named integrity constraints.

A number of kinds of integrity constraint have already been introduced. Optionality conditions for relationship memberships and attributes, cardinality conditions for relationship memberships, permitted values for attributes, and the mutual exclusivity of relationship memberships are all examples of special kinds of integrity constraints. However, these specifications do not exhaustively address all issues that can determine the validity of an entity. To do so, specify a general integrity constraint.

In CA Gen, integrity constraints that are not otherwise addressed are captured as part of a description for an entity type. During activity analysis, ensure that the integrity constraints documented during data analysis are reflected in the process definition.

Two examples of conditions that are addressed by general integrity constraints are:

- **Complex permitted values**—Certain permitted values of an attribute are valid only when other attributes have specific values, or when specific pairings exist. For example, imagine that for a CUSTOMER to have the Status: Preferred. They must have a Credit Rating of Impeccable and at least one outstanding ORDER. An analyst record this, in the description of the entity type CUSTOMER, as: Status is equal to Preferred is valid when: Credit Rating is equal to Impeccable, and places ORDER.
- **Mutually inclusive relationships**—One pairing can exist only if another exists. For example, assume that a PRODUCT can only appear on an ORDER ITEM if it is supplied by a SUPPLIER or is made by a DEPARTMENT. An analyst record this condition, in the description of the entity type PRODUCT, as: Invalid when ordered on order item and not supplied by supplier and not made by a department.

User-Defined Domains

Complex and user-defined domains were mentioned in the primitive domains section. The user-defined domains define sets of possible values that are shared by multiple attributes.

For example, PRODUCTs are available in a limited number of colors. Elsewhere in the business, there is an entity type EMPLOYEE whose security badge can be one of a number of badge colors.

Assuming a standard list of colors, it is possible to have a common definition for both the PRODUCT attribute Color and EMPLOYEE attribute Badge Color, named simply Color, which has the same:

- Primitive domain
- Length
- List of permitted values

Such a definition is named a user-defined domain. The specification of the user-defined domain Color is as in the following table.

Primitive Domain	**Text**
Length	25 characters
Permitted Values	red blue yellow silver gold green

Ask whether the set of permitted values is stable or whether more colors could be added.

Assume that this is unlikely (but not impossible) for badge color but possible for product color. You have another option for modeling this situation: a new entity type COLOR with an attribute Name. The set of COLOR entities includes Red, Blue, and so forth, with the possibility of adding, for example, a Brown entity when the business needs it.

The use of domains provides some benefits over the use of discrete attribute definitions:

- Domain definitions can be shared (using the Copy attribute facility). If, however, the domain changes (a new permitted value is added), the change must be reflected manually in all the separate copied attribute definitions.
- Operations involving attributes from different domains can be restricted to avoid illogical conditions. For example, if there is one domain that is called Price and another named Quantity, it is reasonable to allow multiplication of attributes belonging to the two domains, as in: (ORDER ITEM Amount = PRODUCT Price * ORDER ITEM Quantity).

However, it is probably not reasonable to allow the attributes to be compared (If PRODUCT Price is greater than ORDER ITEM Quantity, …) or added together.

Dates should not be added, multiplied, or divided.

You can represent user-defined domains by observing the following guidelines:

- If possible, use the domain name as the name of the attribute, as in the example that is shown in the following table.

Domain QUANTITY	**Attributes**
	Entity Type ORDER ITEM Attribute QUANTITY Entity Type INVOICE ITEM Attribute QUANTITY Entity Type PRODUCT STOCK Attribute QUANTITY

- When this convention cannot be followed easily, domain information could be recorded as part of the description, or the domain name could be included as part of the attribute name. In the following table, two attributes belong to the Color domain, so some additional qualification is required to distinguish between them.

Domain COLOR	**Attributes**
	Entity Type PRODUCT Attribute COLOR OPTION Entity Type EMPLOYEE Attribute BADGE COLOR Entity Type BUILDING Attribute EXTERIOR COLOR Attribute INTERIOR COLOR

- With reasonable domain names, in practice this convention almost implements itself. In the preceding examples, for instance, the attribute names naturally contain the names of the domains to which they belong. It is difficult to imagine a sensible formation of those attribute names that does not include the domain name. For instance, EMPLOYEE Badge Hue? Or: INVOICE Number of Things?
- An enterprise usefully assemble a set of common domain names. The following table shows some possibilities.

Domain	Type of Values Allowed	Description
Address	Any	Postal or physical address in standard format.
Code	Any	Attribute with value not in natural language, for which one or more other attributes define the meaning of each code value.
Count	Numeric integer	Integer quantity of objects.
Duration	Numeric	Length of time in specified unit of time.
Indicator	Any (typically Yes or No)	Code that has two or three permitted values.
Percentage	Numeric	Ratio expressed as percentage.
Period	Any	Name of a specified length of time.
Price	Numeric	Monetary quantity that can be exchanged for goods or services.
Rate	Numeric	Value that is commonly understood to be a percentage or ratio (for example, interest rate or exchange rate, respectively).
Ratio	Numeric	Proportion of amounts of two objects, expressed as number not as percentage.
Value	Numeric	Monetary quantity that expresses the value of an object.

- Copy the definition of the attribute
- To help manipulate attributes, the copy command allows an attribute's definition to be copied to a different entity type. Through judicious use of this command, you can use an attribute in a particular domain as a basis for defining other attributes in the same domain.

- For example, the ORDER ITEM attribute Quantity could be copied to create the attribute Quantity of the entity type INVOICE ITEM. This is done by copying the definition of Quantity from the entity type ORDER ITEM to the entity type INVOICE ITEM.
- Because CA Gen does not enforce the continued use of this definition, this is not a permanent solution if the permitted values are expected to change.

Composite Attributes

A composite attribute is one that is made up of other attributes.

Attributes of the sort so far covered in this chapter are sometimes named elementary attributes to distinguish them from composite attributes.

An example of a composite attribute is the attribute Phone Number of the entity type CUSTOMER, as shown in the following table.

Entity Type CUSTOMER
Composite Attribute Phone Number
Elementary Attribute Area Code Elementary Attribute Exchange Elementary Attribute Station

Decomposing Phone Number into its elements is unnecessary in most business situations. If, however, the business happens to be a telephone company, there would be a need for Phone Number to be a composite attribute. This allows Phone Number to be treated as a single attribute in some cases, while still making it possible to inspect and modify its elements individually whenever necessary.

CA Gen does support the concept of composite attributes. This is achieved indirectly. The composite attribute can easily be simulated by defining three attributes and using a derivation algorithm to assemble the complete Phone Number.

Multi-Valued Attributes

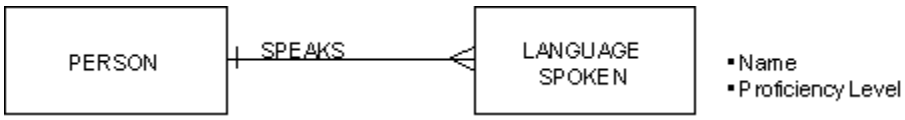
Occasionally, an attribute seems to hold multiple values simultaneously.

For example, consider an entity type PERSON, which has Language Spoken as an attribute. A distinguishing characteristic of a PERSON is, that they speak many languages. So, the attribute Language Spoken would seem to require multiple attribute values for each entity of PERSON.

An attribute that seems to require multiple values simultaneously is named a multivalued attribute.

If, in this example, know the level of proficiency for each language, the case for defining extra entity type that is named LANGUAGE SPOKEN becomes strong.

CA Gen does not, and should not, provide for multivalued attributes. Rather, remove an apparently multivalued attribute to its own entity type and relate it to the original entity type using a 1:M relationship as shown in the following illustration.

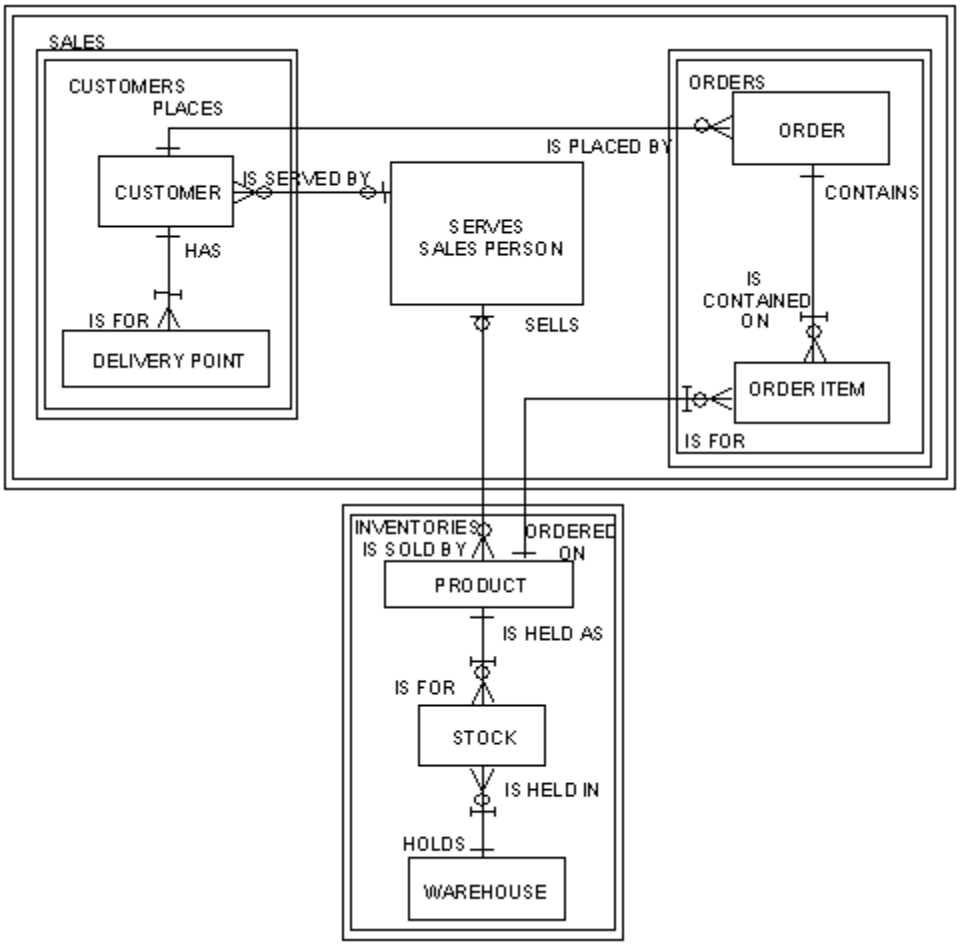


Drawing Entity Relationship Diagrams

Since the Entity Relationship Diagram (ERD) is primarily a means of communication, attention is given to its layout.

In larger Entity Relationship Diagrams, it is sometimes useful to look closely at a small portion of the diagram while hiding much of the surrounding diagram. CA Gen Data Modeling Tool supports variable-sized boxes to represent entity types, and the contraction of subject areas allowing for detail to be hidden. This provides a good deal of flexibility to enhance the aesthetics of the diagram and to support the focused presentation of meaningful diagrams.

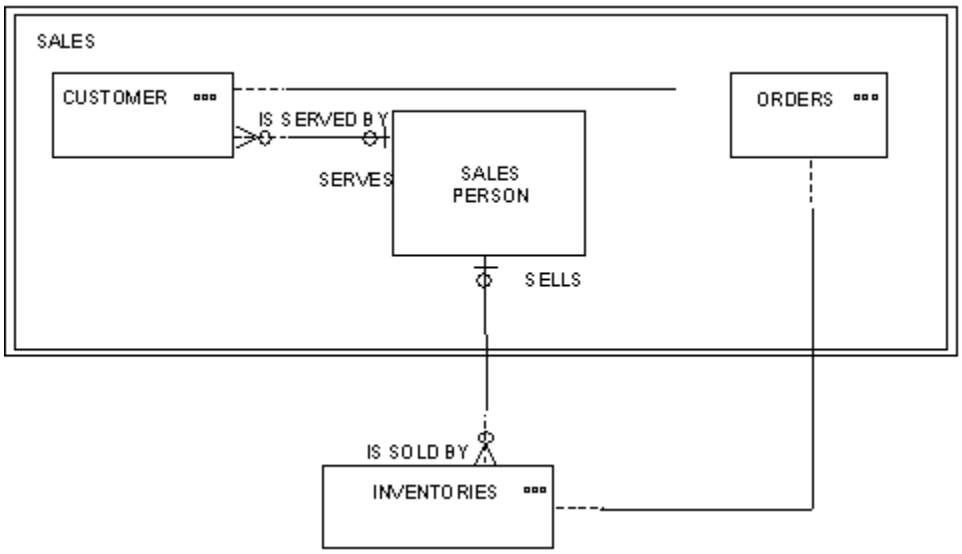
The following illustration shows an Entity Relationship Diagram with four subject areas, SALES, INVENTORIES, CUSTOMERS, and ORDERS.



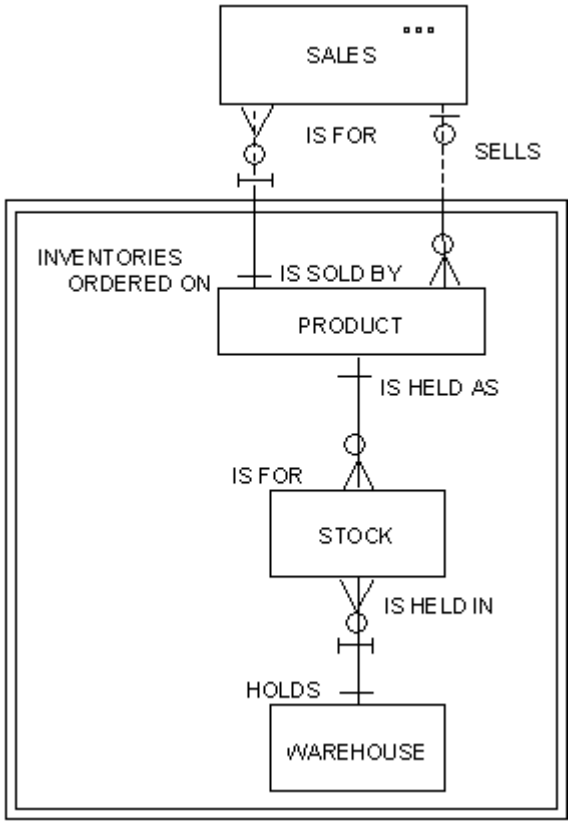
Following is the description of the diagram:

- The subject area SALES includes the subject areas CUSTOMERS and ORDERS and the entity type SALESPERSON.
- The SALESPERSON entity type box has been emphasized by its size and central placement, while other entity types have been made smaller.
- The subject area CUSTOMERS includes the entity types CUSTOMER and DELIVERY POINT.
- The subject area ORDERS includes ORDER and ORDER ITEM.
- The subject area INVENTORIES includes product, stock, and warehouse.

To produce a version of the diagram that further emphasizes the entity type SALESPERSON, the underlying details of CUSTOMERS, ORDERS, and INVENTORY can be hidden by contracting those subject areas, as shown in the following illustration.



To produce a version of the diagram featuring elements of the subject area inventory, only to contract the subject area SALES, as shown in the following illustration.



Summary of Data Modeling Rules

You can verify these rules automatically, for selected model objects or for a whole model, by running Consistency Check. In addition, the Consistency Check software points out a number of the suspect conditions that are mentioned earlier. Those situations that must be addressed before the analysis model can be considered complete are flagged as errors. Any questionable situations with no definite errors are flagged as warnings.

CA Gen enforces some rules automatically and does not permit data modeling objects that contain errors to be used for system design or in the Data Structure Diagram. See the chapter "Block Mode Design" and the chapter "Client/Server Design" for more information.

Rules for Entity Types

- Each entity that is described by an entity type must be uniquely identifiable.
- An entity type must have at least one attribute or two relationship memberships.
- An entity type must participate in at least one relationship.
- An entity type is immediately part of one, and only one, subject area.

Rules for Relationships

- Each relationship associates one or two entity types, and depicts a pairing between exactly two entities.
- A relationship must not have attributes.
- An optional relationship membership may not participate in an identifier.
- A relationship membership with a cardinality of "many" may not participate in an identifier.
- A transferable relationship membership may not participate in an identifier.
- A relationship membership that is involved in a mutually exclusive set may not participate in an identifier.

Rules for Attributes

- An attribute describes exactly one entity type.
- An attribute must have, at most, one value for any entity of the entity type it describes.
- An attribute must not have attributes of its own.

- An optional attribute may not participate in an identifier.
- The initial value of an attribute that participates in an identifier may not be changed.
- A derived attribute may not participate in an identifier.

Rules for Identifiers

- An optional relationship membership may not participate in an identifier.
- A transferable relationship membership may not participate in an identifier.
- A relationship membership that is involved in a mutually exclusive set may not participate in an identifier.
- The initial value of an attribute that participates in an identifier may not be changed.
- An optional attribute or relationship may not participate in an identifier.
- A derived attribute may not participate in an identifier.
- A BLOB attribute may not participate in an identifier.

Rules for Subject Areas

- A lowest level subject area decompose into at least one entity type.
- Each entity type within a subject area is related to at least one other entity type in the same subject area.

Rules for Partitioning and Subtypes

- Each subtype belongs to exactly one partitioning.
- Each partitioning that is not a life-cycle partitioning must be associated with a classifying attribute that belongs to the entity type that it partitions.
- Each subtype must be identifiable by a classifying value, or a range of classifying values.
- A fully enumerated partitioning must divide the entity type it partitions into two or more subtypes.
- Each entity type can have only one life-cycle partitioning and that partitioning must be fully enumerated.
- A subtype within a non life-cycle partitioning must have at least one special attribute or relationship membership.

Results of Data Analysis

- Top-down data analysis produces the following elements of the analysis model:
- An Entity Relationship Diagram describing the data requirements of the development project
- Supporting documentation, which is stored as details of the model objects appearing on the Entity Relationship Diagram

Chapter 5: Analyzing Activities

Activity analysis involves the continued decomposition of activities until the analyst has identified the lowest level processes of interest to the business (elementary processes).

The business model that is developed during analysis integrates three equally important aspects of business requirements:

- Data aspect of the model, which describes things of interest to the business and the relationships between them. See the chapter "Analyzing Data."
- Activity aspect of the model, which records things the business does.
- Interaction aspect of the model, which details how things the business does (activities) affect things of interest to the business (data). See the chapter "Analyzing Interactions."

This aspect serves to confirm, and if necessary modify, the analysis model. It also provides a detailed basis for system design.

You choose to perform process decomposition early in analysis to set or clarify the development scope. You can then progressively extend this decomposition until all elementary processes are identified.

During process decomposition, you verify the results and augment the model by creating Activity Dependency Diagrams and Event Lists. Dependency diagrams show the conditions that processes create, and how processes execute according to conditions that result from the execution of other processes. Event Lists identify the activities that participate in the response by business to external and internal events.

If the business objectives of the development project require it, you model activities as they are currently done. You can later refine the model to obtain definition for an improved or re-engineered business process. This subsequent model is the one that the principal participants must confirm as accurate.

Much of the work of defining processes centers around their interaction with data. The chapter "Analyzing Activities" addresses that topic. For a description of how activities and data are analyzed together. See the chapter "Building the Analysis Model."

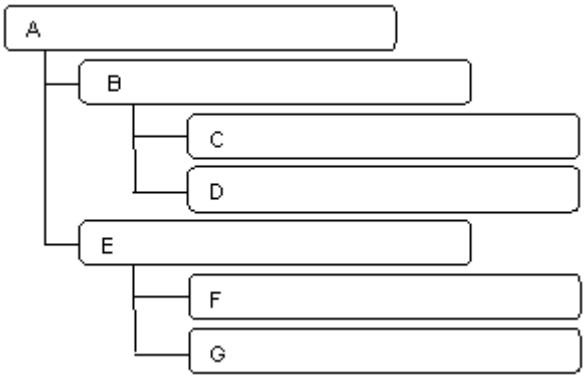
Basic Activity Modeling Concepts

General Nomenclature for Hierarchies

The business activities are as follows:

- High-level activities (or functions)
- Low-level activities (or processes)

A general nomenclature for describing any hierarchical or tree structure is useful when discussing process decomposition. The following illustration shows a typical Activity Hierarchy Diagram.

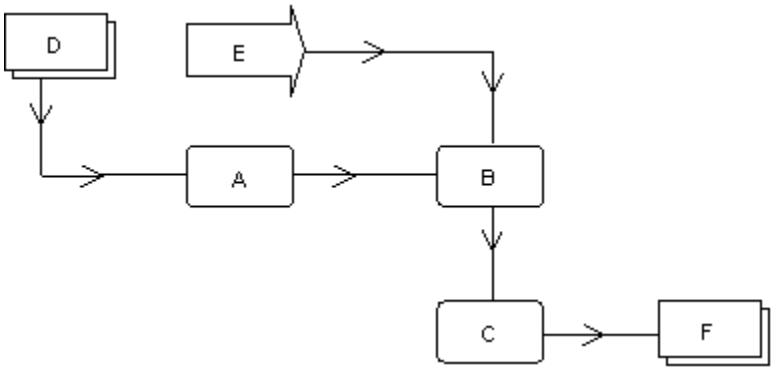


The following table defines the term:

Term	Definition
Root	The root of a hierarchy is its highest level activity. In the previous illustration the single root is A. The root of an activity hierarchy for a CA Gen model must be a function.
Leaf	The leaves of a hierarchy are its lowest level activities. C, D, F, and G are leaves.
Parent	Any activity to which other activities are subordinate is called the parent of the subordinate activities. A is the parent of B and E; B is the parent of C and D; E is the parent of F and G. Any activity that is not a leaf is the parent of some activity.
Child	Any activity that is not the root is subordinate to another activity and is called the child of that activity. C and D are children of B; F and G are children of E; and B and E are children of A.
Sibling	Two or more activities that share a parent are called siblings of one another. C and D are siblings; F and G are siblings; and B and E are siblings.

Basic Terminology for Dependencies and Events

The following illustration introduces some graphical symbols and terminology that is used in dependency and event analysis.



If A, B, and C are processes:

- The activities A, B, and C are interdependent because they are joined by a network of dependencies.
- The directed line (line with an arrowhead) from A to B represents a dependency of B on A.
- The directed line from B to C represents a dependency of C on B.
- An execution of B can be said to depend or enabled by A.
- C depends B.
- A is prerequisite to B.
- B is prerequisite to C.
- D and F are external objects.
- The directed lines from D to A and from C to E represent information flows.
- E is an event.
- The directed line from E to B represents a dependency of B on an occurrence of the event E.
- E initiates or enables B.
- B can be said to execute in response to E.
- E initiates or is prerequisite to B.

Defining Functions

A business function is a group of activities that together completely support one aspect of furthering the mission of the enterprise. Each function describes something the business does, regardless of the structure of the organization.

The highest-level business functions deal with major areas of interest (that is, subject areas) in the enterprise. Most organizations group major activities into five to ten top-level business functions, such as Marketing, Finance, and Manufacturing.

A function that is not subdivided into functions, but into processes, is known as a primitive function.

Any description of a function explains the function does. It is valuable to include the purpose of the function in the description.

The description of a function never describe who, when, where, or how because such information is not fundamental to the existence of the function. Rather, those aspects reflect the approach of the business to executing the activities that make up the function.

In a CA Gen model, a function is the highest level activity (the root activity) in the Activity Hierarchy Diagram.

When producing an Information Strategy Plan, planners subdivide the top-level business functions into smaller, more focused, lower-level business functions. You perform this division of business activities into successively smaller elements in both the planning and analysis.

Continue dividing the activities until you have identified the lowest-level, most fundamental activities. We call these most fundamental activities elementary processes.

Defining Processes

A process is a defined business activity whose executions are identified in terms of the input and output of specific entities or of data about specific entities.

Processes are activities for which a start and an end can be imagined. A process is something that can be said to be executable. An occurrence of a process is named a process execution.

For example, a single execution of a process that is called Take Order deals with the input of certain information about a particular order request and results in the creation of an ORDER entity.

Processes can be discovered by:

- Analyzing life-cycles of the business and of entities within the scope of the development project.
- Examples of entities whose life-cycles are examined for clues to processes include products, orders, and suppliers.
- Identifying the activities that create and manage the business and those entities with which it is concerned.
- This is best achieved by analysts working closely with representative and authoritative business staff who can together agree upon, name, and define activities in a way that is relevant and acceptable to the business.

CA Gen captures the following details about a process:

- Name
- Description
- Definition properties
- Usage properties
- Expected effects
- Information views
- Dependencies

Note: The properties of processes that are discussed here are optional for all but elementary processes. Usage properties, information views, and dependencies are typically recorded only for elementary processes.

Name

The name of a process consist of a verb-noun combination. The noun is typically the name of an entity type or attribute.

For example:

- Take Order
- Create Invoice
- Terminate Contract
- Calculate Account Balance

Order, Invoice, and Contract are likely to be the names of entity types whose occurrences are manipulated during process execution.

Account is an entity type with an attribute Balance that the process Calculate Account Balance modifies.

Description

The definition of a process, which is stored as part of its description in CA Gen, explains the process does and why it does it.

If these facts are recorded, it is clear that they merely reflect the current approach of the business to implementing the process. The definition describe a single execution of the process.

Who performs the process, when it is performed, or how it is performed are questions that are not fundamental to the existence of the process.

A process that is called Take Order, for example, defined as an order is created based on a customer request, with an order item for each different type of product requested.

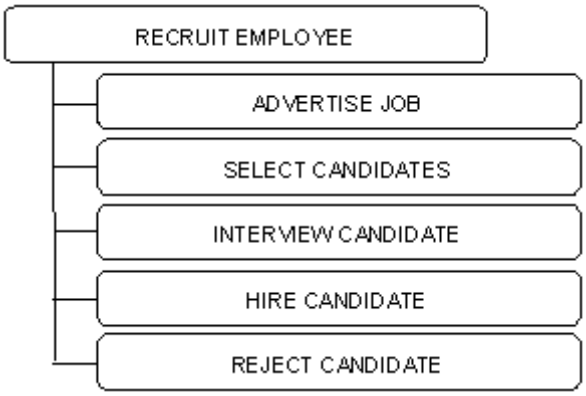
The description is the starting point for defining the actions of the process. It is therefore useful to assemble in the description the set of preconditions for an execution of the process, and the post-conditions that arise from an execution. These conditions are discussed later in Dependency Analysis.

Definition Properties

You record the following definition properties for a process:

- Whether it is an elementary process.
- For a discussion of the unique characteristics of an elementary process see Identifying Elementary Processes.
- Whether the process is repetitive, that is, whether it is likely to execute multiple times whenever a precondition arises that causes it to be executed.
- This property simply reminds you that the logic associated with the process (defined during interaction analysis, described in the chapter "Analyzing Interactions") should allow for repetition. The dependency that is associated with the precondition is named a repetitive dependency.

- For example, suppose that the process Recruit Employee is divided into five subordinate processes, as shown in the following illustration.



- Whenever an employee is recruited, Advertise Job executes once, so it is not repetitive. However, Interview Candidate executes one or more times depending on the number of interviews of selected candidates, so it is repetitive.
- A repetitive process is indicated by a horizontal bar within the process cushion. See Interview Candidate in the illustration.
- The suggested mechanism for implementing the process, or the principal mechanism if there may be more than one.
- This property is typically recorded only for elementary processes. This is a design decision, but when defining a process, you feel for the eventual implementation and can record this information.
- These are the options for implementing a process:
  - **Online**—A terminal operator carry out the process, interacting with some sort of terminal device.
  - **Batch**—A computerized batch job carry out the process, non-interactively.
  - **Manual**—The process is not computerized; this is not an option for elementary processes.
  - **Other**—The process is computerized but use some alternative other than online or batch. This classification of process is rare in business situations. Examples include Real-Time Systems and Process Control Systems.

Usage Properties

Usage properties include the "Expected frequency of occurrence," estimated number of executions of a process in a given time period, and the "Expected growth" of that number of executions over time. This information is useful in later stages, when defining databases, for instance.

Expected Effects

Expected effects define, at the level of entity types, the effect that the execution of an activity can have on data. For each entity type with which the activity deals, the analyst specify whether entities are read, created, updated, or deleted (C, R, U, or D).

When recording expected effects using CA Gen, you must have previously defined the affected entity types.

Considering expected effects prompt you to add further entity types to the model. This is one instance where activity modeling leads into interaction analysis.

You record expected effects for a process in process details, or by direct entry in the cells of the Entity Type/Process Matrix. For example, consider the process Take Order, which probably affects entities of the following types:

- CUSTOMER, since the process must read details of a customer placing an order.
- ORDER, since the order is created.
- ORDER ITEM, since each order likely contains multiple items, each of which is created.
- PRODUCT, since each order item is for a particular product that must be read.
- PRODUCT STOCK, since the stock level of a product must be reduced to reflect the number of items appearing on an order item.

The expected effects, then, appear in the matrix that is illustrated in the following table.

Entity Type	Created	Read	Updated
CUSTOMER		R	
ORDER	C		
ORDER ITEM	C		
PRODUCT		R	
PRODUCT STOCK			U

Remember that delete is rare in business. For legal, audit and statistical reasons, entities are updated to inactive status and not deleted until, often years later, a separate administrative process deletes entities that are no longer needed for any purpose.

Information Views

The information views of a process describe in detail which entities and attributes are referenced or manipulated during execution of the process.

Defining the content of information views closely associates a process with data. Therefore, the data model is stable before information views are defined. The chapter "Analyzing Interactions" includes a detailed discussion of information views.

Dependencies

Dependencies are the conditions that define how the execution of an activity depends resources, material, or information that are created or available as the result of an execution of other activities, or the occurrence of an event.

Process Decomposition

If an Information Architecture already exists, process decomposition is merely a continuation of the function decomposition in planning to discover lowest-level functions or highest-level processes.

A development project then uses, as a starting point, a subset of the Information Architecture established in ISP.

Developing a process hierarchy is a refinement of part of the diagram that is built previously for business functions.

Alternatively, activities of varying levels of detail already have been identified as part of a "business process chain" during Business Process Re-engineering.

If the development project is not expanding part of an already existing Information Architecture, then the analysis model that represents the scope of the project begins with a root function.

Whichever situation is the starting point, you proceed by subdividing the activity hierarchy.

Identifying Functions and Processes

Identify the functions as:

- Reviewing subject areas, described in the chapter "Analyzing Data," to identify those things with which the business must deal or manage.
- For example, sales includes customers and orders, which suggests the functions Customer Management and Sales Management.

- Identifying child activities that manage an object with which a function is concerned.
- For example, the Warehouse Management function include:
  - Warehouse Capacity Planning
  - Warehouse Acquisition
  - Warehouse Operations
  - Warehouse Performance Control
  - Warehouse Disposal
  - Considering the value chain of which a function is part.
  - Using value chain analysis is further described in the chapter "Building the Analysis Model."

Identify the processes as:

- Examining the life-cycles of entities that are managed to meet relevant business and organizational objectives. Entity type life-cycle analysis is described in the chapter "Analyzing Interactions."
- Comparing the activity hierarchy with current procedures identified from information gathering and current systems analysis.
- Identifying the events that affect the business, and identifying activities that are performed in response to those events. See the chapter "Event Analysis" for more information.

Further guidance on starting to identify activities appears in the chapter "Building the Analysis Model."

In analysis, process decomposition is recorded by building an Activity Hierarchy Diagram and continues until the lowest-level, or elementary processes, are identified.

Parallel Decomposition of Activities and Data

This analysis is performed in parallel with data modeling, as described in the chapter "Building the Analysis Model."

When a process is found, further entity types are implied. When a new entity type is described, identify the processes that establish and use entities of that type. In this way, the data and activity aspects of the model can remain complementary. Processes have the data that they need. Entity types that are needed by the business have processes that can maintain them.

In a system development project to improve customer management and retention, for example, the data list and activity hierarchies that are shown in the table have evolved in this way.

Activities	Data
Customer Management Function	Customers Subject Area
Develop New Customer (process)	Customer Entity Type
Establish New Customer (elementary)	
Establish Customer Address (elementary)	Customer Address Entity Type
Develop Customer Contact (elementary)	Customer Contact Entity Type
Support Customer (process)	
Modify Customer Name (elementary)	
Change Customer Address (elementary)	
and so forth	

Identifying Elementary Processes

An elementary process is a process that is the smallest unit of business activity of meaning to the business and that, when complete, leaves the business in a consistent state.

The main challenge during process decomposition lies in taking a business perspective on activities that are typically described by users in terms of their current procedures. This perspective is drawn from the business purpose or value to first model the processes that are currently performed and then all the activities that serve the purpose. This identifies a comprehensive set of elementary processes to be performed.

This ideal or improved set of processes are simpler than what is performed. For example, current procedures are grouped into processes that are performed at a single time, as soon as an order is received, or when an order is due to be delivered.

Activities that add value is performed as soon as possible, activities that incur cost, or that do not add value is performed as late as possible ("just in time"), or even omitted altogether.

The project objectives may involve continuous process improvement or radical business process re-engineering. These objectives guide analysts and business managers in the extent of the change that is desirable from the current procedures and processes.

As expected, a process that is described as elementary cannot be further decomposed, at least not into subordinate processes using the Activity Hierarchy Diagram. So, each elementary process is a leaf in the hierarchy. When the activity hierarchy is complete, all leaves are elementary processes.

Identify elementary processes properly because they form the basis for an implementation as parts of a business system.

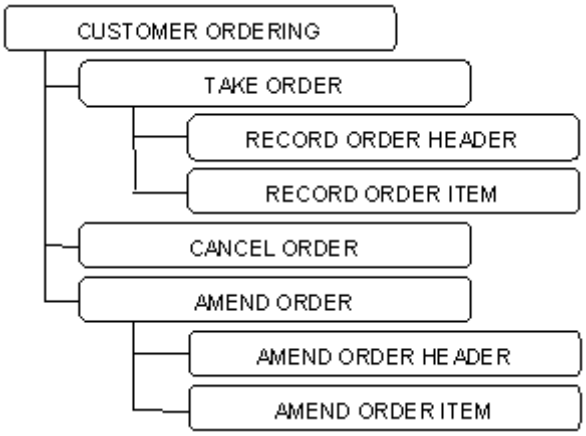
These guidelines are applied to test each process that is suspected of being elementary to ensure that it falls above or below the elementary process level.

- Each execution of the process must produce a result that, to the business, is complete and meaningful.
- Another way of saying this is that each execution must make all necessary changes to business data, and must leave all data in a consistent state, conforming to all data integrity rules.
- A process that violates this rule is below the elementary process level, indicating that decomposition has progressed to too low a level. Such a process is part of a higher-level activity that is an elementary process.
- The process operates on a set of data in common with its sibling processes to which it is closely related.
- The concept that is used to measure this degree of relatedness is named cohesion and is further described in the chapter "Building the Analysis Model."
- Each process must be included in the response of at least one event.
- It either itself be the first process that is directly enabled by the event, or it is indirectly connected to that first process through dependencies.
- None of the elements of process can be executed individually to leave the business data in a consistent state.
- This rule can be applied whenever event analysis has been performed. See the chapter "Event Analysis" for more information.
- A process that violates this rule is above the elementary process level, indicating that the process has elementary processes among its children.

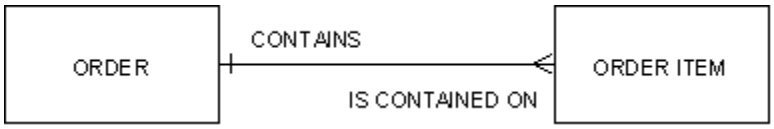
A test for a complete result would check to see that all data actions that must be performed together are performed together. For example, creating an order must include creating at least one order item.

A test for a meaningful result is to ask if the business recognize whether a process has taken place for the entities affected. For example, it is of interest to know that an order has been canceled, but whether the business wants to know that it has been amended need further investigation.

For example, the following illustration shows an Activity Hierarchy Diagram in which all the subordinate activities are processes.



Assume that the fragment of the Entity Relationship Diagram in the following illustration has already been described in the business model.



Assume also that the following descriptions of the business are true:

- When a customer places an order, all the header information for the order is recorded, after which an Order Item is established for each different type of product ordered.
- When an order is corrected, or amended, a correction is typically made either to the header (for example, the customer changes the requested date of shipment) or to a single-line item (for example, the customer request a change to the quantity of product ordered) but not to both.

Is Record Order Header an Elementary Process?

Record Order Header is a process that captures the details of an order entity but none of its order items. Applying the guidelines in this section results in the following observations:

- An execution of Record Order Header by itself does not leave business data in a consistent state. An order cannot exist without order items. The relationship is fully mandatory.
- An execution of Record Order Header without an execution of Record Order Item is unlikely to produce a complete and meaningful result. More likely, the creation of an entire order and all its order items constitutes a result meaningful to the individual taking the order.
- Record Order Header has no elements that are individually executable. It is imaginable that a decomposition of Record Order Header would result in subordinates such as Identify Customer, Check Customer Credit, and Record Date Requested, but it is difficult to imagine the execution of those elements independently of one another.

Because it fails to satisfy the first two guidelines, Record Order Header is clearly not an elementary process; neither is Record Order Item. Since the first two guidelines are violated, the process is sub-elementary. An elementary process higher in the hierarchy is identified.

Now consider the process Customer Ordering. Applying the guidelines for identifying elementary processes yields the following results:

- Each execution of Customer Ordering in this example leaves business data in a consistent state.
- Each execution of Customer Ordering is likely to produce a newly taken order, an order cancellation notice, or an amended order, each of which are complete and meaningful to the user.
- The elements of Customer Ordering can be executed individually. For example, it is sensible to take an order without canceling one in the same execution.

Customer Ordering fails to comply with the last guideline, and so is above the level of being an elementary process. Also, Record Order Header is at too low a level. So Take Order is likely to be an elementary process.

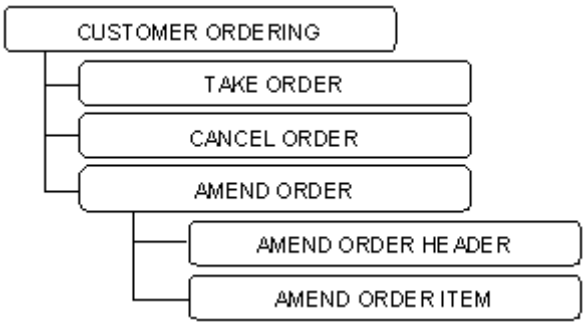
Applying the guidelines to Take Order reveals:

- Each execution of Take Order leaves the data in a consistent state, because both an order and its order items are accepted during the same execution.
- Each execution of Take Order produces a result (that is, an order) that is complete and meaningful to the user in itself.
- Neither element of Take Order (Record Order Header and Record Order Item) can be executed independently of one another, as explained previously.

