XML DESIGN RULES

Background

These design rules have been developed to assist in establishing uniformity in X12 XML business document development and maintenance efforts. The design rules are to be used as stated. There shall be no deviation from them, with all new development and maintenance expected to utilize them.

These design rules uses as its basis the philosophical foundation and general design principles forwarded in the published technical report, ASC X12 Reference Model for XML Design and its description of the Context Inspired Component Architecture (CICA).

This document is intended to be flexible as new situations arise that require new rules or guidelines, or modifications to existing ones. Therefore, individuals involved with XML business document development and maintenance are urged to keep the latest edition of the Design Rules document handy for reference. Modifications and suggestions for improvement of this guideline are welcome and should be sent to Task Group 1 of Subcommittee J, Technical Assessment, care of:

Secretariat, ASC X12
Data Interchange Standards Association, Inc.
333 John Carlyle Street • Suite 600
Alexandria, VA 22314-2852
(703) 548-7005

Table of Contents

PART I – Context Inspired Component Architecture (CICA)

1. Context Inspired Component Architecture

Introduction

The Context Inspired Component Architecture (CICA) offers a method for building electronic business messages using XML. CICA takes the lessons from two decades of EDI experience, and updates that experience to take advantage of XML's flexibility.

Like EDI, CICA presents a standard structure for business messages, and adds in the implementation details required by different industries. However, CICA also allows organizations to apply the same methods for messages used across industries. Thus organizations can relate the terms used in their own industries to potential trading partners in other lines of business. To achieve these somewhat contradictory objectives (consistent structure, with flexibility) presents technical challenges. CICA needs to manage a much larger quantity of information in various contexts, while maintaining a consistent approach to encourage interoperability across those contexts.

The purpose of this section is to provide an introduction to the CICA. It will explain the most basic and important concepts behind the framework and its components. In an attempt to convey the core concepts, we may at times overly simplify some of the explanations. However, with a strong grounding in the core concepts, a more detailed and complex level of understanding can ultimately be achieved.

The general framework of the CICA can be simply stated. A template is created for a business process. A template consists of slots. The slots may be filled with a number of different modules. Modules may be broken down into a number of sections including assemblies, blocks, components and primitives. A unique instance of the completed framework is called a document.

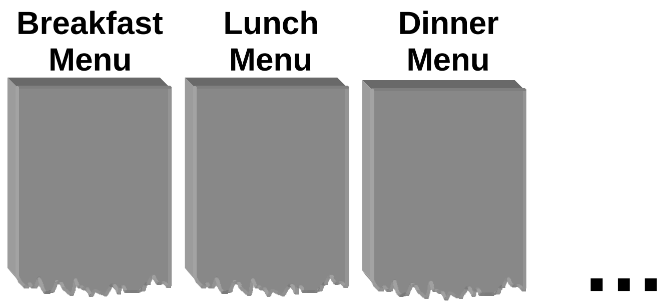
CICA 101

In order to illustrate some of the concepts and relationships, we will use an analogy that many people are familiar with, a restaurant menu.

Many industries face the need to organize and present information in a structure, while allowing for maximum flexibility. In a restaurant, this need expresses itself in creating a menu for its customers. Almost every restaurant, from the corner diner to the gourmet establishment, offers a variety of items. The issue facing the restaurant is how to organize this list of items to be meaningful to the customers. The restaurant could organize food items in a number of different ways: alphabetically, by the main ingredients, by the type of dish offered, by the meal in which they are offered, or the course within the meal.

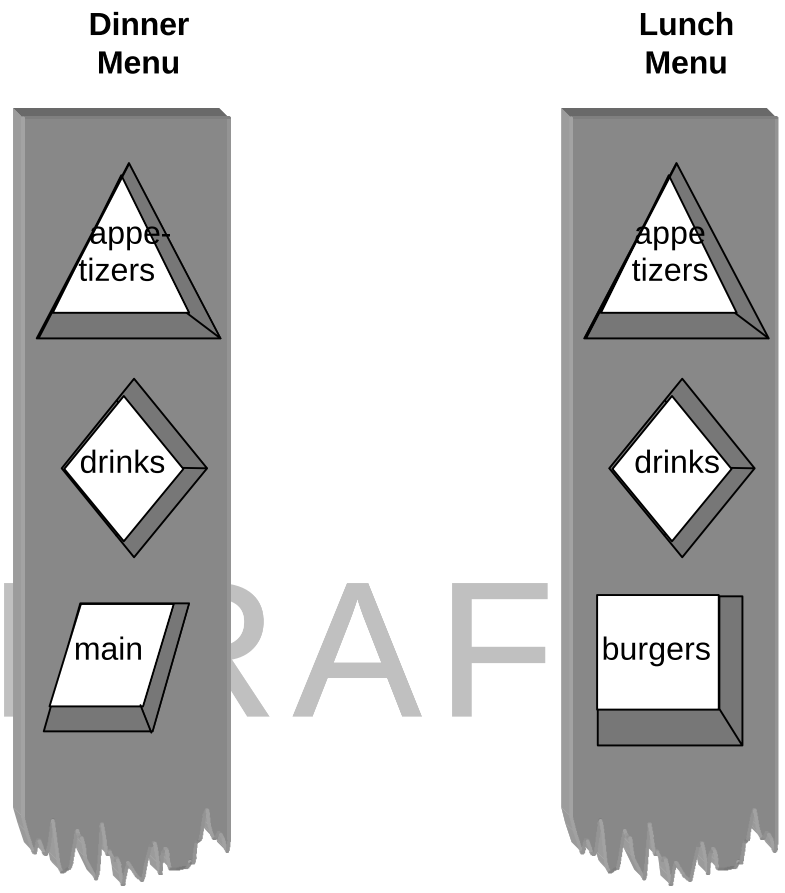
The restaurant might begin by breaking down its offerings into three distinct menus, one each for breakfast, lunch, and dinner. In the CICA context, these menus are analogous to templates. They are similar in that they convey the food offerings to the customer, but are different as they reflect the food preferences generally associated with the different times of day (Figure 1).

Figure 1 Basic menu structure



Within each menu, one can identify different courses. The courses might include an appetizer, main dish, dessert, and beverage. In the CICA context, slots are analogous to the courses reflected on the menu: a slot for appetizer, main dish, dessert, beverage, etc. Some of these may be shared across the different menus. For example, the both the dinner template and the breakfast would probably have a slot for main dish, where as the breakfast template would probably not have a dessert slot. Both may have a beverage slot (Figure 2).

Figure 2. Menu organization



Types of dishes or course selection within a meal can include meat, fish, vegetarian main dishes or hot or cold appetizers. Under each of these categories then come the individual selections. The individual dishes that we use to fill the main dish slot are similar to modules. Each course slot may be filled with a number of modules that are appropriate to the menu template.

But the breakdowns do not end at the individual selections. The selections will generally have prices associated with them as well as more detailed information about the selection, for example a restaurant may offer appetizers, soup, vegetables, and a salad *with* the main course. In other establishments, patrons can specify the type of preparation or amount of seasoning they prefer, or they can indicate substitutes if on a restricted diet. Even on a side dish, the choices can proliferate, for instance selecting the type of dressing on a salad (Figure 3).

The server, in taking the order will list the selections chosen by the customer, and create an individual order. In many cases, the different choices can be mixed from one order to another, thus orders will have different appetizers, soups, salads, main course, and beverages, or just soup and salad, and a main course, or just soup and salad. The idea is to build a meal with interchangeable menu items or components (Figure 4, next page).