Since Take Order complies with every guideline, it is an elementary process.

Consider the Amend Order process. It decomposes in a similar way to Take Order, which was found to be an elementary process. However, based on the underlying set of assumptions, Amend Order Header and Amend Order Item can be executed independently of one another. Given that Amend Order Header and Amend Order Item each produce an amended version of the order, they are elementary processes, each produces meaningful results and leaves the business in a consistent state, even though Record Order Header and Record Order Item were not. This example illustrates that even when decomposition is similar, the designation of elementary processes are different.

The results of this analysis are shown in the following illustration.



The corrected activity hierarchy diagram in the illustration shows that the decomposition of Customer Ordering terminates at the elementary processes Take Order, Cancel Order, Amend Order Header, and Amend Order Item.

Since process decomposition stops at the elementary process level, the sub-elementary processes, Record Order Header, and Record Order Item, have been removed.

Dependency Analysis

Dependency analysis is a powerful technique for refining the content and structure of the activity hierarchy by showing how the executions of processes are related. It indicates which processes must be executed to complete some parts of the activities of the business before execution of other activities, and when they must be performed.

A meaningful dependency analysis also identify:

- Providers of information who can facilitate the execution of processes
- Receivers of information that the processes produce
- Events that enable the execution of processes. See the chapter "Event Analysis" for more information.

This analysis can uncover opportunities for executing activities in parallel, shortening the time that is taken for the business to respond to demands placed on it.

Dependency analysis also helps to identify and validate entity states.

Dependencies between sibling processes are modeled. However, it is not possible, or necessary, to show all possible dependencies between elementary processes that belong to different parent activities.

You verify decomposition by verifying that all sibling processes are interdependent. If they are not, this strongly indicates that the decomposition still be improved. You have not discovered all the sibling processes, have incorrectly grouped processes that should not be siblings, or have erroneously identified an activity as a process.

In a few cases, siblings are not interdependent, but such cases must be regarded with suspicion. Always resolve or explain any anomalies to ensure that elementary processes are correctly identified. Only then can designers correctly implement systems to support the processes.

Dependencies between processes are identified and recorded by building Process Dependency Diagrams using CA Gen Activity Dependency Tool.

Selecting Processes for Dependency Analysis

A Dependency Diagram can be drawn for any process in the analysis model except for elementary processes because elementary processes do not have any children to be depicted on a Dependency Diagram.

For a complex activity hierarchy, dependency analysis can be performed level by level, in parallel with successive decomposition of the activities.

In the hierarchy that is shown in the illustration, Function A is divided into B and E. The Dependency Diagram of Function A is then drawn showing the interactions of B and E. This helps to identify the processes C and D, and F and G.

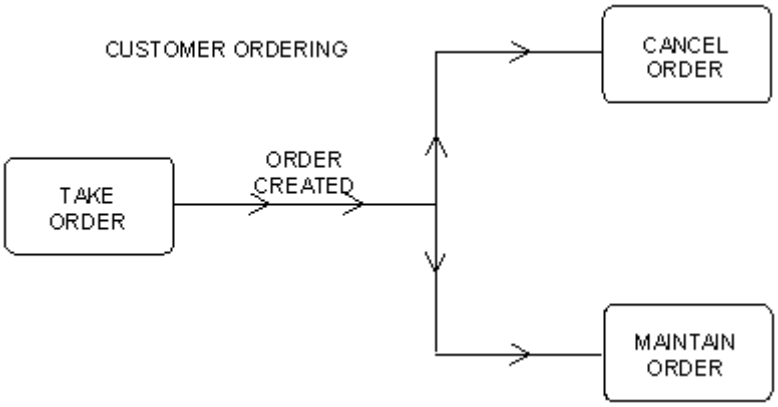
Next, a Dependency Diagram is drawn for B. This diagram shows C and D. Another diagram is drawn for E, showing F and G. Each of these last two diagrams takes into account the dependencies that are already identified when drawing the diagram for A. Drawing these diagrams suggest, in turn, that dependencies be added to B and E.

CA Gen Activity Dependency and Hierarchy Diagramming Tools together support this approach, because it is possible to add activities to the model using either diagram.

Consider the function Inventory Management. Replenish Stock is performed whenever the "stock is low" condition occurs. This condition can occur as a result of any of the processes Issue Stock, Destroy Stock, Check Stock Level, and can Establish Stock Level Policy. In a large hierarchy, it is unlikely that these processes would all be part of the same parent activity, so that they all appear together on one Dependency Diagram. The business performs a regular Check Stock Level process that recognizes "stock is low," which is a sibling of Replenish Stock and can therefore be shown as dependent on Check Stock Level. The precondition "stock is low" can be added to the definition of Replenish Stock. This condition can also be modeled as an event; an event consequence diagram can then show all dependencies that are associated with the event. See the chapter "Event Analysis" for more information.

Dependency analysis tends to be time-consuming. For tightly scoped projects that call for rapid development, you think about dependencies for all processes but decide to draw dependency diagrams only for parents of elementary processes. When the Check command is performed on the model, warning messages appear for activities that have no dependencies. These warnings are ignored.

The previous illustration of a corrected activity hierarchy diagram shows two Dependency Diagrams, one for Customer Ordering and one for Amend Order. The Customer Ordering diagram in the following illustration depicts the interdependency between the two elementary processes Take Order and Cancel Order, and Amend Order, their non-elementary sibling.



The Amend Order diagram (not shown here) depicts the interdependency between the elementary processes Amend Order Header and Amend Order Item.

Interpreting Dependencies

Dependencies do not imply that a set of steps must be executed to execute the parent process. Rather, they reflect a business state that allows execution of dependent processes.

In the previous illustration, the execution of Customer Ordering does not mean that an execution of Take Order must be followed by an execution of Cancel Order or must Amend Order. Rather, the diagram shows that Take Order can leave the business in a state that enables execution of Cancel Order or can Amend Order. In this situation, Take Order is considered prerequisite to Cancel Order and Amend Order, while Cancel Order and Amend Order depends Take Order.

Take Order has a post-condition that, when true, satisfies the preconditions for either Cancel Order or Amend Order. The dependency line can therefore be thought of as a matching of a post-condition of the prerequisite process to the precondition of the dependent process.

In Selecting Processes for Dependency Analysis, if an execution of Take Order completes successfully, resulting in the creation of an order entity and its attendant order item entities, it becomes possible for the order to be affected by Cancel Order or Amend Order. Thus, "order created" is the post-condition of Take Order that matches the precondition of Amend Order or Cancel Order.

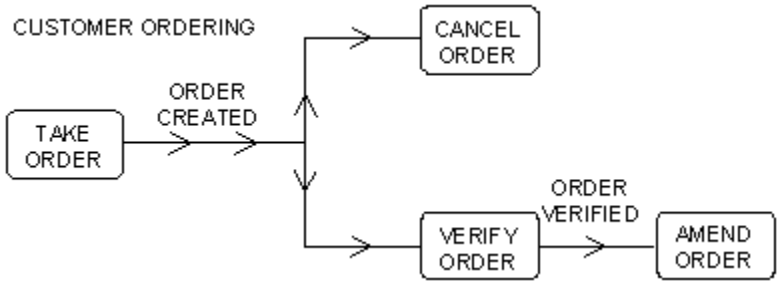
Find that many post-conditions of processes involve a change of state of one or more entities, and that preconditions specify that entities must be in a particular state to enable a process to execute. They later use these state conditions in entity life-cycle analysis, which is described in the chapter "Analyzing Interactions."

Defining Dependencies

For the dependencies, you can record:

- Name
- The dependency name is optional. If specified, it reflect the post and preconditions it matches.
- In Selecting Processes for Dependency Analysis, "order is created" is a suitable name for the dependency.
- Where entity states have been analyzed and identified, use entity state names. These names refer to entity subtypes in a life-cycle partitioning. See the chapter "Analyzing Data." In turn, the conditions suggest more state names.
- Description

- Specify the post and preconditions represented by a dependency in its description and the reason for the dependency. Avoid referencing the processes that are involved because they can be added or deleted from a dependency without actually affecting the post or precondition.
- In Selecting Processes for Dependency Analysis, this description would be too wordy: "An order is created by Take Order that after can be either canceled by Cancel Order or changed by Amend Order."
- It also happens to be a verbal restatement of the Dependency Diagram with the condition included. If you later discover extra elementary process that is named Verify Order that depends on Take Order and is a prerequisite to Amend Order, the post and preconditions for the dependency would remain the same: An order is created. See the following illustration.



- Had the names of the processes that are involved in the description of the dependency been included, it would require modification. Instead, the condition simply be stated as "an order is created."
- Type of dependency

Types of Dependencies

You can clarify the meanings of dependencies on a Dependency Diagram by depicting different types of dependency.

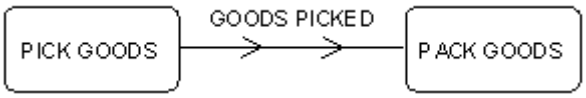
Three primary types of dependency can be distinguished on a Dependency Diagram:

- Sequential
- Parallel
- Mutually exclusive

Sequential Dependency

A sequential dependency involves two processes, one of which depends the other. When the prerequisite process executes, it fulfills its post-condition and satisfies a pre-condition that allow the dependent process to execute.

The following illustration shows a sequential dependency between two processes, Pick Goods and Pack Goods.

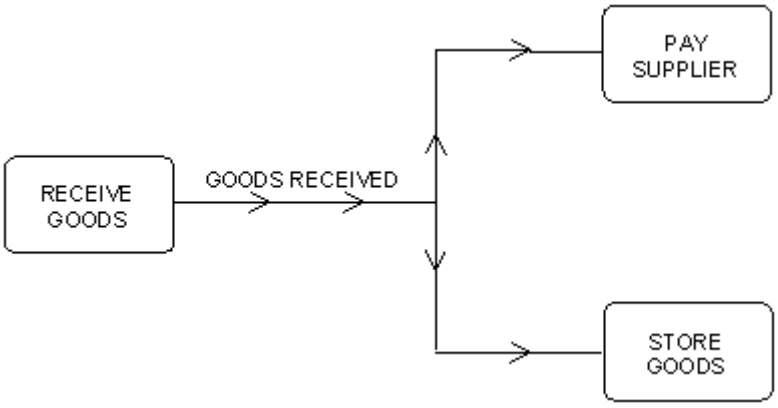


Pick Goods, if it completes successfully, satisfies the post-condition "goods have been picked" which is, in turn, a pre-condition of Pack Goods. In other words, goods cannot be packed until they have been picked from stock. The structure in the illustration reads as: "Pack Goods execute only if goods have been picked."

Parallel Dependency

A parallel dependency involves three or more processes: one is a prerequisite process, and the rest are dependent processes.

When the prerequisite process executes, it fulfills its post-condition and satisfies the pre-condition of each dependent process. Consider the following illustration.



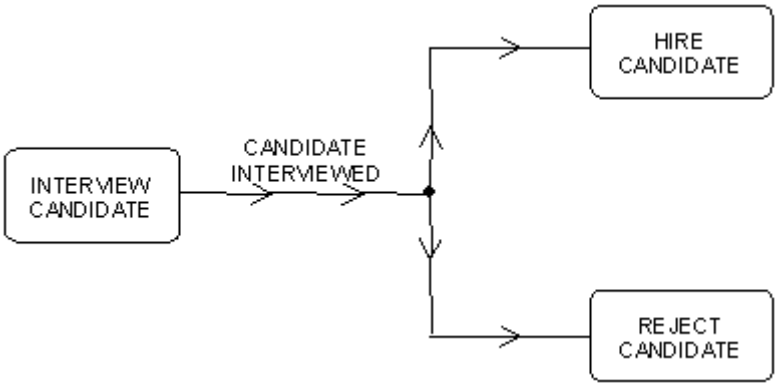
Receive Goods, if it executes successfully, satisfies the condition "goods received," which is a pre-condition of both of the dependent processes Pay Supplier and Store Goods. These two processes can execute independently and possibly simultaneously. The structure reads: "Pay Supplier and Store Goods each execute if goods are received."

Mutually Exclusive Dependency

A mutually exclusive dependency involves three or more processes: one is a prerequisite process, and the rest are dependent processes.

When the prerequisite process executes, it satisfies one of a number of alternative (mutually exclusive) post-conditions, each of which is a precondition of exactly one dependent process. This dependency is quite different from sequential and parallel dependencies because it is associated with multiple alternative conditions.

The structure in the following illustration reads: "Interview Candidate result in either the candidate accepted or the candidate is not accepted." "Hire Candidate execute if candidate is accepted." "Reject Candidate execute if candidate is not accepted."



In the diagram, the dependency Candidate Interviewed represents the matching of these conditions. The darkened circle where the dependency diverges indicates a mutually exclusive dependency, and thought of as testing the post condition to decide which dependent process should execute.

Interview Candidate, if it completes successfully, satisfies either of the conditions "candidate accepted" or "candidate not accepted," but not both.

- Hire Candidate has "candidate accepted" as a precondition.
- Reject Candidate has "candidate not accepted" as a precondition.

The post and preconditions of a mutually exclusive dependency cannot be described meaningfully without including the name of each process that is associated with a particular condition, but this violates the guideline for descriptions. See the Defining Dependencies section in this chapter for more information. In this case, clarity is more important than avoiding the mention of process names.

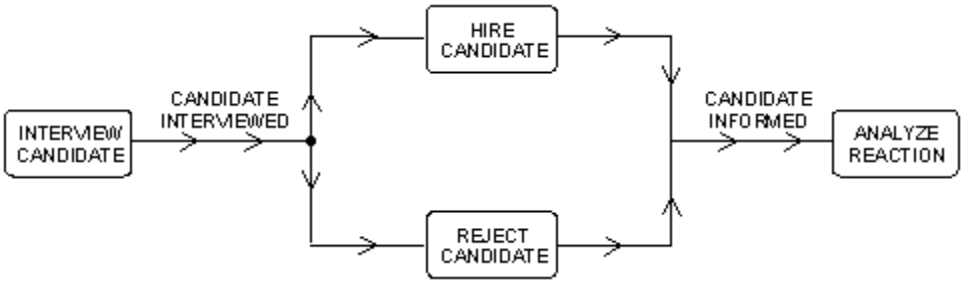
For the condition recorded in the dependency description...

Rather than: "Candidate is accepted or candidate is rejected," use: "Candidate is accepted, enabling Hire Candidate, or candidate is rejected, enabling Reject Candidate."

Mutually exclusive dependencies also differ from other dependencies in that they can be closed.

Closure of a mutually exclusive dependency takes place when two or more dependent processes are prerequisite to a common process.

Suppose the reaction of the candidate to the decision was analyzed, regardless of whether the candidate is accepted or rejected. This situation can be modeled by adding a new process, Analyze Reaction, which depends either Hire Candidate or Reject Candidate, as shown in the following illustration.



An execution of either Hire Candidate or Reject Candidate can satisfy a post-condition "candidate informed of decision" that is also a precondition of Analyze Reaction.

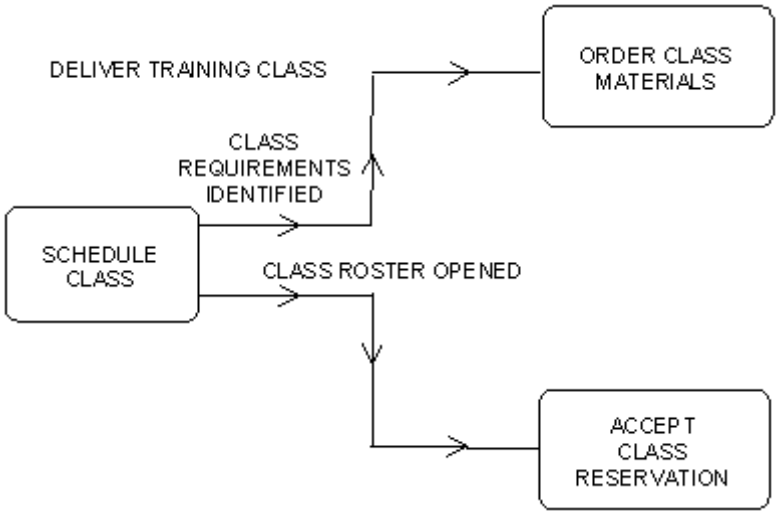
The new dependency closes the mutually exclusive dependency and is depicted by two lines converging and entering Analyze Reaction as a single line.

The closing of the dependency that is shown in the illustration reads: "Analyze Reaction execute if the candidate is informed of the decision, either following Hire Candidate or Reject Candidate."

Some Additional Dependency Concepts

These are less obvious situations that arise in dependency analysis:

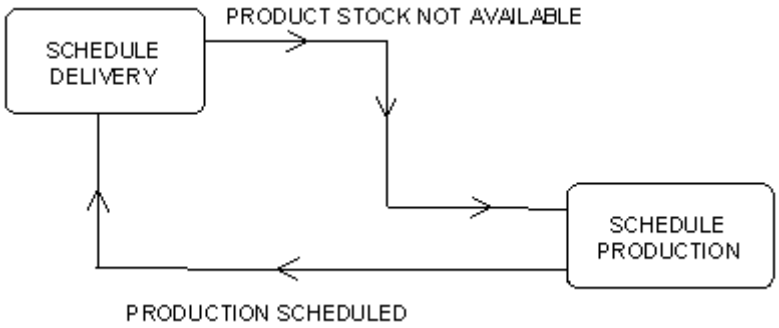
- A single process can be a prerequisite to several dependencies, which is based on different post-conditions.
- In this case, separate dependency lines are drawn. Consider the following illustration.



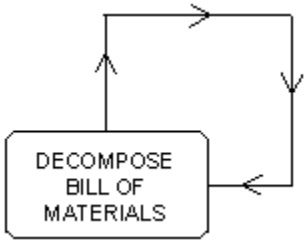
- Assume that Schedule Class in the illustration satisfy two post-conditions.
- Assume that Order Class Materials has the condition "class requirements are identified" as its precondition.
- Assume that Accept Class Reservation has the precondition "class roster opened." This condition is associated with a repetitive dependency, because Accept Class Reservation execute many times as a result of this condition.
- When multiple conditions are not mutually exclusive, multiple enabling dependencies appear in the diagram.
- If the dependent process in a dependency is a repetitive process, the dependency is named a repetitive dependency. This means that when the prerequisite process executes, its post-condition can satisfy the precondition of multiple executions of the dependent process. An example of this is in the following illustration.

- A single execution of Schedule Delivery satisfies the post-condition "product stock not available."

- Multiple executions of Schedule Production (one for each production period) can be executed based on the precondition of Schedule Production "product stock not available."
- There is mutually dependent processes, where each process depends the other process.
- The following illustration expands on the example in the previous illustration to show this situation.



- Assume that Schedule Delivery can result in a condition "product stock not available," which is a precondition of Schedule Production. If enough product stock is available when scheduling the order, it is possible to reserve product stock. If there is not, Schedule Production must be executed repeatedly until enough product is scheduled. When production is scheduled, it is possible to try to schedule the delivery again. In one situation, then, Schedule Production depends Schedule Delivery, while in another, Schedule Delivery depends Schedule Production.
- It is possible to specify a sequential dependency in which a process is dependent upon itself. Such a dependency is named a recursive dependency.
- This indicates that one execution of the process satisfies a post-condition that is a precondition of a subsequent execution of the process. This typically occurs when the process interacts with an entity type that has an involuted relationship, or whose entities are otherwise associated indirectly with other entities of the same entity type.
- Refer to the following illustration.

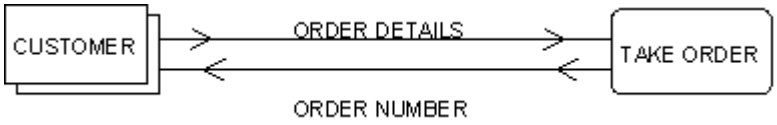


- Assume that Decompose Bill of Materials is a process that decomposes one level of a bill of material structure. Bill of material structures are discussed in the chapter "Building the Analysis Model." To decompose the entire bill of materials, Decompose Bill of Materials must execute one or more times for each level of t he structure.

Defining External Objects

Sometimes a process executes when it receives information from an external source. For example, Take Order executes when an order is received from a customer.

CA Gen allows you to define types of objects from which a process can receive information and to which a process send information. Such objects are called external objects and are depicted on the Process Dependency Diagram as two stacked red boxes. See customer in the following illustration.



The interaction of an external object with a process is shown as a directed line connecting the two. This line is not confused with a dependency; rather, it indicates an information flow.

In CA Gen Activity Dependency Diagrams, information flows are colored cyan (light blue) to distinguish them from dependencies (white).

The illustration depicts the fact that a customer sends the order details that are required by the Take Order process and receives order number information that is produced by the Take Order process.

You can use an external object to show virtually any source of information that is required by a process, or the destination of information that is produced by the process.

In most cases, external objects are considered to be outside the analysis model in which the process appears; hence, the designation "external."

Sometimes they imply business activities or systems external to the analysis model.

Sometimes they are given the name of an entity type that is represented in the associated information flow.

Occasionally a specific entity, such as a bank or government agency, is included as an external object.

Give each external object a short textual name, and optionally, a short description. There are no explicit rules naming external objects. Where the name of a business activity is known, use this name. It is not unusual for an external object to have the same name as an entity type. This does not mean that external objects can represent data stores, or that attributes in an associated information flow are restricted to belonging to one specific entity type.

CA Gen does not enforce a rigorous definition of external objects. When changing the name of an activity or entity type represented also an external object, also change the name of the external object.

Defining Information Flows

CA Gen can capture the following details for each information flow:

- Name
- Description
- Optionality
- Associated Views

Name

The name of an information flow, when specified, is a free-form description of the information that is presented to or received from a process by an external object, depending on the direction of flow.

For example, the flow from customer to Take Order in the previous illustration is named "Order Details" and the one returning from Take Order to customer is named "Order Number." The name does not describe the source or destination of the flow, as that is shown by the external object.

Description

You can record any additional information deemed useful in the description.

If an information flow is optional, you record the conditions under which it is required or forbidden.

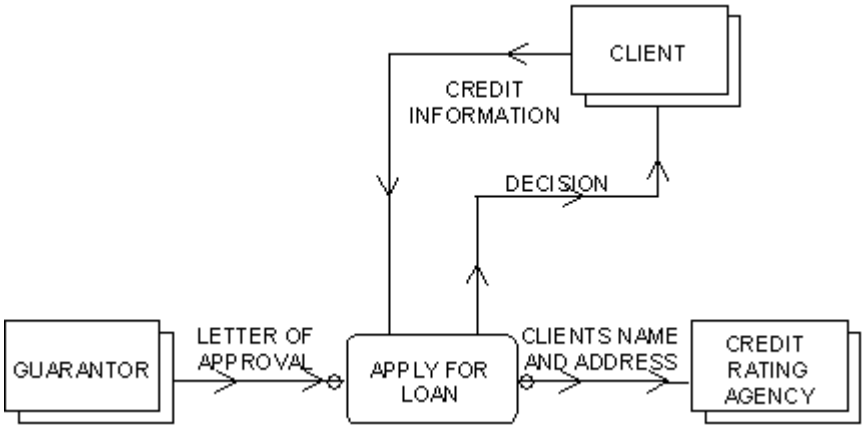
You should not record the content of the information that is passed between the external object and process. If this detail is important, you can record it as associated views.

Optionality

The optionality of an information flow indicates whether information passes between the external object and process at every execution of the process.

An open circle near the destination of the line identifies an information flow as optional in a dependency diagram.

The following illustration shows examples of four information flows associated with the process Apply For Loan.



The flows that run between client and Apply For Loan are mandatory. For each execution of Apply For Loan, "Credit Information" must be received from a client and a "Decision" must be produced for the client.

The other two flows are optional. A "Letter of Approval" is required from a Guarantor in some cases (the optionality condition is left to the imagination of the reader), but not in others. Sometimes a loan application is passed to a Credit Rating Agency.

The conditions under which Apply For Loan requires a Letter of Approval from the Guarantor and under which Apply For Loan passes Name and Address of client to a Credit Rating Agency is documented in the description of Apply for Loan. This information also be left as part of the description of each information view until the logic of Apply for Loan is analyzed in detail.

Process logic analysis is described in the chapter "Analyzing Interactions."

Associated Views

You can specify exactly which elements of the information that is required or produced by the process pass along the information flow.

Associated views for an information flow contain attributes of many different entity types, but these entity types must correspond with the information views and expected effects of the associated process.

See the chapter "Analyzing Interactions" for a detailed discussion of associated views.

Event Analysis

Event analysis is a useful technique for identifying activities that the business must perform to respond to events occurring in its environment, or to regulate the timing of its activities. It can be used both to drive and to confirm process decomposition and the development of the data model.

An event is something that occurs outside the scope of analysis and requires a response from within the scope of analysis.

Event Classification

Events can be classified into two groups:

- **External events**—An external event is an event that results from the execution of some process outside the scope of analysis, which is seen as originating outside the scope.
- **Temporal events**—A temporal event is the arrival of a specific time that enables the execution of one or more processes.

Although the existence of an external event arises from some external activity, the identity and execution of that activity are not visible to the roles and activities within the analysis scope. Only the external event can be recognized.

For example, the external event "Customer Places Order" is the result of some purchasing process that is performed by a customer. The nature of this process is of no interest in analysis and indeed change without affecting the results of analysis. The external event "Customer Places Order" originates outside the enterprise.

Other events can arise within the enterprise but are outside the scope of analysis. For example "Foreign Currency Available" originate from a Treasury function outside the scope of Purchasing, where the process of securing currency is not visible to Purchasing roles or activities.

When you discover an event that results from a process within the scope of analysis, define that event as a post-condition of the process. Conditions are described in the Interpreting Dependencies section in this chapter.

The response include:

- Decisions that are taken by organizational roles within the scope concerning what response to make. For an account of role analysis, see the chapter "Analyzing Interactions."
- Performance of activities within the analysis scope.

When users are describing the response to events by business, distinguish between planned, repeatable responses and on demand reactions, which vary according to the circumstances of each event occurrence.

For example, when the event "Order Delivery is due" occurs, the business always performs the process Deliver Order. However, when "Customer complains that an order has not been delivered," the business does whatever is necessary to satisfy the customer.

Planned the responses are candidates for information system support. Informal, on demand responses are more difficult to automate but examined for opportunities to improve formality. If a customer complaint, perhaps the human decisions cannot be formalized, but the selected response include some formal, repeatable activity. This response executes the process Record Complaint, or even Deliver Order if the business has omitted to do this, or the delivery is found to be lost and must be repeated.

Benefits of Event Analysis

If properly executed, event analysis helps to ensure:

- Clearer definition of project scope by identifying which business situations (represented as events) are addressed by the system and which will not.
- Improved the communication between analysts and users by relating processes in a functional decomposition to familiar business situations.
- More accurate and comprehensive identification of activities by providing an alternative activity discovery process to that used during functional decomposition.
- More user-oriented business systems.
- Identification of activities that are candidates for reuse by the appearance of the activity in multiple event responses.

Defining Events

Events and their responses are incorporated in the analysis model using event arrows or external objects that are connected to the activities that initiate the event response.

Each event must have a name and its full definition include text that provides:

- **Event description**—This is for general descriptive purposes.
- **Response description**—Where there is more than one set of processes that respond to the same event, this description is part of each dependency description that joins the event to the process that initiates a business response.
- **Frequency of occurrence**—This information is used in system design decisions.
- **Required response time**—This information is used in business process re-engineering and system design decisions.

Defining External Events

The source of an external event is represented and defined as "an External Object (as described in the chapter "Dependency Analysis") and a flow of the information that tells the responding business activity what has occurred and what is done."

In the information flow between a process and an external object, for example, CUSTOMER is the source of the event Order Arrives. The information flow Order Details represents the information available to the responding process Take Order.

When using event analysis to confirm existing activity hierarchy and dependencies, you can check the list of external objects and the dependency diagrams to see if the event response is already included in the analysis model.

Defining Temporal Events

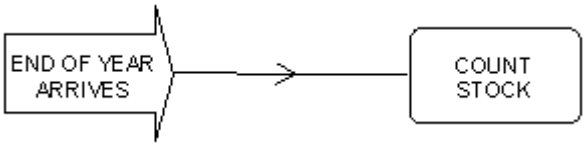
A temporal event appears as a large red arrow on CA Gen Activity Dependency Diagram.

The name of a temporal event is usually a sentence with a subject and verb, for example, End of Year arrives or Order Delivery (is) due.

If something must happen for a process to execute, it can be said that the process "is enabled by the event," or that the event is a "precondition for the process." The event is therefore connected to a single dependent process by a dependency line.

A temporal event can occur arbitrarily; for example, the arrival of a Delivery Date is an event that enables the Deliver Order process. It can also depict some cyclical occurrence, for example, End of Month.

In the following illustration, the process Count Stock is enabled at the End of Year.



Since an event is a precondition for a process, it follows that the condition can arise as the post-condition of another process.

If the other process is a sibling of the processes on the dependency diagram, an event is not needed; a dependency line between the processes suffice. You can use the event arrow to depict dependencies on other activities that appear elsewhere in the activity hierarchy, or even outside the scope of the analysis model.

Note: CA Gen does not automatically control changes of name for events that are used in this way.

Analyzing Events

Event analysis involves these steps:

Step 1 Identify Events

Events are often encountered during information gathering, and they can be found in the operational documentation for current systems. The external events can be identified by examining the interactions from external objects to activities, which is identified during dependency analysis.

You can define a list of events:

- Event
- Corporation opens new store
- Corporation closes store
- Distributor sends a catalog to corporation
- Corporation selects films of interest
- Store decides to purchase tapes from a distributor
- Distributor ships tapes to store
- Store manager changes daily rate for store catalog item
- Customer applies for club membership
- Customer applies for store membership
- Customer reserves tape

- Customer asks to make rental
- Customer returns tape
- Reservations department checks item availability
- It is time to notify overdue customer

You can begin with a preliminary list of the principle events to which the business must respond. Progressively extend and refine this list as you discover further events, compare events with the processes that are involved in responding to events, and find events that are associated with state transitions in the life-cycles of entities. Entity type life-cycle analysis is described in the chapter "Analyzing Interactions."

Step 2 Define Event Response

You can describe the actions that are involved in the response using one of the following techniques:

- Use the names of processes where these are already identified
- Use an ordered list of actions such as the one shown in the following table.

Event	Response
Distributor ships tape to store	Set purchase order line status to "received" Create store catalog item Create store tape copy Set tape aside for reservation Notify store member that reservation is available
Customer reserves tape	Accept reservation Set tape aside for reservation Notify store member that reservation is available
Customer cancels reservation	Delete reservation Set tape aside for reservation Notify store member that reservation is available
It is time to cancel expired reservation	Delete reservation Set tape aside for reservation Notify store member that reservation is available
It is time to notify overdue customer	Set status of rental time to "overdue" Assess additional charge for overdue rental item Notify member with overdue rental items
Customer responds to "overdue" notification	Record response of store member to overdue notification

Event	Response
Customer makes payment on outstanding charges	Collect payment Close rental
Lost tape is returned	Mark store tape copy "available" Refund additional charges collected

A number of elementary processes participate in the response to an event. Processes are added to an event list as shown in the following table.

Event	Response
Distributor ships tape to store	Receive Ordered Tape Set purchase order line status to "received" Create store catalog item Create store tape copy Hold Tape for Reservation Set tape aside for reservation Notify store member that reservation is available
Customer reserves tape	Create Reservation Hold tape for Reservation
Customer cancels reservation	Delete Reservation Hold Tape for Reservation
It is time to cancel expired reservation	Delete Reservation Hold Tape for Reservation
It is time to notify overdue customer	Chase Overdue Rental Set status of rental time to "overdue" Assess additional charge for overdue rental item Notify member with overdue rental items

New processes that are identified are added to the Activity Hierarchy Diagram. Previously identified activities are grouped as candidate actions of a process; these actions are added to the process description. Common actions are identified. They represent processes that respond to more than one event, or possibly common business logic that uses more than one elementary process. See also the discussion of process logic analysis in the chapter "Analyzing Interactions."

Step 3 Specify Dependencies in Event Response

You draw a dependency diagram to show how processes together participate in the response to a single event.

If all the processes are siblings, the event can be shown on a single diagram for the parent activity.

However, it is possible that the responding processes are subordinate to more than parent activity. Several dependency diagrams are required to show separate representations of the event.

If several activities are found that respond to the same event, it is worth asking whether these activities are separate processes, or whether they overlap. This can lead to simplifying the set of processes.

To help understand the complete response to an event, a separate dependency diagram for which the event is the subject is needed. Such a diagram is known as an event consequence diagram and is shown in the following illustration.

Where these processes are coordinated, a single elementary process is enabled by the occurrence of the event and then initiate the response of business.

Step 4 Specify Conditions in Event Response

So that the response of each process to the event can be recorded as part of the process definition, you need to define the circumstances in which each process can be executed. These circumstances are defined as preconditions (described in Dependency Analysis) and can be annotated on an event consequence diagram and defined as part of the dependency. This ensures that the event is taken into account when the process logic is specified. See the chapter "Analyzing Interactions" for a description of how each event contributes to the process definition.

Summary of Activity Modeling Rules

CA Gen enforces some rules automatically. The rules must apply when the analysis model is complete and consistent for those activities that are to be defined as part of a business system. There are other activities in a model that are not yet fully defined or part of the scope of a system.

Rules for Activities

- Each activity must be uniquely named.
- A function appear only once in the activity hierarchy.
- Siblings must be either all functions or all processes.
- An elementary process cannot be decomposed.
- Each activity that is not an elementary process must decompose into at least two subordinate activities.
- The subordinates of an activity must completely describe the activity. For example, if process A decomposes into processes W, X, Y, and Z, then the following statement must be true:

  A = W + X + Y + Z
- Each leaf of the completed hierarchy must be either a manual process or an elementary process.
- Each elementary process must be included in the response to at least one event.

Rules for Processes

Violations of the following guidelines indicate some problem in process decomposition or dependency analysis:

- A process appear more than once in the activity hierarchy, but this is rare.
- An activity decompose into between three and seven subordinate activities.
- Siblings have some interdependence.
- A process must produce some change to the business, and not be merely part of some procedure. Monitoring entities without changing and reporting them are not processes.

Rules for Dependencies

Each dependency must be described by a condition or set of conditions that represent post-conditions of the prerequisite process, and preconditions of the dependent processes.

Rules for Events and External Objects

- Each event is associated with at least one elementary process.
- Each external object must be associated with at least one activity using an information flow.
- An external object is associated with at least one elementary process.

Results of Activity Analysis

Top-down activity analysis produces the following results:

- Activity Hierarchy Diagram depicting the decomposition of processes to the elementary process level
- Activity Dependency Diagrams, at least for each parent with an elementary process as a child
- Event Lists associating events with the activities that respond to them
- Supporting documentation, which is captured as details of the processes, dependencies, external objects, and events appearing on the Activity Hierarchy Diagram and Activity Dependency Diagrams

Chapter 6: Analyzing Interactions

Interaction analysis involves exploring how entity types are affected by elementary processes, and how elementary processes use entity types and their attributes and relationships.

The business model that is developed during analysis integrates three equally important aspects of business requirements:

- Data aspect of the model, which describes things of interest to the business and the relationships between them. See the chapter "Analyzing Data."
- Activity aspect of the model, which records things the business does, or should do. See the chapter "Analyzing Activities."
- Interaction aspect of the model, which details how things the business does (activities) affect things of interest to the business (data).

This aspect serves to confirm, and if necessary modify, the analysis model. It also provides a detailed basis for system design.

Interaction analysis exploits and refines the expected effects of processes on entity types. To represent this use in more detail, two distinct types of analysis are performed:

- Entity type life-cycle analysis
- Entity type life-cycle analysis approaches interactions from the point of view of an entity type: "Which processes use entities of this type?"
- Process logic analysis

Process logic analysis approaches interactions from the point of view of a process: "How are entities that are affected by this particular process?"

Interactions between processes and entity types can be used to group them together in clusters. Clustering is used to produce or confirm groupings of entity types into subject areas, and processes into functions, or scoped into business systems, each of which become the subject of a system development project.

Two further techniques can be used to analyze interactions between activities, data, and elements of the organization, where these interactions are critical to the development of systems:

- Role analysis

You can use role analysis when a system must be designed to fully support specific user activities. This is often the case where client/server applications are to be provided.

- Distribution analysis

You can use distribution analysis where there is a potential need for distributed solutions; therefore, identify where processes and entities are used to help choose the structure of these solutions.

During interaction analysis, you consider only those elementary processes that have been selected as part of a business system for immediate development. To be confident that the business requirements are understood and can be supported by the planned system, you apply these detailed analysis techniques to the most complex elementary processes. As a guideline, consider any process that interacts with more than four entity types.

Analyzing Entity Type Life-Cycles

The subtypes appearing in the life-cycle partitioning represent discrete states of entities at different times during their "lives." Life-cycle analysis examines discrete states of entities at different times during their "lives" to show how process executions can cause an entity to change from one state to another.

Entity type life-cycle analysis helps to ensure that each process:

- Creates an entity state
- Performs possible transition between states
- Deals only with entities in the appropriate states

Consider all entity types that are involved in the development of a business system to see whether life-cycle analysis is performed. During data analysis, not all entity types were given a life-cycle partitioning, either because the lives of their entities are simple or because there are too few differences between states to make it worth defining subtypes.

During activity analysis, considering entity life-cycles was one means of discovering a set of processes. During interaction analysis, life-cycle analysis is not restricted to those entity types where a life-cycle partitioning has already been defined.

Indeed, a more detailed understanding that is developed through interaction analysis cause you to revisit data analysis and define new life-cycle partitioning. Find that an entity has more than one life-cycle because subtypes in its entity type have their own life-cycles, that is, life-cycles that are not shared with other entity types in the same partitioning.

The life-cycle states of an entity is derived from the life-cycles of other related entities. For example, an ORDER is a SHIPPED ORDER if all its ORDER ITEMs are shipped. In this case, it is a matter of business judgment whether the life-cycle of order item is defined formally perhaps where a partial order or batched shipping of orders occur, or only that of order where typically all items are shipped together.

Some processes that affect the states of entities of the type being analyzed lie outside the scope of the project. This require some coordination between development projects to agree a consistent set of processes and entity type life-cycles.

During the early stages of analysis, you often find entity states that overlap and that are not clearly defined or agreed upon by business staff. For complex life-cycles, therefore, seek clear definitions that are practical and acceptable to all groups of system users. This lead to simplifying and improving business processes.

Using CA Gen, entity type life-cycles can be represented in two forms:

- Entity Life-cycle Diagram
- User-defined matrix that is named the Entity State Change Matrix

The chapter "Analyzing Data" described a special partitioning of an entity type, the life-cycle partitioning. To draw a life-cycle diagram using CA Gen, possible states of entities are defined in a life-cycle partitioning of the entity type.

Entity Life and Entity Type Life-Cycle

The distinction between types and occurrences is especially important during life-cycle analysis.

Each life-cycle of entity type is a description of the possible "lives" that are led by entities of that type. However, each entity occurrence leads a life that go through only some of the states that are compatible with the life-cycle of its type.

An entity life is a description of what happens to an entity from the time it becomes of interest to an enterprise to the time it ceases to be of interest to that enterprise.

An entity is considered not to exist until the business becomes interested in it. For example, the person Fred Smith, though 45 years old, is created as a new customer entity because the business had no reason to know about him during the first 44 years and some months of his life. Though Fred Smith live much longer, when the business is no longer interested in him as an entity, he ceases to exist for business purposes.

An entity type life-cycle is a description of what can happen during the lives of entities of one type.

Entity States

The entity type life-cycle consists of a series of entity states, that have been defined as subtypes in the life-cycle partitioning.

An entity can exist in one of four states:

- Creation or starting state
  - The creation state is the initial state of an entity after the business becomes interested in it.
  - An entity type have multiple possible creation states in its life-cycle. This is the point at which the enterprise first records information about entities of importance to the business as it becomes aware of them.
  - Each entity type must have at least one creation state in its life-cycle, but many have more than one.
- Termination or ending state
  - The enterprise lose interest in information about an entity. It can then choose to stop keeping that information readily available, or available at all, and set the information aside in some way. The termination state is this final state in the life of an entity.
  - Each entity type have at least one termination state in its life-cycle, but have more than one.
  - An entity in a termination state, by definition, cannot be changed to another state, although it can be referenced (read).
  - Often, the entity still be of interest to the business when it is in a termination state, in which case the termination state is said to be indefinite, and the entity is not deleted. For example, retired employees remain of interest until some fixed period after death.
  - If the entity is truly no longer of interest to the business, it is deleted. In this case, the termination state is said to be definite.
- Null state
  - Entities exist before the enterprise have information about them available. This unavailability of information is known as a null state.
  - It is convenient to visualize the creation of an entity as involving a transition from a null state to a creation state. For example, the Add Customer process change the state of a customer entity from the null state to a creation state.

  - The null state also be a definite termination state that results from a deletion. The null state never explicitly appears as an entity state subtype in a life-cycle partitioning of an entity type.

- One or more intermediate states that are of interest to the business
  - Any state that is not a creation, termination, or null is named an intermediate state.
  - Intermediate states are represented only if they are of interest to the business, and affect the way that executions of business activities interact with entities. The intermediate states come about because the enterprise adds to, or changes, information that is recorded about an entity or its relationships with other entities. At some point, the differences from the previously recorded information become so significant that they affect the way that business processes with the entity, or even require that different processes may or may not deal with the entity. At these points, the business recognizes that an entity must change its state.
- Until an entity reaches a termination state, it move from state to state without restriction. An entity return to a previous state or skip a state, as long as the life-cycle of its entity type supports such behavior. So, although an entity pass from its creation state to its termination state, it need not pass through all the intermediate states sequentially. Some entities take a complex and repetitive path.

The lives of different entities of the same type can vary greatly from one another, based on the complexity of their entity type life-cycle. While every entity type has a life-cycle, many are so simple that they do not require explicit definition.

The simplest entity type life-cycle has only one explicit state indicating its existence. When an entity of this type is created, it moves from the null state to the creation state Exists. When it is deleted, it returns to the null state as its termination state. A life-cycle partitioning never explicitly partitions entity types with such simple life-cycles.

The following illustration shows an entity type with a simple life-cycle.

A customer is either active, suspended, or does not exist. Its creation state is simply Active, and its termination state is Suspended.

Customer does not need a life-cycle partitioning; nor is it necessary to develop an Entity Life-cycle Diagram for such a trivial case.

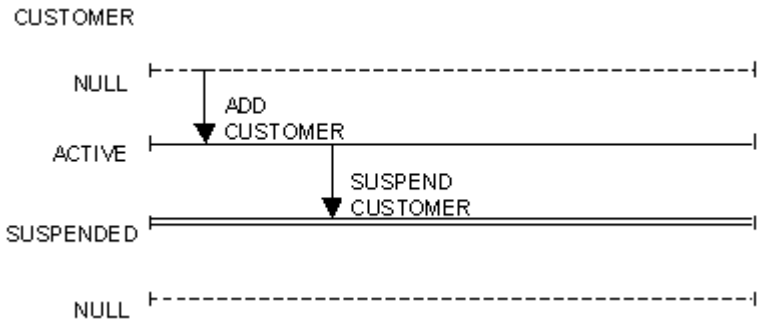
For a given life-cycle partitioning, every entity is in only one state at a given point in time.

Every entity type has at least one creation and one termination state. Since the termination state is null, it is not specified explicitly.

Entity Life-Cycle Diagram

The purpose of an Entity Life-Cycle Diagram, sometimes named the Entity State Transition Diagram, is to depict entity states and the possible state changes for entities of a single type.

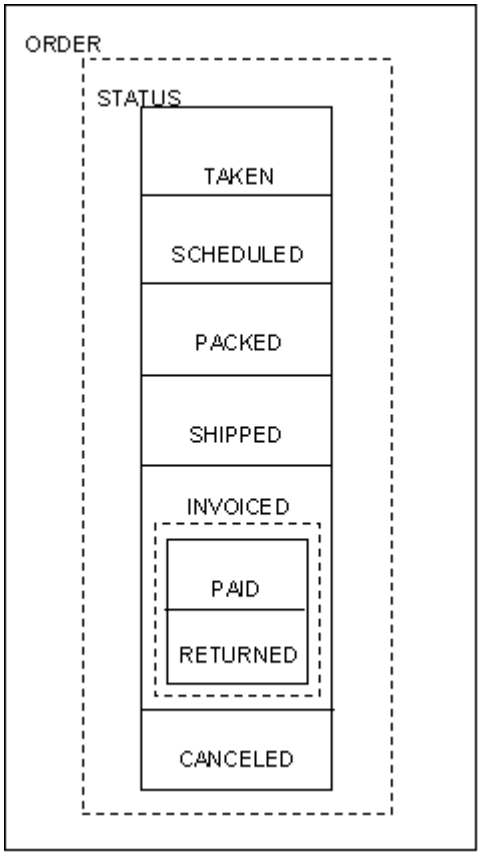
When using CA Gen, you have defined a life-cycle partitioning that includes entity subtypes for each of the discrete states in the life-cycle. CA Gen uses the "fence post" style of diagram as shown in the following illustration.



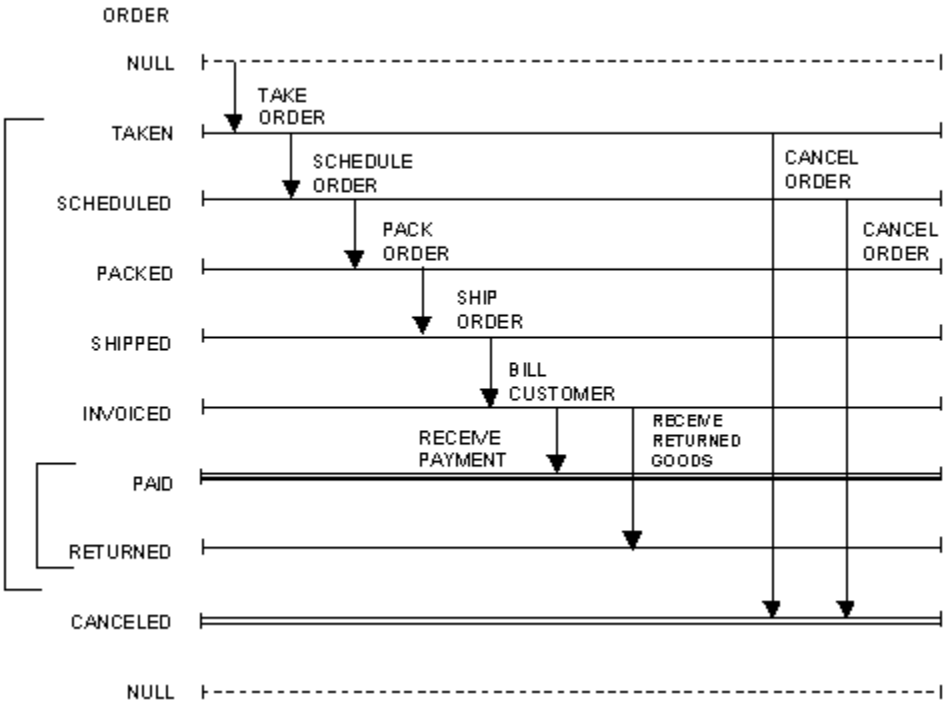
This diagram uses the following symbols:

- A horizontal bar represents each state.
- Entity states are grouped in a partitioning with vertical brackets.
- A vertical arrow represents an elementary process that can cause an entity in one state to move to a destination state.
- Null states are depicted as broken bars.
- A double horizontal bar distinguishes the termination state.

The Entity Relationship Diagram fragment in the following illustration presents a complex life-cycle partitioning of the entity type Order with two life-cycle partitionings and nine discrete entity states.



The entity life-cycle depicts the life-cycle by showing the possible movement of an entity between states. Consider the following illustration of the entity life-cycle for the entity type Order.



Each state in this illustration corresponds to a subtype in the life-cycle partitioning that is depicted in the previous illustration of the life-cycle partitioning of the entity type Order.

Vertical brackets group together the states that are common to a life-cycle partitioning. All the names appearing on the vertical arrows are those of elementary processes in the Activity Hierarchy Diagram.

Notes:

- There is one creation state: Taken.
- The elementary process Take Order moves an order from the null state to the Taken state. Any horizontal bar that is the target of an arrow leading from the null state is a creation state.
- There are four possible termination states:
  - Null state
  - Paid
  - Canceled
  - Returned

- The null state at the bottom of the diagram is always a termination state because this implies deletion of the entity (the ultimate in termination).
- The other three are easily recognizable as termination states because no vertical arrows come from them. Hence, no changes of entity state are possible once any of those states is entered.
- The horizontal bars from which arrows emerge constrain the processes that are represented by those arrows.
- For example, Cancel Order only be executed against orders in either the Taken or Scheduled state. An order that is Packed may not be operated upon by Cancel Order.
- There are two vertical brackets representing the Status partitioning, and a smaller group of the two possible states of Invoiced Orders, Paid or Returned.
- The Entity Life-cycle Diagram shows processes only from the viewpoint of the entity type under consideration.
- For example, the process Bill Customer is likely to create an entity of type invoice, and change the state of an entity of type order from Shipped to Invoiced. Since this diagram relates only to order, invoice is not shown.
- The null state is omitted from the life-cycle partitioning on the Entity Relationship Diagram, but it does appear on the Entity Life-Cycle Diagram.

Entity State Change Matrix

The same level of information that is shown on the previous illustration of the entity life-cycle for the entity type ORDER can appear on a manually prepared Entity State Change Matrix too.

You prepare an Entity State Change Matrix for entity types with complex life-cycles. This helps to show which processes can include changes of state in their post-conditions, and which processes can reference entities in what states.

The matrix format also provides a good check on the completeness of process support for the life-cycle.

An example of this matrix, corresponding to the illustration of the entity life-cycle for the entity Type ORDER, appears in the following illustration.

Cell Values: = Not referenced C = Create D = Delete U = Updates R = Read Only	Entity State	Taken	Scheduled	Packed	Shipped	Invoiced	Paid	Canceled	Returned
Elementary Process									
Take Order		C							
Schedule Order		D	C						
Modify Order		U	U						
Pack Order				C					
Ship Order				D	C				
Bill Customer					D	C			
Receive Payment						D	C		
Receive Returned Goods						D			C
Cancel Order		D	D					C	

The horizontal axis of this matrix shows all possible entity states for the entity type.

The vertical axis contains all the elementary processes that reference the entity type.

Cells indicate whether the elementary process causes a change in state (Delete and Create), Updates an entity in a state without changing the state, or Reads an entity in a state.

Any column with no cell entries represents a termination state.

Any row with no cell entries represents an error—you either omitted cell entries or included processes that did not reference the entity type.

Some processes act on an entity type without changing its state. In this example, modify order changes the contents of a taken or scheduled order without changing its state. Such processes are not shown on the entity life-cycle diagram.

The matrix is easy to check whether possible transitions have been omitted. For example, can Receive Returned Goods execute for Shipped orders and Paid orders?

Other complex situations are more easily shown on a diagram. For example, where several processes cause transitions to the same state, and where different executions of a process lead to one destination state from several source states.

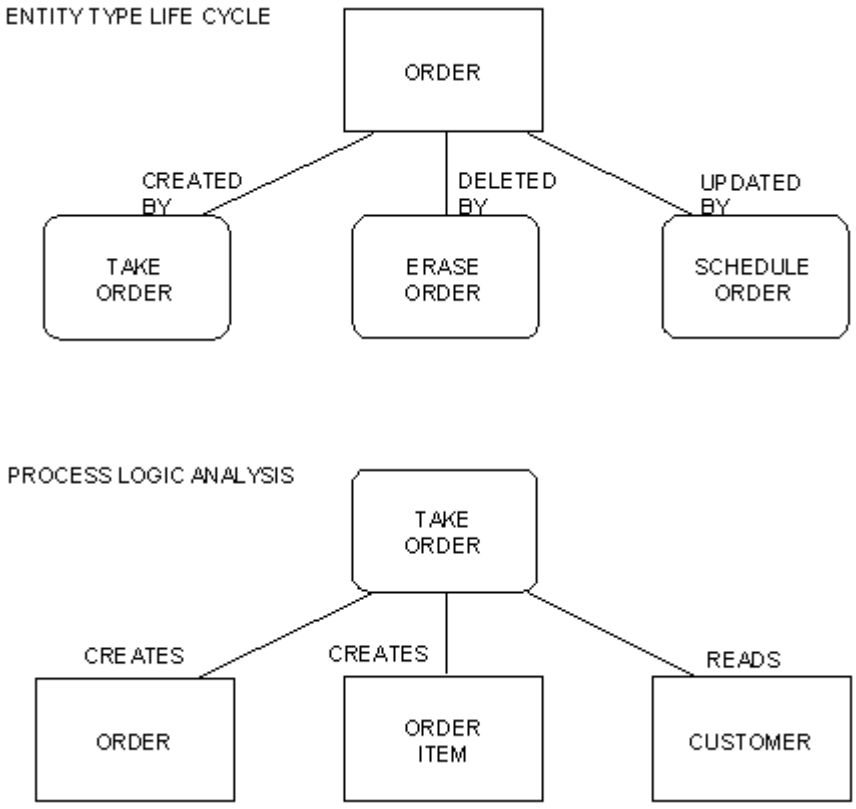
If a process can cause a number of transitions, the process definition explain the conditions that can cause each transition to occur.

Process Logic Analysis

During entity type life-cycle analysis, you focus on one entity type at a time, describing the effects of various processes on entities of that type.

During process logic analysis, you change their frame of reference to elementary processes. You focus on one elementary process at a time, and gradually evolve a deeper understanding by describing its effects on various entity types and detailing the attributes that are used.

The following illustration shows these two perspectives on the interaction between data and activities.



The illustration does not show all the processes that affect order, nor does it show all the entity types that are affected by Take Order, hence the dashed lines. Still, the illustration shows that during entity type life-cycle analysis, the process Take Order is only one of the processes that affect the life-cycle of the entity type order.

During process logic analysis, the entity type order is only one of the entity types with which the process Take Order must deal.

You can use a comparison of these viewpoints to confirm the correctness of the business model, because both viewpoints must be compatible. The use of different perspectives on the interaction model improves the understanding of the business model by both the analyst and the user. The main purpose is to provide a starting point for procedure design.

CA Gen can be used to record the expected effects of processes on entity types, as described in the chapter "Analyzing Activities." Here you can exploit and refine that detail.

CA Gen currently supports one viewpoint for process logic analysis, and one tool for its accomplishment, the Process Action Diagram. This diagram allows the analyst to specify a detailed interaction analysis from a single perspective.

The Process Action Diagram has two main elements:

- Information Views

  Information views are definitions of the entities that are seen by a process. These views can be defined by detailing a process, in the Activity Hierarchy, or in a Dependency Diagram. They can also be entered into the Process Action Diagram.

- Actions

  Statements describing actions, or formal rigorous statements of the actions that are performed by a process, can be:

  - Entered under the guidance of the Action Diagram Tool
  - Expanded from the expected effects of a process when the Action Diagram is first entered
  - Generated as part of a complete process using the process synthesis command

Analyzing information views and developing a full Process Action Diagram requires a rigor that is more appropriate to the design of systems procedures than to analysis, and it can consume a great amount of time. It is useful to develop action diagrams to define and agree complex business logic, verify the ability of data to support it, and associate it with a specific entity type. This allows action diagrams that support operations on an entity type to be displayed using the Data Model Browser tool.

Decide whether to develop action diagrams and to what level of detail. You also decide to define information views for complex processes only.

Important! Perform process logic analysis only on elementary processes, not on their parent processes. In the remainder of this chapter, the term process denotes an elementary process only, unless the text makes an explicit distinction.

Defining Information Views

An information view is a collection of entity type attributes of interest to the process in some context or other. An information view is the means by which a process sees information about entities.

It is important to understand the subtle difference between an entity and information about an entity.

The chapter "Analyzing Data" explained that an entity represents a real thing. For example, an entity of the type customer is a real person named Fred Smith.

Information about an entity is merely some collection of data that is associated with the entity and of interest to the business. For example, Name, Address, and Credit Rating of Fred Smith are probably of interest to the business, while other characteristics may not be, such as IQ, Height, and Favorite Food. So, the definition of an entity type that is developed during data analysis describes the characteristics of entities of that type about which the business want to store information.

This distinction is important because you specify the details of an elementary process in terms of information about entities, not entities themselves. For example, in process logic and Process Action Diagram terminology, an entity is said to be "created" once the business learns of it. Obviously, Fred Smith is not created once the business learns of his existence. The term create refers to the recording of information about Fred Smith, who is an entity of the type customer.

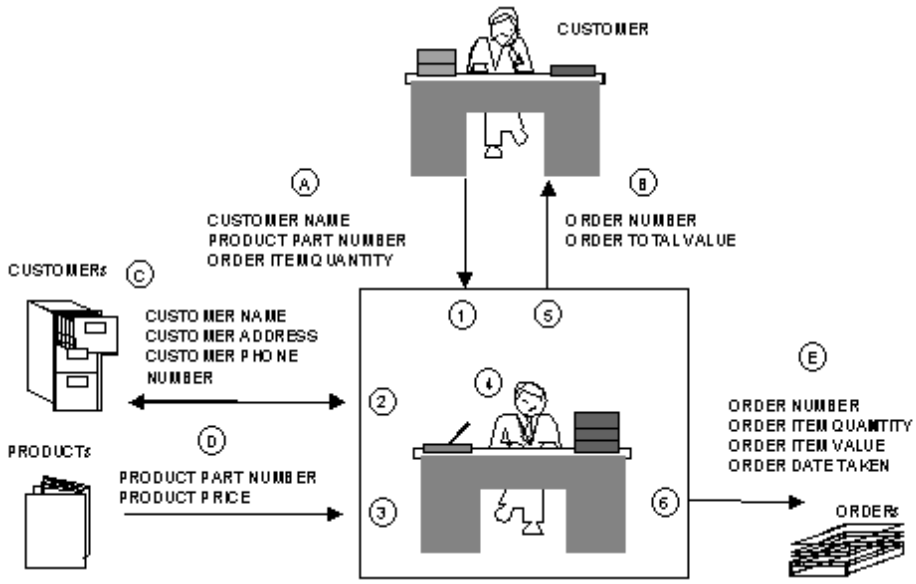
Note: In the following discussion, the term entity is used to represent information about entities, and to denote the entities themselves, with distinctions made where necessary.

Types of Information Views

A view is a window through which a process can communicate information about entities. There are four types of information views:

- Import
  - An import view is view through which a process receive information when it begins execution, and that contains the minimum information that is needed to initiate a process.
  - In the analysis model, imports arrive from another process in the model or from outside the scope of the project.
- Export
  - An export view is a view through which a process provide information when it ends its execution.
  - The view show the values of data at the end of execution, or, in addition, it explain the successful or exceptional result of the execution.
  - In the analysis model, exports either pass to another process in the model (manual or automated) or are used outside the scope of the project.
- Entity Action
  - An entity action view is a view through which a process inspect or manipulate stored information about entities during its execution.
- Local
  - A local view is a view in which a process temporarily save information during its execution.

For example, imagine the manual Take Order process that is depicted in the following illustration.



The large box around the clerk in the illustration represents the boundaries of the elementary Take Order process.

Refer to the following table for an explanation of the key activities in this process.

Key	Activity
1	Customers contact the store and provide the clerk with their Name, a list of products, and the Quantity of each product they wish to order.
2	The clerk walks to a filing cabinet containing records for all customers with which the store deals, and checks to make certain that the customer is on file. If the customer is not on file, the order cannot be taken until certain information about the customer is recorded in the file.
3	For each product requested, the clerk looks up its price in the product book and records it on the appropriate order item. The clerk also calculates the value of the order item by multiplying the product Price by the Quantity Ordered.
4	The clerk sums the Value of each order item to determine the Total Value of the order.
5	The clerk then contacts the customer and tells them the order Number and the Total Value of the order.
6	The clerk types up an order for all the items the customer wishes to purchase, and places it in a basket containing open orders. Assume that the order Number is a pre-printed unique number on the order form.

Customer is an external object. It is a real entity that both provide information to and receives information from Take Order. However, it is "external" to the stored information about the business.

Take Order deals with four entity types:

- CUSTOMER
- ORDER
- PRODUCT
- ORDER ITEM

The Take Order process has two views of customer:

- Import view (A)
- Entity action view (C)

Both are views of the same customer entity; they are simply used for different purposes.

The Take Order process relates to entities in three ways:

- It receives information about a customer entity and several product entities from a customer.
  - Since these views can be thought of as being "imported" by Take Order, they are named import views.
  - The circled letter A in the illustration indicates the import views of Take Order.
- It produces information about an order that is returned to the customer entity at its conclusion.
  - Since this view can be thought of as being "exported" by Take Order, it is named an export view.
  - The circled letter B in the illustration indicates the export view of the Take Order process.
- It manipulates stored information about entities.
  - With customer (circled letter C) and product (circled letter D), information is referenced.
  - With order (circled letter E) and order item, information is created or newly stored.
  - A view of stored information about an entity that manipulates a process is named an entity action view.

Another type of view, named a local view, is not used by Take Order. The local views are used infrequently during analysis because they usually represent temporary information that is used within a process.

Defining Entity Views

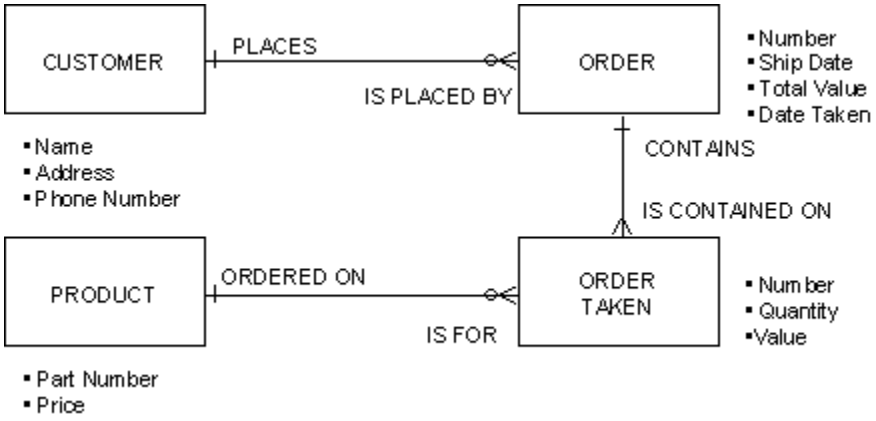
An entity view is a view of a single entity, and should not be confused with an entity action view. See the Types of Information Views section in this chapter for more information.

Types of Entity Views

An entity view can be any of four types:

- Import view
- Export view
- Entity action view
- Local view

The following illustration of the Entity Relationship Diagram fragment for the process Take Order shows the four entity types used by the Take Order process depicted in the illustration of the manual Take Order process. The following illustration includes the names of attributes.



The following table lists the entity types for which views are required for Take Order.

Entity View	Entity Type
Import	
	CUSTOMER
	PRODUCT
	ORDER ITEM
Export	

	ORDER
Entity Action	
	CUSTOMER
	ORDER
	ORDER ITEM
	PRODUCT

Naming Entity Views

There are two views of entity type CUSTOMER. When defining the detailed process logic for Take Order, you will need to refer to elements of each of these two views separately. However, there is no way yet to tell them apart.

CA Gen provides for these cases by allowing each entity view to have a name. Whenever detailed process logic refers to an entity view, the name takes the form:

```
entity-view-name entity-type-name
```

Entity view names always act as modifiers of the entity type name. Therefore, if the import view of CUSTOMER were named Import, the entity view would be referred to as "Import CUSTOMER" in the detailed process logic.

This combination of entity view name and entity type name must be unique for a process.

For each entity type requiring a view in a process, one view may be unnamed. As long as only one is unnamed, it is still distinguishable from other views of the same entity type.

By convention, entity action views are normally left unnamed unless there are multiple entity action views of the same entity type. This convention improves the readability of the detailed logic that appears in an action diagram.

The following table shows details of the views of the Take Order process.

Entity View	Entity Type	Entity Attribute(s)
Import	CUSTOMER	NAME
Import	PRODUCT	NUMBER
Import	ORDER ITEM	QUANTITY

Entity View	Entity Type	Entity Attribute(s)
Export	ORDER	NUMBER SHIPDATE TOTAL VALUE
(unnamed)	CUSTOMER	NAME NUMBER
(unnamed)	ORDER	NUMBER
(unnamed)	ORDER ITEM	QUANTITY VALUE
(unnamed)	PRODUCT	NUMBER PRICE

The following notation of the view details is used in CA Gen:

Import View	Take Order Process
view of	IMPORT
entity	CUSTOMER
attr	NAME

- view of precedes an entity view name
- entity precedes an entity type name
- attr precedes an attribute name

Entity View Definition

CA Gen can capture the following details for each entity view:

- **Name—As** shown in the previous example, the entity view name should be a modifier of the entity type name. The combination of entity view name and entity type name must be unique within the views of the process.
- **Description**—In CA Gen, the description for a view of any kind is optional. It is necessary only when the purpose of the view is not clear from its name.
- **Optionality**—You may specify optionality only for import views.

- **Mandatory import views**—If an import entity view is specified as mandatory, each of its attribute views specified as mandatory must have a value when the process to which it belongs begins execution.
- **Optional import views**—If an import entity view is specified as optional, none of its attribute views need have a value, even if specified as mandatory for the entity type.

The optionality condition for an optional entity view may be specified in the same way as the optionality conditions for attributes or relationship memberships. This condition can be stored as part of the entity view description.

Defining Attribute Views

Each entity view contains one or more of the attributes of the entity type of the view based on the needs of the process. For example, the Entity Relationship Diagram fragment for the process Take Order shows that the entity type CUSTOMER contains three attributes:

- Name
- Address
- Phone Number

Based on the illustration of the manual Take Order process, the entity view of customer in the import views of Take Order includes only the Name attribute. So, the only information required for a customer is the value of its Name.

The only property available for an attribute view is its optionality within the import views.

If an import attribute view is specified as mandatory, it must have a value when process execution begins. You must therefore be sure that a value for every mandatory attribute is available to the enterprise when an entity is first recognized (that is, needs to be created) by a process.

Defining Group Views

The views for the Take Order Process do not completely show the structure of the data. In the original description of the Take Order process, it was clear that multiple products could be ordered on a single order. The import view must therefore allow for multiple occurrences of product for its Number and order item for Quantity. This is accomplished by using a repeating group view.

A group view is strictly defined as a collection of one or more entity views. A group view is the only kind of view that may be designated as repeating. So, any time a process imports or exports a list of entities, the list is included as a repeating group view.

To complete the views for Take Order, you must add a group view to which the entity views Import product and Import order item will belong.

The following tables show the finished views for Take Order. Notice that the letter "r" precedes the group view Import Product Information indicating a repeated group view.

Import Views	**Take Order Process**
view of	IMPORT
entity	CUSTOMER
attr	NAME
group (r)	IMPORT PRODUCT INFORMATION
view of	IMPORT
entity	PRODUCT
attr	NUMBER
view of	IMPORT
entity	ORDER ITEM
attr	QUANTITY

Export Views	**Take Order Process**
view of	EXPORT
entity	ORDER
attr	NUMBER
attr	SHIP DATE
attr	TOTAL VALUE

Entity Action Views	**Take Order Process**
view of	(unnamed)
entity	CUSTOMER
attr	NAME
attr	NUMBER
view of	(unnamed)
entity	ORDER
attr	NUMBER
view of	(unnamed)
entity	ORDER ITEM
attr	QUANTITY
attr	VALUE

Entity Action Views	Take Order Process
view of	(unnamed)
entity	PRODUCT
attr	NUMBER
attr	PRICE

CA Gen can capture the following details about group views:

- Name
- Description
- Cardinality
- Minimum cardinality
- Maximum cardinality
- Average cardinality
- Optionality (import views only)
- Method of subscripting

Group views may not appear in the entity action views for a process.

Name

Each group view name should accurately reflect its contents. Unlike entity view names, group view names always stand alone in the detailed process logic, so they should specify the role of their elements, for example, input, update, and output.

This explains the name Import Product Information in the table previous table rather than simply Product Information.

Description

In CA Gen, the description for a view of any kind is optional. It is necessary only when the purpose of the view is not clear from its name.

Cardinality

Cardinality indicates whether the group view repeats. The cardinality may be:

- One (non-repeating)
- One or more (repeating)

In practice, detailed process logic never needs to reference non-repeating group views. Instead, it references their entity views. The analyst and business person may, however, name a group of entity views to provide clear documentation about parts of the data that are used together.

For each repeating group view, you must also specify the following information that will be used in the design and construction of a system:

- **Minimum cardinality**—The least number of occurrences it can contain
- **Maximum cardinality**—The greatest number of occurrences it can contain
- **Average cardinality**—The average number of occurrences it can contain

For minimum and maximum cardinality, you may indicate whether the value specified is an estimate—there will probably never be more than ten or an absolute limit—there must never be more than ten.

Optionality

You may specify optionality for import views only.

- Mandatory group views
- If an import group view is specified as mandatory, each element specified as mandatory must have a value when the process to which the view belongs begins execution. For example, in the following table, Group View 1 consists of Group View 2 and Entity View 3.

Group	View
Group	GROUP VIEW 1
group	GROUP VIEW 2
entity	ENTITY VIEW 3

- If Group View 1 is mandatory, then Group View 2 is also mandatory. All mandatory attributes that appear in all its entity views are also mandatory. So the mandatory attributes of Entity View 3 must have values when the process to which Group View 1 belongs begins to execute.
- Optional group views
- For an optional group view, you may specify the optionality condition in the same way as the optionality conditions for attributes or relationship memberships. This condition is stored as part of the group view description.

Subscripting

Subscripting is the means by which a particular occurrence of an element of a repeating group view is identified by a statement in an Action Diagram.

Each repeating group view, that is, a group view with a cardinality of one or more, must be defined as either implicitly or explicitly subscripted.

In design, repeating group views are usually handled using implicit subscripting because this gives better control of selection within a group.

In analysis, implicit subscripting is sufficient for the handling of repeating group views.

A process must set values for explicit subscripts; only in very complex process logic is there a need for explicit subscripting. Subscripting can be changed to explicit during system design if necessary.

Analyzing Process Logic

To help make the transition from the high level of detail at which activity analysis stops to the low level of detail required in the Process Action Diagram, you need to detail the logic of an elementary process.

Beginning with a very general definition, you add detail at each step.

In the final step, you combine all these details to form a Process Action Diagram, which describes aspects of process behavior based on a relevant subset of the Entity Relationship Diagram.

Building a complete action diagram is often deferred until design, especially when the analysis model is not completely stable, or when business logic is not fully agreed upon and subject to change during design prototyping.

Meanwhile, a description of the business logic may be recorded as part of the process definition, with additions being made as process logic is analyzed and actions and business rules are formulated.

The formulation of detailed process logic benefits from such a formal step-by-step approach because of its extremely detailed nature and the precise language used to record it (either deferred until design, or during analysis) in the Process Action Diagram.

Preparing for Process Logic Analysis

You may specify process logic for each elementary process that will be supported by the business system to be developed. This is especially useful with a complex process, one that interacts with many entity types or that performs iterative or complex calculation. It is also useful for a process whose performance is time-critical or is likely to make exceptional demands on technical facilities, for example, using a large number of entities during a process execution.

No information views or expected effects are required before beginning this analysis, since they will be defined or reviewed as necessary. Typically the expected effects have been defined for each elementary process.

Other indications of the data that may be needed, established, or changed may be deduced from any pre-conditions and post-conditions identified during process dependency analysis. See the chapter "Analyzing Activities."

Where the process logic is reasonably straightforward, it may be possible to shortcut process logic analysis steps by using CA Gen automatic action diagram generation. This is called stereotyping.

Automatic action diagram generation allows you to build an Action Diagram to either create, update, delete, read, or list entities of the type selected. This is typically a primary entity type, as in Step 1 of Analyzing Process Logic.

Stereotyping leads you through a dialog in which you select actions on relationships and related entities. This process is described in detail in the toolset documentation. Expected effects on entity types are automatically recorded based on your decisions. The resulting action diagrams can then be selected as operations on the entity type, which is shown in the Data Model Browser.

When the process logic is more complex, stereotyping can be used to establish a starting point for the development of entity action views and action diagrams after the initial formulation of process logic is complete. See Step 4 Determine Actions on Attributes and Relationships section in this chapter for more information.

CA Gen can also generate an action block for a process based on its recorded expected effects on entity types, but stereotyping is the technique assumed here.

Performing Process Logic Analysis

These are the steps for analyzing process logic:

Step 1 Identify Primary Entity Types

This is the first step in creating the Process Logic Diagram. You can use this diagram to sketch out process logic before specifying it precisely in the Action Diagram.

The general approach to building a Process Logic Diagram is to superimpose actions to be performed in the process on a subset of the Entity Relationship Diagram, showing entity types to be affected.

The purpose of this step and Step 2 is to scope this Entity Relationship Diagram fragment.

The sample process used in this discussion is the Take Order process. The relevant Entity Relationship Diagram fragment is an expansion of the one shown in the Entity Relationship Diagram fragment for the process Take Order.

The primary entity types are either already identified as the objects to be worked on by the process, defined as expected effects of the processes, or, simply those entity types that immediately spring to the mind of the analyst when visualizing the process.

With the Take Order process, for example, the entity type order is obviously required. Even if the requirements for product, order item and customer do not occur to you, you can identify them in the next step. At this point, you need to at least note that Take Order uses the entity type order.

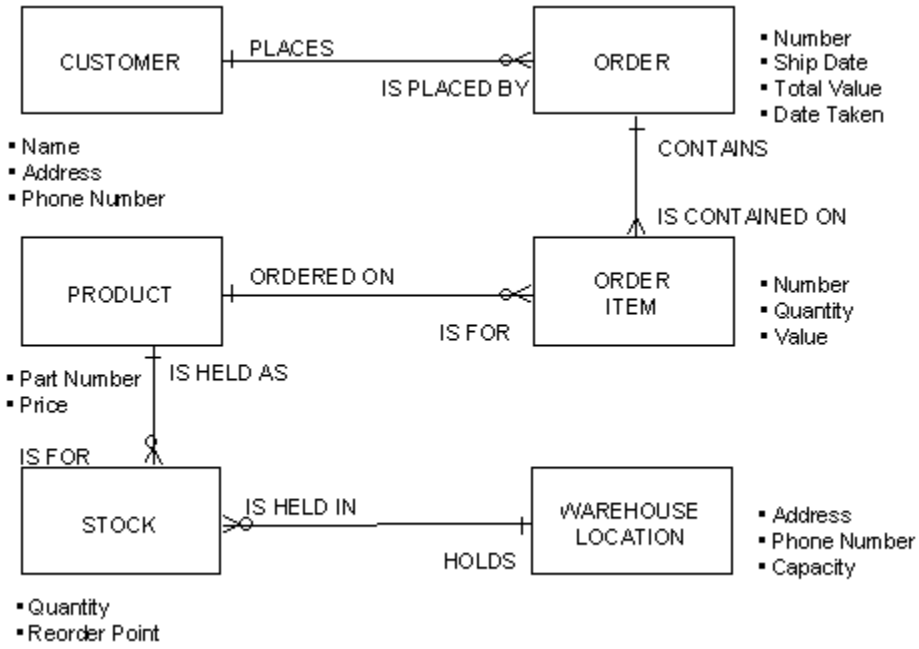
If you follow the parallel decomposition approach, you will often find that the primary entity type is the central entity type in its subject area, which is naturally the first place to begin the next step. The chapter "Building the Analysis Model" describes the parallel decomposition approach.

Step 2 Examine the Primary Entity Types Neighborhoods

To finish developing the Entity Relationship Diagram subset, you must inspect the neighborhood of the primary entity types to find additional entity types required by the process.

The neighborhood of an entity type is the set of all entity types directly related to the subject entity type.

For each such entity type discovered, you also examine the neighborhood of that entity type. This activity is repeated until the neighborhoods of all affected entity types have been considered.



Step 1 identified ORDER as the only primary entity type for this process. Based on the illustration perform these steps:

1. Examine the neighborhood of order. It contains two entity types, order item and customer.
2. Customer places an order and is therefore of interest to Take Order. Record that customer is important.
3. ORDERs consist of order items. Since the relationship between order and order item is fully mandatory, they must both be created in the same process. Order item is therefore of interest to Take Order. Record that order item is important.
4. Examine the neighborhood of customer.

   According to the illustration, CUSTOMER has only one entity type in its neighborhood, the original primary entity type order. Since you have already identified order as important and since customer has no more relationships, that set of neighborhoods is exhausted.

5. Examine the neighborhood of order item.

   Besides order, order item has one entity type in its neighborhood: product. Since the kind of product ordered on an order item can only be identified by the pairing of the order item with a product, clearly product is important to Take Order also. Record this fact.

   The relationship membership order item is for product, is mandatory. This indicates that you must establish the pairing between order item and product in the same elementary process in which the order item is created.