Figure 3. Reusing menu selections

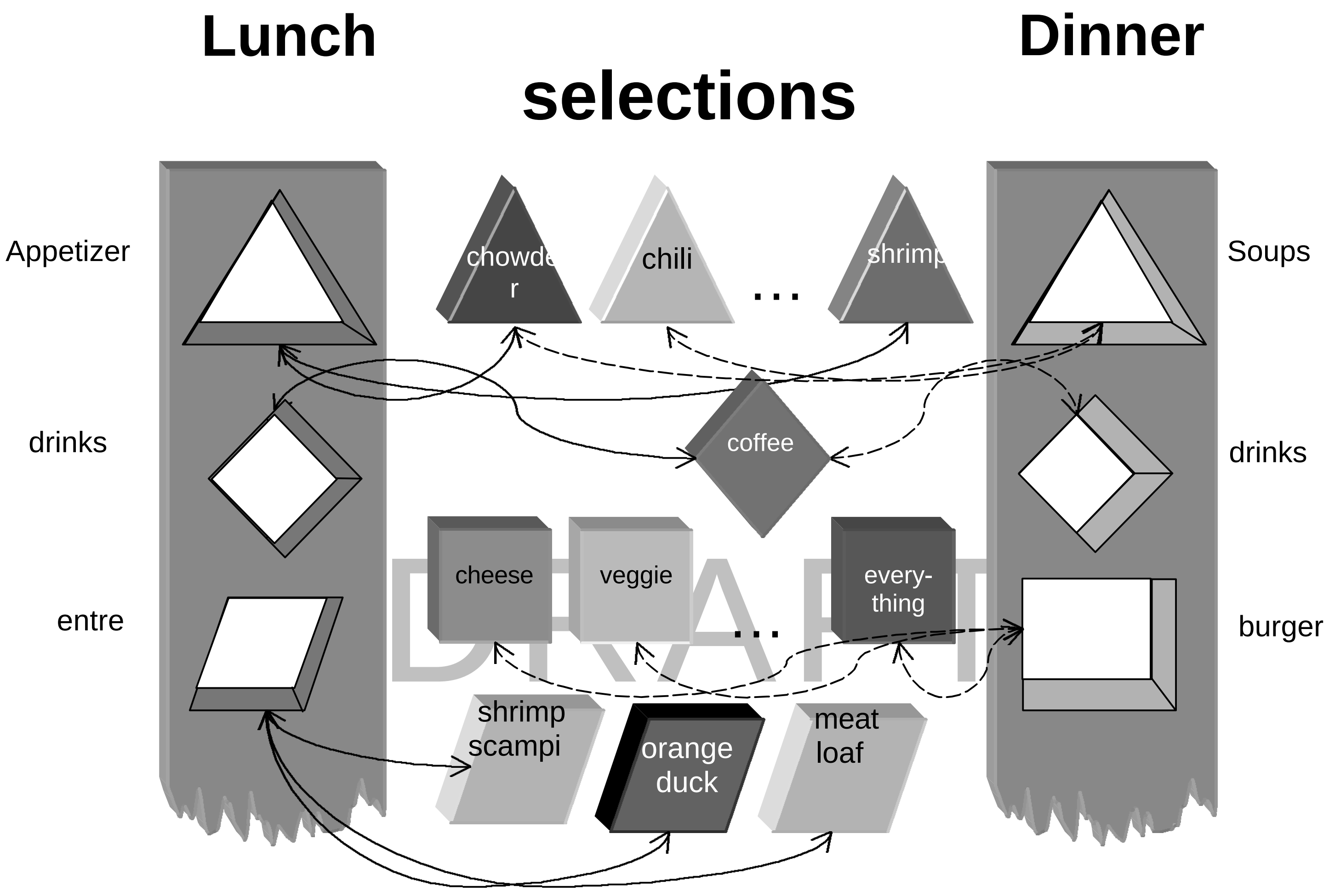
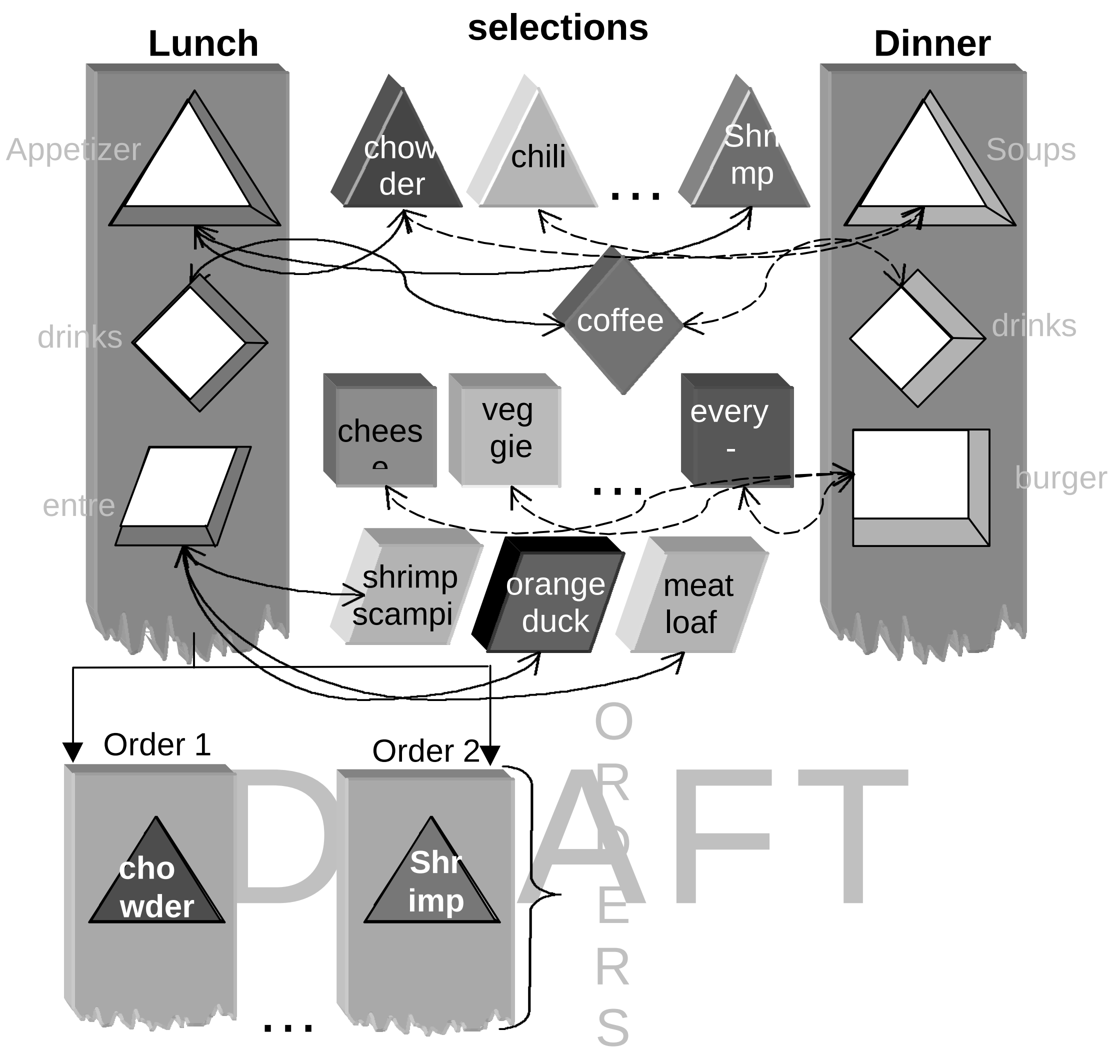


Figure 4. Creating customer orders with reusable items



While writing a restaurant or food service menu may seem light years away from building electronic business messages, many of the same principles apply:

- A disciplined underlying structure – menus for all of the meals follow the same basic hierarchy, making it easier to generate new menus
- Encouragement of item reuse – e.g., the side-salad with dinner is also offered at lunch; the cooks can use the same ingredients and methods, thus simplifying their work
- A structured approach to reuse – cooks planning the meals know in advance the extent of reuse available, and servers know the limits of substitutions; for example, breakfast and lunch generally have fewer courses than dinner, limiting the items available at the earlier meals

With electronic business messages, these same principles can achieve comparable benefits in organization and simplification. A disciplined underlying structure provides a basic hierarchy for electronic business messages, much like it does for menu planners. The structure provides the cook with a way of organizing menus, but does not limit the creativity of the final product. Likewise with e-business messages, having an underlying structure offers a way for trading partners to consistently organize the data they exchange, but does not limit the kinds of business that the trading partners can undertake.

The reuse of items enables both the menu planners and e-business message architect to simplify their efforts and make better use of their resources. If the food service planners can provide to customers a wide selection of items by offering variations on a basic set of ingredients, they meet the customers' needs for variety while simplifying their work and probably making better use of their time. With e-business messages, the use of interchangeable data items offers a way of making maximum use of IT resources and simplifying the message contents. The CICA architecture builds electronic business messages by choosing from the interchangeable or reusable items and combining them at different levels.

In building business messages, the message uses a process analogous to the restaurant menu. The message collects components, important pieces of information, needed by the trading partner to perform actions beneficial to the parties to the transaction. In most cases, the industry in which the companies operate will have processes and terminology defined, much like the way the business environment of a food service defines the meals it serves: a fancy expense-account restaurant will not likely serve breakfast, for example.

One can draw parallels between the hierarchy in restaurant menus and the CICA architecture:

Menu hierarchy	**CICA architecture**
Customer order (baked salmon, ranch dressing on salad)	Document (event invoice goods)
Menu (lunch menu)	Template (event invoice)
Menu category (lunch, main course)	Template slot (Line Product)
Menu selection (baked salmon, includes side salad)	Module (Catalog Goods)
Dish combination (baked salmon plus side salad)	Assembly (Commodity Goods)
Completed dish (baked salmon)	Block (Product)
Food item (salmon)	Component (Product ID)
Item ingredient (dressing on salad)	Primitive (ID Number Type)

In both cases, the structure defines an overall hierarchy for the operators to organize their menus on one hand or goods invoice on the other. The interchangeable parts make it possible to reuse items on different menus or in other business messages. A company could use the unit price block, for example, in various messages, such as quotation or purchase order, as well as invoice.

2. General Information and Rules

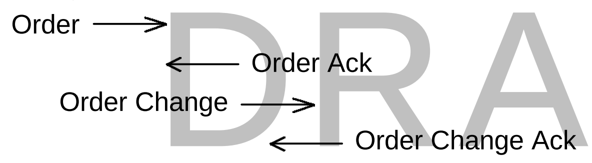
The information in this section applies to the entire document, and is placed in this section to assist in the general overview. The information may repeat in subsequent sections.

Business Process

In CICA, the use of 'business process' is used in the most general sense of the term. In this context, business processes are general from two different perspectives. Business processes are both industry independent and independent of specific message flows. Business processes are bigger than a message, that is, completion of a sequence of steps is required to constitute a business exchange. This does not, however, require that a business process include multiple business documents, it simply acknowledges the fact that some business exchanges use a combination of external mechanisms with 'silence equaling acceptance' an understood part of the business process.

In this context, business processes refer to 'door-to-door' processes limited to the public or exposed portions of the process external to the organization's firewall, if you will. When an organization receives a request, it might carry out any number of internal steps to determine, for instance, whether it wants to do business with at trading partner, or whether it can do business with that trading partner. The steps an organization takes to make such decisions are internal and proprietary to the organization often resulting in competitive advantage. These internal steps are not considered within the scope of the CICA definition for business process. Of course, once a decision has been made, the organization's response as to the request is very much a part of the business process.

This leads to the balance between how general and how specific. In other words, when are two business processes 'the same' and when are they 'different. The question is illustrated in the following set of messages.



From this small set of messages, several specific sequences can be produced, as follows.

1. Order, Silent Ack, done
2. Order, Order Ack, done
3. Order, Order Change, done
4. Order, Order Ack, Order Change, Order Change Ack

As illustrated above, even when there are a variety of sequences that can be followed it does not necessarily represent a different order process. Conversely, having the same sequence of messages does not imply the same business process, for example a situation with orders between a production supplier and its customers versus casual orders between trading partners without a prior business relationship. So, where does that leave us? A set sequence of messages does not guarantee us a unique business process action. A different sequence of messages does not guarantee us that a message is different. So, what does? A set of indicators,

- Pre conditions/state
- Trigger Event
- Post conditions/state

together act as indicators determining whether a message is unique, or the same as other business process usages. This test helps determine the set of UNIQUE messages, based on business process. For each of these, in CICA, Templates are created that is, they are specific to a set of key indicators that are used to determine use/uniqueness.

This is a significant departure from X12 current EDI practices. The X12 philosophy is to have a single transaction set for each business purpose or action for example a single Invoice. While in practice this is not actually done, it remains the prevailing X12 philosophy. CICA recognizes the value of and has as its objective to utilize a single means to fulfill a single purpose. Dissimilarities create complexity, which CICA is equally committed to minimize.

Two primary factors lead to complexity in X12's EDI Transaction Sets:

1.) Supporting the needs for multiple business processes within the same Business Document.

General business functions, such as invoicing, are my many perceived to be the same business process – an invoice is an invoice. This is not necessarily the case with the differences in business circumstances associated with the generation of one invoice can be different than that of another, affecting the high level organization. Having to support dual business processes with the use of a single business document leads to ambiguous business message design often resulting in more than one position to place the same information. This resulting complexity is dealt with by CICA in the following ways:

a. each business process different need for a different high level organization gets a different Template

b. Templates only contain Slots, and loose Modules are constructed to fit into those Slots. This allows the same Module to be used in multiple Templates, minimizing the cost of having Templates. This design is dramatically different for current EDI practices.

2.) Differences in information needs due to the differences in products and/or services that are the subject of the business document.

CICA has an elegant solution for this, allowing support for details sufficient to meet the needs of implementation.

a. Modules can be built out of reusable constructs, which can be fully customized to meet the needs of each user.

b. Modules coexist within CICA in that multiple Modules are linked to the same Slot, allowing for peer Modules to be associated with the same role or purpose.

c. This granularity approach further provide for the natural information flow through a business process, where the same information appears within multiple messages within the process. The same Module therefore can be used in many business documents.

Structure Rules Overview

There are three tests that can be applied when comparing two candidate information constructs to determine the level to which they are related. These are Form, Fit and Function, and they are taken from the Parts world where they are used to determine when a new part number needs to be assigned. These tests, while the same for each CICA construct, have slightly different implications depending on the semantic abstraction of the construct. Modules, the most semantically specific construct, are more sensitive to purpose and usage and a little less impacted by structure. In contrast, Blocks contain abstract semantics and are affected more by structure. These details affect how to apply these tests and the resulting rules. The general concepts are presented below.

For eBusiness considerations, Form, Fit and Function are defined as follows:

2.1.1. **FORM**: Physical – the structure, contents and components of the information structures being specified. For example, parts have names and so do people. People have first, middle and last names, whereas a part has a single name, part name. The difference in Form makes these two types of names different. In contrast, you might have a Student First Name and Student Last Name, compared with a Patient First Name and Patient Last Name. Form-wise, these two examples are the same.

2.1.2. **FIT**: Identity-Meaning-Specificity – Two organizations or industries that share the common element named Part Number have reason to believe that there is some commonality. Sometimes two uses of an identically named item do not provide the same level of specificity, and therefore these items are not the same thing. In ebXML, a case using a Vehicle Identification Number, "VIN" was used. Different organizations use the VIN, but they may be referring to a different subset of sub-components. Each subsection of the component parts of the VIN, for the same vehicle, is different information. Can all of these different subsets of the same base number all be called VIN – no! Other examples are found with Part Number, with different levels of specificity found with a construct called Part Number. For these to be considered the same, they must specify the same level of specificity.

2.1.3. **FUNCTION:** Purpose or how used. – When comparing two information constructs, such as Product, there is a common purpose or usage – which motivates treating them as 'the same', even though the detail used to specify various Products can vary widely. In some cases, the various Product descriptions are similar in form, but in many instances, this is not the case. Efforts to merge dissimilar definitions results in ambiguity, which later needs to be disambiguated. In the CICA architecture, through the use of abstract layers and links, these Functionally related constructs are associated, without imposing ambiguous merging. The product specification for a Widget is dramatically different from the product specification for Visiting Healthcare services.

Detailed Structure Rules

2.1.4. The levels of equality that are true determines how closely related two information constructs are. Consider the following:

2.1.4.1. Condition 1:

FORM = YES

FIT = YES

FUNCTION = YES

When all three tests are true, then with 100% certainty we can determine that the two are the same thing, the constructs are semantically equal. Examples of this situation are Shipper, Seller, or Supplier. These are different industry-specific terms for a semantically equivalent party playing a role. Frequently the descriptive details are exactly the same; and when that case is true, they are semantic equals in every sense.

2.1.4.2. Condition 2:

FORM = NO

FIT = NO

FUNCTION = YES

When equality is based on function alone, the two information constructs appear below a common parent structure. For example, in the travel industry you have rooms in hotels and passenger seats on flights. Although they are specified with different data elements and are called different things, they are used in the same manner in a business process/message. Thus, the two appear beneath a common parent [at some level], possibly human service products.

2.1.4.3. Condition 3:

FORM = YES

FIT = NO

FUNCTION = NO

This case is very common in EDI today, and is well supported. The X12 N1 loop specifies the name, ID and address of any party, person or organization. The fundamental difference is that in the CICA architecture, Blocks are specified for the various data arrangements [different where a party is an individual versus an organization]. Further, this is independent of whether the construct can represent many purposes, which is the expected case. Therefore, in terms of Blocks, it is expected to have a single block [Party with First, Middle and Last Name] used for many specific parties: Passenger, Patient, or Student.

2.1.4.4. Condition 4:

FORM = NO

FIT = YES

FUNCTION = NO

This is the case where an information construct serves the same semantics in two different settings/business conditions, but it is used differently and has different components.

2.1.4.5. Condition 5:

FORM = YES

FIT = NO

FUNCTION = YES

In the automotive industry, Part Number is used to specify the desired product.

Manufacturer A has a significant digit part number which is really a composite of several identifiers: base + change number + color number + location on vehicle + etc.

Manufacturer B and others have a part number too, but it refers only to the base. Separate additional values are required which include: change number, color number, location on vehicle, etc.

Both of these are related in that they are used to specify THE part, but they are NOT semantically equal. They do not provide the same level of specificity. Therefore, although they are used for the same base purpose, they cannot be used interchangeably and therefore, they are not the same.

2.1.4.6. Condition 6:

FORM = YES

FIT = YES

FUNCTION = NO

This case happens primarily when multiple business processes are involved. Consider a scenario where a Doctor is treating Patients versus a scenario of a business process where a Clinic is communicating its Assets – its staff. In both cases the form and fit are the same, but the function is different. It is unclear what structural implications this case has.

2.1.4.7. Condition 7:

FORM = NO

FIT = YES

FUNCTION = YES

In this case there is a difference in form, as is the case with Person Name versus Organization Name. Both cases are serving the function to specify the Party. Last Name does not equal Organization Name, because they don't deliver the same level of precision. In order to achieve the same level of "Fit", it is Organization Name = Last Name + Middle Name + First Name. Fit ensures semantic equality.

General Rules

2.1.5. **Req Designator** values are one of the following:

- 2.1.5.1.1. **R** = Required
- 2.1.5.1.2. **O** = Optional
- 2.1.5.1.3. **A** = **I**nclusive Or – at least one, can be more than one
- 2.1.5.1.4. **X** = Exclusive Or – one and only one

2.1.6. **MinOccurs** values MUST BE integer numerics equal or greater than zero.

2.1.7. **MaxOccurs** values MUST BE either an indication as unbounded, or an integer numeric greater than zero.

2.1.8. **Names**, **XML Names**, and **Usage Names**, and **XML Usage Names** MUST BE be camelcase.

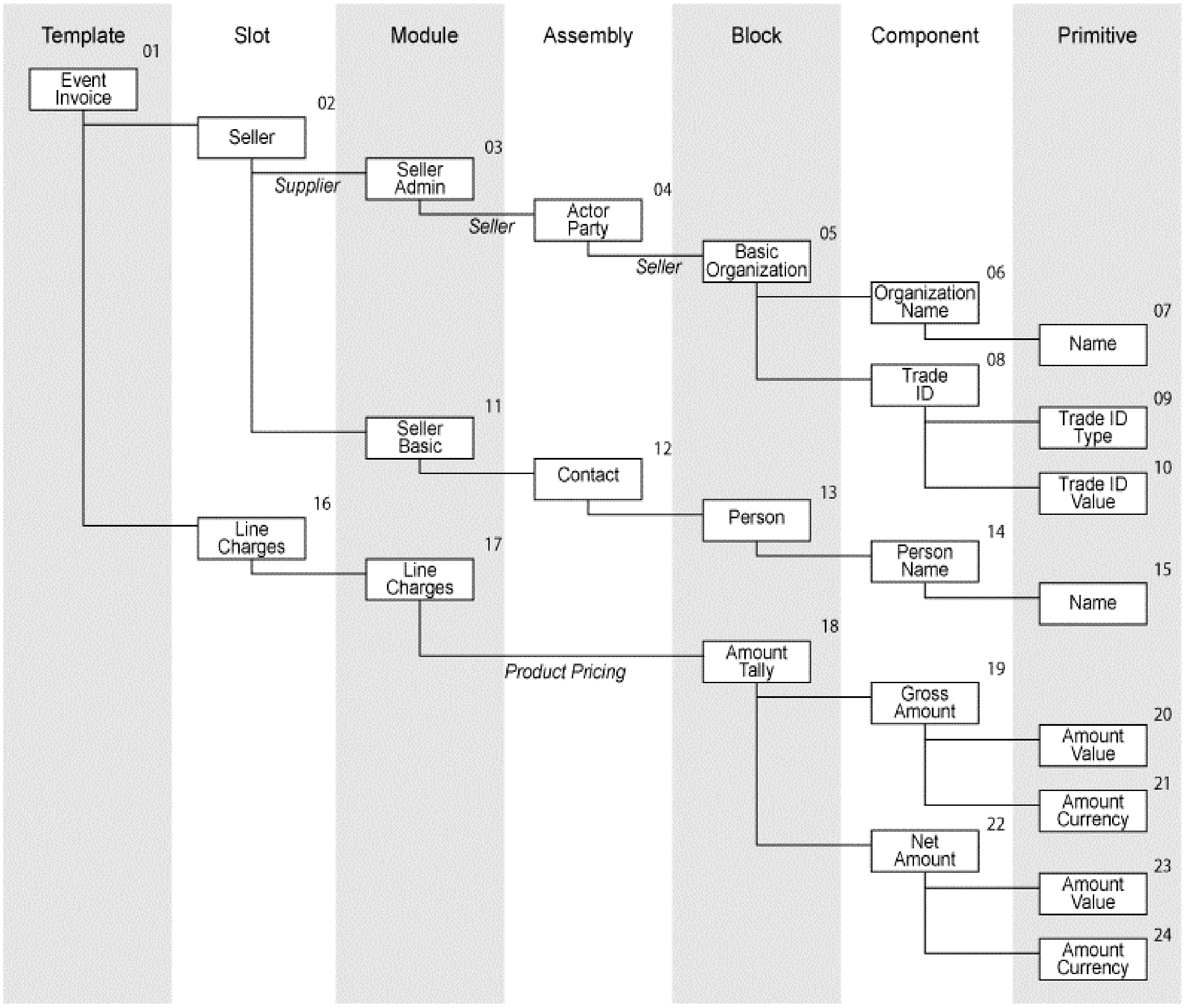
2.1.9. **XML Names** MUST start with a letter (A-Z) and consist only of those characters included in the NMTOKEN datatype as defined in the W3C Schema definition.