6. Examine the neighborhood of product.

   Besides order item, the only entity type in the neighborhood of product is stock. After careful consideration, you and the business staff determine that order taking does not change stock levels. Thus, stock is not important to the Take Order process. This is a simplified example. In practice, Take Order may alter an attribute of stock called Free Stock Level.

   Because none of the elements of the neighborhood of product is important to Take Order, you have exhausted all potential neighborhood entity types.

The entity types discovered during this step are:

- ORDER
- CUSTOMER
- ORDER ITEM
- PRODUCT

The final activity in Step 2 is to draw a fragment of an Entity Relationship Diagram showing the entity types required for the process.

Usually, the set of entity types with which a single process deals is small enough to be contained in a page print or plot from the Entity Relationship Diagram. In the Take Order example, the Entity Relationship Diagram fragment and the expanded Entity Relationship Diagram fragment both happen to show the exact required subset of the Entity Relationship Diagram. If information views have already been analyzed, this set of entity types should resemble the composite list of import and export views of the process. You must resolve any anomalies, such as an entity type missing from this diagram but listed as an export view.

Step 3 Determine the Entity Actions

In this step, you annotate the Entity Relationship Diagram with the actions to be performed on the identified entity types. These actions should be significant for the business; actions to correct data errors, for example, should be left for the system designer.

These are the entity actions that may be performed:

- **Create**—Stores information about an entity that previously did not exist.
- **Read**—Makes available stored information about an entity.
- **Update**—Modifies stored information about an entity.
- **Delete**—Erases stored information about an entity.

  Delete is very rare in business. For legal, audit and statistical purposes, entities are usually updated to "inactive" status or placed in an inactive termination state a nd not deleted until (often years later) a separate administrative process discards entities that are no longer needed.

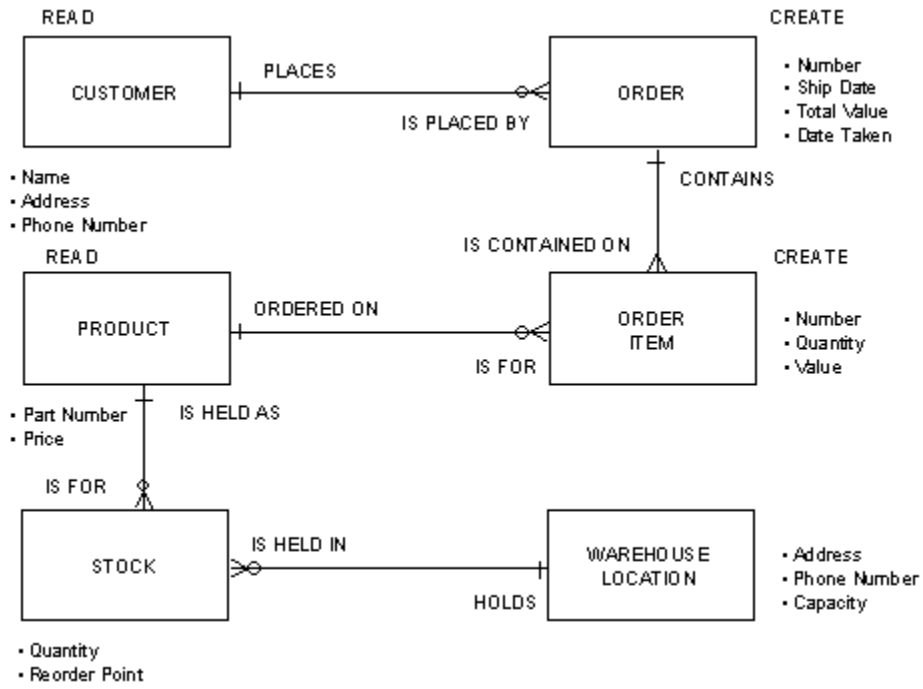
The entity actions required in the Take Order process are:

- ORDER must be Created
- CUSTOMER must be Read
- ORDER ITEM must be Created
- PRODUCT must be Read

This list of actions on entity types contains exactly the same information as the expected effects of the process. If expected effects have already been recorded, you should verify them against this list.

CA Gen reflects the content of an action diagram for an elementary process in the expected effects of that process, but in the case of product, the action diagram contains a Read statement, for product, and the expected effect is Update because the set of product relationship memberships is modified.

The following illustration shows the annotated Entity Relationship Diagram fragment, from now on referred to as a Process Logic Diagram, for the Take Order process.



Step 4 Determine Actions on Attributes and Relationships

The Create and Update entity actions may have attribute and relationship membership actions subordinate to them. In this step, you identify these subordinate actions and update the Process Logic Diagram to show them.

- A Create action can contain subordinate set actions that assign values to attributes. A Create action must set each mandatory attribute of the affected entity type. In addition, it may set any optional attributes as required by the process.
- A Create action may also establish (Associate) pairings. As with attribute actions, a Create action normally contains "Associates" for all mandatory relationship memberships. However, with fully mandatory relationships, this is not always possible until occurrences of each associated entity type have been created.
- An Update action can modify any attribute of the affected entity type, except an attribute that is an identifier or part of an identifier. This modification can assign a value to an attribute, or remove any value from an attribute. An Update action may also establish (Associate) or eliminate (Disassociate) pairings.
- Relationship actions can also occur independently of Create or Update actions, for example when modifying an order.

In the Take Order process, Create actions operate on two entity types: order and order item. Assume that the attributes of those entity types have the optionality shown in the following table:

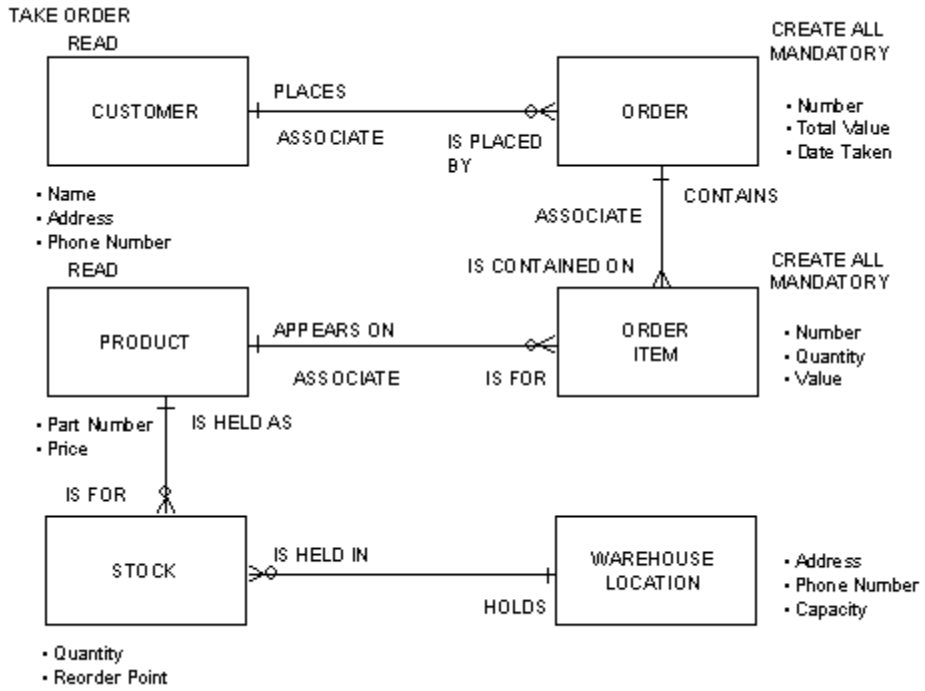
ORDER	**Optionality**
NUMBER	(mandatory)
SHIP	(optional)
DATE	(mandatory)
TOTAL	(mandatory)
VALUE	
DATE	
TAKEN	

ORDER ITEM	**Optionality**
NUMBER	(mandatory)
QUANTITY	(mandatory)
VALUE	(mandatory)

The only optional attribute of either order or order item is Ship Date. You must set all mandatory attributes of both order and order item in the statements that Create them.

As for Ship Date, it will probably not be set until the order is filled and shipped, so it is not set in the Take Order process.

The following illustration shows the Process Logic Diagram for Take Order annotated with subordinate attribute actions to the Create actions for order and order item.



Also in the example based on the illustration of the Entity Relationship Diagram fragment for the process Take Order, the relationship between order and order item is fully mandatory. Assuming that you create an order before its order items, the Create for order cannot include an Associate to order item; the order item does not yet exist. You must establish the relationship between order and order item when each order item is Created.

The rule to remember is that all mandatory relationships must be established by the time the elementary process finishes executing. An individual entity action might leave things inconsistent for a moment, but the total collection of actions that form a process must leave the business in a completely consistent state.

For the Take Order process, you must consider the relationships of order and order item. It so happens that each relationship membership of order and order item is mandatory so they must all be established. The Process Logic Diagram for Take Order after detailing actions shows the Process Logic Diagram for Take Order, annotated to include attribute and relationship actions.

Step 5 Determine Sequence of Actions

After identifying all actions on entity types, attributes and relationships, you can determine the sequence in which those actions should be performed during process execution. This sequence is recorded on the Process Logic Diagram by numbering each of the entity and relationship actions to be specified.

Before two entities can be paired, they should both be available through a Read or a Create. In general, you specify Read actions for existing entities and then Create actions for entities with which they will be paired.

Sometimes the sequence of actions is irrelevant. This situation typically occurs when sets of actions can either be performed in parallel with, or exclusive of, one another. In such cases, you must assign an arbitrary sequence to the actions in question.

Process Logic Diagram notation is primarily oriented to showing a linear sequence of entity actions. If necessary, the notation can be extended to show details of conditional, repetitive, or recursive processing, as shown in the illustration of the Process Logic Diagram for Take Order after detailing actions.

Step 6 Detail the Actions

Before entities can be Read, a set of criteria for selecting them from among other entities of the same type needs to be established. In this step, you identify these criteria and use them to annotate the Process Logic Diagram.

Selection criteria can include any combination of:

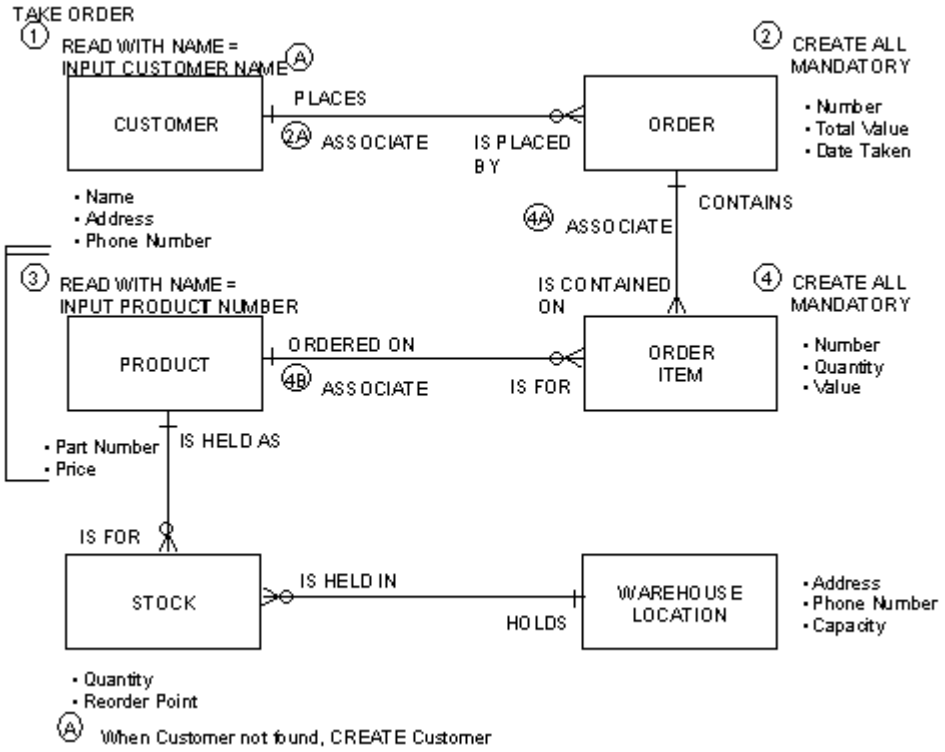
- An attribute value of the entity being Read
- An attribute value of an entity directly or indirectly related to the entity being Read
- The existence of a relationship

You develop selection criteria by combining these elements using a series of ands and ors to fully identify the entity to be retrieved. For more information about developing selection criteria, see the Read action statement in the toolset documentation.

In the Take Order example, entities of two types will be Read:

- **CUSTOMER**—The customer to be retrieved is the one whose Name is the same as the Requesting customer Name
- **PRODUCT**—The products to be retrieved are those with Numbers equal to the list of product Numbers in the Requested product and Requested order item.

The following illustration shows the Process Logic Diagram for Take Order after detailing actions annotated with entity selection criteria.



The Process Logic Diagram may not show details of repetitive and conditional actions, so important process logic questions may remain:

- How are optional or exceptional conditions to be handled?

  An optionality symbol can be used with a letter that identifies a note describing, for example, that the process should Read a certain entity only under certain circumstances, or what happens if, during the execution of the Take Order process. There is no customer associated with the customer Number requested.

- How is repetition to be handled?

  For example, a list of product Part Numbers and order item Quantities serves as input to the Take Order process. The repetition bracket on the left of the illustration specifies that actions on product and order item should occur repeatedly.

These questions will be addressed more rigorously when building the Process Action Diagram.

Other actions you may identify during this step involve the use of common logic. These are formulated in CA Gen using action blocks that may be used by action diagrams for many processes and later in design by procedures. For example, part of the order item VALUE may be calculated using a Sales Tax algorithm.

Step 7 Begin the Process Action Diagram

Complete action diagrams should not normally need to be developed during analysis. There may be some circumstances when complex business logic needs to be captured, for example, business rules for calculating interest due. The ability of the data to support this can be verified by specifying information views and possibly by specifying an algorithm in note form. Otherwise, action diagrams should be detailed in design when developing the procedures and common action logic that will support them.

CA Gen creates an action diagram for every process that is designated elementary. Entity actions may be added during stereotyping, generated from expected effects or added by you. If no actions are added, the Check command will report an error. It will then be impossible for the designer to use the process until action details have been added.

For now, you can describe the process as part of the process definition, or as a set of notes in the Action Diagram for the process. Developing an action diagram is described in detail in the chapter "Block Mode Design" and the chapter "Client/Server Design."

You may now also define entity action views. At least one entity action view is required for each entity type participating in an entity action. Entity action views are discussed in defining Information Views. The phrase "participating in an entity action" is precise. Sometimes an entity type is only mentioned as part of the selection criteria for an entity of some other type. Regardless of this, an entity action view is required for it.

For example, a Read that finds any customer who placed an order for a certain product requires an entity action view for all four entity types (customer, order, order item, and product) even though attributes are only retrieved for customer. This is because these other entity types and relationships are needed to identify each customer.

Using Interaction Clustering to Refine Project Scope

Cluster analysis is a technique for grouping things based on some set of common characteristics.

In analysis, clustering is used to group elementary processes based on their use of data. The main technique is clustering based on expected effects, using CRUD (Create, Read, Update, and Delete) values for the interactions between processes and entity types.

After clustering, each cluster represents a set of data and activities that should be handled by one integrated system because it includes all the processes that create or modify a set of tightly related data.

This technique is particularly suited to developing a set of procedures that together can maintain closely related data as a service to a wide range of applications that can then exploit that data.

Confirming and refining the scope of business systems to be implemented may be critical in a large development project that will implement many systems or that must progressively implement support for groups of processes. Cluster analysis helps to ensure that each group of elementary processes as it is implemented has available the entity types that are needed.

If you follow the principles of parallel decomposition of data and activities, you should find that there is a close correspondence between groupings of entity types into subject areas, and elementary processes into higher level activities. See the chapter "Building the Analysis Model." In such a case, it may not be necessary to perform any cluster analysis. The groupings are functionally what the scope of the development project requires. If it becomes useful to explore boundary issues to determine where to place an entity type or elementary process, you can perform cluster analysis as a confirmation technique.

CA Gen supports techniques for scoping in analysis based on grouping entity types and activities into clusters of tightly interacting model objects in a matrix. The result of this grouping is represented in the clustered Entity Type/Elementary Process Matrix.

CA Gen supports both the recording and clustering of this matrix. The cell values with which this matrix is populated are interactions between activities and data. These values either derive from the expected effects defined for elementary processes or may be entered directly into the matrix. CA Gen clusters, as well as possible, based upon the currently recorded interactions.

Often you have additional background information. Repeated validation and correction cycles using user-selected parameters allow you to explore alternatives and to identify and correct errors both in defining interactions and in handling the Entity/Activity Matrix.

The final step requires human intelligence to tidy up and validate the final clustered result. The technique therefore includes final manual definition of business areas and business systems after initial automatic clustering.

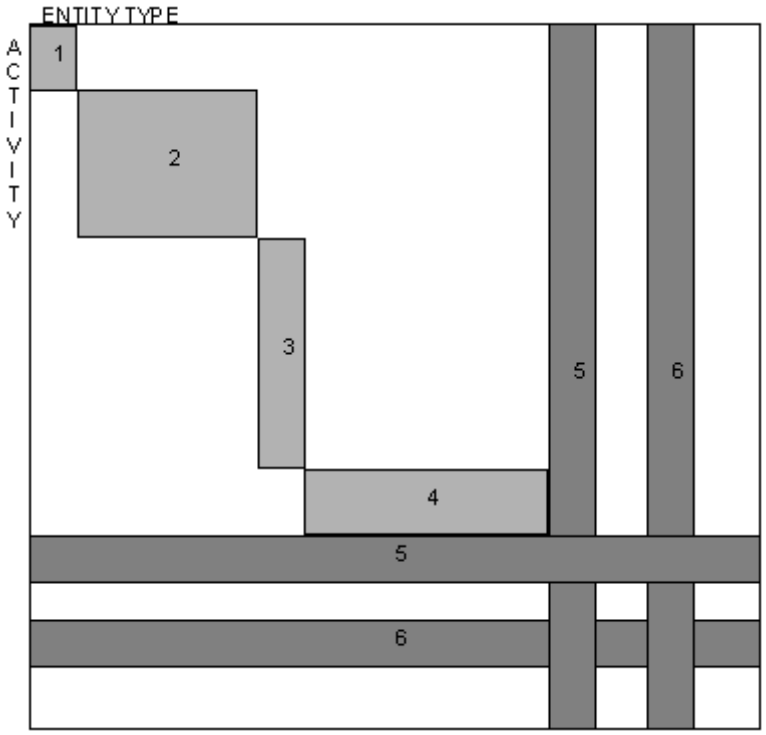
Basic Interaction Clustering Concepts

Interaction Clustering can be performed on two matrices:

- Entity Type Used By Function
- Entity Type Used By Elementary Process

The rows and columns of these matrices are rearranged to show a "staircase" of clusters from top left to bottom right.

A clustered matrix may contain one or more of the elements shown in the following illustration.



For a description of this matrix, see the following table.

Key	Comment	Description
1	Ideal cluster	The ideal result of interaction clustering is many small clusters of 1 to 16 cells. Expect 2-10 cells in a cluster to contain Cs or Us. Be suspicious, however, if many clusters contain only one cell. This may indicate that the Information Architecture is insufficiently detailed.
2	Too large a cluster	A cluster with more than 16 cells is generally too large. It may be caused by one or more dense activities, entity types, or both, that is, rows, columns, or both with many Creates and Updates.
3	Many rows but few columns	A very tall and narrow cluster often indicates that the rows are at too low a level of detail compared with the columns.
4	Many columns but few rows	A very short and fat cluster often indicates that the columns are at too low a level of detail compared with the rows.

Key	Comment	Description
5	Dense columns or rows	These indicate activities or entity types that have previously been identified (automatically or manually) as dense, and excluded from subsequent clustering iterations. Each column or row should be treated as a separate item. Either it should be a cluster in its own right, or it has been removed from clustering for investigation of activity or entity type definition, and clarification or redefinition of each interaction.
6	Unclustered columns or rows	Activities or entity types cannot be included in a cluster because either they have no C or U interactions, or their only C and U interactions are in dense columns or rows.

The scope of a cluster identifies the smallest group of entity types and activities that together should form the scope of a system. Each cluster is de-coupled as far as possible from all other clusters.

A development project may safely be targeted at the "ideal" scope of a cluster with the minimum need for development coordination with other projects. However, where clusters are small or closely related in the business, or existing systems being replaced cover a wider scope, then a larger optimum scope can be formed by taking two or more complete clusters.

Often, the initial requirement scope of a project, or of a system that is being replaced, does not neatly match the ideal cluster scope. While adjusting the target scope is easy, it is a management role to balance the ideal information structure against the business demands and transition requirements.

Performing Interaction Clustering

Guidelines for performing interaction clustering:

- Start clustering as early as possible during analysis. Do not wait until all the details are available and agreed. The interim results will help to clarify ideas and to direct follow-up actions to improve the processes and entity types.
- Scope repeatedly. Scoping is not a one-time procedure.
- Use the results of initial scoping to narrow down the scope of analysis.
- Use details of the scope of the systems to selectively complete analysis only for those processes that form part of the priority business system. Subsequent projects will fill in the details for other systems. CA Gen will help to ensure coordination and consistency between these subsequent projects.

The following steps are for performing interaction clustering:

Step 1 Cluster the Elementary Processes

This simply involves executing the cluster command while accessing the Entity Type Used By Elementary Process Matrix.

The following illustration shows the matrix before the clustering operation, with expected effects included.

Cell Values: = Not referenced C = Create D = Delete U = Update R = Read / Entity Type	Activity	AMEND CONTRACT	ANALYZE RESEARCH	ASSESS CREDIT	COMMUNICATE	CONCEIVE PRODUCT	DEVELOP PROPOSAL	DEVELOP SPECIFICATION	DEVELOP TECHNICS SALE	ESTABLISH TARGET	GAIN COMMITMENT	IDENTIFY NEEDS	IMPLEMENT PRODUCT	MANAGER	MANAGE PRODUCT GROUP	MANAGE VERTICAL MARKET	NEGOTIATE SALE	PLAN RESEARCH	PURCHASE	SET PRICES	TRACE DESIGN
ADVERTISING MEDIUM					U									C							
COMMITMENT	C						C	C			C						C		C		
COMPETITOR												C									
COMPETITION PRODUCT												C									
CUSTOMER SURVEY											U		C			C					
DESIGN PROJECT									C												U
GEOGRAPHY ZONE																					
MARKET																C					
MARKET NEED		U										C									
MARKET RESEARCH		U																C			
PORTFOLIO STRATEGY											U				C					C	
PRODUCT						C							U							R	
PRODUCT APPLICATION						C							U								
PRODUCT GROUP						C							C		U						
PRODUCT PRICE																				C	
PURCHASE ORDER	C						U	C			U						U		C		
SALES TARGET										C											
VENDOR																			C		
VENDOR EMPLOYEE																			C		
VENDOR PRODUCT																			C		

For clarity, only C and U expected effects are shown, but the matrices can contain R and D as well. You should complete any missing expected effects before clustering.

Following clustering, expect to see a matrix more like the one in the illustration which shows one specific R to illustrate an example issue.

Step 2 Adjust the Clusters if Necessary

Sometimes the clusters automatically calculated by CA Gen are not as tidy as those in the following illustration.

Cell Values: = Not referenced C = Create D = Delete U = Update R = Read Entity Type	Activity	AMEND CONTRACT	DEVELOP PROPOSAL	DEVELOP SPECIFICATION	GAIN COMMITMENT	NEGOTIATE SALE	PURCHASE	CONCEIVE PRODUCT	IMPLEMENT PRODUCT	MANAGE VERTICAL MARKET	MANAGE PRODUCT GROUP	SET PRICES	MANAGER	COMMUNICATE	IDENTIFY NEEDS	ANALYZE RESEARCH	PLAN RESEARCH	ESTABLISH SALE TARGET	DEVELOP TECHNICS SALE	TRACE DESIGN	ASSESS CREDIT
COMMITMENT		C	C	C	C	C	C														
PURCHASE ORDER		C	U	C	U	U	C														
VENDOR							C														
VENDOR EMPLOYEE							C														
VENDOR PRODUCT							C														
PRODUCT								C	U												
PRODUCT APPLICATION								C	U												
PRODUCT GROUP								C	C												
CUSTOMER SURVEY					U				C	C											
MARKET										C											
PORTFOLIO STRATEGY					U						C	C									
PRODUCT PRICE												C									
ADVERTISING MEDIUM													C	U							
COMPETITOR															C						
COMPETITOR PRODUCT															C						
MARKKET NEED															C	U					
MARKET RESEARCH																U	C				
SALES TARGET																		C			
DESIGN PROJECT																C			C	U	
GEOGRAPHY ZONE																					

You may need to investigate the definitions of processes and expected effects where processes appear in unexpected clusters and to intervene to allocate dense or unclustered processes or entity types. If this is the case, use rhw Matrix Processor facilities to move rows and columns around until the appearance of the clusters is improved.

Step 3 Determine Business System Boundaries

Finally, the results of clustering are used to identify business systems. Consider the following illustration.

Cell Values: = Not referenced C = Create D = Delete U = Update R = Read	Activity	AMEND CONTRACT	DEVELOP PROPOSAL	DEVELOP SPECIFICATION	GAIN COMMITMENT	NEGOTIATE SALE	PURCHASE	CONCEIVE PRODUCT	IMPLEMENT PRODUCT	MANAGE VERTICAL MARKET	MANAGE PRODUCT GROUP	SET PRICES	MANAGER	COMMUNICATE	IDENTIFY NEEDS	ANALYZE RESEARCH	PLAN RESEARCH	ESTABLISH SALE TARGET	DEVELOP TECHNICS SALE	TRACE DESIGN	ASSESS CREDIT
Entity Type																					
COMMITMENT		C	C	C	C	C	C									Purchasing					
PURCHASE ORDER		C	U	C	U	U	C														
VENDOR							C														
VENDOR EMPLOYEE							C														
VENDOR PRODUCT							C														
PRODUCT								C	U			R									
PRODUCT APPLICATION								C	U												
PRODUCT GROUP								C	C							???					
CUSTOMER SURVEY					U				C	C											
MARKET										C											
PORTFOLIO STRATEGY					U						C	C				Pricing					
PRODUCT PRICE												C									
ADVERTISING MEDIUM													C	U		Advertising					
COMPETITOR					Competitive Analysis										C						
COMPETITOR PRODUCT															C						
MARKKET NEED															C	U					
MARKET RESEARCH						Market Research										U	C				
SALES TARGET						Targeting												C			
DESIGN PROJECT						Project Support													C	U	
GEOGRAPHY ZONE																					

The results of interaction clustering in the illustration show that the analysis model includes eight design clusters:

- Purchasing
- ???
- Pricing
- Advertising
- Competitive analysis

- Market Research
- Targeting
- Project Support

At least four issues need to be resolved:

- If the member activities of the design area cluster 2 are not normally associated, then possibly there has been an analysis fault, or a typing/documentation error. Should the activity Implement Product create the entity type customer survey? Discussion must be held with end users of the member activities to verify the expected effects (CRUD), and perhaps name the cluster Product Development.
- There is some doubt about whether the entity type geographic zone is relevant to the project scope (or analysis model for which the matrix is drawn) because it has no Create activity. Again, consult end users.
- Questions arise about the activity Assess Credit being part of the scope. Consult end users.
- There needs to be an investigation of implementation priorities, coupled with a review of the Read interaction dependencies between clusters. For example, the Pricing cluster reads the entity type product created and updated by the newly named Product Development cluster.

It would be technically simpler to implement the second design area cluster Product Development before the Pricing design area. However business priorities may require Pricing to be implemented first. If so, then an existing file of products will have to be accessed through bridging software until the second cluster is resolved and the fully defined product is implemented.

Before assuming that the cluster analysis is complete, consider comparing the results of this analysis with any existing Business Systems Architecture developed during Planning. Although the business systems just derived are much more detailed than those predicted during Planning, their shapes should be roughly the same. If not, reconcile the two views, either by revising the Business Systems Architecture or by changing the grouping of processes.

Since a business system is the basis for a development project, it must be sized appropriately for that project. Combine clusters to produce an appropriate development scope that:

- Covers business requirements
- Is of manageable size for the skills and experience available within the development team
- Delivers worthwhile business benefits when implemented; these benefits must therefore be identified and quantified

Analyzing Roles

Role analysis helps to establish who performs what activities, and who uses what data.

Role analysis helps to:

- Check that all elementary processes needed by the users have been identified
- Plan the deployment of data, processes, or both
- Provide a basis for the design of system procedures to support roles
- Plan the training of end users

From the results of role analysis, business management may also if needed review the activities of the enterprise and identify opportunities to re-engineer the business.

This section describes the use of matrices to compare roles with processes and entity types. Other matrices may also be useful here, for example, the Process/Organization Unit Matrix, Organizational Unit/Role Matrix; and Organizational Unit/Location Matrix.

For matrices involving location, see Analyzing Distribution. Role analysis is developed further during design, using the user task design technique as described in the chapter "Block Mode Design" and the chapter "Client/Server Design."

Basic Role Analysis Concepts

A *role* is a function played by one or more persons in the enterprise to perform a particular task. For example, an accounts clerk is needed to chase an order. One person in the enterprise usually takes on more than one role to do their job. The job of Warehouse Manager, for instance, may require the jobholder to take on several roles, such as a Purchaser, and even a Warehouse Stocker when a large consignment is delivered.

For the organizational units within the scope of the development project, current and planned job descriptions provide a useful starting point for a list of roles. It is important to allow for future growth, simplification, and diversification in roles, especially if the development project is associated with some business process re-engineering or improvement.

The main tasks or areas of responsibility for each job description may equate to a role. A job may require several roles to be undertaken. However, some roles, such as electronic mail user, may relate to generally performed tasks rather than to particular areas of responsibility.

Analyzing Roles and Processes

Use a Process/Role Matrix, as shown in the following illustration, to analyze which processes are performed by what roles. For each role that performs a process, insert an X or weighting (1-9) at the intersection.

KEY Blank = Not Referenced X,1-9 = Included	**Role**	MARKETEER	ACCOUNTS CLERK	SECRETARY	CLERK	SALESMAN	MAIL ADMINISTRATOR	MANAGER
Process								
ENTER NEW CUSTOMER					X	X		
ENTER NEW POLICY					X			
CANCEL POLICY					X			
ACCEPT POLICY					X			
RENEW POLICY					X			
ACCEPT PAYMENT			X					
PAYOUT			X					
SET UP MAIL ADDRESS							X	
MAIL PROMOTIONS				X				
CREATE NEW PRODUCTS		X						

If a role performs no process, then that role probably does not have a part in the project scope.

A process not performed by any role may indicate that the analysis is incomplete. Either some current role has been included in the matrix, or possibly a new role needs to be defined.

Analyzing Roles and Entity Types

Use an Entity Type/Role Matrix to analyze which roles use what data, as in the following illustration.

KEY
C = Create
D = Delete
U = Update
R = Read

Entity Type							
POLICY		U		C			
CUSTOMER		U		R	C		
PAYMENT		C					
SALES PERSON			C	R	U		
MESSAGE	C	C	C	C	C	C	C
PRODUCT	C						
MAIL LIST	R	R	C	R	R	C	R
MAIL ADDRESS	R	R	R	R	R	C	R
DIARY	R		U		U	C	R
APPOINTMENT	R		C		C		R

For each role that accesses a particular entity, use one of the symbols, listed in priority order, in the following table.

Symbol	Represents	Description
C	Create	Role creates the entity.
D	Delete	Role deletes the entity.
U	Update	Role updates the entity.
R	Read only	Role requires read-only access to the entity.

If an entity type has no Create action, check which role is responsible for creating the entity.

If an entity is created by more than one role, check which role should be the owner of that entity type. Alternatively, check to see whether the entity type could be partitioned by role ownership.

Analyzing Distribution

Distribution analysis identifies and describes where in the enterprise processes are currently or expected to be performed.

In information system development, distribution analysis is performed only when the activities being analyzed are performed at different locations. As a consequence, the location of data and processing is identified as a design issue. It indicates the kinds of locations for each process and, by implication, where entities of each type are used.

Distribution analysis focuses on the distribution of processes on the assumption that if there is no process distribution, data distribution is meaningless, other than for physical security purposes.

The analysis techniques involve building matrices. You may build two intermediate matrices for preliminary analysis, presentation and review with users:

- Process/Location Matrix
- Entity Type/Location Matrix

The final deliverables of this analysis are:

- Expected process frequencies, entity type volumes, and growth, by location
- Estimating assumptions (optional)
- Process Frequency Table
- Entity Volume Table

CA Gen focuses on recording total volumes and frequencies, but these detailed volumes can be produced easily using simple spreadsheet facilities.

Performing Distribution Analysis

To analyze distribution, perform the following steps as needed:

Step 1 Identify Organizations and Locations

A location is a physical place at which activities of interest to the enterprise may be performed, either by the enterprise itself or by external organizational units.

- List the locations where processes in the business area are performed. As a starting point, use the locations identified in the Business Systems Architecture or the Technical Architecture.
- Occasionally, a business activity may be subcontracted and performed at a location belonging to another enterprise. For example, a warehouse would be of interest whether it is owned by the enterprise, or used by the enterprise but managed by a specialist supplier. Include locations that are external to the business if processes are performed or planned to be performed there.
- For reasons of competitive advantage, and to extend control of the value-chain, the enterprise may seek to support the activities of other enterprises at locations belonging to those other enterprises. Such locations may also be included in the analysis if the activities and their supporting data are added to the business model.
- A location does not necessarily correspond exactly to a physical address. For example, headquarters and a regional sales office of a company may share a postal address. A location need not even be fixed geographically. A ship or a car of a salesperson, especially with the telecommunications links of today, may be a perfectly acceptable location. A location is not an organizational unit, although it may sometimes share a name with a unit, for example, Southern Sales Office.
- There may be many locations that perform the same role, for example, sales offices, factories, and warehouses. The location type groups together those locations that perform similar roles and share broadly similar frequencies of activities and volumes of data. A location type is a classification of locations based on the similarity of their role or purpose, and of their level of activity.
- Identify the roles performed at locations and, if relevant, summarize similar locations together to form location types. If there are groupings already recognized by the business, for example, northern region or bulk warehouses, use them until significant variations suggest the need for subdivision.
- Record the number of locations known to exist for each location type and record user estimates or business planning assumptions of expected growth of locations of each type. Locations that serve a similar purpose but have markedly different process frequencies should not be grouped into a location type.

Step 2 Analyze Activity Distribution

Record what elementary processes are performed at which location, or location type, using a Process/Location Matrix such as the one shown in the following illustration.

KEY Blank = Not Referenced X,1-9 = Included **Process**	**Role**	NEW YORK PRODUCTIONS	DALLAS ACCOUNTS	LOS ANGELES	SAN FRANCISCO	DALLAS	NEW YORK	SALT LAKE CITY	LOS ANGELES SALES	SAN FRANCISCO SALES	DALLAS SALES	NEW YORK SALES	SALT LAKE CITY SALES	DALLAS MAIL ADMIN
ENTER NEW CUSTOMER			X	X	X	X	X	X	X	X	X	X		
ENTER NEW POLICY			X	X	X	X	X							
CANCEL POLICY			X	X	X	X	X							
ACCEPT POLICY			X	X	X	X	X							
RENEW POLICY			X	X	X	X	X							
ACCEPT PAYMENT		X												
PAYOUT		X												
READ MAIL	X	X	X	X	X	X	X	X	X	X	X	X	X	
SEND MESSAGE	X	X	X	X	X	X	X	X	X	X	X	X	X	
READ DIARY	X		X	X	X	X	X	X	X	X	X	X		
ARRANGE APPOINTMENT	X		X	X	X	X	X	X	X	X	X	X		
SET UP MAIL ADDRESS														X
MAIL PROMOTIONS	X													
CREATE NEW PRODUCTS	X													

- Record the distribution information by determining where the equivalent procedure is performed, or by obtaining the opinions of users on where it should be performed, if it is currently not done.
- If the enterprise has already decided to relocate a process, then its intended location is identified and its current location(s) and location types are recorded.
- When considering where processes should be performed, it may help to review any events that enable processes to determine at what location the business becomes aware that an event has taken place. This technique may be useful when the enterprise is considering relocating the performance of some activities, perhaps as part of re-engineering a business process.
- Business processes may be supported by more than one procedure. These procedures may occur at more than one location or location type over which the process is distributed at present.
- If a process currently has no equivalent procedure which is rare, then record its anticipated location.
- Omit from subsequent analysis any locations or location types that are not used by processes within the scope of analysis.

Step 3 Analyze Data Distribution

This is performed only if distributed data support for the business is part of the Technical Architecture, and if refinement of the data store distribution is a critical design issue.

- For each entity type and location/location type, determine the locations of processes that Create, Read, Update and Delete entities of the entity type.
- Create an Entity Type/Location Matrix, such as that shown in the following illustration. As starting points, use the Process/Entity Type Matrix (or the expected effects of each process) and the Process/Location Matrix.

KEY C = Create D = Delete U = Update R = Read **Entity Type**	**Location**	NEW YORK PROMOTIONS	DALLAS ACCOUNTS	LOS ANGELES	SAN FRANCISCO	DALLAS	NEW YORK	SALT LAKE CITY	LOS ANGELES SALES	SAN FRANCISCO SALES	DALLAS SALES	NEW YORK SALES	SALT LAKE CITY SALES	DALLAS MAIL ADMIN
POLICY			R	U	U	C	C	U						
CUSTOMER			U	U	U	U	U	U	C	C	C	C	C	
PAYMENT			C	R	R	R	R	R						
SALES PERSON				R	R	R	R	R	C	C	C	C	C	
MESSAGE	C	C	C	C	C	C	C	C	C	C	C	C	C	C
PRODUCT	C	R	R	R	R	R	R	R	R	R	R	R	R	R
MAIL LIST	C	C	C	C	C	C	C	C	C	C	C	C	C	C
MAIL ADDRESS	R	R	R	R	R	R	R	R	R	R	R	R	R	C
DIARY	U	U	U	U	U	U	U	U	U	U	U	U	U	C
APPOINTMENT	C	C	C	C	C	C	C	C	C	C	C	C	C	C

Step 4 Summarize Volumetric Data

In this step, you record or estimate the frequency of execution of each elementary process at each location, or location type, using a Process Frequency Table such as the one in the following illustration.