2.1.10. **Go for clarity over cleverness!** Clarity and ease of use/adoption are the top priorities.

2.5 Example

For the purposes of this document, a complete, top to bottom example has been constructed in order to illustrate the concepts. Note that each box is numbered in order to easily refer to subsection within the example.

Figure 5, Example.



PART II – CICA Constructs

1. Core Component Type [CCT]

CCT – Definition

CCTs are defined within the CCTS, and are used in their entirety within the CICA architecture. According to the CCTS, a "*Core Component*, which consists of one and only one *Content Component*, that carries the actual content plus one or more *Supplementary Components* giving an essential extra definition to the *Content Component*. *Core Component Types* do not have *Business Semantics*."

CCTs are at the base of the CICA architecture, and are as defined in the CCTS. CCTs are finite in quantity; currently there are 10 CCTs.

CCT – Content

CICA utilizes the CCTS listing of CCTs. Each CCT contains a single value CCTC, where the actual business information is stored. In addition, one or more supplementary CCTCs are defined for each CCT. The table is as follows:

CCT Dictionary Entry Name	Definition	Remarks	CCTC Components
Amount. Type	A number of monetary units specified in a currency where the unit of currency is explicit or implied.		• Amount. Content • Amount Currency. Identifier • Amount Currency. Code List Version. Identifier
Binary Object. Type	A set of finite-length sequences of binary octets.	Shall also be used for *Data Types* representing graphics (i.e., diagram, graph, mathematical curves or similar representations), pictures (i.e. visual representation of a person, object, or scene), sound, video, etc.	• Binary Object. Content • Binary Object. Format. Text • Binary Object. Mime. Code • Binary Object. Encoding. Code • Binary Object. Character Set. Code • Binary Object. Uniform Resource. Identifier • Binary Object. Filename. Text

CCT Dictionary Entry Name	Definition	Remarks	CCT Components
Code. Type	A character string (letters, figures or symbols) that for brevity and / or language independence may be used to represent or replace a definitive value or text of an *Attribute* together with relevant supplementary information.	Should not be used if the character string identifies an instance of an *Object Class* or an object in the real world, in which case the Identifier. Type should be used.	• Code. Content • Code List. Identifier • Code List. Agency. Identifier • Code List. Agency Name. Text • Code List. Name. Text • Code List. Version. Identifier • Code. Name. Text • Language. Identifier • Code List. Uniform Resource. Identifier • Code List Scheme. Uniform Resource. Identifier
Date Time. Type	A particular point in the progression of time together with relevant supplementary information.	Can be used for a date and/or time.	• Date Time. Content • Date Time. Format. Text
Identifier. Type	A character string to identify and distinguish uniquely, one instance of an object in an identification scheme from all other objects in the same scheme together with relevant supplementary information.		• Identifier. Content • Identification Scheme. Identifier • Identification Scheme. Name. Text • Identification Scheme Agency. Identifier • Identification Scheme. Agency Name. Text • Identification Scheme. Version. Identifier • Identification Scheme Data. Uniform Resource. Identifier • Identification Scheme. Uniform Resource. Identifier
Indicator. Type	A list of two mutually exclusive Boolean values that express the only possible states of a *Property*.		• Indicator. Content • Indicator. Format. Text
Measure. Type	A numeric value determined by measuring an object along with the specified unit of measure.		• Measure. Content • Measure Unit. Code • Measure Unit. Code List Version. Identifier
Numeric. Type	Numeric information that is assigned or is determined by calculation, counting, or sequencing. It does not require a unit of quantity or unit of measure.	May or may not be decimal	• Numeric. Content • Numeric. Format. Text

Quantity. Type	A counted number of non-monetary units possibly including fractions.		• Quantity. Content • Quantity. Unit. Code • Quantity Unit. Code List. Identifier • Quantity Unit. Code List Agency. Identifier • Quantity Unit. Code List Agency Name. Text
Text. Type	A character string (i.e. a finite set of characters) generally in the form of words of a language.	Shall also be used for names (i.e. word or phrase that constitutes the distinctive designation of a person, place, thing or concept).	• Text. Content • Language. Identifier • Language. Locale. Identifier

CCT – Use

CCTs are used as the basis for deriving a set of CICA Primitives. As CCTs are very general, almost data types, yet at a more semantic level. Therefore, CCTs act as semantic anchor point within the CICA architecture linking all usages and derivations

CCT – Examples

As an example, take the CCT for Amount. The content or value of this component is defined as a number of monetary units of a particular currency. One supplementary component indicates the currency, expressed as a 3-letter alphabetic code value (i.e. USD=U.S. dollars, CAN=Canadian dollars). Another supplementary component identifies the version of the code list of the currency codes, i.e. the publication that states the USD stands for U.S. dollars, CAN stands for Canadian dollars, etc. Refer to the UN/CEFACT ebXML Core Components Technical Specification, version 1.9.

2. Primitives

Primitive – Definition

Primitives occupy the next lowest level, immediately above CCTs within the CICA architecture. Primitives specify a slightly more specific semantic purpose than that of the CCT, yet remain generally reusable. Primitives are created by taking a CCT and deriving a set of semantically specific usages for each CCT. A Primitive, like CCTs, contains a single, discrete piece of information. Each Primitive is derived from one ebXML Core Component Type (CCT), but multiple Primitives are derived from a single CCT. In other words, Primitives are subclasses of ebXML CCTs.

Primitive – Content

A Primitive consists of a single value CCTC based on the ebXML Core Component Type. To put this in object-oriented terms, each Primitive is a subclass of the CCT and inherits from the CCT its list of CCTCs. The Primitive is assigned a name different than the name of the CCT it inherits from, reflecting its unique semantic intent.

Primitive – Use

A Primitive is used as a member of Components and Blocks (higher level constructs).

Primitive – Examples

The following are examples of Primitives:

- Weight Measure
- Total Amount
- Party Identifier
- Net Quantity

Primitive – Design Rules

2.1. Semantic Rules:

2.1.1.MUST have a unique Primitive Name and system assigned ID value.

2.1.2.MUST be derived from a single ebXML Core Component Type (CCT).

2.1.3.MAY be derived from the same Core Component Type as one or more other Primitives.

2.1.4.SHOULD represent a single, discrete piece of information.

2.1.5.SHOULD represent a unique piece of information across all Primitives.

2.2. **Primitive Property Rules:**

2.2.1.**ID –** identifier assigned to this Primitive, assigned by the X12 CICA Database Tool. Will be unique across all Primitives. Example: P00001

2.2.2.PrimitiveName – logical name assigned to this Primitive. Must be unique across all Primitives. Example: Name Text

2.2.3. **PrimitiveXMLName** – physical/syntactic name assigned this Primitive. Example: NameText
2.2.4. MUST be unique across all Primitives.
2.2.5. MUST comply with the X12.7 Naming Conventions.
2.2.6. **PrimitiveType** – ebXML Core Component Type (CCT) that this Primitive is derived from. Example: TextType

2.2.7. **PrimitiveDescription** – free-form text that describes the meaning and purpose of the Primitive. Example: Free-form text that represent the name of a Party.

2.3. Primitive Usage Rules

2.3.1. A Primitive may only be used in Blocks or Components.

2.3.2. When used, it is assigned a UsageName and an XMLUsageName.

3. Components

Component – Definition

The purpose of a Component is to fully express either an Identity or Characteristic, which becomes a part of the definition of the next higher-level entity, the Block. Components are a collection of two (2) or more Primitives, which together specify either an Identity or Characteristic, for a higher-level construct. Components occupy the next level above Primitives in the CICA architecture.

Component – Content

A Component is composed of two or more members, each of which is a Primitive. In object-oriented terms, each Component member is "derived" from a Primitive.

Component – Use

A Component is used as a member of Blocks.

Component – Examples

The following examples are cases where a single identity or characteristic is conveyed by more than one piece of information:

- Person Name - consists of First Name, Middle Name, Last Name
- Mailing Address – consists of Street Address, City, State, etc.

Some Components express a range of values for some object. A range specifies, in some manner, a minimum value and a maximum value, and typically will be of the same data type (i.e. dates, amounts, text, etc.).

Examples:

- Date Range - consists of Start Date and End Date.
- Price Range – consists of a Minimum Price and a Maximum Price.
- Product Code Range – consists of a Start Product Code and an End Product Code (i.e. to restrict the list of products of interest by a range of code values).

Given the example:

C00001 **Component Name:** Person Name
Component XMLName: PersonName
Component Description: The name of an individual person.

Pos
PName
Usage Name

01
NameText
First Name

02
NameText
Middle Name

03
NameText
Last Name

- This Component's member set consists of {First Name, Middle Name, Last Name}.
- Each of the following is a member of this Component: First Name, Middle Name, and Last Name.
- Each member inherits from the Primitive Name NameText.
- Component - Attributes

Component – Design Rules

The statements below define a Component. Note that term "member set" refers to the list of members that make up the Component. Each Component member, a Primitive, is uniquely defined by its Usage Name, (not by the Primitive it is derived from).

3.1. Component Semantic Rules

3.1.1. MUST consist of at least two Primitives.

3.1.2. MAY consist of more than two Primitives.

3.1.3. MUST have a unique member set across all Components, as defined by the Usage Names.

3.1.4. MAY have a member set that is a subset or superset of another Component's member set.

3.1.5. MAY have more than one member that inherits from the same Primitive.

3.1.6. MAY be used in one or more Blocks.

3.1.7. MUST represent a specific Identity or Characteristic of an object.

3.1.8. SHOULD represent a unique Identity or Characteristic across all Components.

3.1.9. SHOULD have an abstract naming that is not specific to a single business context.

3.2. Component Syntactic Rules

3.2.1. **ComponentName** – logical/semantic name assigned to this Component. Example: Person Name

3.2.2.**ComponentXMLName –** physical/syntactic name assigned this Component. Example: PersonName

3.2.3.**ComponentDescription** – free-form text that describes the meaning and purpose of the Component. Example: Full name of an individual person.

3.3. Each **Component Member** semantic Rules:

3.3.1.MUST inherit from a single Primitive (identified by Primitive ID/Name)

3.3.2.MAY inherit from the same Primitive as another Member.

3.3.3.MUST have a unique Usage Name within the Component.

3.3.4.MAY have the same Usage Name as a member of a different Component.

3.3.5.MUST be designated as either Required or Optional, when used in a Module.

3.3.6.SHOULD represent a single, discrete piece of information that is a subset of the specific Identity or Characteristic represented by the Component as a whole.

3.4. **Component Member** Properties:

3.4.1.**Position** – sequential number that indicates the position of the member within the Component. Example: 01

3.4.2.**UsageName** – name assigned to this member. Example: First Name

3.4.3.MUST be unique within the Component.

3.4.4.**XMLUsageName** – physical/syntactic name assigned to this member. Example: FirstName

3.4.5.**PrimitiveName** – Name of the Primitive that this member inherits from. Example: Name

Only filled in when used in Model

3.4.6.**Req Designator** – indicates whether or not this member is required, and if there are any conditional rules that apply to this member. Valid values are:

R = Required

O = Optional

4. Blocks

Block - Definition

Blocks specify a single Party, Location, Resource or Event, in a semantically precise, yet business context neutral manner. Blocks are semantically precise, because their composition is specific for the type of noun being specified. Blocks are business context neutral in that they specify a general noun, and are not specific to the specific noun. Blocks occupy the next level above Components in the CICA architecture. Each Block, like any noun, answers the questions "who", "what", "where", "when", or "why". The Block members specify either Identity or Characteristic information that apply to this entity.

Blocks are context-neutral, that is, they are independent of a specific business use, and are reusable in higher-level constructs.

Blocks play a pivotal role in CICA.

Block – Content

A Block is composed of one or more members, each of which specifies a particular Identity or Characteristic of the Block. Each Block Member is either the use of a Component or a Primitive. The Block Member is assigned a usage name.

Each Block Member is assigned a designator that indicates if it is required or optional. The minimum and maximum number of occurrences for each Block member is also specified. Detailed descriptions of these attributes are provided in the Attributes section.

Block – Use

A Block is used in two different places in CICA. First, sets of Blocks can be grouped together to create larger neutral constructs – Assemblies. These constructs are useful for creating related groupings of Blocks, such as when you need a Party with a Location. But, these are completely reusable and neutral, the same way Blocks are.

Blocks are also used to construct Modules, which are context specific implementable constructs.

As a standalone entity, a Block represents a logical grouping of related data items that can be referenced and reused as a single entity.

But, when a Block is placed into a Module, either directly or via an Assembly containing the Block, the Block takes on additional characteristics. The additional characteristics further define or restrict the allowable data content. For example, a Block instance can specify minimum and maximum lengths on individual data fields at the lowest level (i.e. Primitives), and it can specify which Block members are required and how many times they may (or must) occur. Some of the characteristics applied to the Block instance override those established to the abstract Block, others are applied only at the Block instance and are not part of the abstract Block definition.

> **<u>From a technical perspective</u>**, a Block is an abstract super-class. Each Module use will require certain properties be assigned [min length, max length, max repeat, etc.]. Each different set of properties is a separate sub-class of the abstract super-class.
>
> **<u>From a Business Perspective</u>**, when a Block is used in a Module, that is the point in time when its specific business use in known. For example, a Person Party Block can be used to create a Student module, in an education business context. Thus, a set of properties is assigned to Block, and all of the Block Members, as deemed necessary for the specific Student use. The next Block use, can either use the set of properties previously assigned, or create a new set of properties.

Block – Examples

The following examples are cases where a Block would be used:

- Person – consists of a Person Name, Social Security Number, Height, Weight, Hair Color, Eye Color, etc.
- Product – consists of a Product Identifier, Product Description, Product Class Code, etc.
- Location – consists of an address – Street Address, City, State, Postal Code, Country, etc.
- Account – consists of Account Number, Account Type, Account Status, etc.

Given the example:

Block Name: Person Party
Block XMLName: Person Party
Block Type: Party
Block Description: Describes an individual person.

Pos	Prim/Comp Name	Usage Name	Use
01	Person Name	Person Name	I
02	IDNumber	Social Security Number	I
03	Description	Hair Color	C

- This Block's member set consists of {Person Name, Social Security Number, and Hair Color}.
- Each member inherits from the Component or Primitive represented by the ID value and Name indicated in the Member ID/Name column.

Block – Properties

The statements below define a Block. Note that the term "member set" refers to the list of members that make up the Block. Each member in the set is uniquely defined by its Usage Name (not by the Primitive or Component being used).

4.1. Block Semantic Rules:

4.1.1. MUST consist of at least one member.

4.1.2. MAY consist of more than one member.

4.1.3. MUST have a unique member set across all Blocks, as defined by the Usage Names.

4.1.4.MAY have a member set that is a subset or superset of another Block's member set.

4.1.5.MAY have more than one member that is a usage of the same Primitive or Component.

4.1.6.MAY be used in one or more Assemblies.

4.1.7.MAY be used in one or more Modules.

4.1.8.MUST represent a specific Party, Resource, Location, or Event.

4.1.9.MUST be business context neutral and independent of specific business use.

4.2. Block Member Semantic Rules:

4.2.1.MUST be a use of a single Component or Primitive (identified by Member Name).

4.2.2.MAY contain multiple usages of the same Component or Primitive, but each instance must have a different Usage Name.

4.2.3.MUST have a unique Usage Name within the Block.

4.2.4.MAY have the same Usage Name as a member in a different Block.

4.2.5.MUST be designated as either an **I**dentity or **C**haracteristic (see definition below).

4.2.6.MUST represent a single **I**dentity or Characteristic of the entity the Block represents.

4.3. Block Property Rules, Abstract super class

4.3.1.**BlockName** – the logical/semantic name assigned to this Block. Example: Person Party

4.3.1.1. MUST be unique across all Blocks.

4.3.2.**BlockXMLName** – the physical/syntactic name assigned this Block. Example: PersonParty

4.3.2.1. MUST be unique across all Blocks.

4.3.3.**BlockType** – indicates the general type of this Block, for classification/reporting purposes. Valid values are:

4.3.3.1. **Party –** describes a person, organization, or other entity; i.e. answers a "who" question

4.3.3.2. **Resource –** describes a thing or object, i.e. answers a "what" question

4.3.3.3. **Location –** describes a place, i.e. answers a "where" question

4.3.3.4. **Event –** describes an event or activity, i.e. answers "when" and "what happened" questions, can be associated with a particular occurrence of some event, usually associated with a specific time or timeframe.

4.3.4.**BlockDescription** – free-form text that describes the meaning and purpose of the Block. Example: Describes an individual person.

4.4. Each Block Member has a set of attributes, as follows:

4.4.1.**Position** – sequential number that indicates the position of the member within the Block. Example: 01

4.4.2.**Name** – the name of the Component or Primitive that this member is derived from. Examples: Person Name, Name

4.4.3.**UsageName** – the name assigned to this member that reflects its usage within the Block. Example: Person Name.

4.4.4.**XMLUsageName** – the XML name that corresponds to the Usage Name above. Example: PersonName

4.4.5.**Use** – indicates if this member is an **I**dentity or Characteristic. Valid values are:

4.4.5.1. **I** = **I**dentity

4.4.5.2. **C** = Characteristic

An Identity generally distinguishes one instance of this object from another, and is frequently unique across the universe of objects, although this is not a requirement. The entity may or may not follow a given numbering scheme or format. Examples of Identity include the following:

Block Type	Block Name	Identity Examples
Party	Person	Person Name, Social Security Num
Resource	Product	UPC Code
Location	Property	Parcel Number
Event	Graduation	Graduation Date

A Characteristic describes a particular quality or attribute of the object, much like an adjective describes a particular quality or attribute of a noun. Elements containing free-form text are more likely to be classified as Characteristic rather than Identity, although this is not a rule. Examples of Characteristic include the following:

Block Type	Block Name	Characteristic Examples
Party	Person	Eye Color
Resource	Product	Product Description
Location	Property	Property Description
Event	Graduation	Type of Degree (Attainment), i.e. Bachelor’s.

5. Assemblies

Assembly - Definition

Assemblies occupy the next level above Blocks in the CICA architecture. An Assembly represents a group of logically related party, resource, location, and/or event entities in a business document. Each Assembly can be thought of as a group of nouns that forms areusable unit. Assemblies are context-neutral, that is, they are independent of a specific business use, and are reusable within other Assemblies as well as within the next higher leve construct – Modules.

There are three types of data relationships expressed by Assemblies:

- Consists-of: where the members operate together as a unit to create a larger reusable grouping.
- Kinds-of: where the members are peers in that they perform the same role.
- Expansion/Contraction: where the members are related in that the instance data for the similar/same peers ???? further explanation

Assembly – Content

An Assembly is composed of one or more members, each of which specifies a sub-group of the entire Assembly. Each Assembly member is associated with either another Assembly or a Block - to put this in object-oriented terms, each Assembly member is “inherited” or “derived” from an Assembly or a Block. The Assembly member is assigned a name which can be the same as or different than the name of the object it inherits from. Typically it is assigned a different name that reflects its specific usage; hence the term applied to the name of the member is “Usage Name”.