KEY Volume 100/Month Growth %perannum Location Process	New York Promotions	growth	Dallas accounts	growth	San Francisco	growth	San Francisco sales	growth	Dallas	growth	all	locations	growth
Enter new customer					86	9	92	8	73	9	84	10	8
Enter new policy									256	9	251	2	9
Cancel policy					19	8			33	1	26	5	4
Accept policy					53	7			52	2	52	5	5
Renew policy					81	8			191	6	156	5	7
Accept payment			421	5							421	1	5
payout			152	5	25	6			42	2	74	6	3
Read mail	417	9	399	9	319	15	497	15	521	11	451	13	12
Send message	159	8	145	8	140	15	177	15	198	11	163	13	12
Read diary	3.4	5			1.1	12	2.3	12	2.1	9	2.3	11	7
Arrange appointment	6.5	5			0.5	12	1.6	12	1.5	8	1.1	11	9
Set up mail address											0.1	1	1
Mail promotions	47	10									47	1	10
Create new products	0.54	5									0.54	1	5

In this table, the entries show process frequency, which is the number of times in a given period a process is performed at a location or location type.

- Examine the Process Frequency Table and the Entity Type/Location Matrix. Use the entity creation and deletion frequency to record or estimate the volumes of entity types expected to be included in data stores at each location or location type.
- Record the estimates in the entity volume table. In this table, the entries show entity volume: the number of entities of interest to a location or a business area.

An example is shown in the following illustration.

KEY Volume 100/Month Growth %perannum Location Entity Type	New York Promotions	growth	Dallas accounts	growth	San Francisco	growth	San Francisco sales	growth	Dallas	growth	all	locations	growth
policy			197	9	276	8			293	9	233	6	8
customer			201	8	276	7	345	8	293	9	301	11	9
payment			573	5	25	6			42	2	213	6	4
Sales person							0.71	7			0.52	5	5
message	364	6	406	7	425	8	282	5	554	6	456	13	7
product	5.1	5			1.7	4	2.3	3	3.4	4	3.1	11	5
Mail list	129	5	163	5	105	6	97	3	173	2	165	13	3
Mail address	146	9	193	9	212	15	113	15	257	11	234	13	12
diary	4.3	8			0.8	15	2	15	1.7	11	2.2	12	12
appointment	10.2	5			1.7	12	4.1	12	3.8	9	4.7	12	7

- Process frequencies and entity volumes are recorded as current figures. All the figures are based on the same time interval, for example, a month.
- Growth is extrapolated out to the planning horizon for the systems to be developed.
- The total of processes or entity types for all locations may not equal the total of individual locations, if some processes are performed, or entities are used, at more than one location.
- Record the number of locations of each type in which activities of the enterprise are performed. Together with any anticipated growth factor, this is documented as a property of the location type. Record the basis of any calculations and any assumptions made. Where there are no current figures take the best estimates of users.

Methods for assessing volumes and frequencies:

- When analyzing frequency, use any agreed upon business strategy for distribution and the Technical Architecture developed during planning to choose the appropriate level of analysis. Use the least detailed analysis that still satisfies the needs of planning and design.
- The frequency of a process that creates an entity may be derived from the volume of new entities observed or expected in the period.
- The frequency of a process that is triggered by an event is derived from the frequency of that event.
- The frequency of process executions may also be assessed using the optionality and cardinality of dependencies with other processes whose frequency is already known.

- The volume of entities of a type may be derived from information about the frequency of the processes that create and delete them, and the period of time for which entities are of interest to a location. For example, a warehouse is only interested in planned and recent deliveries, while a head office may need to analyze deliveries over a long period.
- Volumes and frequency may vary by time period, for example, the volume of orders taken may rise before a holiday period. Analysts are interested in maximum likely volumes and frequencies. For this reason, it is important to discover the expected growth. This may be derived by extrapolation from previous growth, based on assumptions connected with planned business developments, or affected by expected changes in the business environment.
- The number and growth of individual locations of each type have already been recorded. The total growth of volumes and frequencies out to the planning horizon can then be calculated by multiplication.

When likely changes in location or frequency are already known, a review of distribution stability may be usefully performed during distribution analysis, or may be performed separately as one aspect of stability analysis, which is covered in the chapter "Starting Object-Oriented Analysis."

Summary of Interaction Modeling Rules

This chapter has described a number of interaction modeling rules, some explicitly and others implicitly. This section summarizes those rules.

The rules must apply to elementary processes that are to be defined as part of a business system. There may be other activities in a model that are not yet fully defined or are not yet part of the scope of a system.

Rules for Entity Life-Cycles

The following rules apply to life-cycles but are not directly enforced by CA Gen:

- Each entity has a life that must pass through at least two states: the creation state and the termination state. The creation state is its first state; the termination state is its last state.
- A null state is never shown as an entity state subtype in the life-cycle partitioning.
- An entity may only change states as the result of a process execution.
- Each entity must, at any point in time, be in one and only one state for each life-cycle.
- For each transition between entity states, there must be an elementary process that may perform that transition.

Rules for Elementary Processes

The following rules are in addition to the rules in the chapter "Analyzing Activities":

- Each elementary process must have an action diagram that includes at least one import view, at least one export view, and one or more entity action views.
- An elementary process may provide an operation or service to entities of only one type through executing the associated action block.
- Each elementary process (where role analysis is performed) should be performed by at least one role.

Rules for Information Views

CA Gen enforces these rules when an information view is defined:

- Each entity view must refer to an entity type.
- Each part of an entity view must refer to an attribute of the relevant entity type.

Rules for Roles

Where role analysis is performed, each role should perform at least one elementary process.

Results of Interaction Analysis

Interaction analysis produces the following results, which represent the interactions for a priority part of the business being analyzed:

- Entity Life-cycle Diagram or Entity State Change Matrix for each entity type with a sufficiently complex life-cycle
- Process Action Diagram for each elementary process that is to be supported by a business system. The level of detail is a policy decision for the development project, but for complex elementary processes, the diagram should include:

  Set of import views, export views, and entity action views

  Notes about actions based on the Process Logic Diagram. The Process Logic Diagram discussed in this chapter is merely an intermediate result useful for discussion with business staff before developing a detailed, formal Action Diagram
- Clustered Entity Type/Elementary Process Matrix, showing groupings of activities into business systems, each of which will become the subject of a system development project
- Process/Role Matrix

- Entity Type/Role Matrix
- (Optional) Process/Location Matrix
- (Optional) Entity Type/Location Matrix
- (Optional) Expected process frequencies, entity type volumes, and growth, by location
- (Optional) Estimating Assumptions
- (Optional) Process Frequency Table
- (Optional) Entity Volume Table

Chapter 7: Analyzing Current Systems

Current systems analysis examines existing systems that support the business area and are to be replaced by, interfaced with, or reused by, new systems. The technique focuses on current data structures and procedures and the support provided by existing systems to business processes.

Current systems analysis focuses on entity types and business processes. It is typically performed separately, but in parallel with, data and activity analysis. However, in a small development project, it may be useful to analyze a current system together with data and activity analysis. This provides continuous confirmation of the business model and development project decisions to replace or reuse current procedures and data stores.

In a large development project, a survey of current systems may be performed early in analysis. However, where it is planned to reduce the initially defined project scope to support a more focused project, it may be more practical to postpone a more detailed analysis until the scope of the development project has been refined.

Where it has been identified that a number of current systems collectively support several business areas, for instance, during planning, then current systems analysis may be assigned to the highest priority project, or may even be planned as a separate project which is then coordinated with all the affected system development projects.

In all of these scenarios, the objectives of performing current systems analysis before completing analysis, are to:

- Validate understanding.

  Analyzing current systems helps completeness checking during modeling, and in model confirmation at the end of Analysis. This verifies scope and identifies any omissions.

- Plan or confirm the plan for transition from existing to new systems.

  This may involve planning to replace all or part of the current systems, or interworking with them. Especially where data is to be reused, it may involve evaluation of the accuracy and consistency of data in current data stores.

- Prepare for initial data conversion and subsequent interfacing between current and new systems.

  Identifying differences between existing data structures, system use of data, and activity dependencies helps in developing strategies to resolve conflicts that may occur when data from existing systems is converted or is used by new systems.

Current systems analysis techniques produce a complete representation of data, processes and their interactions by developing one or more current system business models.

Systems that represent a compatible view of the business may be combined into a single model. This model can then be used as the basis for completeness checking, transition planning, and preparation for conversion and interfacing procedures.

If data in existing data stores is to be reused (such data is therefore sometimes called "legacy data"), then the model may become part of a new system, which may exploit client/server techniques for instance, to make current data available through server procedures to new client procedures.

The techniques employed are those of reverse engineering. These techniques can also be applied to re-engineering a system to change the technology used, or applied to analyzing, selecting, and planning the introduction of a packaged application.

Selecting Systems to Analyze

Two matrices help you identify existing systems that support processes in the analysis model:

- **Current Information System/Business Area Matrix**—From this matrix, you select all systems that support processes in the defined business area. For example, during analysis of the Purchasing business area, the systems that should be selected are: Sales Forecasting, Purchasing, and Materials Requirements.

Current Info System / Business Area	PRODUCT MANAGEMENT	PURCHASING	WAREHOUSING	ACCOUNTING	SALES	MARKETING	DISTRIBUTION	PERSONNEL
INVENTORY CONTROL	X		X		X		X	
SALES ORDER PROCESSING				X	X			
PRODUCT SCHEULING	X							
SALES FORECASTING	X	X		X	X	X		
PURCHASING		X	X	X	X	X		
PAYROLL				X				X
MATERIALS REQUIREMENT PLNG	X	X			X			

- **Current Information System/Current Data Store Matrix**—From this matrix, you determine the data stores to be analyzed in the defined business area. These are the data stores used by the three selected systems: all except the Employee data store.

Cell Values: = Not Referenced C = Create D = Delete U = Update R = Read	**Current Data Store**	CUSTOMER MASTER	PRODUCT MASTER	INVENTORY	SUPPLIER MASTER	PURCHASING	SALES	EMPLOYEE
Current Info System								
INVENTORY CONTROL			R	C	R			
SALES ORDER PROCESSING		C		R			C	
PRODUCT SCHEULING			R	R		R		
SALES FORECASTING		R	R				R	
PURCHASING			C	R	C	C	R	
PAYROLL								R
MATERIALS REQUIREMENT PLNG			R	R	R		R	

These matrices may have been produced as part of an Information Strategy Plan or during initial project scoping. These matrices are supported in the planning and analysis toolsets.

You also review the scope and intention of the project with regard to current systems and data stores.

Current systems analysis confirms or qualifies these intended project outcomes:

- **Retire a current system completely**—Current systems analysis must produce an understanding of the functionality currently provided by the system.
- **Modify or retire a current system in part**—Here, current systems analysis is used to examine the parts of the system in sufficient detail to identify implications of this decision.
- **Interface with a current system, reading only, or also updating**—If there is an interface file or suitable input transaction file, this needs to be analyzed.
- **Use a current data store, reading only, or also updating**—The data store needs to be analyzed in detail.

You should probably examine the following existing systems:

- Current and planned systems that interact with data stores relevant to the business area, whether they are batch or online
- Systems that produce relevant specified output from these data stores for any purpose except occasional inquiry
- If the data store is not to be replaced, only those output processes that support the business area need be examined
- Application packages that access these data stores
- Manual procedures that operate on essential data stores independently, or are otherwise associated with the above systems
- When analyzing non-computerized systems or associated manual procedures, you should examine only procedures significant to the business. These do not include routine clerical tasks, unless they affect the content of data files or the form of information presented to or collected from end users.

The level of detail of the following analysis depends on whether the intention is to interface, access, enhance, or replace current systems and data stores completely or in part. It also depends on whether the original system development was done using CA Gen. For the systems and data stores to be analyzed, assemble and catalog system documentation, and assess the quality of this existing documentation and identify any further preparation actions.

- For a current system developed using CA Gen, obtain and analyze the appropriate model. Model objects from it can be merged or adopted as needed.
- For a system that has been analyzed already for another development project, obtain the analysis and check it for relevance and completeness.
- For a system with poor documentation, some documentation improvement may be necessary before analysis begins.

Collecting Information About Current Systems

Information about current systems is needed for reviewing their usefulness and performance, and for more detailed current systems analysis. The information should be available from those who currently maintain the system, and from those who use it and verified it as complete and up-to-date.

The information needed includes:

- System and data structure, which covers the procedures supported and types of data that are stored, available as definitions and data flow diagrams within user and operations manuals.
- Support and maintenance history, and outstanding enhancement requests.

Surveying Current Systems Performance

Often, it is useful to review how well current systems satisfy business needs, assessing those systems that are part of the project scope.

Inevitably, users will comment on these systems during facilitated sessions or interviews. Conducting a more formal analysis to help decide:

- Whether to replace all or part of a system and its data
- Whether to reuse all or part of a system and its data
- What aspects of system performance and data quality are important to the business, for example, is accurate and complete data available when needed?

When the current systems environment has been assessed as part of an Information Strategy Plan, that assessment may be reviewed and reused where it is still valid.

A survey can be conducted using the following steps:

Step 1 Define the Assessment

The following tables show an example set of criteria using a survey form.

System Assessment Criterion	Assessment of:	Score
Functionality	To what extent does the system meet the business requirements?	
Ease of use	How easy is the system to use for the end user community?	
Response	Does the system perform the required actions when needed?	
Availability	Can the system be used when and where the users need it?	
Ease of Operation	How easy is the system to manage by user administrators and IT practitioners?	
Ease of Support	How well do the system and any supporting organization provide Help information and advice to the end user community?	

System Assessment Criterion	Assessment of:	Score
Maintainability	How easy is the system to maintain and enhance when the business needs it?	

Data Criterion	Assessment of:	Score
Completeness	Is all the data needed by the business available?	
Accuracy	Does the data reflect the facts?	
Timeliness	Is the data as up to date as all users need it to be?	
Consistency	Are the data values as consistent within a system or between systems as the business needs them to be?	
Presentation	Can the data be laid out in a way and in a medium that all users find easy to use?	

Reusability Criterion	Assessment of:	Score
	Can any of the data and its supporting procedures be used to support other business activities? What improvements are needed to make reuse worthwhile?	
Conclusion		Overall Score

Scoring Key
4 = Excellent - No improvement needed
3 = Good - Minor improvement is possible
2 = Inadequate - Significant investment should be considered
1 = Poor - Replacement should be considered

The criteria are selected to support the objectives of the assessment. This always includes meeting business requirements in some form and the accuracy and consistency of data. Other criteria depend on whether, for example, ease of support is an issue. Decide whether to assess data separately from other aspects of the system.

Step 2 Select Survey Participants

Select a representative sample of the users of the system. This always includes the management and supervisors of those who use the system, as well as those who maintain and use data and use the system procedures. Other participants may be selected if, for example, maintaining or supporting the system is likely to be an issue.

Step 3 Gather System Assessment

The survey is circulated to users and the results are collected. Respondents are asked to list system problems and to score each of the criteria according to a scheme such as the one used in the following illustration.

Criterion	Assessment of: Purchasing System	Score (avg)
System Criteria		
Functionality	Should be able to select alternative suppliers by product cost. Poor linking to Delivery data prevents effective assessment of supplier delivery performance.	3
Ease of use	Purchase Order confirmation dialog sequence is not intuitive.	1
Response	Purchase Order authorization is often delayed.	2
Availability	It is not possible to enter new purchase orders while delivery notes are being printed.	4
Ease of operation	There are no problems.	3
Ease of Support	Documentation is out of date.	3
Maintainability	This system has been much amended, and further amendment incurs unwanted side effects.	3
Data Criteria		

Criterion	Assessment of: Purchasing System	Score (avg)
Completeness	More information on Supplier Terms would be useful.	3
Accuracy	Product structures are recorded as a hierarchy only.	3
Timeliness	New products are added very late.	2
Consistency	Common product components are defined differently in cost and manufacturing files for product. See accuracy problem.	4
Presentation	It should be possible to sort purchase order items by delivery date.	2
Reusability	Supplier and Supply Contract are candidates for non-manufacturing purchasing, but only after data problems have been resolved.	
Conclusion		
*Overall Score		3

*See the scoring key in the previous table.

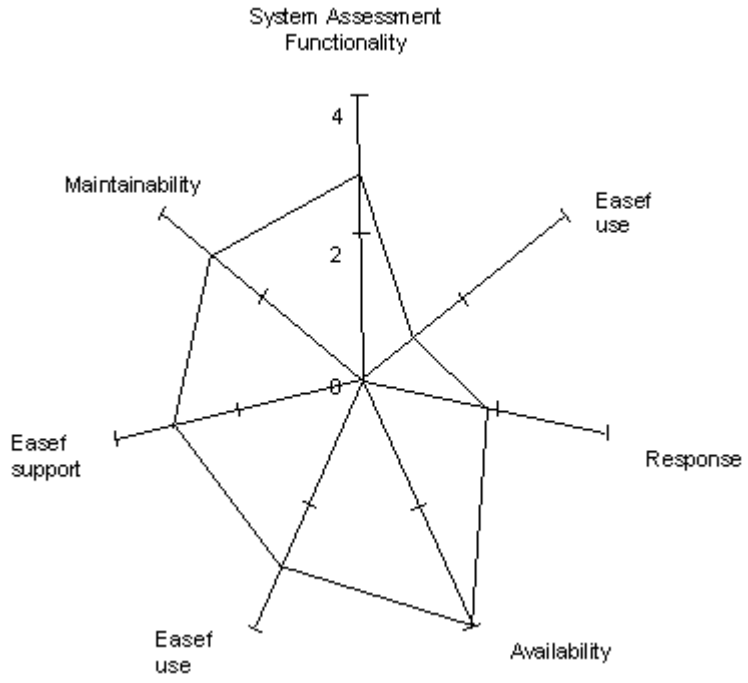
Step 4 Summarize System Assessments

You produce a single list of problems associated with each system based on all the comments received from the survey and from other information gathering activities.

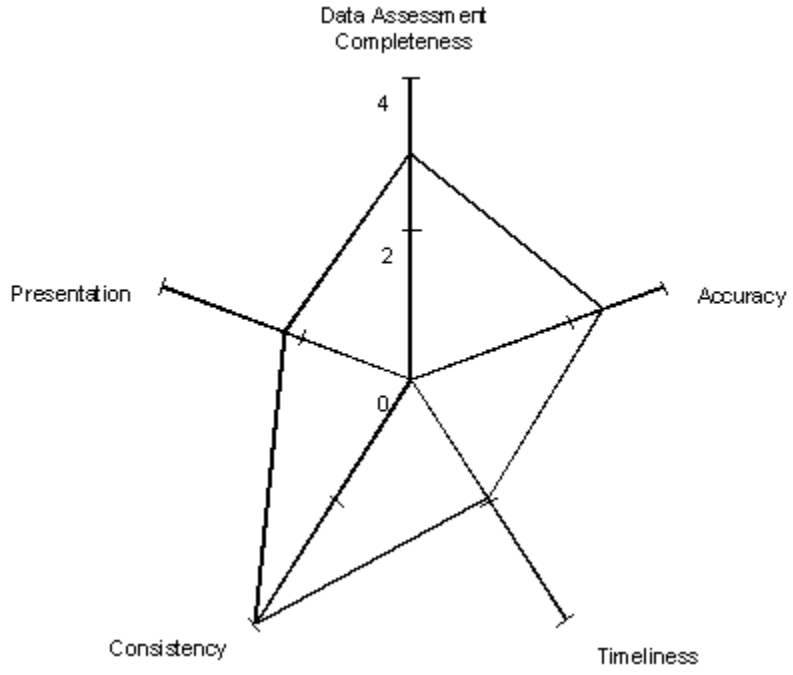
Associate the problems, if possible, with each of the selected assessment criteria.

The average of all responses can be conveniently summarized as a chart. Low scores indicate problems that should be addressed in future solutions. The following illustrations are presented as radar, or footprint diagrams, for a system that adequately meets user requirements, but whose interface could be improved, as suggested by the low scores for "system ease of use" and "data presentation."

The following shows an assessment of a Current System.



The following shows an assessment of Current Data.



Analyzing Current System Procedures

During current systems procedure analysis, you build a definition of current procedures and examine their structure and dependencies. You also document data flow between procedures.

Procedure analysis within current systems analysis resembles process decomposition and dependency analysis. See the chapter "Analyzing Activities" for more information.

You create a Procedure Hierarchy and one or more Data Flow Diagrams. These diagrams serve to check the completeness of the analysis model. A subset of the current Business Systems Architecture model or a high-level Data Flow Diagram serves as a useful starting point for understanding the context of the current systems to be analyzed, among other systems.

The Procedure Hierarchy and Data Flow Diagrams are prepared manually or by using some automated means other than CA Gen.

Note: If a CA Gen model is used, for instance to build a detailed model of the interactions between procedures and the implied data model, this model must never be confused with a model that analyzes business requirements or defines a system to be developed using CA Gen.

Basic Concepts for Procedure Analysis

Some key concepts related to procedures and data flows are included here, as well as an introduction to the concept of a data view.

- Procedure is a method of carrying out one or more elementary processes. In current systems analysis, the word "procedure" includes both computer programs and manual activities.
- In procedure analysis, you include only meaningful collections of procedures or processes; these are programs referenced in procedure manuals and manual activities that users perceive as meaningful units. It should not be necessary to describe a system below the level of procedures that execute discretely, or to include error correction procedures.
- Data Store is a repository of data of which users are aware and from which data can be read repeatedly and non-destructively. A data store can be permanent or temporary, depending on the system.
- In current systems procedure analysis, you identify the files, databases, and clerical stores that the selected procedures use or update. Although data stores are sometimes temporary, you should not include transitory files, such as sort files, in the list of data stores. Current systems analysis techniques apply to business data repositories, which are longer-lived files.

- Data View and Layout
  - A data view is an organized collection of fields that is meaningful to a procedure, business system, or organizational unit.
  - A person or program normally requires only a subset of the data in a system. This subset, called a data view, is a limited view of the information.
  - Data views appear within layouts, for example, within screens, files, manual and computerized forms, and reports. A layout is therefore a grouping of fields used to present data to a module or user.
  - A layout can contain more than one data view. For example, a report may contain data about several things. A data view is the part of a layout required for a particular procedure.
- Data Flow is a requirement for a data view to pass between two designed elements, each being a business system, procedure, data store, or external object. In other words, a data flow represents the passing of a data view between two procedures or between a procedure and a data store. It represents how a dependency has been implemented.
  - Expected Effects are the high level definition of the effect (Create, Read, Update, Delete, or "CRUD") that an activity can have on the entities in the implied data model.

Using a Procedure Hierarchy

The Procedure Hierarchy is a type of indented list in which you document the hierarchy of current procedures, as observed or described in documentation.

The example hierarchy in the following illustration lists the procedures performed in part of a current purchasing system.

Procedure Hierarchy
PURCHASING SYSTEM
DEMAND ASSESSMENT
CALCULATE SALES REQUIREMENT
DETERMINE INVENTORY LEVEL
DETERMINE ECONOMIC ORDER QUANTITY
PLACE PURCHASE ORDER
SELECT PRODUCT
SELECT SUPPLIER
REGISTER PURCHASE ORDER
GOODS RECEIPT
CHECK IN DELIVERY
RETURN DELIVERY ERRORS
STORE GOODS
CALCULATE NEW INVENTORY
PURCHASES PAYMENT
REGISTER SUPPLIER INVOICE
PAY SUPPLIER
CLOSE PURCHASE ORDER

Procedure Hierarchy

Guidelines for procedure hierarchies:

- List one procedure per line.
- Use equal indentation for procedures of the same rank.
- List activities of interest to the business, therefore omitting batch control and list procedures, and online menus.
- List job steps within a batch procedure.
- List transactions within an online (sub) system based on the menu.
- Follow the sequence of procedure manuals and other documentation.
- Use the same names as current documentation. If these names are not meaningful, use names of significance to the business to help comparison with the business model and record system identifiers in the descriptions. If a name is duplicated, qualify each occurrence to indicate the current system context.

Using a Data Flow Diagram

By showing the flow of data between data stores and procedures on the Procedure Hierarchy, the Data Flow Diagram highlights data interactions between procedures and the sequence of procedure execution.

Data Flow Diagrams represent successive levels of detail for the current system.

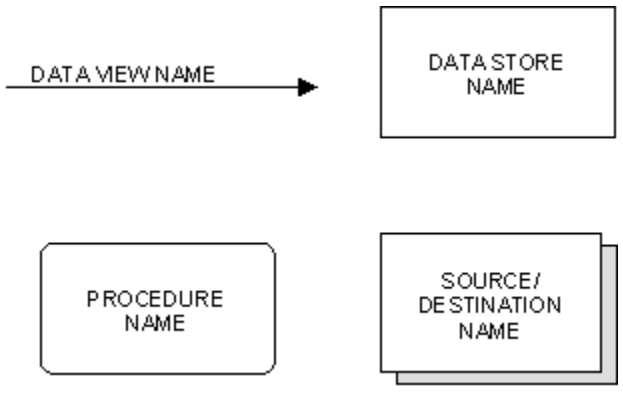
The flows shown on a Data Flow Diagram are listed in the following table.

From	To
Procedure	Data store
Data store	Procedure
Procedure	Procedure
External object or System	Procedure
Procedure	External Object or System

Data flow diagramming may be useful for complex current systems, for example, where the sequence of a number of procedures, or the interaction of batch and online procedures, is important. In this case, it may be useful to annotate the diagram with the mechanism of each procedure: batch, online, or manual. There is no set notation for recording this property.

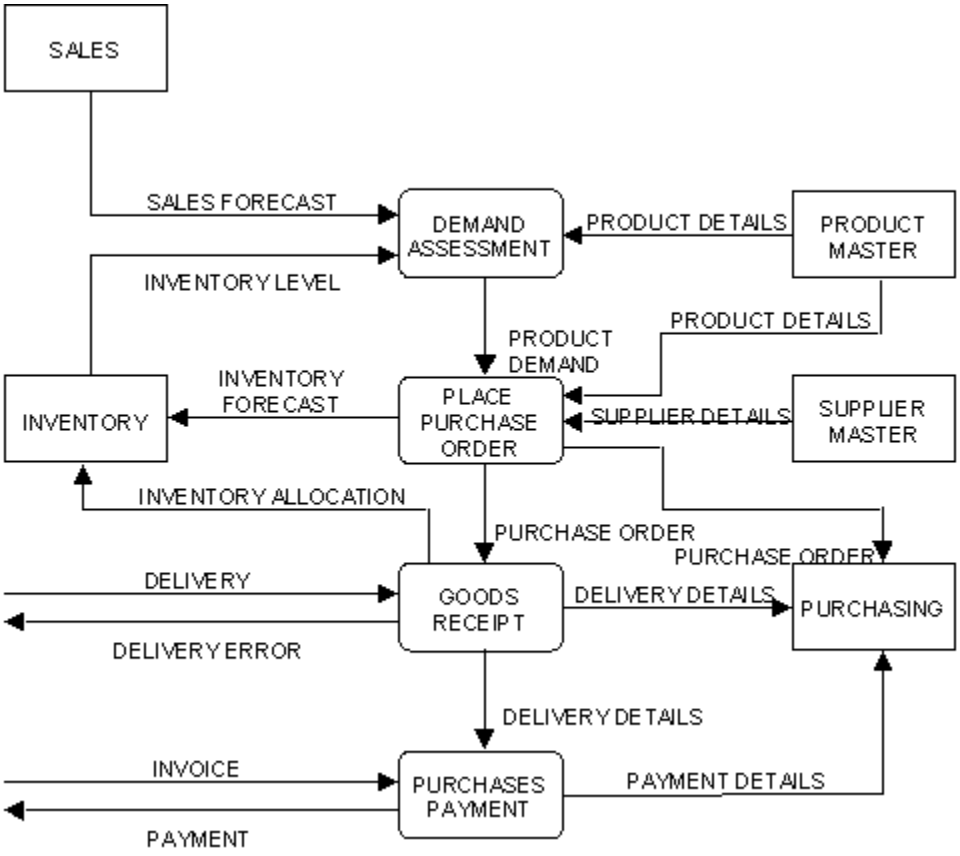
Data Flow Diagram Conventions

The following illustration shows the conventions of Data Flow Diagrams.



Arrows on the lines represent flows into and from data stores, as well as flows between activities. The direction of the arrows indicates the direction of the flow. For instance, the downward direction of the arrow head indicates the flow from Demand Assessment to Place Purchase Order.

Consider the top level Data Flow Diagram in the following illustration.



This Data Flow Diagram includes four subsystems:

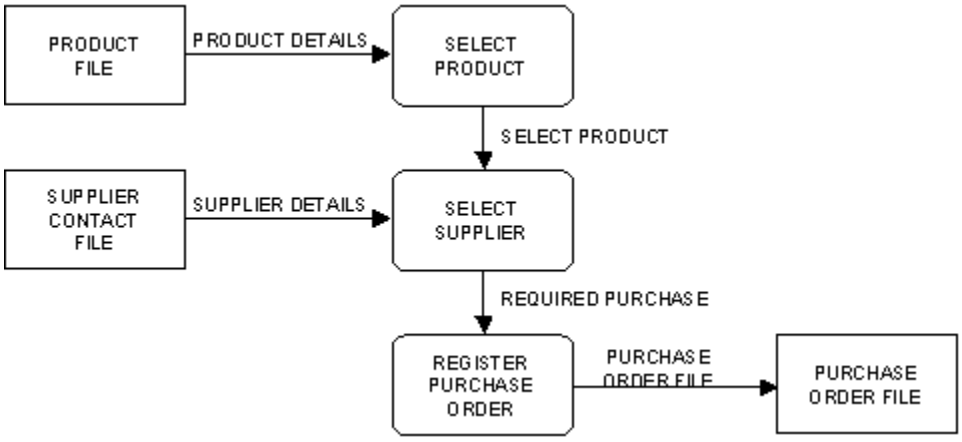
- Demand Assessment
- Place Purchase Order
- Goods Receipt
- Purchases Payment

The diagram also includes five principal data stores:

- Sales
- Inventory
- Product Master

- Supplier Master
- Purchasing

The following illustration shows more detail of the Place Purchase Order system.



The high-level data stores in the previous top level illustration now appear in this illustration as three, more detailed data stores.

Developing the Data Flow Diagram

As a starting point, there may already be a Business Systems Architecture Diagram produced during planning, or a Data Flow Diagram for the system showing its procedures.

A complete architecture for business systems should include a diagram for the current systems as a basis for further system development to meet business requirements.

Where systems are being retained (sometimes called legacy systems), current systems also appear on later versions of the Business Systems Architecture. Diagrams for this architecture may not be very detailed, but should provide a basis for drawing a high-level Data Flow Diagram showing interactions of all the current systems to be analyzed.

Complex systems may require more than one level of Data Flow Diagram. If so, develop a series of these diagrams to include each appropriate level of detail. These diagrams might comprise: a single high-level context diagram showing each relevant system; a diagram for each system, showing subsystems or collections of procedures that are performed together; and a diagram for each subsystem showing how the procedures interact.

Guidelines for developing data flow diagrams:

- Keep the lower level diagrams consistent with the higher level ones. For example, a lower level Data Flow Diagram cannot contain more inputs and outputs than a higher level diagram. A data store that appears on a higher level data flow diagram must also appear on each lower level diagram.
- Label procedures with existing names.
- Label data flows with existing transaction file names where appropriate, or names of subject areas and entity types.
- If necessary, annotate each procedure as manual, batch, or online.
- Do not relate procedures to specific tools.
- Do not include procedures that are system maintenance activities, such as Log Transaction or Reorganize Database.
- Represent only the normal, regular state, not start-up procedures.
- Ignore trivial error paths.
- Retain enough detail to allow planners to specify how the transition from current systems to new systems will be executed.

Analyzing Current Data

The main purpose of current data analysis is to model the data used by one or more current systems. Data is represented in the current system both in a structured form, as in current data stores, and as data in user views, forms, reports and displays.

You begin with data stores, then move to these other views of data. The result is an implied data model that may describe one or more data stores, or the total view of data supported by one or more current systems. There may be a single current system model, but if systems are known or found to be incompatible, there may be more than one.

The models are assembled by asking a series of simple questions about each data item to ensure that it is dependent on the appropriate identifier or key. The questions are those used in the normalization technique described in the chapter "Analyzing Data."

The current system model is built up progressively. This technique is sometimes called canonical synthesis because it proceeds by progressively adding together two or more models of the data.

The models may be represented graphically or as a list. The method suggested here is to use CA Gen to develop a separate model of the data, using the diagram or entity list as convenient.

If the model is to be used to generate procedures that access current data, or procedures for data conversion or bridging, then in addition to the synthesized (implied) data model, a data store analysis model is retained.

Note: You must be careful not to remove redundant record types, identifiers, and fields from such a model.

Analyzing Data Stores

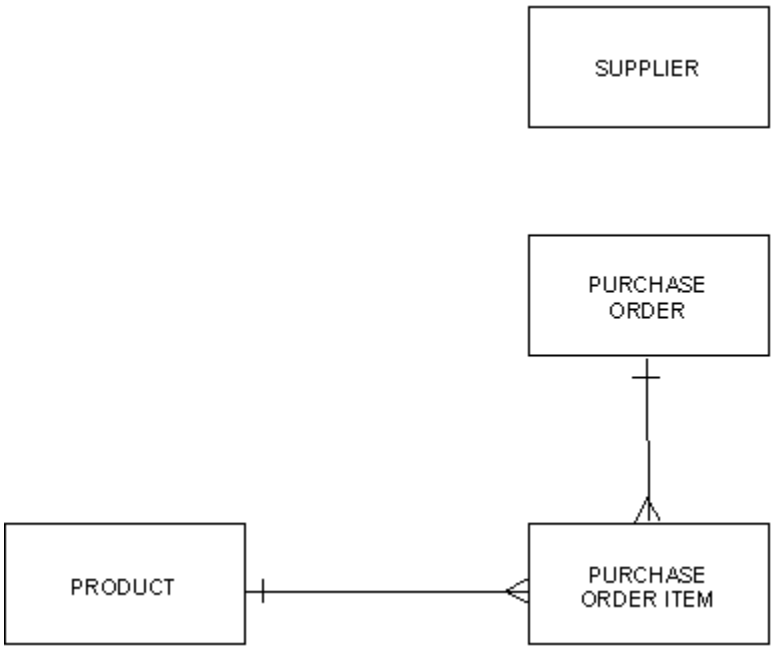
As a repository of data used by a current information system, a data store can be permanent or temporary, depending on the system.

In current systems analysis, you identify the data structures along with their parts and associations that fall within the defined scope by examining business data repositories, longer-lived files, and possibly also files that interface or communicate between systems. You should not examine transitory files, such as sort files, in the list of data stores.

The content structure of a data store can vary from the very simple "flat file" to complex databases. Start by noting record types, identifiers, and other non-key fields. The structure of the Purchasing data store is shown in the following table.

Record Type	Identifier	Non-Key Field
Purchase Order	Purchase Order Number	Date Raised By Delivery Notes Total Sub Total Sales Tax
Purchase Order Item	Purchase Order Number Product - Code	Quantity Total Price
Product	Product - Code	Description Item Price
Supplier	Supplier - Number	Name Address

The following illustration shows an Entity Relationship Diagram based on the Purchasing data store structure.



Although the entity type SUPPLIER has been identified, the analysis of the data store may not provide enough information to deduce all the relationships needed to compose a fully integrated Entity Relationship Diagram.

Analyzing User Views

A person or program normally requires only a subset of the data in a system. Those data views of interest to users are of particular significance to analysts as evidence of what the system is currently intended to handle. By identifying these views, you can begin to understand which types of data are likely to be relevant to particular procedures.

A user view is a special case of data view. It is a collection of associated fields of interest to a user during procedure execution.

During user view analysis, you collect a full representation of data used in current systems and the associations between the data. When studying user views, it is important to examine a selection of examples that have been completed.

User views appear within layouts such as screens, manual files, manual and computerized forms, and reports.

A layout can contain more than one data view. For example, a form may contain data about several things. The example in the following illustration contains data about purchase order, delivery, product and supplier, among others.

PURCHASE ORDER NUMBER A00854

SUPPLIER NO. 68732 SUPPLIER ADDRESS: ABC CORPORATION LANSDOWN BOULEVARD NEWTON	RAISED BY: L. HODGES DATE: SEPTEMBER 5, 1999	DELIVERY NOTES: DO NOT DELIVER BEFORE: SEPTEMBER 26, 1999

PRODUCT CODE:	PRODUCT DESCRIPTION	QUANTITY	ITEM PRICE		TOTAL PRICE	
PD12345	25MM CARBIDE DRILL BIT	10	07	99	79	90
PD47653	36MM GALVANIZED SCREW	24	00	12	02	88

AUTHORIZED BY: ----------

SUBTOTAL	82	78
SALES TAX @ 17.5%	14	49
TOTAL	97	27

Different procedures may be concerned with different parts of each user view. User view analysis provides a secondary source of information for modeling current system data, and as such, it acts as a confirmation for data and their associations gained from the analysis of current information system data stores.

In the illustration, Supplier is indeed shown to be relevant to the Purchasing data store. You can also identify its relationship with the other entity types previously found.

Developing an Implied Data Model

You develop a model from the record types, keys, and fields of each data store and data view. The conventions used to model this data are exactly the same as those used during data analysis.

Guidelines for developing the data model:

- Include each key group as an entity type.

  A key group is a key and a set of fields that are dependent on that key. For example, in the Purchasing data store structure, Supplier Name and Address depend on Supplier Number, so they become attributes of entity type SUPPLIER, with Supplier Number as the identifier. Within a concatenated key, each key component defines a separate key group. Each of the separate key groups will be related to the key group containing the concatenation.

- Define relationships between all key groups.

  PURCHASE ORDER has a one-to-many relationship with PURCHASE ORDER ITEM. A relationship cannot always be defined with certainty. Supplier, for example, cannot be associated with any other entity type based solely on its key.

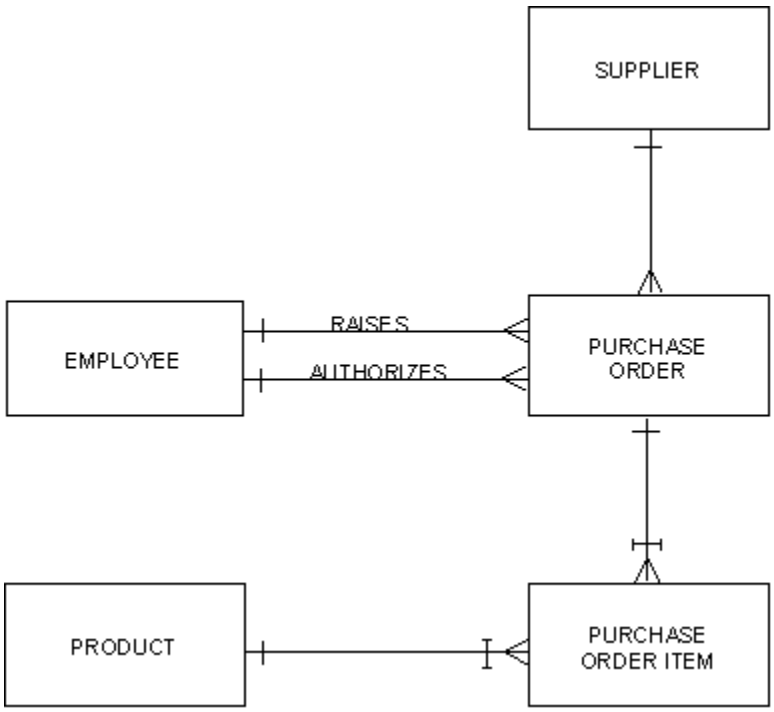
- Include every relationship.