Each member is assigned a designator that indicates if it is required, optional, or “conditionally required”, i.e. depending on the existence of other members in the Assembly. The minimum and maximum number of occurrences for each member is also specified. Detailed descriptions of these attributes are provided in the Attributes section.

Assembly – Use

An Assembly is used as a member of other Assemblies and Modules, much the same way Blocks and Assemblies are used as members of Assemblies. As a standalone entity, an Assembly is *abstract*, i.e. it represents a logical grouping of related data items that can be referenced and reused as a single entity.

But, when an Assembly is placed into a Module, the Assembly takes on additional characteristics – in essence, a new *instance* of the *abstract* Assembly is created that fits the requirements for the particular Module it is being used in. The additional characteristics further define or restrict the allowable data content. For example, an Assembly instance can specify which members are required and how many times they may (or must) occur. The characteristics applied to the Assembly instance override those established to the abstract Assembly.

Assembly - Examples

The following entities would be represented by an Assembly:

- Delivery Party – consists of a Person and Address.
- Product Status – consists of a Product and Quantity Information.
- Patient Record – consists of a Person and Appointment Events
- Account History – consists of Account information and Transaction Events
- Contact Information – consists of contact information of various types – Phone, Email, Mailing, etc. (each is a different entity (noun).

Assembly – Properties

The statements below define an Assembly. Note that the term "member set" refers to the list of members that make up the Assembly. Each member in the set is uniquely defined by its Usage Name (not by the Primitive or Component it is inherited from).

An Assembly:

- MUST consist of at least two members.
- MAY consist of more than two members.
- MUST have a unique member set across all Assemblies, as defined by the Usage Names.
- MAY have a member set that is a subset or superset of another Assembly's member set.
- MAY have more than one member that inherits from the same Assembly or Block.
- MAY be used in one or more Assemblies and/or Modules.
- MUST represent a semtantically unique entity of in terms of composition.
- SHOULD have an abstract meaning that is not specific to a single business context.

An Assembly Member:

- MUST inherit from a single Assembly or Block (identified by ID).
- MAY inherit from the same Assembly or Block as another Member.
- MUST have a unique Usage Name within the Assembly.
- MAY have the same Usage Name as a member in a different Assembly.
- MUST be designated as either **R**equired, **O**ptional, **A**ny, or e**X**clusive-or (see definition below).
- MUST be designated with a minimum and maximum number of occurrences. The maximum number of occurrences MUST be equal to or greater than the minimum number of occurrences.

Assembly - Attributes

As described earlier, there are basically two types of Assemblies:

1. the default, or abstract Assembly that serves as a base level definition of a grouping of parties, resources, locations, and/or events.
2. the specific, or instance Assembly, which is based on an abstract Assembly then further defined by overriding attributes at the Assembly level.

The following attributes apply to the Assembly as a whole:

ID – an identifier assigned to this Assembly. Must be unique across all Assemblies. Example: A00001.

AssemblyName – the logical/semantic name assigned to this Block. Example: Delivery Party

1. MUST be unique across all Assemblies.

AssemblyXMLName – the physical/syntactic name assigned this Assembly. Example: DeliveryParty

- MUST be unique across all Assemblies.
- MUST comply with the following syntax-dependent restrictions: it must be allowable as the name of an XML complexType.
- MAY be a “default value” that derived from the corresponding logical name, Name, by removing spaces, using upper camel case format, removing characters not supported by NMToken (see General Rules), removing articles and conjunctions (e.g., “a”, “for”, “and”, etc.), and applying standard abbreviations where appropriate (see Appendix D of the XML X12 Technical Report). The X12 XML Database Tool will automatically generate this default value, which can be overridden by the user.

AssemblyDescription – free-form text that describes the meaning and purpose of the Assembly. Example: Describes the name and location of a person.

Each member of the Assembly has a set of attributes, as follows:

Position – sequential number that indicates the position of the member within the Assembly. Example: 01

ID – the ID of the Assembly or Block that this member is derived from. Examples: A00002, B00002

Name – the name of the Assembly or Block that this member is derived from. Examples: Ship To, Person Party

UsageName – the name assigned to this member that reflects its usage within the Assembly. Example: Ship To Party

XMLUsageName – the XML name that corresponds to the Usage Name above. Example: ShipToParty

Req Designator – indicates whether or not this member is required, and if there are any conditional logic rules that apply to this member. Conditional logic refers to the concept that the inclusion or exclusion of a given member depends on the existence or non-existence of one or more other members within the same construct.

Valid values are:

R = Required

O = Optional

A = At least one of the members coded with 'A' is required, more than one is allowed.

X = One and only one of the members coded with 'X' is required.

Min Occurs – indicates the minimum number of occurrences for this member. If the Req Designator is R, this value should be 1 or greater. If the Req Designator is O, A, or X, this value should be 0 or greater.

Max Occurs – indicates the maximum number of occurrences for this member. This value should be equal to or greater than the Min Occurs value. A value greater than 1 allows the member to repeat up to the number of times specified by the Max Occurs value. For example, if the Min Occurs value is 1 and the Max Occurs value is 5, the member can occur 1, 2, 3, 4, or 5 times.

Assembly – Design Rules

5.1. Assembly Semantic Design Rules

5.1.1. Assemblies may only be composed of the following

5.1.1.1. Two-or-more Blocks

5.1.1.2. Two-or-more Assemblies

5.1.1.3. One-or-more Blocks and one-or-more other Assemblies

5.1.2. Assemblies are made up of two or more blocks and/or assemblies.

5.1.3. Assemblies are Neutral constructs; therefore their contents and naming are abstract and industry independent.

5.1.4. There are three types of data relationships which exist which are expressed at the Assembly level,

5.1.4.1. Consists-of where the children operate together as a unit to create a larger reusable grouping. Examples include:

5.1.4.1.1. A Person with a Location

5.1.4.1.2. A Person with an Event

5.1.4.1.3. A Location with an Event

5.1.4.2. Kinds-of where the children are peers in that they perform the same role, Examples include:

5.1.4.2.1. A set of ways to specify how to contact, phone number, fax number, email, etc.

5.1.4.2.2. A set of parties, one of which is performing a role, such as a person or an organization, which has acted in a role. Specifically, this applies in Healthcare where a Doctor or a Professional Corporation is collecting payment.

5.1.4.3. Expansion/Contraction. The children are related in that the instance data for the similar/same peers. Examples:

5.1.4.3.1. In healthcare, similar structures used to specify a Patient & Subscriber are used. In some cases, the Patient is the Subscriber, and in EDI notes are used to specify the suppression of repeated instance data between the related structures

5.1.4.3.2. In procurement, the Buyer, Payer, and Ship-to can be the same instance information. Suppression of the repeated instance details are preferred.

5.2. Assembly Syntactic Design Rules

5.2.1. **AssemblyName**.

5.2.1.1. Must be a semantically descriptive name for the Assembly.

5.2.1.2. Shall be used as the primary means of identification of assemblies in the DISA database.

5.2.1.3. Shall be maintained as unique in the X12 XML standards development process.

5.2.2. **AssemblyXMLName**. A meaningful name for the Assembly, derived from its AssemblyName, suitable for use as an XML element name in a message. In practice, the AssemblyXMLName may be derived from its corresponding AssemblyName by removing spaces, using upper camel case format, and removing articles and conjunctions (e.g., “a”, “for”, “and”, etc.) and punctuation (e.g., slashes “/” and hyphens, etc.). May be, but not required to be, identical with the AssemblyName.

5.2.3. **AssemblyType**. Specifies one of two types, a consists-of or a kinds-of relationship. A consists-of relationship is when the set of elements that make up the Assembly together form a larger unit, such as Party + Location. Kinds-of relationship is when the elements are semantic peers – each can perform the same function in the exchange, as is the case with Phone and Email each providing a mechanism for contacting a party.

5.2.3.1. C = Consists of

5.2.3.2. K = Kinds of

5.2.4. **AssemblySubtype.** Used for Kinds of relationships, distinguishing between mutually exclusive and at least one of.

5.2.4.1. X = Mutually exclusive

5.2.4.2. O = At least one of

5.2.5. **Assembly Class.** Each Assembly is formed to specify information about a primary class of information: Party, Resource, Event or Location. The Class of the first element within the Assembly is typically the same as the Assembly Class.

5.2.5.1. P = Party

5.2.5.2. R = Resource

5.2.5.3. E = Event

5.2.5.4. L = Location

5.2.6. **Assembly Description.** Free form text used to describe the Assembly.

5.3. Assembly Contents Design Rules

5.3.1. BlockName or AssemblyName

5.3.2. **Block or Assembly Flag**. Flag used to indicate whether the entry is a Block or Assembly

5.3.3. **UsageName**. Specifies an over-ride name for the Block or Assembly, when used in this position within the Assembly. This usage name is constrained to be a general name, in order to meet the reuse constraints on the Assembly.

5.3.4. **Position**. Used to indicate sequential position within Assembly

5.3.5.**RequirementsFlag**. The value must be compatible with the MinOccurs and MaxOccurs values in this AssemblyListEntry

5.3.5.1. C = Conditional

5.3.5.2. M = Mandatory

5.3.5.3. O = Optional

5.3.6.**MinOccurs**. The value must also be compatible with the RequirementsFlag and MaxOccurs values in this Assembly List Entry

5.3.6.1. MinOccurs – Numeric

5.3.7.**MaxOccurs**. The value must also be compatible with the RequirementsFlag and MinOccurs values in this entry.

5.3.7.1. MaxOccurs - Numeric

5.3.7.2. Infinity designated with [PDP]

5.4. Assembly Design Guidelines

There are three types of data relationships which exist which are expressed at the Assembly level,

Consists-of where the children operate together as a unit to create a larger reusable grouping. Examples include:

A Person with a Location

A Person with an Event

A Location with an Event

Kinds-of where the children are peers in that they perform the same role, Examples include:

A set of ways to specify how to contact, phone number, fax number, email, etc.

A set of parties, one of which is performing a role, such as a person or an organization, which has acted in a role. Specifically, this applies in Healthcare where a Doctor or a Professional Corporation is collecting payment.

Expansion/Contraction. The children are related in that the instance data for the similar/same peers. Examples:

In healthcare, similar structures used to specify a Patient & Subscriber are used. In some cases, the Patient is the Subscriber, and in EDI notes are used to specify the suppression of repeated instance data between the related structures

In procurement, the Buyer, Payer, and Ship-to can be the same instance information. Suppression of the repeated instance details is preferred.

6. Templates

Template - Definition

A Template is a framework or skeleton of a business document for a particular business process requirement. The business process determines the Template's high-level composition and use. The Template defines the general structure and content of a business document, from which specific implementations of business documents are based. For example, an Event Based Invoice Template is created for a specific business process use of an Invoice. Its contents describe the general document used by parties to request payment for goods and/or services. Specific Invoice documents are developed based on that Invoice Template.

Template - Content

A Template is composed of an ordered list of one or more members, called Slots. Slots are named using neutral terms for the logical business purpose they identify, and act as placeholders for Modules. A Template Slot is linked with a list of Modules, and each Module is selected to use in the Slot, under different business conditions. A Document is built by assigning a Module to each Template Slot.

A Template is divided into three sections:

- Header – applies to the entire business process and specifies the business context and parties to the business exchange.
- Detail – contains a set of data relevant to the business process of the message.
- Summary – summarizes the information contained in the detail (use of this section is generally discouraged).

Template - Use

The Template provides a big picture view, of all the possible Modules used in each Slot. This construct is a fundamental innovation in the CICA document assembly model. All contextual uses of the Template can be viewed through this construct. The Template is conceptually the focal point of the architecture, bridging between "neutral" constructs below the line, and "implementable" constructs above the line, as shown in the figure below:

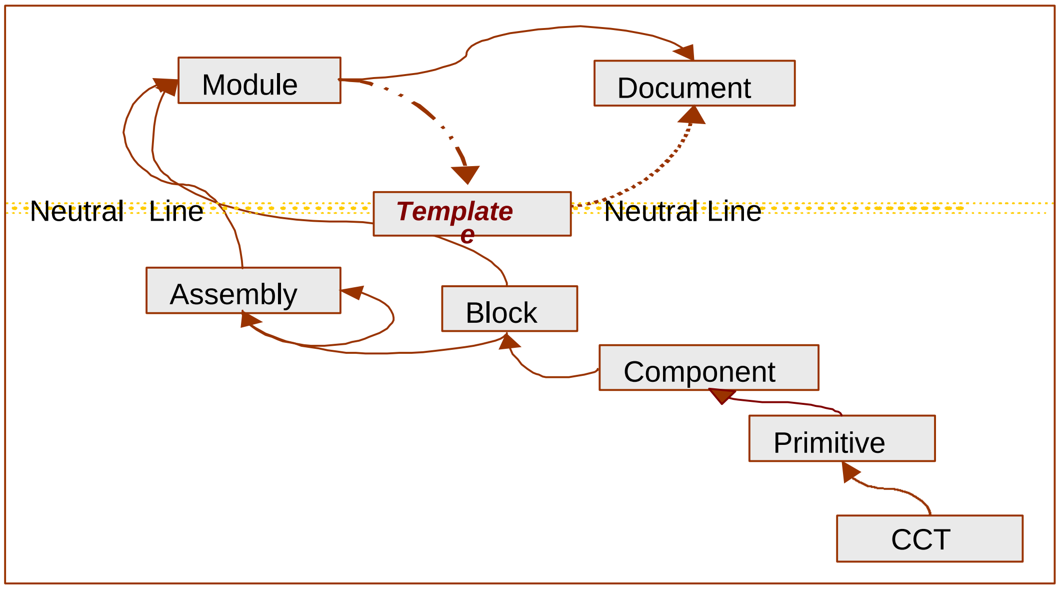


Figure 6, CICA Layers and Relationships

Examples

Templates can be created for the following general business documents:

- Event Invoice
- Bill of Lading
- Payment Request

If there is significant variation in the structure of a general business document, then multiple Templates could be created to accommodate these. For example, there are many types of Payment Requests, which can be categorized by the payment instrument to be used (ACH, wire transfer, credit card, etc.). A separate Template could be created for each, i.e.

- ACH Payment Request
- Wire Payment Request
- Credit Card Payment Request

Template - Design Rules

6.1. Semantic Design Rules

6.1.1. A Template MAY be assigned to zero or more Documents.

6.1.2. A Template MUST be specified for each Document.

6.1.3. A Template MUST consist of one or more logically ordered and named Slots. These slots are reserved for Modules.

6.1.4. New Templates can only be created when the business process specifies a new condition. This determination is made based on preconditions, trigger event, post conditions, and constraints.

6.1.5. A Template MUST consist of one or more Slots, each of which is a member of either the Header, Detail, or Summary area.

6.1.5.1. The Header area consists of Slots that contain information that applies to the entire business document and specifies the business context and parties to the business exchange.

6.1.5.2. The Detail area consists of one or more Slots that contain information that describes the detail content of the message. The Slots in the Detail area may be repeated as a unit in a Document, i.e. the unit as a whole represents a single line item, and repeats for each line item in the message.

6.1.5.3. The summary area consists of Slots that summarize information about the Detail area.

6.1.5.4. The Slots pertaining to each AREA must be together.

6.1.5.5. Areas must be ordered, Header first, Detail second, Summary last.

6.1.6. Each area in the Template MAY be defined to have zero or more Slots.

6.2. Syntactic Design Rules

6.2.1. **TemplateName** – the logical/semantic name assigned to the Template.

6.2.1.1. It MUST be unique across all entities within the CICA Architecture.

6.2.2. **TemplateXMLName** – the physical/syntactic name assigned to the Template; it corresponds to Name, above.

6.2.2.1. It MUST be unique across all entities within the CICA Architecture.

6.2.2.2. It MUST comply with the naming conventions in X12.7.

6.2.2.3. It MAY have a “default value” mechanically generated by the CICA Database Tool, which can be overridden.

6.2.3. **TemplateDescription** – free-form text that describes the meaning and purpose of the Template.

6.2.4. **TemplateFamily** – indicates the general category the Template falls within; it is a way to group "peer" Templates that cover the same general business purpose but are required because the associated documents vary in structure. For example, if separate Templates were needed for ACH Payment Request, Wires Payment Request, and Credit Card Payment Request, they would all be assigned the same Family of 'Payment Request'.

6.2.5. **BusinessProcess** – identifies of the Business Process to which this Template applies. The ebXML catalog of Common Business Processes [version 1.0] is a potential source. TAS will determine an initial set of values to choose amongst.

6.2.6. **BusinessProcessFamily** – identifies of the family of Business Processes to which this Template applies. . TAS will determine an initial set of values to choose amongst.

6.2.7. **BusinessSubprocess** – identifies the sub process of the overall business process within which this Template is used. For example, Event-Based Invoicing is a subprocess of Invoicing. TAS will determine an initial set of values to choose amongst.

6.2.8. **BusinessSubprocessFamily –** identifies a relationship amongst peer sub processes. For example if Event Based Invoicing were a Business Sub process, then it would be a member of the Invoicing family of subprocesses. . TAS will determine an initial set of values to choose amongst.

6.2.9. **TriggeringEventDescription.** This describes the event in the business process being served that triggers the need to generate a message (Document) having this Template. This description may also include the range of expected responses to receipt of the generated message.

6.2.10. **ResponsibleSubCommittee** – identifies the ASC-X12 Subcommittee with primary responsibility for maintenance of this Template.

6.2.11. **SlotList -** Ordered list of one or more Slots which make up the Template. See Member Design Rules, below.

Slots within Template – Design Rules

6.3. Semantic Design Rules

6.3.1. Each named Slot within the Template SHALL fulfill a specific function in the business process defined by the Template.

6.3.2. Each named Slot SHALL, as its primary purpose, specify a single one of the following as related to the business process being identified:

6.3.2.1. Who (party: person / organization),
6.3.2.2. What (resource: financial, product [tangible or service])
6.3.2.3. Where (place: physical location)
6.3.2.4. When (event: past / present/ future).

6.3.3. A Slot, as defined by its Slot Name, MAY appear in multiple Templates.

6.3.4. Each Slot within a Template MUST accommodate one or more Modules.

6.3.5. Slots SHALL be named using a neutral term for the logical business process they identify.

6.3.6. Each Slot within a Template SHALL have a name that is unique.

6.4. Syntactic Design Rules

6.4.1. **SlotName** – the name of the Slot within the Template. This term should be specific to the general business process, and not specific to an industry.

6.4.1.1. It MUST be unique within the Template.

6.4.2. **SlotXMLName** – the physical/syntactic name assigned to the Slot; it corresponds to Name, above.

6.4.2.1. It MUST be unique within the Template.

6.4.2.2. It MAY be the same as a Slot of a different Template.

6.4.2.3. It MUST comply with the naming conventions in X12.7.

6.4.2.4. It MAY have a “default value” mechanically generated by the CICA Database Tool, which can be overridden.

6.4.3. **SlotPurpose** – free-form text description describing the purpose of the Slot, from the perspective of the business process. It is a description of the purpose served by candidate Modules that fill the Slot.

6.4.4. **SlotDetailFlag** – indicates which area in the Template the Slot belongs to.

6.4.4.1. **H** = Header

6.4.4.2. **D** = Detail

6.4.4.3. **S** = Summary

6.4.5. **ReqDesignator** – indicates whether or not the Slot is required (mandatory) or optional.

6.4.5.1. It MUST be one of the following values:

6.4.5.1.1. **R** = Required

6.4.5.1.2. **O** = Optional

6.4.6. **MinOccurs** – indicates the minimum number of times this Slot may occur within the Template. If the Slot is in the Detail area, this represents the minimum number of times the Slot may repeat within <u>each</u> occurrence/iteration of the entire Detail area.

6.4.6.1. It MUST be a positive integer, i.e. equal to or greater than zero.

6.4.7. **MaxOccurs** – indicates the maximum number of times this Slot may occur within the Template. If the Slot is in the Detail area, this represents the maximum number of times the Slot may repeat within <u>each</u> occurrence/iteration of the entire Detail area.

6.4.7.1. It MUST be either unbounded, or a positive integer, and equal to or greater than the **MinOccurs** value.

6.4.8. **SlotModuleLinkage –** the following identifiers must be supplied for each Module that can occupy the Slot.