  The implied data model includes many relationships, some of which may be redundant. See the chapter "Analyzing Data" for information on redundancy. During comparison checking, determine which relationships to keep.

- Resolve each many-to-many relationship for which additional attributes are needed about the relationship (for an implied data model only).

  In each of these cases, create an associative entity type. Use the identifying attributes of the entity types involved as a composite identifier, and then add the necessary relationships and attributes.

In the Purchasing data store structure, a purchase order may contain many PRODUCTs. Likewise, a PRODUCT may appear on many purchase orders. This is resolved by creating an associative entity type PURCHASE ORDER ITEM, which has for its key the two identifiers PRODUCT and PURCHASE ORDER as well as the attribute Quantity. This is shown in the following illustration.



- Do not represent the optionality of relationships.

  In the example, it was inferred that a product is ordered by means of many purchase order items. Optionality allows for an instance when a particular product is never purchased perhaps because it is obsolete and never appears on a purchase order item.

- Remove duplicate attributes that are not identifiers.

  Perform this for an implied data model only. To remove duplicates without losing information, you may need to create relationship memberships between the entity type of the duplicate attribute and an entity type currently identified by or containing the attribute.

  First check whether the required memberships already exist directly or indirectly. The Purchasing data store in the example contains in Purchase Order fields for Supplier Name and Supplier Address, which appear also in the Supplier record type and can therefore be removed.

Analyzing Current System Interactions

Once current system procedure analysis is completed, and a model for current systems is available, current systems analysis can be refined further by recording interactions in more detail. This helps to identify the business impact of changes in data structures that occur as a result of implementing new information systems. This additional detail is especially needed if a system is to be replaced in part.

The expected effect of a current system on an entity within a data store can be summarized using two matrices:

- A new or amended Current Information System/Current Data Store Matrix

Cell Values: = Not Referenced C = Create D = Delete U = Update R = Read	**Current Data Store**	CUSTOMER MASTER	PRODUCT MASTER	INVENTORY	SUPPLIER MASTER	PURCHASING	SALES	EMPLOYEE	PRODUCT TRAINING	SALES FORECAST
Current Info System										
INVENTORY CONTROL			R	C	R					
SALES ORDER PROCESSING		C		R			C			
PRODUCT SCHEULING			R	R		R				C
SALES FORECASTING		R	R				R		C	
PURCHASING			C	R	C	C	R			
PAYROLL								R		
MATERIALS REQUIREMENT PLNG			R	R	R		R			

- In the analysis model, an Entity Type/Current Data Store Matrix

Process	Current Data Store	CUSTOMER MASTER	PRODUCT MASTER	INVENTORY	SUPPLIER MASTER	PURCHASING	SALES	EMPLOYEE
CUSTOMER		X						
PRODUCT			X					
WAREHOUSE				X				
STOCK ITEM				X				
SUPPLIER					X			
PURCHASE ORDER						X		
PURCHASE ORDER ITEM						X		
SALES ORDER							X	
SALES ORDER ITEM							X	
EMPLOYEE						X	X	X

If the relevant facts are included in a CA Gen model, use the Matrix Tool.

This analysis identifies relevant entity types, their existence in current data stores, and their maintenance by the existing systems. It may also become apparent that some current data stores contain duplicate data. This detail is used in confirming the analysis model, for investigating current data stores that are already redundant, or that will become so. It is also used in refining the plan for Transition from existing to new systems. This plan may involve evaluating the accuracy and consistency of data in current data stores, based on a comparison of the current system model and the analysis model.

Summary of Current Systems Analysis Guidelines

This chapter has described a number of guidelines for developing current system models, some explicitly and others implicitly. This section summarizes those rules.

Because these guidelines are performed manually, they cannot be automatically enforced by CA Gen. Where some part of a current system model is represented using a CA Gen model, CA Gen rules may be over rigorous when the model is not intended to be used in generating parts of a new system.

Guidelines for Current System Activities

- Each activity must be uniquely named.
- Each leaf of the hierarchy must be an elementary process.
- The subordinates of an activity must completely describe business support provided by the activity. They should not include batch control and list procedures, and online menus.
- Each lowest level activity should be a manual procedure, a job step within a batch procedure, or an online transaction.

Guidelines for Data Flow Diagrams

- The subject and activity parts of a Data Flow Diagram should appear in the procedure hierarchy of a current system activity model.
- The subject of a Current System Data Flow Diagram may be:
  - Collection of current systems relevant to a business area
  - Single current system relevant to a business area
  - Subsystem of a current system
- The activities depicted in a Current System Data Flow Diagram may be:
  - Current system
  - Subsystem of a current system
  - Current system procedure that supports an activity of interest to the business
- A data flow may be from:
  - Procedure to Data Store
  - Data Store to Procedure
  - Procedure to Procedure
  - External Object or System to Procedure
  - Procedure to External Object or System
- Lower-level diagrams should be consistent with the higher level ones.
  - A lower-level Data Flow Diagram cannot contain more inputs and outputs than a higher level diagram.
  - A data store that appears on a higher level must also appear on each lower level diagram.

Guidelines for Current Data Stores

A data store should be capable of being read repeatedly and therefore should be a permanent data store or an interface file.

Guidelines for Implied Entity Types

- An implied entity type corresponds to a key and a set of fields that are dependent on that key and nothing else.
- Define relationships between all key groups.
- Purchase Order has a one-to-many relationship with Purchase Order Item, but a relationship cannot always be defined with certainty. Supplier, for example, cannot be associated with any other entity type based solely on its key.
- Include every relationship.

The implied data model includes many relationships, some of which may be redundant. See the chapter "Building the Analysis Model" for a discussion of redundancy.

During comparison checking, determine which relationships to keep.

- Many-to-many relationships that have attributes are resolved by using associative entity types.
- An associative entity type must have for its key the identifiers of all implied entity types that it associates.
- Optionality of relationships is undefined.
- Duplicate attributes must be identifiers.

Results of Current Systems Analysis

The results of current systems analysis are:

- Current Systems Assessment Summary
- Current System Activity Model containing:
  - Hierarchy of procedures to the level of transactions and batch job steps
  - Data Flow Diagram showing data stores and the interactions of current systems
  - (Optional) Data Flow Diagrams for individual current systems or subsystems showing the interaction of procedures

- (Optional) one or more Current System Data Store Models containing definitions of implied entity types or current record types, if this model is needed to generate conversion or bridging procedures
- (Optional) Implied Data Model, if current system data analysis is conducted separately for later comparison with the analysis model=

Chapter 8: Building the Analysis Model

There are a variety of ways in which the data, activity, and interaction analysis techniques can be combined to develop a detailed understanding of the problem space at the conceptual level. Still, experience has shown that some ways of employing analysis techniques yield better results than others.

This chapter provides guidance in the practical use of those techniques within the context of a project. In particular, it offers a more detailed description of what practitioners regard as the most consistently successful approach to building an analysis model: parallel decomposition.

Parallel decomposition helps to ensure that all aspects of the model are complementary, that data exists to support the activities, and that the model represents all activities needed to support the business objectives for the development project.

Principles of Parallel Decomposition

In analysis circles, it is common for practitioners to describe themselves as being either "data oriented" or "process oriented."

Data-oriented analysts typically begin an analysis project by building an entity-relationship diagram. After most of the data modeling work is done, they begin to identify the set of processes needed to manipulate the data based on the way they have modeled it. Some proponents of such data-oriented approaches even go so far as to say that the process model "drops out" of the data model.

Process-oriented analysts might begin an analysis project by creating a detailed functional decomposition accompanied by a thick sheaf of dependency diagrams. Once the details of the process interactions are well understood, they might model the data required to enable those processes.

Both schools of thought have their adherents, but in practice it turns out that each is lacking. Experience has shown that those who concentrate solely on defining data will miss some important processes and thereby, some important data. Likewise, those who focus on processes alone are likely to forget some critical element of data and, by implication, the processes that use them.

One obvious way to overcome the limitations of these one-dimensional approaches is to interleave the analysis of data and activities so that they can be used to confirm one another. By carefully adding detail to both sides of the model simultaneously, you can produce a more accurate and stable view of the part of the business being modeled. In practice, models built in this way tend to require less rework, be more complete in the initial pass, and generally exhibit higher quality than models built using either a data or a process emphasis.

This approach to defining and refining data and activities at the same time is called parallel decomposition. Parallel decomposition results in the definition of a set of business object types that include entity types and the processes that affect them.

Decomposition was introduced in the chapter "Analyzing Activities." The topic was activity decomposition: the breaking of a function or process into sub-functions or processes to refine one's understanding of the activities undertaken by the business.

There is a corresponding concept in the realm of data. Although not treated explicitly in the chapter "Analyzing Data" for data analysis, subject areas can decompose into subordinate subject areas or entity types just as functions can decompose into subordinate functions or processes.

In each successive layer of decomposition, whether of activities or data, more detail is added to the model, and its representation of business reality becomes more precise.

Criticism of Parallel Decomposition

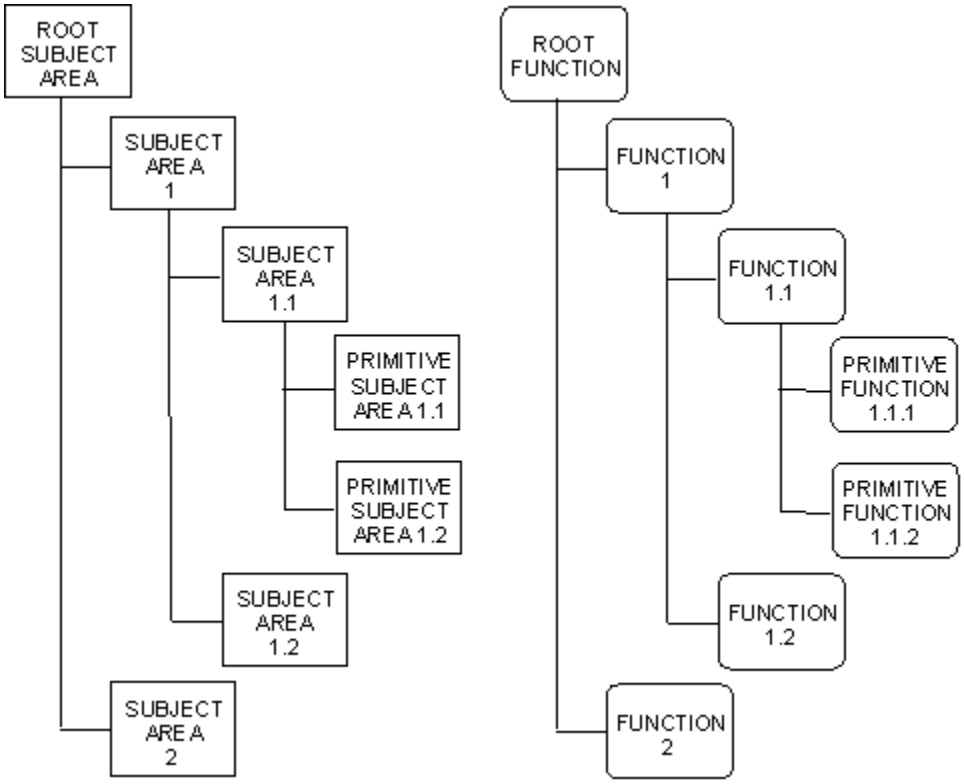
Before delving into the details of the parallel decomposition approach, it is important to address a common criticism of decomposition: decomposition, it is argued, is an inferior modeling practice because its results are not *repeatable*. That is, two analysts given the same subject are likely to decompose it in two different ways. As a result, decomposition is considered by its critics to be a slipshod, haphazard approach to subdividing a problem space.

The parallel decomposition approach successfully addresses this issue of non-repeatability. Cross-checking between data and activities, and following guidelines for subdividing by business object type classification and by business object type life-cycle, together with confirmation by event analysis, help to minimize the randomness generally associated with decomposition. Confirmation by event analysis is described in the chapter "Analyzing Activities."

Principle of Parallelism

Correspondence and Balance

The principle of parallelism is very straightforward: during parallel decomposition, there is to be a one-to-one correspondence between elements on the activity side of the model and those on the data side. This results in a set of corresponding isomorphic structures, like those shown in the following illustration.



The decomposition begins with two model objects:

- Root subject area
- Root function

These objects represent the broadest categories of data and activities that characterize the problem space under investigation. If you cannot identify a root function or subject area immediately, do not lose heart. The advice in the section en Getting Started with Parallel Decomposition will help to establish a starting point.

An example root function for a business area related to manufacturing might be Manufacturing.

An example of a root subject area for the same business area might be Manufacturing also.

This is not unusual. At the upper levels of decomposition it is often difficult to invent meaningful names that encompass all the concepts that must be dealt with without becoming long-winded, so the root function and root subject area will often share the same name. "Things To Do Associated With the Manufacturing of Products" and "Things of Use When Manufacturing Products" are truer reflections of the contents of the root function and subject area, respectively, but the simpler name "Manufacturing" sums everything up pretty nicely.

As detail is added to the model, then, a 1:1 correspondence is maintained between functions and subject areas.

There is another school of thought on this point. Some analysts prefer to relax this constraint and insist only on 1:1 correspondence between the first level of decomposition (the highest level functions/subject areas) and the last (primitive functions and subject areas). In other words, activity and data decomposition can be allowed to follow different paths in the intervening levels as long as they correspond exactly at the top and bottom of the decomposition. This approach, while potentially viable, is really a shortcut that can lead to less stable, less reliable structures than true parallel decomposition. As a result, a complete 1:1 correspondence is recommended over this less rigorous approach.

The mechanism for subdividing functions and subject areas into their elements is described in Performing Decomposition, but for now the important point to remember is this: each function discovered corresponds directly to exactly one subject area, and each subject area discovered corresponds to exactly one function.

The number of levels in the decomposition will depend on the complexity of the problem space. It is typical to find between three and five layers of function between the root function and subject area and the primitive function and primitive subject area of which they are composed. This is depicted in the previous illustration of partial parallel decomposition.

Primitive Subject Areas

A primitive subject area is a subject area that includes exactly one central entity type and its dependent entity types.

A primitive subject area cannot be decomposed into subject areas. The next level of decomposition will result in the discovery of fully normalized entity types.

A primitive subject area includes a central entity type and, usually, a set of dependent entity types that reflect some useful business concept.

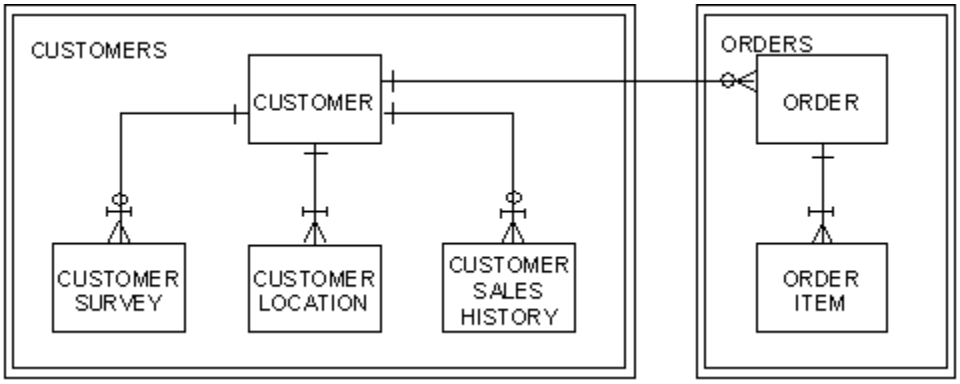
A central entity type is the focal point of a primitive subject area. An occurrence of a central entity plus the occurrences of its dependents constitute an occurrence of the primitive subject area in which it is contained.

A dependent entity type is an entity type whose occurrences have no meaning without the existence of an occurrence of a central entity type.

It makes sense to think of an occurrence of a primitive subject area.

Primitive subject areas define the data contents of a business object type.

For example, consider the following illustration.



In the illustration, customers is a primitive subject area that contains four entity types:

- CUSTOMER
- CUSTOMER SURVEY
- CUSTOMER LOCATION
- CUSTOMER SALES HISTORY

Looking at customers, it is plain that its main focus is on the entity type customer because all the other entity types within that subject area are dependent on it. In fact, they simply add detail to customer:

- CUSTOMER LOCATION
- CUSTOMER SALES HISTORY
- CUSTOMER SURVEY

These items are merely pieces of information about a customer. However, because the entity relationship model requires that attributes be fully normalized, these pieces of information must be separated into their own entity types. See the chapter "Analyzing Data" for a further discussion of normalization.

Since each customer can have multiple locations, the customer location information must appear as its own entity type connected by a 1:M relationship.

The same is true for customer sales history and customer survey.

The illustration shows that customer is the central entity type in the primitive subject area customers.

Customer survey, customer location, and customer sales history are dependent entity types.

It makes sense to think of an occurrence of the primitive subject area of the customer: it includes all information related to a single customer.

Primitive Functions

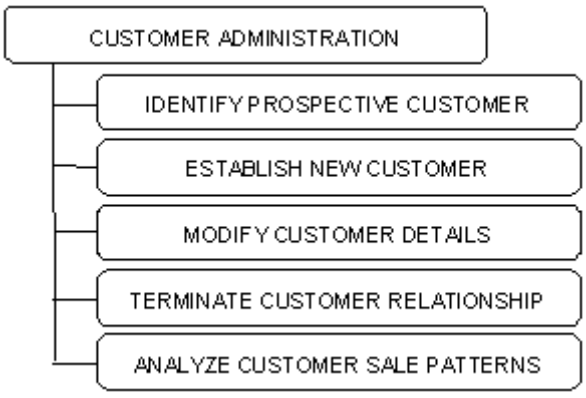
A primitive function is a function whose constituent processes are responsible for managing the lives of entities of a single central entity type.

Each primitive function corresponds to exactly one primitive subject area.

A primitive function may not itself be decomposed into functions. Instead, it is decomposed into processes.

The constituent processes of the primitive function are responsible for managing the occurrences of the central entity type of its corresponding primitive subject area as they move through their lives. See the chapter "Analyzing Interactions" for a discussion of entity type life-cycles.

For example, the following illustration shows a primitive function named Customer Administration that decomposes into several processes.

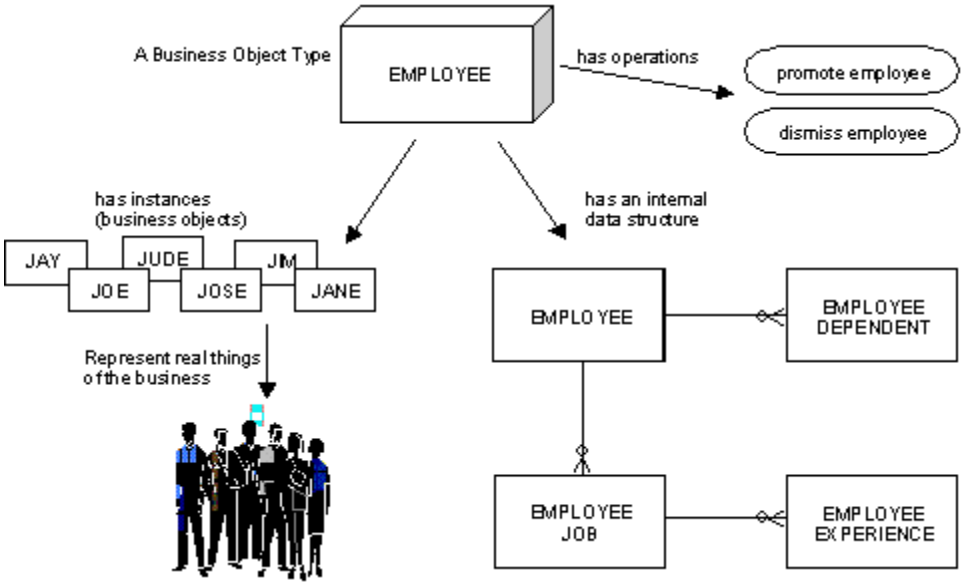


These processes clearly manage the information related to occurrences of customer. They will also, in the course of managing this information, manipulate occurrences of dependent entity types, but this is incidental to their main focus on the central entity type.

The processes into which a primitive function decomposes can be further decomposed into sub-processes and, eventually, into elementary processes.

Finally, a primitive function defines the activities of a business object type. A business object type is a representation of some type of thing a business needs to keep track of while running its business. This representation includes both data definitions and processing rules.

An occurrence of a business object type is a business object. See the following illustration.



In parallel decomposition terms, a primitive subject area provides the data representation of a business object type while a primitive function provides the processing representation.

Which Came First: Data or Activities?

When using parallel decomposition, it really does not matter whether you start with data or activities at any given level of decomposition. As long as you maintain the one-to-one correspondence between function and subject area and analyze both the activity and the data legs of the decomposition at the same level, the two aspects of the model will tend to confirm one another.

Extent of Parallel Decomposition

The one-to-one correspondence of parallel decomposition begins with the root function and subject area and continues until the primitive functions and subject areas are identified. However, this strict correspondence is discontinued at the next level of decomposition: the level in which processes and entity types appear.

Refining the activity model will continue with decomposition confirmed by dependency analysis until the elementary processes required to manipulate the entity types within the corresponding primitive subject area are defined.

Refinement of the data model continues through normalization of attributes and discovery of relationships within the primitive subject area.

However, there are still some guidelines that govern the relationship between the elements of primitive subject areas and primitive functions despite the absence of true parallelism. These guidelines are discussed in Parallel Decomposition Heuristics.

Getting Started With Parallel Decomposition

In some cases it can be difficult to tell exactly where to start with parallel decomposition. Project boundaries are rarely as tidy as analysts might wish. That is why, in Principles of Parallel Decomposition, the subject of parallel decomposition was described as a "problem space."

If you begin in the classically recommended manner, the results will be an Information Strategy Plan in which business areas are clearly defined. In such a case, each analysis project focuses on a single business area.

The reality is that most analysis projects do not begin with a pre-scoped, logically consistent business area definition. Since such projects can be any shape or size, a term like "problem space," which is sufficiently general to apply to any analysis effort, is necessary to define their scope.

There is often no question of where to begin the analysis. If the goal of the analysis project is to solve some immediate business need, the scope of the project is probably already well known. If the scope of analysis is some business area identified during planning, the analysis project scope should also be well understood.

Value Chain Analysis

Even when the project scope is predetermined, it is sometimes difficult to take the first step in decomposition. In such cases, value chain analysis can be used to help clarify the situation.

In his discussion of the value chain (Competitive Advantage, New York Free Press, 1985), Michael Porter proposes a taxonomy into which all business activities can be classified. This taxonomy, when converted into a functional decomposition, can form a starting point for virtually any project.

Porter proposes five primary functions:

- In-bound logistics (receiving, storing, materials handling)
- Operations (machining, packaging, assembling)
- Out-bound logistics (storing, distribution)
- Marketing and Sales (advertising, promotion, selling)
- Service (installation, repairs, parts supply)

Several support functions will exist to provide a foundation for the primary ones. They must be identified by the analyst, but probably include these proposed by Porter:

- Firm infrastructure
- Human resource management
- Technology development
- Procurement

Together, these functions can be used as a first-cut decomposition, as depicted in the following illustration.

You should feel free to use different names than those appearing in the illustration. Each business has its own nomenclature and every attempt should be made to conform to it.

When the customized, value-chain-oriented decomposition is complete, you can then identify where the project scope fits. Once that is understood, you can use the techniques described in Outlining Activities to continue the decomposition.

For example, assume that the analyst has been given the task of specifying an application that will track employee training. Based on the illustration, employee training is probably best considered a sub-function of Human Resource Management.

Alternatively, assume that a business re-engineering project has pointed out the need for an entirely new purchasing process. Purchasing is a sub-function of Procurement.

Outlining Activities

Activity outlining can be used to continue the analysis begun using value chain analysis.

An activity outline is simply an indented list that identifies natural groupings of activities, where the definition of "natural" is very subjective.

The outlining technique provides a less formal mechanism for organizing ideas, and so is suitable for use when "brainstorming" a list of candidate business activities.

The following list is an example outline of an Accounts Payable function.

- Payee Management
  - Establish Payee
  - Approve Payee
- Voucher Management
  - Request for Vouchers
  - Payment of Vouchers

Of course, an outline must be regarded as nothing more than a first cut at organizing ideas. After the outline is complete, it remains necessary to perform formal parallel decomposition. This can now be done with a better understanding of the problem space.

Performing Decomposition

Decomposition involves breaking down activities and data into their elements. The objective of this process is to form groups of activities and data that enjoy a high level of cohesion and a low level of coupling.

- Cohesion describes how closely related the elements within each level of decomposition are. For a function, cohesion is a measure of the relatedness of its sub-functions.
- Cohesive functions generally operate on a common set of data. For a subject area, cohesion is a measure of the relatedness of its subordinate entity types.
- Cohesive subject areas are generally operated on by a common set of functions.
- Coupling describes how closely related each element in the decomposition is with elements that are not siblings.
- Loosely coupled functions have few dependencies and loosely coupled subject areas have few connecting relationships.
- A function should have more dependencies with its siblings than with functions that are not siblings.
- A subject area should have more relationships with its siblings in the decomposition than with subject areas that are not siblings.

The techniques described in "Decomposing Activities" and "Decomposing Data" tend to create highly cohesive, loosely coupled groupings of activity and data at each level of decomposition. The result is a highly stable and consistent structure that enables the efficient assignment of resources, ready coordination of development efforts, and effective project management.

Decomposing Activities

Activity decomposition is recorded using an Activity Decomposition Diagram, described in the chapter "Analyzing Activities." In CA Gen, this diagram is drawn using the Activity Hierarchy Diagramming tool.

Activity Dependency Diagrams, drawn using the Activity Dependency Diagramming tool, may be used to confirm decomposition.

Life-Cycle Decomposition

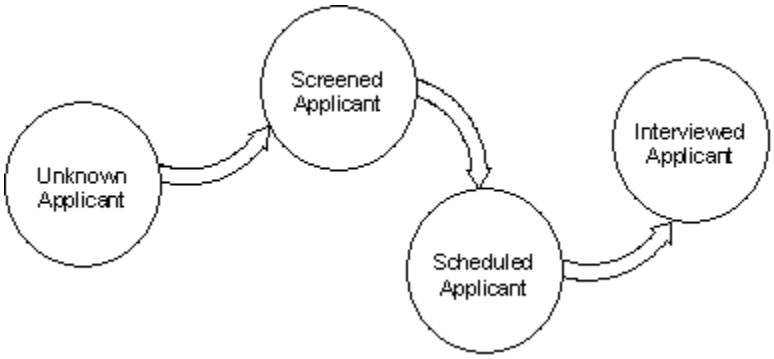
The most reliable way to decompose activities is based on the life-cycle of a proposed business object type. This involves deciding exactly what the business objects are.

At high levels of the decomposition, it is not always obvious what the business object types are going to be. For example, a high-level subject area named human resources would certainly include the data representation of a business object type (that is, a primitive subject area/primitive function combination) called employee. However, a few levels of decomposition might be required before it became clear that offer, training course, and benefits are also business object types within human resources. It might even be true that employee is eventually divided into several smaller business object types.

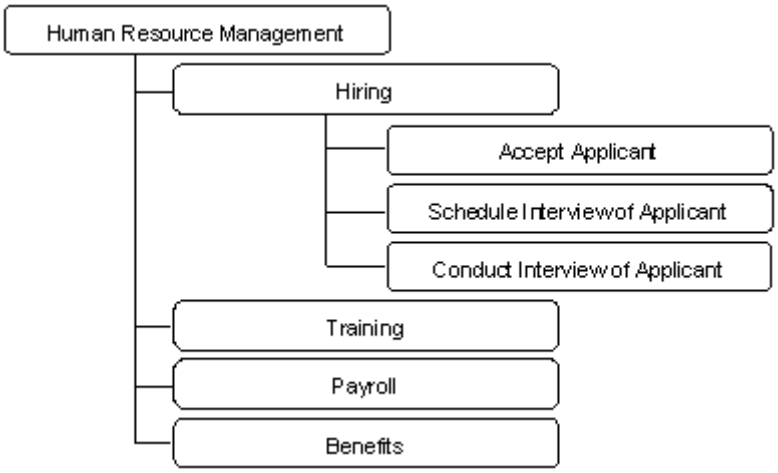
The result is a paradox of sorts: business object types cannot be identified with full confidence until decomposition completes, but business object types are needed to complete decomposition. The solution is to identify high-level proposed business object types that reflect reasonable assumptions about the model.

Take, for example, the sub-function of the Human Resource Management function called Hiring. It would be reasonable to assume that Hiring deals with occurrences of a business object named applicant. To decompose Hiring into its elements, consider how applicant moves through its life-cycle. This is not a formal life-cycle analysis in which all possibilities for an entity type are considered. It is simply a recognition that there are certain phases through which an applicant proceeds during the Hiring process.

A potential life-cycle for applicant is shown as the following illustration.



The Hiring function can be decomposed into the sub-functions required to lead occurrences of applicant through its life-cycle. The following illustration shows an example of a decomposition of Human Resource Management that includes a decomposition of Hiring into its elements based on the life-cycle of applicant.



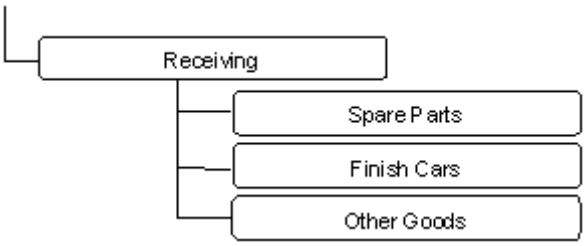
Whenever possible, decomposition of activities should be accomplished using this approach.

Decomposition by Business Object Type Classification

In some cases, particularly at higher levels of the decomposition hierarchy, it may be difficult to discern a clear life-cycle for a single proposed business object type.

Some high level functions are so broad that they include several readily identifiable proposed business object types, each of which have their own life-cycle. In such cases, functions can be decomposed based on their support for specific proposed business object types.

Consider, for example, the RECEIVING function of a company that sells cars. Assume that the company must receive both finished cars and spare parts and that finished cars and spare parts both undergo a different process when they are received. In this case, the RECEIVING function cannot be associated with the life-cycle of some proposed business object type. Instead, the RECEIVING function is associated with multiple life-cycles of multiple business object types, namely SPARE PARTS and FINISHED CARS. As a result, RECEIVING is decomposed based on the classification of its proposed business object types, as shown in the following illustration.



This sort of decomposition should be used only until a function is discovered to handle a single identifiable proposed business object type. From that point, the life-cycle of the proposed business object type should govern decomposition.

Confirming Decomposition with Dependency Analysis

The principles of cohesion and coupling can be tested at each leg of the decomposition by drawing dependency diagrams, which are described in the chapter "Analyzing Activities."

Activities that are siblings should be dependent on one another in some fashion; that is, their parent should be cohesive.

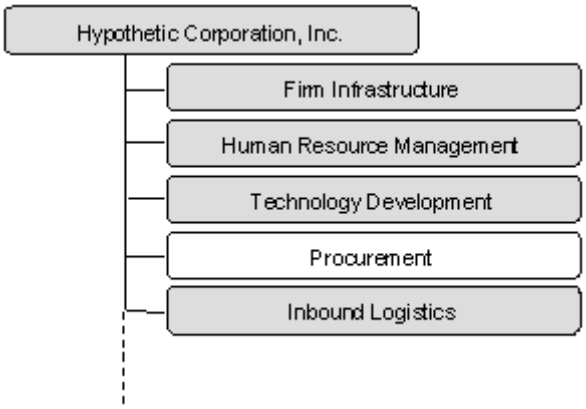
At the same time, siblings should have few dependencies with functions with which they are not siblings; that is, their parent should be loosely coupled.

Any decomposition that results in siblings that are not interdependent or are highly dependent on non-sibling functions should be questioned.

Example Activity Decomposition

This example illustrates the decomposition of a Procurement function for a hypothetical business.

Assume that Procurement is a highest level function, as depicted in the following illustration.

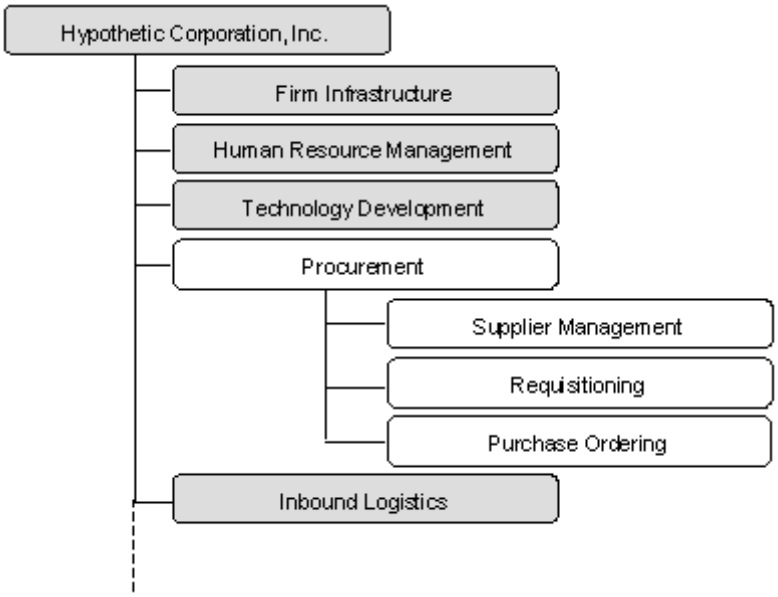


PROCUREMENT is a sufficiently broad grouping of activities that business object type classification would best be used to decompose it.

After some investigation, assume that the following proposed business object types are tentatively identified:

- SUPPLIER
- REQUISITION
- PURCHASE ORDER

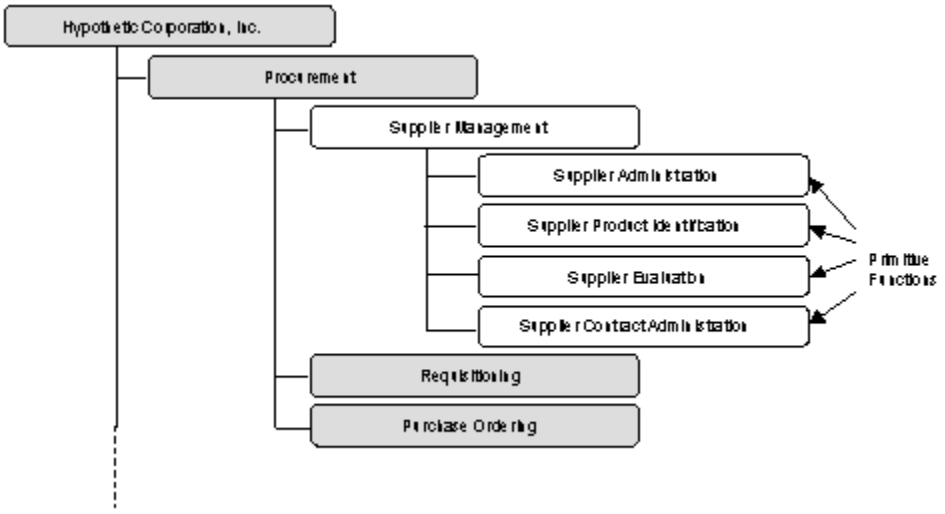
The resulting decomposition of Procurement based on this classification of proposed business object types might look as shown in the following illustration.



Each function deals with one proposed business object type:

- Supplier Management deals with supplier
- Requisitioning deals with requisition
- Purchase Ordering deals with purchase order

Now take a look at the Supplier Management function. After some investigation, it turns out that Supplier Management involves several proposed business object types, too. In addition to supplier (the obvious one), it must deal with supplier products, supplier evaluations, and supplier contracts. The following illustration shows the resulting decomposition of Supplier Management based on this classification of proposed business object types.



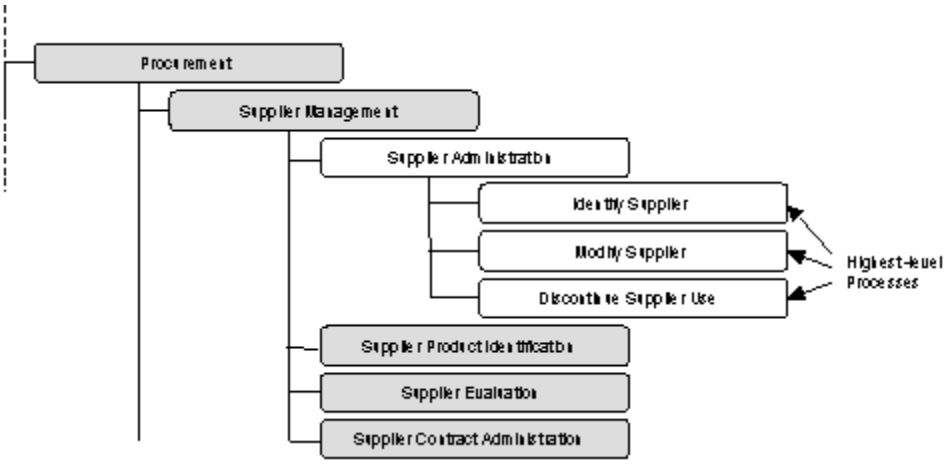
After further analysis, assume that it becomes obvious that these proposed business object types are actual business object types. This might happen as the result of parallel decomposition in which the proposed business object type is discovered to have the characteristics of a primitive subject area (that is, having a clearly identifiable central entity type). If this is true, the functions at this level of decomposition are primitive functions (that is, responsible for managing the lives of entities of each of the central entity types).

The next level of decomposition, then, must yield processes rather than functions. Look at Supplier Administration, which is clearly concerned with the business object type supplier, whose central entity type is also supplier.

Assume that each supplier has a very simple life-cycle:

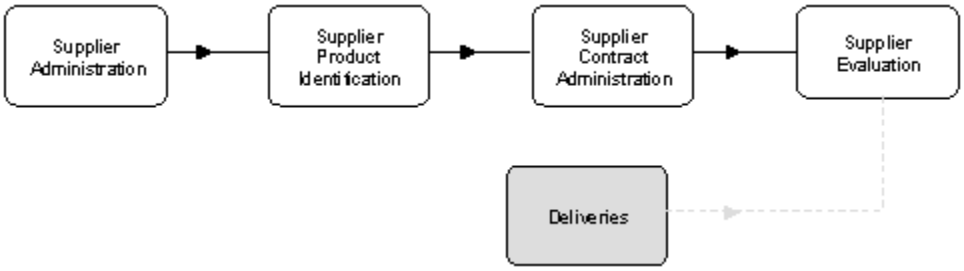
- A supplier can be identified as a vendor of products needed by the enterprise
- Characteristics of supplier can change over time
- The enterprise can lose interest in the supplier

Based on this life-cycle, the highest level processes described for Supplier Administration might be those shown in the following illustration.



Dependency diagrams drawn at each of these legs will reveal high cohesion and loose coupling, thereby confirming the correctness of the decomposition.

For example, the Dependency Diagram shown in the following illustration shows how the siblings of Supplier Management are related.



Supplier products cannot be identified before suppliers are, and a supplier contract cannot be created until the supplier products are known.

Supplier Evaluation relies on the supplier contract as well as some additional information about deliveries maintained by a different sub-function of Procurement. Note that the dependency of Supplier Evaluation on a non-sibling function is more than offset by a high level of cohesion among its siblings.

Decomposing Data

Data decomposition is recorded using a Data Decomposition Diagram, which can be built with the Data Model List tool. The Data Model Diagram, which shows relationships between subject areas and between entity types, is used to confirm the decomposition.

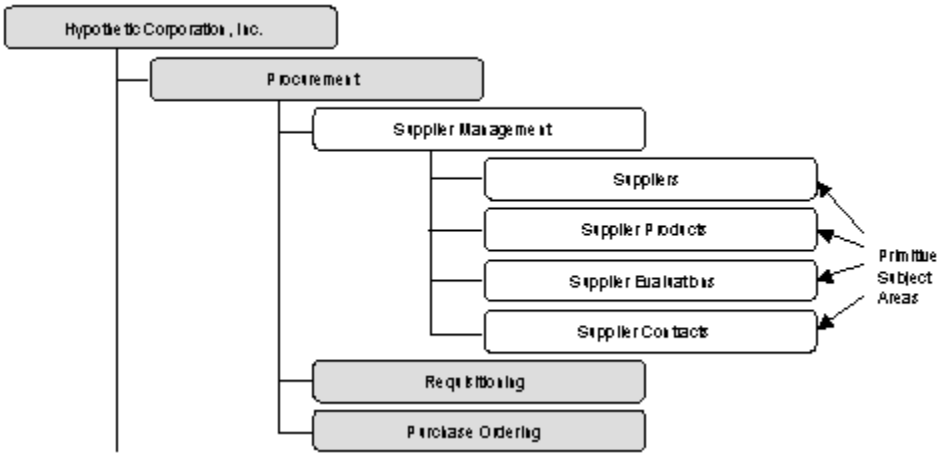
The same principles used for decomposition of activities are used for decomposing data. Business object type classification yields corresponding structures on both sides of the model, and decomposing by life-cycle on the activity side often reveals new business object types on the data side.

Following the Procurement example in Decomposing Activities, the subject area procurement, corresponding to the like-named function, is the starting point for data decomposition.

PROCUREMENT decomposes into a set of subject areas based on reasonable, proposed business object types:

- Supplier Management
- Requisitioning
- Purchase Ordering

Supplier management can be further classified into subject areas based on the distinct proposed business object types suppliers, supplier products, supplier contracts, and supplier evaluations. This decomposition is depicted in the following illustration.



Each of the subject areas has a single central entity type for which instances can be imagined, that is, supplier, supplier product, and so forth. The functions associated with each subject area can be considered to deal with entities of that type as they make their way through life. Since these conditions are true, the lowest level subject areas in the illustration must be primitive subject areas.

Identifying Dependent Entity Types Using Normalization

On the data side of the model, decomposition ends at the primitive subject area. Each primitive subject area is composed of a central entity type and its dependents.

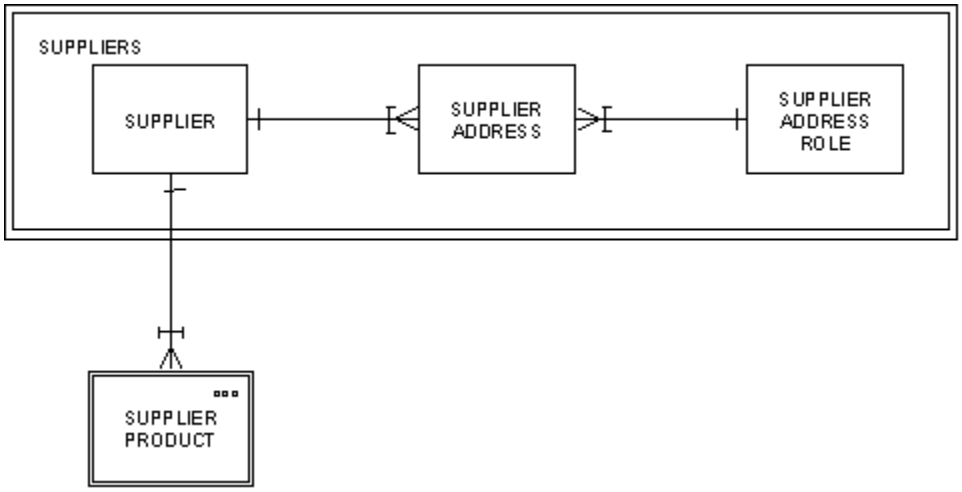
The most common mechanism used to model data once decomposition concludes is normalization, described in the chapter "Analyzing Data."

For example, consider the primitive subject area suppliers. It is identified as a primitive subject area because it is represented by a single central entity type, supplier. Business users of the application are likely to recognize supplier as a significant concept. However, supplier may not be fully normalized as it stands.

For example, each supplier might have multiple addresses. Each address might fulfill a particular role. The supplier might have one address for payment, one address to contact for shipping queries, and another address to contact for product problems.

Assume that the identifier for a single occurrence of an address of supplier is based on who the supplier is and which role the address fulfills. Further assume that the business has identified a standard set of roles for which each supplier must provide an address and that the identifier of each is a designed attribute.

Following the rules of normalization, supplier, supplier address and supplier address role must be separated into their own entity types, as shown in the following illustration.



Supplier is the central entity type; supplier address and supplier address role are its dependents.

Parallel Decomposition Heuristics

While not all are inviolable rules, any decomposition that fails to conform to these heuristics should be considered suspect:

- Each level of decomposition should yield between three and seven subordinate elements.
- If more than seven subordinate elements are discovered, introduction of an intervening super-ordinate level should be considered.
- Each primitive subject area should generally yield no more than 10 entity types.
- If more than 10 entity types are found, there is probably another central entity type in the primitive subject area.
- The expected number of entity types per primitive subject area in a typical model is eight.
- It is unusual (although not impossible) for a primitive subject area to yield fewer than six entity types.
- In such a case, it is likely that the entity types have been misclassified.
- The average number of elementary processes into which a primitive function eventually decomposes in a typical model is 27.
- That is, each primitive function decomposes into three high-level processes, which decompose into three processes apiece, which in turn decompose into three elementary processes each.
- A primitive subject area and its associated primitive function should be correspondingly complex.
- If the primitive subject area is simple (that is, includes fewer than the expected eight entity types), the associated primitive function should also be simple (that is, decompose into fewer than the expected 27 elementary processes).
- If the primitive subject area is complex, the primitive function should also be complex.
- There is no strict rule here, but if a model includes only three entity types that require 60 elementary processes to deal with them, something is askew.
- When decomposing a primitive function into its directly subordinate processes, those processes should be clearly recognized as dealing with the central entity type of the corresponding primitive subject area.
- At the next level of process decomposition, processes may be discovered that deal with both the central entity type and its dependents.
- This guideline helps to confirm that the primitive subject area and its corresponding primitive function truly correspond.

Refining the Model

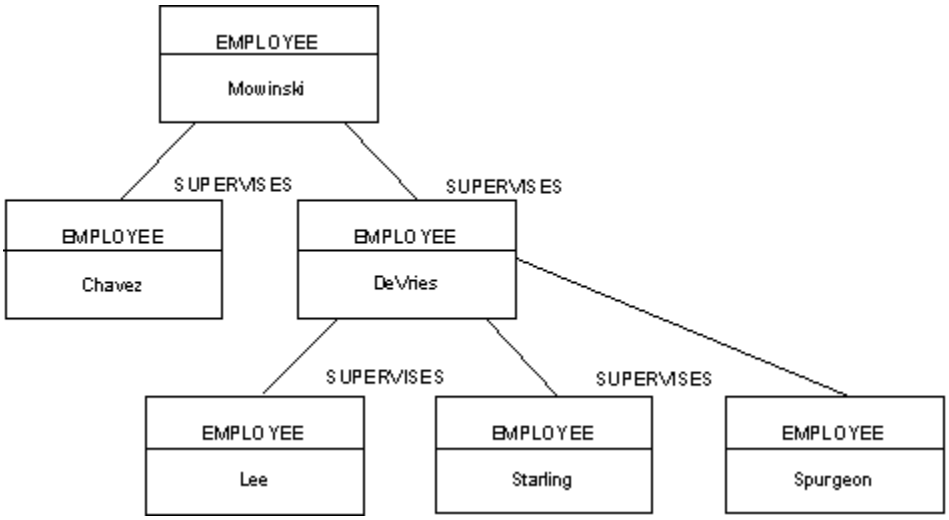
You may encounter some of these common, complex data modeling situations while building the analysis model, and some circumstances that may call for refining the model.

Modeling Hierarchies and Networks

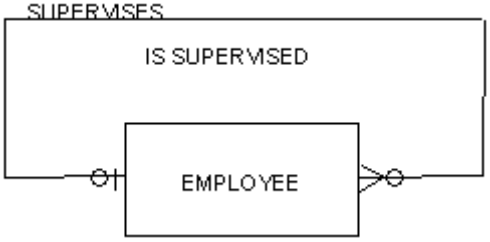
Recall that a relationship is a reason relevant to the enterprise for associating entities of one or two entity types.

Up to now, this guide has used examples of relationships between two separate entity types. However, it is clear from the definition that entities from the same entity type can also be paired based on a relationship. The most frequent use of such a relationship is to represent either a hierarchy or a network of entities, for example, in an organizational hierarchy.

The following illustration represents a situation where one employee can supervise one or more other employees who can, in turn, supervise one or more other employees.



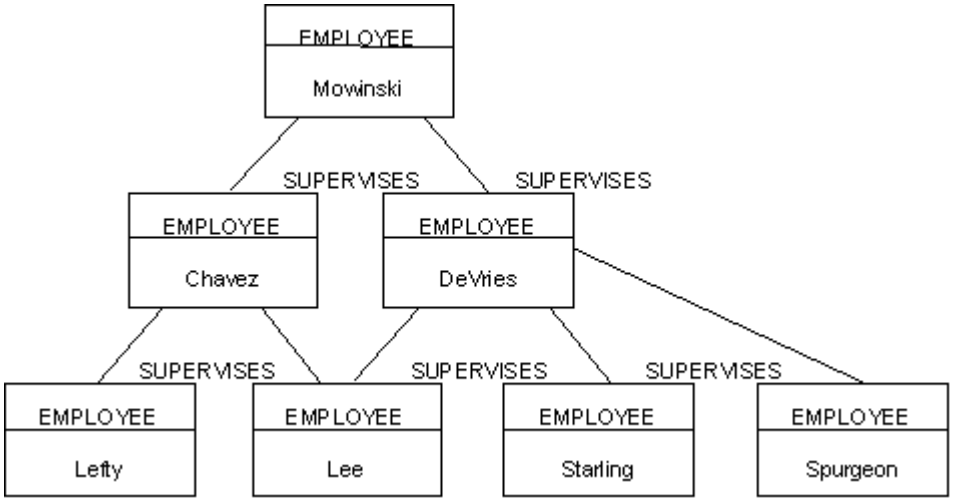
Such a hierarchy of employee entities can be modeled as shown in the Entity Relationship Diagram fragment in the following illustration.



This involuted relationship is also called a looped relationship. Note that the relationship must be fully optional to avoid a situation in which the hierarchy continues forever. The employee at the top of the hierarchy has no pairing because of its IS SUPERVISED BY relationship membership. The employees at the bottom have no pairings based on their supervises relationship membership.

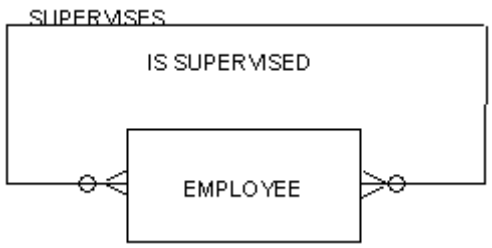
Each employee is supervised by at most one employee. By implication, it is also true that each employee entity can only appear once in the hierarchy.

The following illustration shows entity occurrences for a less formally structured business situation in which a single individual, named Lee, has multiple supervisors. These employee entities do not participate in a true hierarchy. Rather, they are arranged in a general network.



In a network, an entity can participate in multiple pairings based on either membership of its involuted relationship. Obviously, the model fragment presented in the following illustration of an involuted relationship is too restrictive to allow for such a condition.

The style of involuted, fully optional, M:N relationship shown in the following illustration can be used to support any general network and can also support the retention of successive occurrences of supervision over a period of time.



Consider a Bill of Materials structure. A Bill of Materials is generally shown as a hierarchy. In reality, however, it is a network, since a single part can appear multiple times, as shown below.

- Part Type A made from:
  - Seven of Part Type B
  - Three of Part Type C
- Part Type C made from:
  - Four of Part Type B
  - One of Part Type D

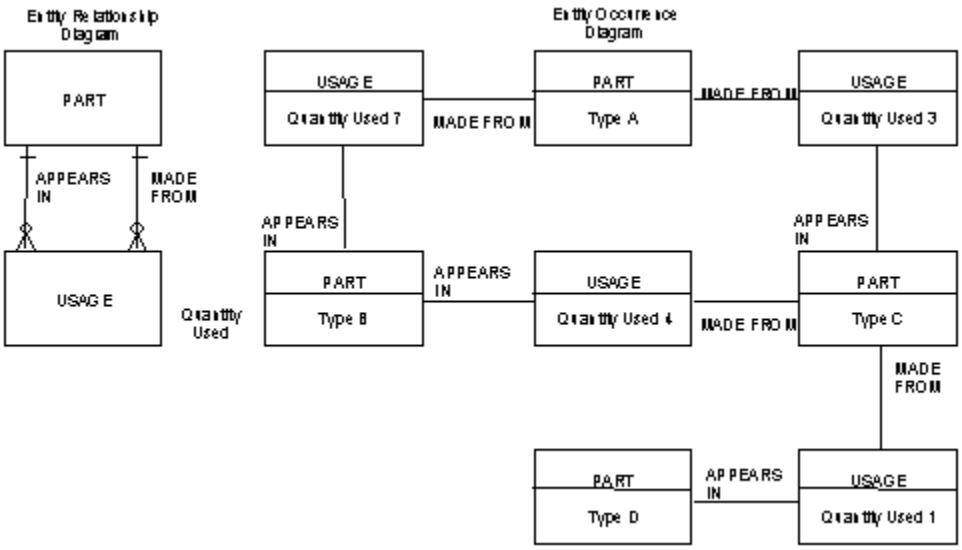
This structure is slightly different from the example of the general network.

Here, each time a part appears in the structure, it is accompanied by a piece of related information, Quantity Used. For example, in the table Bill of Materials Structure:

- Seven parts of type B are required to make a single part of type A
- Four parts of type B are required for a single part of type C of which three are required for each type A part

Obviously, Quantity Used is an attribute of something besides the entity type part. Otherwise, each part, with all of its attendant relationships and attributes, would have to be repeated each time it appeared in the structure, with the only difference being the value of Quantity Used.

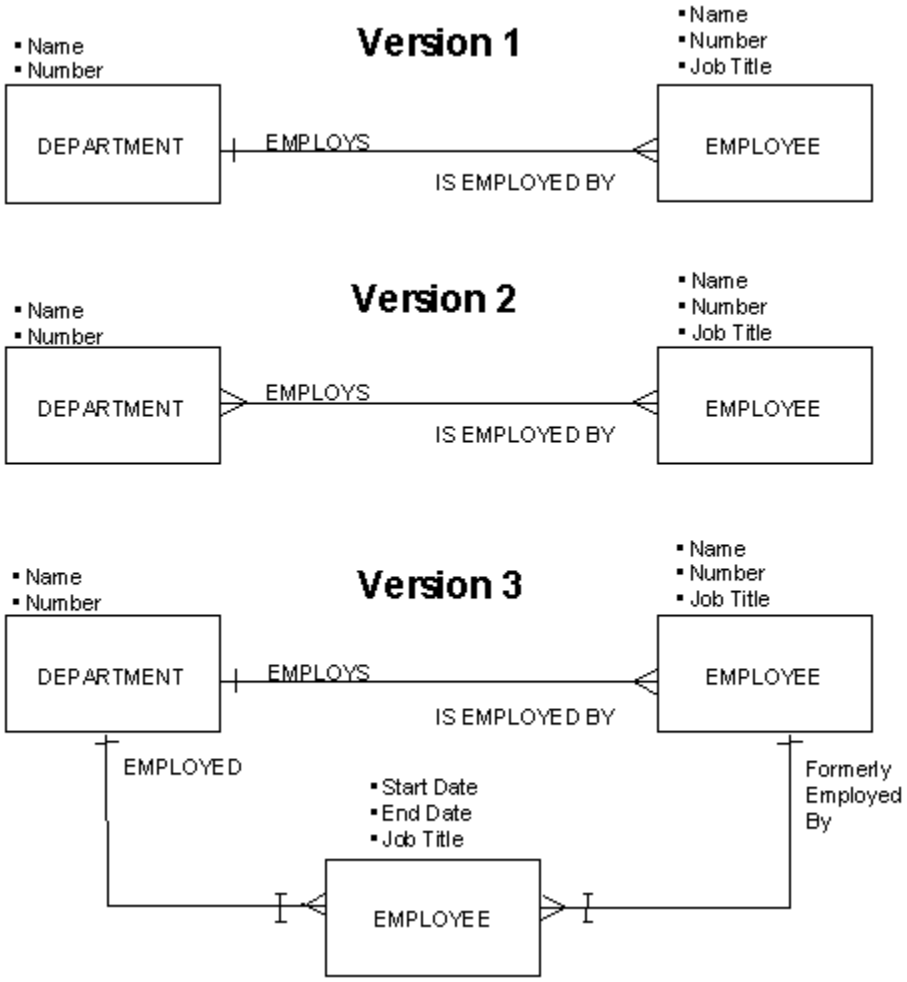
To solve this problem, you can introduce an entity type whose sole function is to hold Quantity Used and any other attributes associated with the specific usage of a part in the structure. The following illustration shows a model fragment that uses this technique, and the set of occurrences that support this illustration.



Modeling History and Time

Modeling the existence of entities over time needs more thought from the analyst than representing simply what is found at a single point in time. Converting a "snapshot" model may involve adding entity types, adding or moving attributes, and reconsidering the optionality of attributes and relationship memberships.

Consider, for example, the following three fragments of an Entity Relationship Diagram.



The ERDs in the illustration show versions in the understanding by the analyst of the situation in which departments employ employees.

- **Version 1**—For a single point in time, the model fragment for Version 1 represents this situation well enough. A department employs many employees, each of who has a Job Title.

Considering the history of this situation, the analyst realizes that many employees have been employed in a number of departments, and therefore a department has employed many more employees than it has at present. The analyst must check that this is indeed a business requirement before further elaborating the model.

- **Version 2**—This fragment corresponds to the change to the model, but there are information needs which are not yet represented:
  - Job Titles of employees in their previous departments
  - When they worked in those departments

This is an example of hidden information, which is discussed in the section Information Hidden in an M:N Relationship.

- **Version 3**—Adding an employment entity type to the first fragment, to give Version 3, allows the information needs to be satisfied.

The analyst also notes that there are now two attributes named Job Title, and so changes the name of employee Job Title.

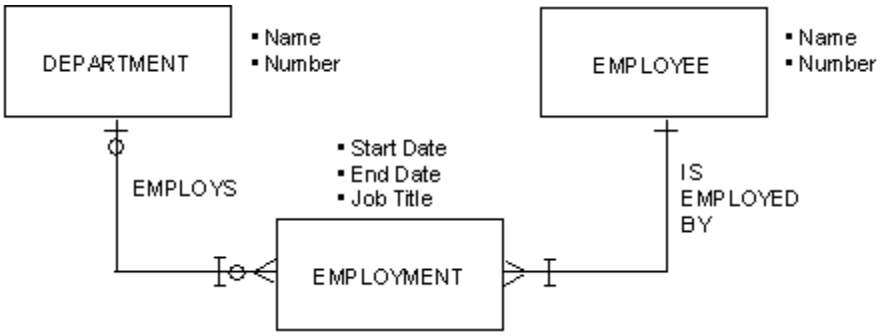
Employment is identified by its Start Date and by relationship memberships with department and employee. In fact, employment, with its Start Date and End Date attributes, is a typical active entity type.

The analyst then realizes that for the employment entities of today, there are no End Date values, so End Date is an optional attribute.

Business staff point out that they need to plan new departments and future employment, and so do not agree with the relationship names employed and formerly employed by. While changing these names, the analyst realizes that the relationship department employs employee is redundant, since the entity type employment and its relationships can represent the past, the future, and the present. Redundant relationships are discussed in Refining the Model. Nor is employee Current Job Title now needed.

Since a new planned department may not yet have any planned employment, the employs relationship membership must be optional. Employment Start Date can have a value in the future, so no further change to the model is needed.

With this improved understanding and consensus, the analyst makes further refinements to produce the fragment shown in the following illustration.

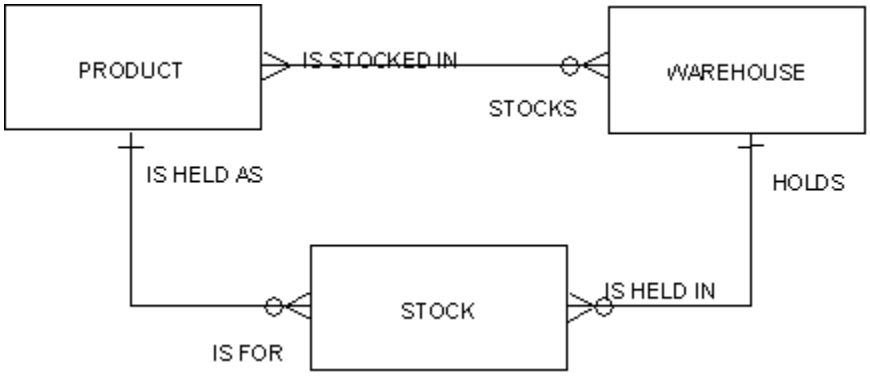


Handling Unusual Situations

You may encounter some of these questionable, unusual, or impossible situations when analyzing the business environment.

Redundant Relationships

A redundant relationship is a relationship conveying no information that cannot be deduced from other relationships. Consider the Entity Relationship Diagram fragment shown in the following illustration.



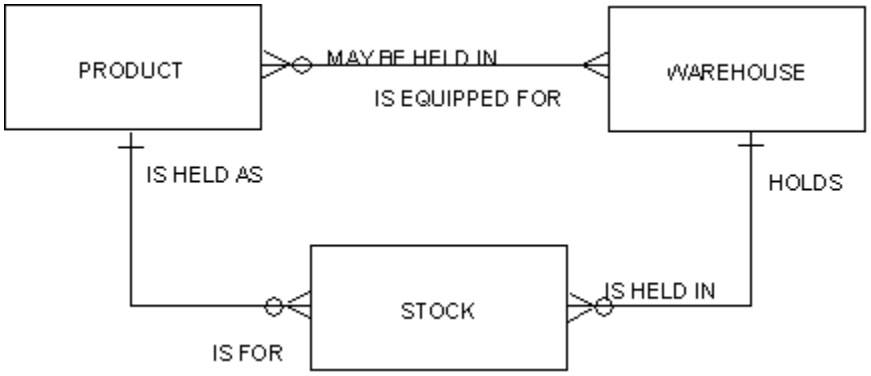
The M:N relationship between product and warehouse records that a product is stocked in a particular warehouse.

Assume that if a product is stocked at one or more warehouses, the Quantity of product at a particular warehouse is held as an attribute of the entity type stock. If this is true, the analyst can deduce that a product is stored at a warehouse from the pairing of the product with stock. STOCK, in turn, is paired with a warehouse. The relationship product is stocked in warehouse conveys no new information and is therefore redundant.

You should eliminate redundant relationships. Such relationships are superfluous and require duplicate operations to properly record the information reflected in the model. For example, each time a product is to be stocked in a new warehouse, the product must be paired both with a warehouse and a stock entity. When the product's stock is exhausted, two pairings must be deleted instead of just one.

Note: Exercise caution when removing apparently redundant relationships.

Imagine, for example, a situation only slightly different from the one in the previous illustration. Assume that certain warehouses are equipped to handle certain products and that the ability of a warehouse to stock a particular product, whether it actually carries it, is of interest to the business. The following illustration depicts this situation.



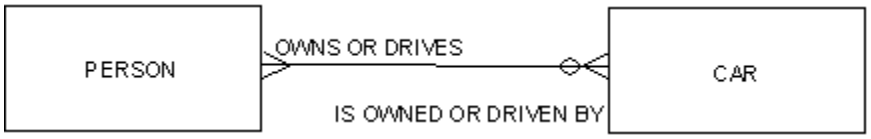
Here, the relationship between product and warehouse is not redundant, although it might appear to be at first glance.

You should always watch for this situation. Businesses commonly need to distinguish between the actual or current positions, and allowable or potential positions. When searching for relationships, try to determine whether this distinction is important.

Relationships with Multiple Meanings

Each relationship reflects a reason for associating two entities. Sometimes, there are multiple reasons for joining two entities of the same two entity types. If so, the model should contain one relationship for each reason.

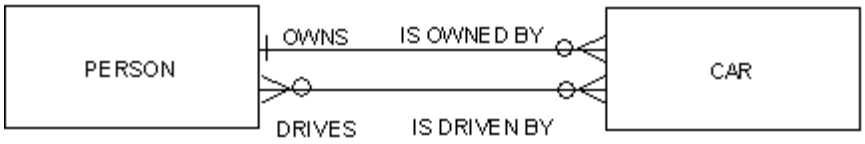
For example, consider the following illustration.



In this illustration, the M:N relationship between person and car has two distinct meanings:

- A person can own one or more cars which must, in turn, be owned by just one person
- A person can drive one or more cars that can, in turn, be driven by many persons.

The single M:N relationship fails to distinguish between two different reasons for associating person and car. It should be expanded into two relationships, as shown in the following illustration.

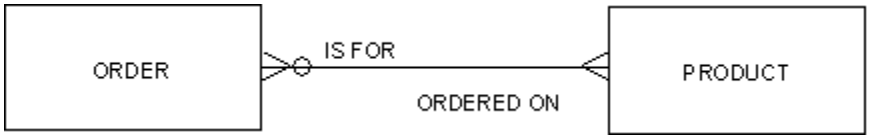


Two or more relationships that join the same entity types are called parallel relationships.

Information Hidden in an M:N Relationship

The injudicious use of M:N relationships can sometimes lead to the loss of important information.

Although M:N relationships sometimes occur naturally, there is often information of interest hidden by the relationship itself. For example, consider the following illustration.



The illustration shows that an order is for one or more products, and that a product appears on one or more orders. While this is true, some important details have been omitted. For example, what is the quantity of each product appearing on a given order? You cannot extract this information from this illustration.

Useful information that is not apparent because of the existence of an M:N relationship is called intersection data.

The need for intersection data calls for the addition of an entity type to hold that data as attributes. By adding an intersection (or associative) entity type and replacing the M:N relationship with two 1:M relationships, you can retain all the information conveyed by the original M:N relationship and also represent additional facts about the intersection.

In the following illustration, the entity type order item has been added to hold the attribute Quantity for a given product on a given order.

As a matter of good practice, carefully evaluate each M:N relationship to ensure that it results in no required information being hidden. It is unusual for a fully detailed business model to retain any M:N relationships.

1:1 Fully Mandatory Relationships

Analysts should be wary of any 1:1 relationship defined as fully mandatory (both relationship memberships are mandatory). Unless there is a good, solid business reason for identifying the entity types separately, they should be combined into a single entity type.

Solitary Entities

An entity that is the only occurrence of its entity type is called a solitary entity.

The existence of solitary entities often indicates incorrect analysis of some kind. Evaluate solitary entities carefully.

For example, a model that depends on there being only one occurrence of some entity type, say factory, is time dependent. If more factories are added in the future, the model will require modification.

In other cases, where there is and will be only one occurrence of an entity type, ensure it is not a special instance of another, more general entity type. For example, an entity type representing the entire corporation has only one occurrence, and so might well fit into an entity type called organizational unit.

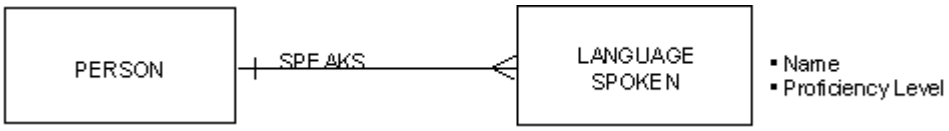
Multi-Valued Attributes

Occasionally, an attribute seems to need to hold multiple values simultaneously.

For example, consider an entity type person, which has Language Spoken as an attribute. A distinguishing characteristic of a person is, of course, that they may speak many languages. So, the attribute Language Spoken would seem to require multiple attribute values for each entity of person.

An attribute that seems to require multiple values simultaneously is called a multi-valued attribute. If, in this example, you need to know the level of proficiency for each language, the case for defining an additional entity type called language spoken becomes very strong.

Data modeling does not, and should not, provide for multi-valued attributes. Rather, you should remove an apparently multi-valued attribute to its own entity type and relate it to the original entity type through a 1:M relationship. This is shown in the following illustration.



Entity Types With Too Few Relationships and Attributes

Each entity type must have at least one attribute and one relationship membership. This makes sense because an entity type with only one relationship membership and no attributes can convey no additional information. Therefore, certain combinations of relationships and attributes should be checked carefully:

- Entity types with no attributes
  - Evaluate these carefully to ensure that there really are values associated with them; otherwise, they should be removed from the model. Also ensure that the pairings in which they participate are truly of interest to the business.
  - An entity type with no attributes is commonly an associative entity type that resolves one or more M:N relationships. There is always the possibility of discovering some attributes later on, such as the time and reason for the association. Adding an entity type or a relationship has a much greater impact on design aspects of the model such as Action Diagrams, the screen designs, and the Data Structure Diagram than adding an attribute to an entity type.
- Entity types with only one attribute
  - These are very rare, but occasionally they are valid as might be the case with a multi-valued attribute. Always check that the sole attribute does not belong to some other entity type.
- Isolated entity types
  - These have no relationships, and so the information provided by their attributes cannot be related to any other entity types in the business. This is an unlikely situation, so the definition is probably incorrect.
  - Most things of value to a business relate to some other aspects of the business. Early in building a model, important reference entities such as tax rates may appear to be isolated but are eventually associated with entity types such as state or product group.

- If data design work has been done, you may find that some designer-added entity types appear in the model. Where such entity types appear, there may be some that are isolated, such as those that represent code tables. These isolated entity types may safely remain in the model. If, after careful consideration, you discover no relationships, you should remove the entity type from the model.

Results of Building the Analysis Model

The results of building the analysis model are:

- Entity Relationship Diagram describing the data requirements of the development project
- Supporting documentation, stored as details of the elements of the Entity Relationship Diagram
- Activity Hierarchy Diagram depicting the decomposition of processes to the elementary process level
- Dependency Diagram for each activity with an elementary process as a child
- Supporting documentation, which is captured as details of the processes, dependencies, external objects and events appearing on the Activity Hierarchy Diagram and Activity Dependency Diagrams.

Chapter 9: Starting Object-Oriented Analysis

This chapter describes the object-oriented methodology, its features, and the implementation of object-object analysis in CA Gen.

Object-Oriented Methodology Concepts

Object-oriented methodologies typically require diagrams that show a message is passed between objects. CA Gen puts the information in these diagrams in the form of dialog flows and USE actions held within the model. Object-oriented methodologies typically recommend some form of state transition diagram, to study the phases in the life of an instance of an object type. This is supported by the oriented approaches for studying the required functionality of applications. This bears considerable similarity with activity dependency analysis and event analysis, and the activity dependency diagram can, if required, be used to document use cases.

Object models are expected to take advantage of inheritance and polymorphism. Inheritance is supported by CA Gen, while polymorphism is not.

Inheritance

Inheritance enables the reuse of data definitions and functionality. Subtypes inherit all the operation, attributes, and relationships of their parent types. Object models may include much generalized object types that enable full advantage of the inheritance mechanism to be taken.

Polymorphism

Polymorphism is typically used to describe the capability that an operation can have different methods (different logic) for each subtype that inherits it, and that the correct method gets automatically chosen at runtime. This can be achieved using CA Gen by including a Case of Subtype structure within operation logic. What cannot be done with CA Gen is the addition of new subtypes and new methods for the inherited operations of subtypes without editing and recompiling some of the existing logic. One of the benefits of the best object-oriented systems is the ease with which models and application functionality can be extended without the danger of corrupting existing functionality—making it possible to grow applications.

Object Modeling with CA Gen

The following table presents the vocabulary of object modeling methods with the equivalent or broadly equivalent terms from CA Gen.

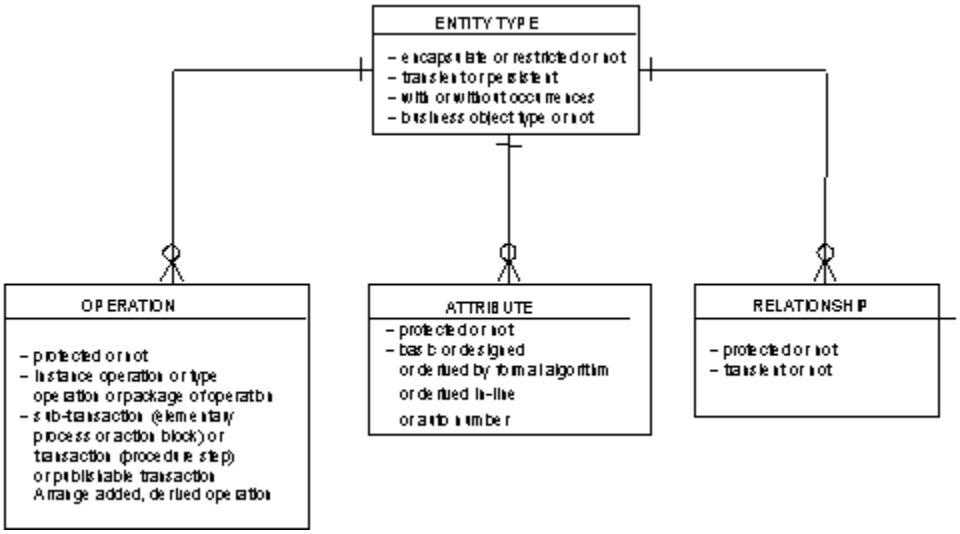
Object-Oriented Term	CA Gen Term
Object Type or Class	Entity type that owns operations
Instance or Object	Occurrence or Instance
Operation or Service or Method or Function	Operation
Operation Implementation or Method	Action Diagram Logic Statements
Operation Signature	Operation Imports and Exports
Attribute	Attribute
Association	Relationship
Multiplicity	Cardinality
Message	Dialog Flow or USE action
Package or Category or Subsystem	Subject Area
Subclass	Subtype
Disjoint or non-overlapping subclass set	Partitioning
Discriminator	Classifying Attribute

Object modeling is an alternative technique for modeling an area of a business. It bears many similarities with the data/activity analysis approach, but it does differ in some important ways:

- Business entities and activities are integrated into a single model. This tighter integration of models produces a robust model early in the development process.
- Business entities are perceived to own and perform business activities. In fact, each aspect of the functionality of the business area should be regarded as owned by some business entity (usually called an object or instance). We call each discrete unit of functionality an operation of the business entity.

- In object modeling you document generic classes of business entities, rather than individual business entities. These classes continue to be termed entity types, although they are variously termed object types or classes or concepts in object modeling methods.
- In object models, entity types are perceived to be packages of functionality that maintain data about themselves. They also continue to correspond to types of things (tangible or intangible) that the business has to monitor. For example, the entity type Customer performs operations like Change Credit Rating and Offer Discount. It also keeps data about itself such as Name, Address, Payment Statistics, and Sales Statistics. This data can only be accessed through the operation of the business entity.
- An operation of an entity may need information about entities of other types, or may need to update the data of other types. For example, when an Order is changed, the credit rating of the Customer may need to be checked and the Customer statistics may need to be changed. An operation of one entity often needs to use the operations of other entities. The Change Order operation is not allowed to directly manipulate data but must be done under CA Gen control using CA Gen operations.
- Each entity is regarded as a black box of functionality: each entity encapsulates its data. This style of modeling ensures entity types are relatively independent, which encourages reuse and promotes the maintainability and software quality of applications based on object models.
- To summarize: an object model describes the problem domain as a mesh of intercommunication, functioning, black-box entity types, instead of by a cross-related, entity and activity models.

The following illustration should help you understand features that enable entity types to be regarded as object types (or classes) and hence to be used as building blocks within objects models.



Using Object-Oriented Features

The following sections describe the operation and encapsulation features of object-oriented analysis.

Operation

An operation is a unit of functionality owned by an entity type. An entity type that owns functionality is known as an object type or class. An operation may only be owned by one entity type.

In an object model, all functionality should be defined as operations of entity types. When a new elementary process or action block is identified, you should register it as an operation of an entity type. Following this procedure will result in a more tightly integrated model and existing functionality that is easier to find.

Another beneficial practice is to allow the project to build either a conventional (non-OO) model of free-standing action diagrams, or an object model, in which entity types own all functionality.

The identification of new functionality can lead to the discovery of new entity types. Object models need not be normalized. See the discussion on transient entity types. Utility action blocks, which only operate on work set attributes, need not be defined as operations.

Operations must be classified as one of the following:

- Instance
- Type
- Packages

Instance Operation

Instance operation is an operation that is focused on a single existing occurrence (instance) of the owning entity type. Operations that update or delete an existing entity occurrence (instance) should ideally be registered as instance operations. Operations that read an existing instance and then retrieve data about that instance or its associated instance (or update associated instances) should also be registered as instance operations.

Type Operation

A type operation is an operation that is not focused on a single existing occurrence (instance) of the owning entity type. A type operation is focused on a single non-existent occurrence (for example, a list operation or a find operation employing a non-unique search key.)

Operation Package

An Operation Package is a procedure step that represents a packaging together of several operations. Procedure steps may offer several discrete units of functionality, and the consumer uses an input command to distinguish which piece of functionality is to be executed. Each command, then, effectively represents a separate operation within the procedure step. See the design book for more information concerning this type of procedure step.

Protected Operations

Protected operations may only be used by a calling action diagram where the caller is:

- An operation of the (protected operation's) owning entity type
- An operation of a subtype or supertype of the (protected operation's) owning entity type

Encapsulation

Encapsulation means that the entity type's implementation (action diagrams and data storage) can be changed without impacting the users of the entity type. Persistent views of an encapsulated entity type may only occur within its own operations, or in the operations of a supertype or subtype. Hence create, update, delete, and read actions on occurrences of this type are confined to operations of the type, its subtype or supertype.

An encapsulated entity type may not have persistent relationships. Hence associate, disassociate, and transfer actions are not available.

The features of encapsulated entity types may be protected. Protected features are only visible within the operations of the entity type owning the feature. By default, all basic and designed attributes of an encapsulated entity type are protected unless the attribute is a part of a primary identifier.

In projects performing object modeling, the recommended encapsulation level is restricted. But in some situations, encapsulated will be preferred:

- Where the data storage for the entity type occurrences is not a CA Gen generated table
- Where it is known that implementation of the entity type is liable to change
- Where the benefits of strict encapsulation outweigh the costs of having to handle relationships and RI through user-written logic.

Encapsulation Options

There are three types of encapsulation from which to choose:

- Open
- Restricted
- Encapsulated

The encapsulation option you select affects:

- **Feature visibility**—The features of an entity type are its operations, attributes, and relationships. If a feature is protected, it is visible (available) either within the operations of the entity type that owns the feature or within the operations of its subtypes or supertypes. While the owning entity type of an attribute or operation is clear, the owner of a relationship needs more explanation and is covered below.
- **Action restrictions**—The actions on an entity are CRUD (create, delete, update, read) and DAT (disassociate, associate, and transfer). Encapsulation options restrict which actions are permitted in any given action diagram. The restricted and encapsulated options require more development effort. The pay-back concerns ease-of-change, maintainability, and data integrity.

Open Encapsulation

The entity type is not encapsulated. Use this form of encapsulation for detailing (also known as "dependent" or "characteristic entity types") entity types in object models where the project has decided to encapsulate business object types rather than individual entity types.

Restricted Encapsulation

Some of the restrictions inherent in full encapsulation are applied. Updates must be handled by an entity type's own operations, while any action diagram may perform a read. The features of the entity type can be protected. This choice is not pure encapsulation. It cannot bring all the benefits. It does not bring all the costs.

The restricted option confines the maintenance of an entity type to a few operations, improving the integrity of the type, the modularization of logic and the organization of developer responsibilities.

The features of restricted entity types may be protected. Protected features are only visible within the operations of the entity type owning the feature. By default, all basic and designed attributes of a restricted entity type are protected, unless the attribute is a part of the primary identifier of the type.

Transient Entity Types

A transient entity type is one whose occurrences are not physically stored. Entity types that are implemented by a corresponding relational database table are termed persistent. Transient types have no corresponding relational database table.

While there may be a simple one-to-one correspondence between a transient entity type and a persistent entity type, other mappings may prove useful. For example the data of several different transient types can be stored in one table corresponding to one persistent type. Or a transient type may gather together the data stored in several tables.

Since transient types do not represent persistent storage, there is no need for a set of transient types to be normalized.

Transient views (this term covers transient import, transient export and local views) may be defined for transient entity types. Persistent views (this term covers entity action, persistent import and persistent export views) may not. By way of contrast, persistent entity types may have both persistent and transient views.

Business Object Types

A business object type is an entity type that:

- Owns one or more transaction operations
- Is meaningful to business personnel
- Has multiple occurrences

When an entity type is given a transaction operation (that is, a procedure step is registered as an operation of the entity type) that entity type is automatically marked as a business object type.

When building an object/data model in a top-down fashion, mark the main entity types as business object types, since these are the entity types we expect to offer transaction operations.

No Occurrence Entity Types

A No Occurrence Entity Type is an entity type that owns operations, attributes, and relationships, but which has no set of occurrences. The entity type represents an object. One may think of this as an entity type with no instances, or a single instance that does not belong to any type.

The entity type must be transient. All its operations must be type operations.

Use No Occurrence Entity Types to represent in models any package of functionality that does not correspond to a type of thing, but which correspond to a single thing. Examples are, the functions of an existing system or the functions of a transient type that supports utility operations (that reformat dates for example) or the functions of a type that has a single occurrence, where there is little or no chance there will ever be multiple occurrences; such as The Company, Head Office, or The Government.

Where there is a single occurrence at the moment, but further occurrences may occur in the future, then it is preferable to record the entity type as multiple occurrence, and enter this minimum, maximum, and average occurrences as one.

By way of contrast, most entity types will represent types of things, for example Customer, Employee, Property for which multiple occurrences (instances) will exist at runtime. Operations will usually be focused on single occurrences of the type (instance operations).

Attributes

The following sections describe the various types of attributes for an entity.

Derived

A derived attribute is one that is not stored. Derived attributes are the "attribute equivalent" of transient entity types and relationships. Developers may prefer to think of them as transient attributes.

Both persistent and transient entity types may own derived attributes. The attributes of a transient entity type must all be derived.

Persistent entity types may also own derived as well as persistent attributes. Derived attributes allow persistent types to display a degree of denormalization, without the need to store redundant data. For example, the Employee entity type may own the attributes Age and Boss's _Name. The first of these is derived from Date_Of_Birth, a persistent attribute of Employee; the second is derived from Head_Name, a persistent attribute of the associated entity type Department.

The consumer of an operation need not be concerned whether the attributes imported or exported by that operation are derived or persistent.

Protected

A protected attribute is one that is only visible to the implementations of the owning entity type. Attributes are protected so they can be altered, removed or added to the implementation of an entity type, without impacting the rest of the application. Ideally, encapsulated and restricted entity types protect all their persistent attributes, except for the primary identifier, since this enables consumers to have a "handle" on an occurrence of the entity type.

You should protect all non-derived attributes of restricted or encapsulated entity types, except for the primary identifier. This should enable the implementation of the entity type to be changed without impacting the consumers of the entity type.

Where the project is confident that a particular attribute is very stable, and it is very unlikely to be dropped or altered in any way, then the attribute may be unprotected to simplify the action diagramming. If the storage of the attribute is changed in future, the developer may alter the attribute to derived at that time, and introduce new persistent attributes to store the values.

Protected derived attributes may be used for attributes used in intermediate calculations, where there is no requirement to reveal this value to consumers.

Relationships

The following sections describe the various types of relationships that exist between the entities.

Transient

A transient relationship is one that is not stored. A transient relationship may, of course, be stored by adding a persistent attribute to an entity type, to hold the identifier of some other persistent entity type. The validity of this foreign identifier has to be maintained by explicit action diagram statements (in operations).

Transient relationships are "documentation only." Compare a transient relationship with a CA Gen managed relationship, in which CA Gen automatically generates foreign keys in the relationship table, and manages referential integrity using database features, or CA Gen generated code. The user has the convenience of referring to the relationship in Read actions, and performing Associate, Disassociate, and Transfer actions on the relationship.

Transient relationships can be used to show business relationships existing between persistent types, but which are not managed by CA Gen. The user must write specific logic to make relationship occurrences persistent, and to maintain referential integrity. However, the modeler may simply wish to depict a relationship in the diagram that is redundant or derivable, as an aid to understanding the diagram. In this case, there is no requirement for user logic to maintain the relationship.

One reason for not using CA Gen to implement the relationship is to achieve full encapsulation. Once two-entity types share a CA Gen managed relationship, they cannot be fully encapsulated, since a change to the implementation of one entity type impacts the implementation of the other.

Transient relationships are also useful for depicting relationships between transient and persistent types. Transient relationships allow the user to define integrity constraints involving relationships (that is, optionality, cardinality, and identifier) although Compose does not enforce such constraints.

Protected

Protected relationships are only visible to the implementations of their two participating entity types. Protected relationships can be altered, removed, or added to the implementation of their two participating entity types, without impacting the rest of the application.

Encapsulated entity types do not support CA Gen implemented relationships. Relationships involving one or two restricted entity types are automatically protected by CA Gen.

A protected relationship can only be referenced (in read statements) in the operations of the (one or two) participating entity types of that relationship. Only the owning entity type may perform DAT actions on protected relationships.

Chapter 10: Confirming Analysis

It is important to verify that the model is correct and complete at intervals throughout analysis.

Model confirmations become increasingly important as you near the completion of this development phase. It is especially important to ensure the model is ready for continued refinement during the next phase of system development. Apply confirmation techniques systematically before proceeding with design activities.

CA Gen provides facilities to support confirmation techniques. However, confirmation can never be completely automated. In particular, the Consistency Check facility performs much of this checking automatically. The toolset documentation explains this facility in detail.

Consistency Check can be run against:

- A single model object, such as an entity type or process
- Process portions of the model
- All the data against all the activities
- The entire model

In design, CA Gen performs automated checks and will not accept inconsistent model objects. The model must be completed and corrected before finishing analysis on any part of the model that a system will support.

The automated consistency check facility verifies that the model conforms to CA Gen rules and conventions. The confirmation techniques described in this chapter help the analyst to ensure that the meaning of the model is correct as well.

By the end of analysis, you should also have produced, confirmed, or modified the following:

- Prioritized plan for business system development, identifying the projected implementation sequence for business systems
- Prioritized plan for each business system, identifying the projected implementation sequence for processes
- Formulation of the technical requirements for each system

Checking for Completeness

You perform completeness checking to confirm that the model for the business area under analysis is complete. In particular, you address these questions:

- Have all the processes, dependencies, entity types, attributes, and relationships been found?
- Does the model include processes to create, delete, update, and reference each entity type, attribute, and relationship?

The result of completeness checking is a complete but unconfirmed business model. To ensure confirmation, you must also check the model for correctness and stability. Completeness checking includes two main techniques:

- Comparison checking
- Interaction cross-checking

Comparison Checking

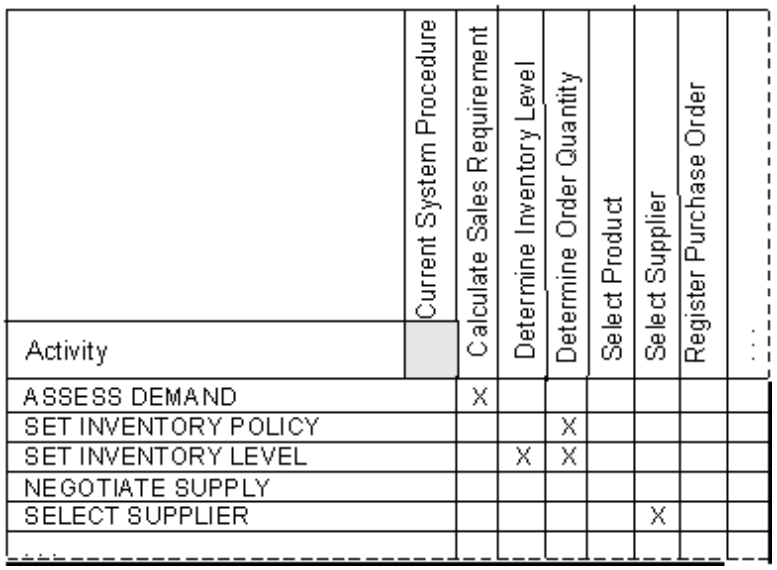
Comparison checking involves comparing the new business model with current systems models. This activity helps to find further analysis objects, particularly attributes and low-level processes.

The following table shows some likely model objects for comparison.

Business Area Model	Current Systems Models
Subject Area	Subject Database
Entity Type	Record, Data Store, Key Group
Attribute	Field, Data Element, Implied Relationship
Relationship	Field, Implied Relationship
Function	System
Process	Procedure, Program
Information View	Data View, Data Store
Dependency	Data Flow

The chapter titled "Analyzing Current Systems" describes comparison of entity types with current data stores. One example of a comparison, used for confirming the analysis model and planning for transition, is that of elementary processes in the activity model compared with current system procedures.

Comparison checking uses the matrix shown in part in the following illustration.



Activity	Current System Procedure	Calculate Sales Requirement	Determine Inventory Level	Determine Order Quantity	Select Product	Select Supplier	Register Purchase Order	. . .
ASSESS DEMAND		X						
SET INVENTORY POLICY				X				
SET INVENTORY LEVEL			X	X				
NEGOTIATE SUPPLY								
SELECT SUPPLIER						X		
. . .								

Comparison checking can reveal the following situations:

- Procedure supports no process in the analysis model.
- The activity model may need to be amended, or the procedure may support a process that is outside the scope of the current development project, in which case the project should not plan to replace the current system in its entirety.
- No procedure supports this process.
- The current system does not support the process, but it may be worth checking other current systems.
- A procedure supports this process.
- It may still be worth checking the process definition for omissions suggested by the procedure documentation.
- Several procedures support this process.
- There may be genuine overlap in current systems, but it is still worth checking the process definition to see if the process is really elementary, or whether it represents several business processes.
- A procedure supports several processes.
- It is worth checking the process definitions to see if the business processes really are separate.

Interaction Cross-Checking

Interactions show how business activities use data analysis objects. They summarize the involvement of processes with entity types, attributes or relationships.

Three interaction checks can be developed, using:

- Elementary Process/Entity Type Matrix
- Elementary Process/Relationship Table
- Elementary Process/Attribute Table

An Elementary Process/Entity Type Matrix indicates whether every defined entity type is fully used. CA Gen can produce the Elementary Process/Entity Type Matrix, as described in the chapter titled "Analyzing Interactions."

The tables can be produced as needed using a spreadsheet or similar utility.

You perform these checks to determine whether the new business model contains a process to:

- Create, Read, Update, or Delete each entity type
- Associate and Disassociate each relationship
- Set values for attributes and to reference, change, or remove these values

The Elementary Process/Entity Type Matrix is useful throughout analysis. Relationships and attributes, however, are finalized later, through interaction analysis and checking against the findings of current systems analysis.

Using the Elementary Process/Entity Type Matrix

CA Gen automatically populates the Elementary Process/Entity Type Matrix with the list of elementary processes along one axis and entity types along the other.

Cell values are populated from the expected effects (Create, Delete, Update, Read) of the elementary process. They do not come from entity action statements in the Action Diagram.

A cell may contain one letter only and shows the highest value if more than one action is involved:

- Create takes precedence over Delete.
- Delete takes precedence over Update.
- Update takes precedence over Read.

Consider the following illustration:

Cell Values: = Not Referenced C = Create D = Delete U = Update R = Read	Entity Type	COURSE	SEGMENT	ITEM	VERSION	PRESENTATION	SESSION	CUSTOMER	STUDENT	REGISTRATION	ATTENDANCE	PAYMENT	LECTURER	SUBJECT	ROOM
Activity															
AGREE COURSE OUTLINE		C	C											R	
PREPARE SEGMENT		U	R	C	C								R		
PRODUCE SEGMENT			R	U									R		
MAINTAIN SEGMENT			U	R	C								R		
PLAN PROGRAM		R	R			C	C						R	R	R
ADVERTISE		R				R								R	
DIRECT MAIL SHOT		R	R			R		R	R					R	
SELL PRESENTATION		R	R			C	C	C						R	
REGISTER ATTENDANCE						R		C	C	C					
PRESENT SESSION									R	R	C		R		R
GRADE STUDENT									R	U	U				
CANCEL PRESENTATION						D	D			D					
PREPARE INVOICE															
RECEIVE PAYMENT								R		R					
HIRE LECTURER													C	R	

This matrix reveals several problems within a current model:

- Two processes create the entity type version. It is worth examining whether the two processes should be combined.
- The entity type payment is not used. Either it is not required or the processes that use it are incorrectly defined.
- The process Prepare Invoice uses no entity types. According to the matrix, the process does nothing.

Using the Elementary Process/Relationship Matrix

- Each row in the matrix represents a process.
- Each column represents a relationship.
- A cell may contain only one letter. Letters indicate action types.

The following illustration shows an example of this type of matrix.

Key: a = associate d = disassociate t = transfer Activity / Entity Type	segments about subject	course consists of segment	segment of system	item development version	lecture prepared version	course presented presentation	presentation consist of session	items used for lecture	lectures presented by lecturer	customer commissioned present.	customer employed student	student applies by registration	presentation requested registration	registration permit attendance	attendance at session	customer makes payment	payment covers registration	lecturer can teach subject
AGREE COURSE OUTLINE	a	a																
PREPARE SEGMENT	t		a	a	a													
PRODUCE SEGMENT			t	t	t													
MAINTAIN SEGMENT			d	a	a													
PLAN PROGRAM		t				a	a	a	a	a								
ADVERTISE	t	t				t												
DIRECT MAIL SHOT	t	t				t						t						
SELL PRESENTATION	t	t				a	a	a			a							
REGISTER ATTENDANCE												d	a	a				
PRESENT SESSION									t	t			t			a		
GRADE STUDENT													t	t	t	t		
CANCEL PRESENTATION						t	t					d		a				
PREPARE INVOICE											t	t	t	t				
RECEIVE PAYMENT																a	a	
HIRE LECTURER																		a

The example in the illustration raises several questions about the current model:

- The relationships presentation consists of session, and course presented as presentation, are associated in two processes Plan Program and Sell Presentation. This double association suggests potential duplication.
- The matrix shows transfers involved with many relationships in the model, whereas most relationships are, in fact, fixed. You must investigate this discrepancy.

Using the Elementary Process/Attribute Matrix

Each row represents a process, and each column represents an attribute. This matrix includes attributes from one entity type only. Letters in cells indicate action types.

Consider the following illustration.

Key: s = set or change value r = reference x = remove a value Entity Type Activity	session number	session name	course number	course presentation sequence number	session presentation date/time	room	lecture number	lecturer	session lecturer notes (text)	session student notes (text)	session abstract for publicity (text)
AGREE COURSE OUTLINE											
PREPARE SEGMENT	s	s	s								
PRODUCE SEGMENT								s	s		
MAINTAIN SEGMENT		s						s	s		
PLAN PROGRAM	r	r	s	s	s	s					
ADVERTISE	r	r	r	r	r		r				s
DIRECT MAIL SHOT							r				r
SELL PRESENTATION	r	r	r	r	r		r				r
REGISTER ATTENDANCE			r	r	r						
PRESENT SESSION									t	t	
GRADE STUDENT	r	r	r		r		r				
CANCEL PRESENTATION				x	x	x	x	x			
PREPARE INVOICE											
RECEIVE PAYMENT											
HIRE LECTURER	r	r	r	r	r						

The example in this illustration raises questions about the current model:

- The process Cancel Presentation removes the attribute Lecturer Number, but no value has been given to the attribute. It has not been set.
- The processes Agree on Course Outline, Present Session, Prepare Invoice, and Receive Payment use no attributes. Is this consistent with the Elementary Process/Entity Type Matrix? If so, what do the processes do?

Checking for Consistency

Consistency checking confirms that the analysis model conforms to CA Gen rules. Where a model does conform to these rules, it is said to be in a consistent state.

CA Gen automates consistency checking of the model at the data, activity, and interaction levels. It can selectively perform the checks that are applicable to analysis.

After consistency checking is performed, the analysis model can be assumed to be consistent with CA Gen rules. However, a model that is in a consistent state may still not fully and accurately represent the business area.

When to Perform Consistency Checks

A usual sequence of events is for you to:

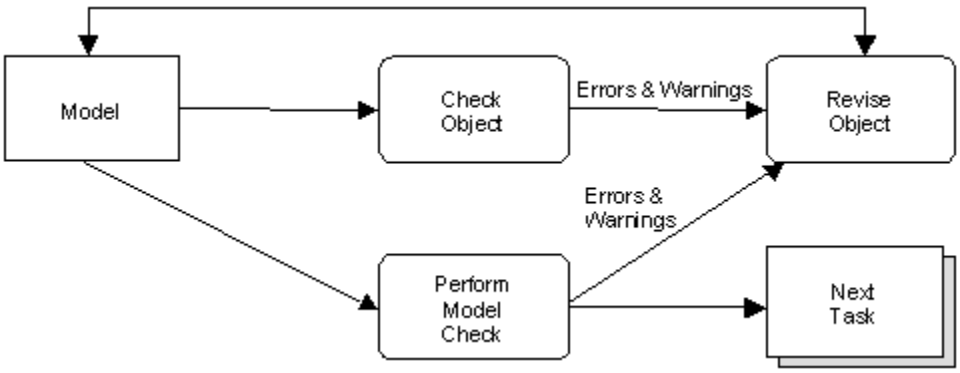
- Develop, in parallel, the data and activity aspects of the analysis model
- Perform checks at intervals
- Analyze interactions and the completeness of the model

Because the modeling activities also enable you to detail interactions between activities and data, it is normal to require a progressively more detailed and consistent model of the data as a prerequisite to interaction analysis.

It is useful to perform consistency checks on the data and activity objects at regular intervals during analysis, and especially before:

- Completing an aspect of analysis and beginning a new analysis activity, for instance, developing a progressively more detailed and consistent model of the data and activities before moving to interaction analysis.
- Review
- Moving to design
- Business system definition

These tasks are shown in the following illustration.



Consequences and Levels of Inconsistency

The Analysis Toolset automates consistency checking at the level of a model, and for model objects you select. It reports the rules to which elements of the model do not appear to conform.

Each problem encountered is classified according to likely severity. This level of severity affects whether system development is able to proceed.

These are the levels of severity, listed in the following table from greatest to least severity.

Severity Level	Description
Warning	Conditions indicating that context is important. For example, if X is required before Y, a warning on Y appears, but you may still be able to perform transformation and system and database generation.
Severe Warning	Conditions in the model that have a greater degree of severity than warnings. In most instances, severe warnings cause errors during system implementation.
Error	Conditions that must be corrected before the model moves to the next stage of system development.

Resolving Inconsistencies

When CA Gen reports inconsistencies at all levels of the analysis model, it lists the current anomalies at a given point in time.

An inconsistency may have a cascade effect and generate other consequential inconsistencies in different parts of the model. It is therefore sensible to approach the resolution of inconsistencies in a structured and ordered manner, concentrating first on a model object and subsequently expanding to the neighborhood, diagram, and model.

The following sections suggest a sequence for resolving inconsistencies. Following this sequence maximizes the likelihood of resolving consequential inconsistencies and minimizes iterations of the checking and revision cycle.

Data Model Inconsistencies

- **Entity Properties**—These are usually at the warning level, relating to the metrics of the number of occurrences.
- **Attributes**—These are inconsistencies relating to attribute definition of domain, field size and the definition of permitted and default values.
- **Relationships**—These are warnings related to the cardinality and optionality of the relationship and about the existence of many to many and one to one relationships.
- **Entity identifiers**—The definition of attributes, relationships, or both, as the identifier for an entity can only be performed if the correct attributes and relationships exist.

Attributes must exist in the correct domain and not be derived.

Optional relationship must not be used as identifiers.

Failing to record attributes for an entity type generates several consistency check messages, which are presented in the following table.

Type of Message	Message
Error	Each entity type must have an identifier to use the DSD.
Severe Warning	Each entity type must have an attribute for codegen.
Error	An entity type with no subtypes must have at least one identifier.
Error	An entity type must have at least one attribute or relationship.
Error	Each occurrence on an entity type must have at least one attribute or two relationships.

The subsequent definition of attributes satisfies a number of these inconsistencies. Checking the data model again produces the messages shown in the following table.

Type of Message	Message
Error	Each entity type must have an identifier to use the DSD.
Error	An entity type with no subtypes must have at least one identifier.

Once an identifier for the entity type has been recorded, then the inconsistencies will have been resolved, whether that identifier includes attribute(s) and/or relationships.

Clearly, the resolution of inconsistencies in the suggested sequence does minimize the number of iterations of the checking and revision cycle, maximizing the number of consequential inconsistencies resolved.

Activity and Interaction Model Inconsistencies

- **Activity hierarchy**—Warnings relate to situations where non-elementary activities do not decompose into more than one lower level activity.
- **Properties and usage properties**—This includes activity occurrence metrics, their suggested implementation mechanism, and whether they are repetitive, elementary, or both
- **Elementary processes**—Expected effects, process logic and information views should be detailed for all elementary processes that are to be included in a business system.
- **Expected effects**—The correct definition of expected effects, throughout the hierarchy, with a manual check that the sum of the expected effects of subordinate activities is equivalent to the expected effects for their parent activities. This is necessary for confirmation of decomposition and the identification of the expected interactions elementary processes have with entities.
- **Elementary process dependency**—The modeling of process dependencies cannot be performed with any degree of consistency unless all the previous stages of activity modeling have been made stable. Only the correct definitions of elementary processes and expected effects allow inconsistencies at this level to be resolved.
- **Information views**—Once process dependency is performed, the inclusion of required information views to satisfy dependency information view requirements should form the final series of consistency checks and result in a consistent activity model.

Activity objects such as elementary processes are checked for consistency with other processes in the hierarchy, events, dependencies, and external objects.

For example, in the Activity Hierarchy Diagramming Tool, failure to properly decompose activities to elementary process level, and to completely record the expected effects those activities will have on entities, generates several consistency check messages. These may include the messages shown in the following table.

Type of Message	Message
Warning	A function must decompose into at least two activities.
Warning	A non-elementary process must decompose into at least two processes.
Warning	The expected effects of CRUD for a subordinate activity should be expected of its parent activities.
Warning	An entity type must have at least one attribute or relationship.
Warning	The expected effects of CRUD for a parent activity should be expected of its subordinate activities.

The correct decomposition of activities allows you to correctly determine expected effects for activities. If decomposition is incomplete or inconsistent, then expected effects also reveal inconsistencies.

Once decomposition is completed, the recording of consistent Expected Effects is straightforward. The following table lists messages that would typically be produced in this situation.

Type of Message	Message
Warning	The Expected Effects of CRUD for a subordinate activity should be expected of its parent activities.
Warning	The Expected Effects of CRUD for a parent activity should be expected of its subordinate activities.

Expected effects inconsistencies can be resolved by matching the expected effects of parent activities with those of its subordinates. The expected effects of a parent activity must always be composed of the sum of the expected effects of all its subordinate activities.

Inconsistencies may still exist, however. These inconsistencies may not necessarily be due to inaccurate modeling or understanding of the business. You must nevertheless confirm that any inconsistencies still in existence at the end of the analysis phase are not likely to impede further work.

Checking for Correctness

You perform correctness checking to confirm that the analysis model accurately represents the business requirements. The project can then use this verified model in subsequent stages of system development.

In particular, you address these questions to ensure a full and accurate representation:

- Have attributes been grouped with the correct entity?
- Does the model exclude all unnecessary elements?
- Does the model look right to the end users?

Correctness checking includes these techniques:

- Process dependency checking
- Normalization
- Redundancy checking
- Quantity cross-checking
- Structured walkthroughs

Process Dependency Checking

You use process dependency checking to verify the activity portion of the analysis model by confirming the following:

- Every elementary process appears in a Process Dependency Diagram (optional).
- Every elementary process is dependent on at least one of an event, an import view, and another process.
- Every elementary process has entity actions defined and at least one input and output. An output may be an export view or an entity view associated with a Create, Delete, or Update action.
- Process Dependency Diagrams are consistent with one another.

Normalization

Normalization is a step-by-step process used to ensure that you have assigned each attribute to the proper entity type and defined sufficient entity types. This technique is more fully described in the chapter titled "Analyzing Data."

Normalization can be verified based on understanding what users accept are the functional dependencies of attributes. Without using normalization terminology, you check with users to ensure that:

- Every attribute is dependent upon the key (1NF), the whole key (2NF), and nothing but the key (3NF).
- Every collection of attributes is a 3NF entity type.
- Every entity type is a 3NF collection of attributes.

Redundancy Checking

You use redundancy checking to search for unnecessary attributes and relationships.

Attributes and relationships are unnecessary when they already appear in the model under some other guise or can be deduced from other objects in the model.

You also identify overlapping entity types and duplicated processes.

Quantity Cross-Checking

You use quantity cross-checking to check the consistency of all quantitative information gathered about objects in the analysis model.

In particular, you check two items:

- Relationship cardinality

  When two entity types are associated, the entity volume figures must be consistent with the cardinality of the relationship associating them.

  For example, if customer has 15 occurrences, and order has 150 occurrences, the cardinality of the relationship between customer and order should average 10 (150 divided by 15).

  The cardinality of a relationship depends on the optionality of the entity types it associates. The following illustration shows the cardinality of a relationship between entities A and B under three conditions.

Condition	Average Occurrences of B Associated With Each A:
A and B are both mandatory	Is equal to B divided by A
A is optional and B is mandatory	Is less than B divided by A
A is mandatory and B is optional	Is greater than B divided by A

- Subtype volume

  The number of entities of a given type cannot exceed the combined total of maximums for all subtypes in a partitioning. Each mandatory partitioning must have the same combined total.

Structured Walkthroughs

Structured walkthroughs are formal, step-by-step inspections of analysis results.

You present the walkthroughs verbally to the business personnel most familiar with the business area being analyzed.

Structured walkthroughs can be used at any stage of analysis.

Within correctness checking, the walkthroughs help users to verify that the analysis model correctly and completely reflects the business needs. It is especially useful for the following:

- Entity Relationship Diagram
- Activity Hierarchy Diagram
- Comparison Check Report
- Stability Analysis Report

During the walkthrough, those who developed the model systematically explain the meaning and interpretation of the model, the modeling decisions that have been made, and the impact of those decisions.

Participants check, discuss, and agree on each element of the model.

The level of detail varies according to the interests of the business participants.

Analyzing Model Stability

Stability analysis is the study of the impact of potential business changes on the analysis area, especially on the entity and process models.

During this analysis, you identify modifications that will result in a resilient analysis model. One that will not need to change, even when business requirements change. Although total resilience may never be achieved, you can and must strive to make the model as resilient as possible.

These are the steps for analyzing analysis model stability:

1. Identify likely changes to the business.
2. Study the impact of such changes on the analysis model.
3. Modify the model to lessen the impact of possible changes.
4. Prepare Stability Analysis Report (if necessary) describing likely changes to the business, their impact, and any modifications to the model.

Modifications to the model could include additional data, as well as representing as data those business rules that are subject to variation.

For example, order authorization limits can be modeled as an entity type authorization, relationships with employee, order, and purchase order. This is preferable to expressing rules as part of process logic.

Refining the Project Plan

After completing any major aspect of analysis, and especially when finally confirming an analysis model, you should ensure that the analysis results are reflected as needed in the system development plan. This may involve:

- Selecting elementary processes to implement
- Verifying process clusters, finalizing business system boundaries, and defining business systems using CA Gen
- Determining the business system implementation sequence
- Determining the process implementation sequence within each business system

Selecting Elementary Processes to Implement

You need to confirm the continued business benefit of providing information system support to all elementary processes. If their implementation cannot be justified, they should be removed from the implementation plan.

The technique for identifying low-benefit processes is straightforward, requiring the following steps:

1. Before changing the implementation plan, check Dependency Diagrams to discover if processes of otherwise questionable benefit may be a prerequisite to many high-benefit processes; such processes should remain in the plan.
2. Review the clustered Entity Type/Elementary Process Matrix and the Process/Role Matrix for the remaining questionable processes, to determine whether failure to implement any of the processes will result in logical inconsistency.

   For example, if a low-benefit process creates an entity later referenced by many high-benefit processes, the cost of not implementing the process may be greater than the cost of implementing it.

   Omission of low benefit processes may weaken data or process support for a role that performs high-benefit processes.
3. Review current system and data assessments and matrices against the data and elementary processes supported by those systems. This type of review is described in the chapter titled "Analyzing Current Systems."

   These assessments and matrices may indicate that data is already available and does not need new system support to maintain it.

   Any elementary processes that exist only to maintain that data may be removed from the implementation plan.
4. Developers specifying business systems to be implemented in a client/server environment may find it useful to review the clustered Entity Type/Elementary Process Matrix for opportunities to improve data and system reuse.
5. It may be found that:

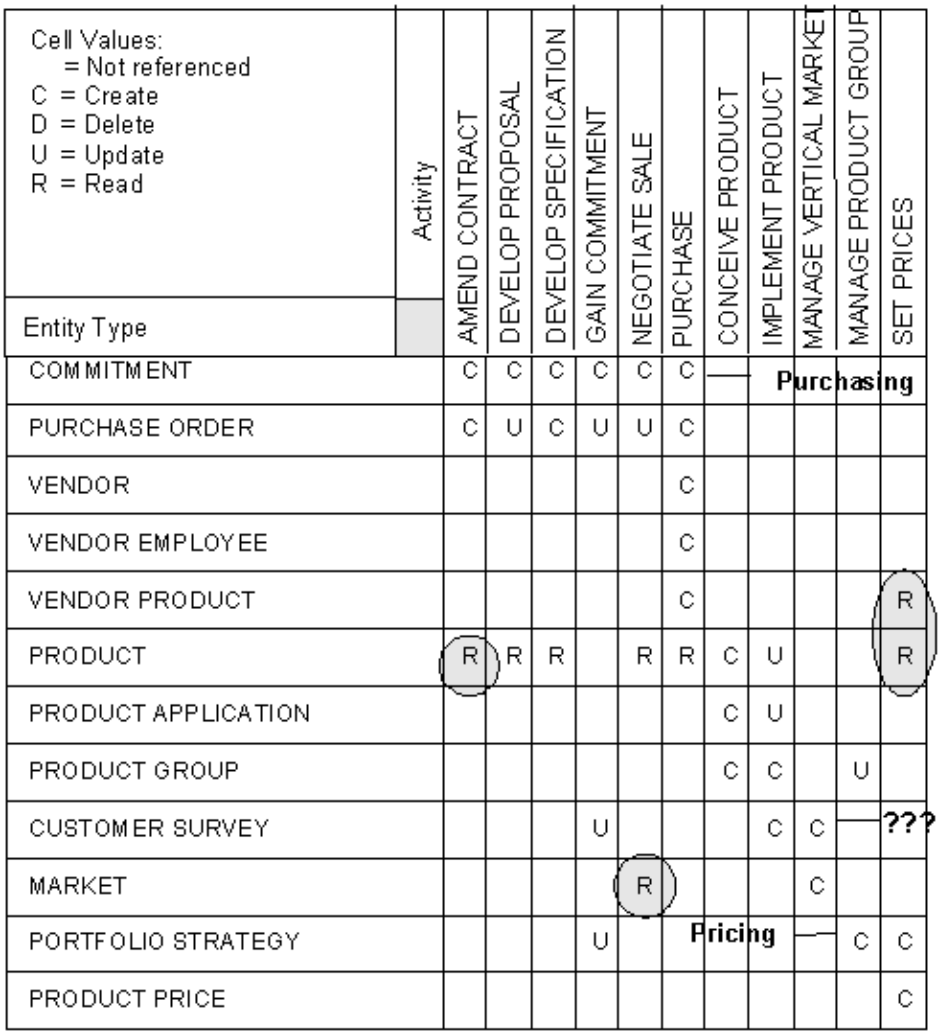
   - Processes closely clustered by their creation and maintenance of data widely used by many other processes are candidates for support by business systems that include server procedures, which can be used by many client procedures.
   - Processes that need many entity types (possibly created by several different clusters) are candidates for support by business systems composed of client procedures that reuse data maintained by a combination of current systems and provided by common server procedures.

Determining the Sequence for Business System Implementation

Barring other considerations, you should implement business systems according to their position in the business life cycle. That is, if any procedure in business system B requires information captured in business system A, then business system A should be implemented before business system B.

This ordering, sometimes called the natural implementation sequence, can be deduced from dependencies in the activity model and from the clustered Entity Type/Elementary Process Matrix.

For an example, see the following illustration.



Read interactions that lie outside clusters potentially represent the need of the activities in one cluster for information created by activities in another cluster. These "reads" represent a transition dependency.

For example, the process Set Prices reads the entity type vendor product, to check vendor product price, which is created by the Purchase process in the Purchasing cluster. If a Purchasing system is implemented before Pricing, this poses no problem.

Sometimes, deviation from this natural sequence becomes necessary. Many non-technical factors can render the natural sequence impractical, such as management and user pressures, the desire to implement higher benefit systems first, and a variety of resource constraints.

For example, Purchasing may be the highest business priority. However, the matrix in the illustration also shows that many of the processes grouped into the Purchasing cluster read product and product price. You should therefore discover whether the information available from current systems is sufficient for the purposes of Purchasing.

Any time there is a deviation from the natural sequence, there is a risk of introducing logical inconsistencies that, in turn, require additional resources to resolve. Consider this additional cost (the cost of implementing systems out of their natural order) whenever such a deviation is proposed and offer the business a choice of implementation strategy, costs, and risks.

Determining the Sequence of Process Implementation

Ideally, the natural implementation sequence (see the Determining the Sequence for Business System Implementation section in this chapter) should be the sequence in which elementary processes are implemented within the first priority business system. This sequence can be determined from the order in which elementary processes are specified on the Dependency Diagram.

In many cases, the natural sequence is ambiguous because of mutually exclusive and parallel dependencies within the Process Dependency Diagram. You can sequence these ambiguous elementary processes by placing those with higher benefits first.

The implementation sequence of processes within a business system is subject to the same non-technical pressure for deviation as the implementation sequence for business systems themselves. As with business systems, you need to address and acknowledge the cost of introducing logical inconsistencies before finalizing the decision to deviate from the natural sequence.

Results of Analysis Confirmation

The results of confirming an analysis model include:

- Additions to the analysis model to show the support of entity types by current data stores, and optionally, of elementary processes by current system procedures
- Verified, resilient analysis model
- Comparison Check Report which includes matrices, as appropriate, comparing elements of the analysis model with current systems and data stores
- Interaction matrices that check processes against entity types and (optionally) relationships and attributes

- (Optional) Stability Analysis Report that:
  - Tabulates the changes postulated
  - Outlines the impact of each change on the analysis model
  - Describes the modifications made to cope with each change
- (Optional) Minutes of walkthroughs that record agreement of business participants on the results of analysis, and any change actions
- Identified or refined business system development actions and an implementation sequence for the development project, using the results of clustering and cost/benefit analysis
- This definition may include an implementation sequence for elementary processes in the first priority business system.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W