6.4.8.1. **ModuleName** – the name of the Module as defined in the CICA Architecture. Must conform to the naming conventions in X12.7.

6.4.8.2. **ModuleUsageName** – the logical/semantic name assigned to the Module that reflects its usage within the Slot in the Template.

6.4.8.2.1. It MUST be unique across all Modules within the Template.

6.4.8.2.2. It MAY be the same as a Module of a different Template.

6.4.8.3. **ModuleXMLUsageName** – the physical/syntactic name assigned to the Module member that reflects its usage within the Slot in the Template; it corresponds to Usage Name, above.

6.4.8.3.1. It MUST be unique across all Modules within the Template.

6.4.8.3.2. It MAY be the same as a Module in a different Template.

6.4.8.3.3. It MUST comply with the naming conventions in X12.7.

6.4.8.3.4. It MAY have a “default value” mechanically generated by the CICA Database Tool, which can be overridden.

6.4.8.4. **ContextCategoryValuePair –** provides for a list of ebXML context rules that apply to the selection of this Module to occupy the Slot. These could potentially be used by a rules engine to programmatically determine which Module is appropriate for a Document that addresses given business process. At this time, no such engine has been implemented; hence this entry is purely informational.

7. Modules

Module - Definition

A Module represents a set of context-specific, related data that serves a specific purpose in a business document. Modules used within a specific business document satisfy the objective of the business document. Because Modules are above the neutral line, they always represent a detailed definition of the data, rather than an abstract definition.

Module – Content

A Module is composed of one or more members, each of which specifies a sub-group of the entire Module. Each Module member is derived from either an Assembly or a Block, i.e. the Assembly or Block definition that contains the additional characteristics that further define or restrict the allowable data content. This is because the Module is context-specific – above the neutral line – and therefore requires a fully qualified Assembly or Block rather than an abstract representation. To this point, each Module should be:

- specific such that can be implemented without further qualification as to what each piece of information means
- concise such that further explanation is not required in order to determine what pieces of information are used.

Each Module member is assigned a name that is different than the name of the object it is derived from. The name reflects its specific usage; hence the term applied to the name of the member is "Usage Name".

Each Module member is assigned a designator that indicates if it is required, optional, and another designator conditional usage, i.e. depending on the existence of other members in the Module. The minimum and maximum number of occurrences for each member is also specified. Detailed descriptions of these attributes are provided in the Syntactic Design Rules.

Modules - Use

A Module occupies a Slot in a Template. A Module can be linked with multiple Templates and therefore can be used in multiple Documents, providing that the specific definition and characteristics of the Module apply in all cases.

Modules - Examples

The following are represented by a Module:

- Invoice Event – consists of invoice number, date, purchase order reference, and other details relevant to a particular invoice, i.e. conveys basic "header" information of the invoice.
- Buyer – consists of name, location, contact information, and other details relevant to the buying party.
- Invoice Line Goods – consists of a product, units, and other details relevant to a line item on an invoice, i.e. conveys the "detail" information of the invoice.

Module – Design Rules

The statements below define a Module. Note that the term “member set” refers to the list of members that make up the Module. Each member in the set is uniquely defined by its Usage Name (not by the Block or Assembly it is derived from).

7.1. Semantic Design Rules

7.1.1. A **Module**:

7.1.1.1. MUST consist of at least one member.

7.1.1.2. MAY consist of more than one member.

7.1.1.3. MUST have a unique member set across all Modules, as defined by the Usage Names.

7.1.1.4. MAY have a member set that is a subset or superset of another Module’s member set.

7.1.1.5. MAY have more than one member that inherits from the same Assembly or Block.

7.1.1.6. MAY be designated to fill one or more Slots within a single Template or multiple Templates.

7.1.1.7. MUST be a semantically unique entity of in terms of composition.

7.1.1.8. MUST answer a specific semantic question within the business process (e.g., Who/What/When/Where/Why).

7.1.1.9. MUST conform to the purpose of the Slot(s) in which it will be used.

7.1.1.10. MAY be reusable within a business process and across business processes.

7.1.1.11. MUST be specific such that further qualification is not required as to what each piece of information means.

7.1.1.12. MUST be concise such that further explanation is not required in order to determine which pieces of information are used.

7.1.2. A **Module Member**:

7.1.2.1. MUST be derived only from an Assembly or a Block.

7.1.2.2. MAY be derived from the same Assembly or Block as another Member within the same Module.

7.1.2.3. MUST have a unique Usage Name within the Module.

7.1.2.4. MAY have the same Usage Name as a member in a different Module.

7.1.2.5. MUST be designated as either **R**equired or **O**ptional.

7.1.2.6. MUST be designated with a minimum and maximum number of occurrences.

7.1.2.7. MAY have a maximum number of occurrences that is “unbounded”, meaning an unlimited number.

7.1.2.8. MUST have a maximum number of occurrences that is equal to or greater than the minimum number of occurrences (unbounded is considered to be greater than or equal to any finite value).

7.2. Syntactic Design Rules

7.2.1. A **Module** is defined by the following:

7.2.1.1. **ModuleName**.

7.2.1.1.1. MUST be semantically descriptive.

7.2.1.1.2. MUST uniquely identify the Module in the CICA database.

7.2.1.1.3. MUST be unique across the entire CICA database (i.e. all levels).

7.2.1.2. **ModuleXMLName**.

7.2.1.2.1. MUST be a meaningful name for the Module.

7.2.1.2.2. MAY be derived from or identical to the ModuleName.

7.2.1.2.3. MUST be suitable for use as an XML element name in a message.

7.2.1.2.4. MAY be derived from its corresponding ModuleName by applying naming conventions

7.2.1.3. **ModuleDescription**.

7.2.1.3.1. MUST be a text paragraph describing the use, intent and overall use of the Module.

7.2.1.4. **Module Instance Relationship**.

7.2.1.4.1. SHOULD specify whether there is a semantic relationship between the instance information of the peers within the grouping. For example, a Buyer consists of 3 sub-roles, the Buyer, the Payer & the Ship-to. Therefore, there is relationship amongst the instance information, i.e., the same person or organization can be the playing all three sub-roles: Buyer, Payer and Ship-to.

7.2.1.5. **Module Class**.

7.2.1.5.1. MUST indicate which primary class of information this Module conveys: Party, Resource, Event or Location. The Class of the first member within the Module is typically the same class as the Module Class.

7.2.1.6. **ResponsibleSubCommittee**.

7.2.1.6.1. MUST designate the ASC X12 Subcommittee with primary responsibility for maintenance.

7.2.2.A **Module Member** is defined by the following:

7.2.2.1. **UsageName.**

7.2.2.1.1. MUST be unique within the Module. (redundant with Semantic rules)

7.2.2.1.2. MUST be semantically descriptive of how the member is being used in the Module.

7.2.2.2. **AssemblyorBlock**

7.2.2.2.1. MUST be A or B to specify whether the member is an Assembly or a Block.

7.2.2.3. **RequirementsFlag**.

7.2.2.3.1. MUST be either R for Required, or O for Optional.

7.2.2.3.2. MUST be compatible with the MinOccurs and MaxOccurs values for this member.

7.2.2.4. **MinOccurs**

7.2.2.4.1. MUST be 0 or a positive integer.

7.2.2.5. **MaxOccurs**

7.2.2.5.1. MAY be either 0 or a positive integer AND greater than or equal to the MinOccurs –OR– “unbounded”, meaning an unlimited number.

8. Documents

Document - Definition

A Document is a distinct, detailed message specification that is reflective of a context-specific business process. A Document is derived from a Template and its Slots, with context specific Modules occupying each Slot. A Document is defined by starting with a Template and for each Slot in the Template, selecting the single Module that is required for that particular Document.

Content

A Document is composed of the specific Modules that have been designated to fill each Slot in the Template that has been assigned to the Document.

Use

A Document definition is the complete specification of the message. From the Document, syntax specific representations are generated --- such as XML.

Examples

The following examples represent Documents:

- Delivery-Based Goods Invoice
- Statement-Based Service Invoice
- ACH Payment Request

Document - Design Rules

8.1. Semantic Design Rules

8.1.1. A **Document**:

8.1.1.1. MUST represent an exchange of data that fulfills a single purpose in a business process.

8.1.1.2. MUST contain a specific semantically complete definition.

8.1.1.3. MUST be composed of:

8.1.1.4. A link to a specific Template.

8.1.1.5. A link to a specific Module for every Slot in the Template, made from the choices among the candidate Modules for each Slot in the Template.

8.1.1.6. A set of context references that drove the Module choices.

8.1.2. A **Document Member**:

8.1.2.1. MUST be a specific Module must be designated to fill the Slot in the Template that is assigned to the Document.

8.1.2.1.1. MUST fulfill a specific function in the business process defined by the Document.

A Module, as defined by its Module Name, may appear in multiple Documents.

8.2. Syntactic Design Rules

8.2.1. A **Document** is defined by the following:

8.2.1.1. **DocumentName.**

8.2.1.1.1. MUST represent the logical/semantic name assigned to the Document.

8.2.1.1.2. MUST be unique across all entities within the CICA Architecture.

8.2.1.2. **DocumentXMLName.**

8.2.1.2.1. MUST represent the physical/syntactic name assigned to the Document; it corresponds to DocumentName, above.

8.2.1.2.2. MUST be unique across all entities within the CICA Architecture.

8.2.1.2.3. MUST comply with the naming conventions in X12.7.

8.2.1.2.4. MAY have a “default value” mechanically generated by the CICA Database Tool, which can be overridden.

8.2.1.3. **DocumentDescription.**

8.2.1.3.1. MUST be free-form text that describes the meaning, use, and overall purpose of the Document.

8.2.1.4. **DetailMaxOccurs**.

8.2.1.4.1. MUST indicate the maximum number of times the Detail Area can repeat.

8.2.1.4.2. MUST be equal to the corresponding value in the specified Template, or a “hardening” of it (e.g., unbounded in the Template, and 25 here).

8.2.1.5. **ResponsibleSubCommittee**.

8.2.1.5.1. MUST designate the ASC X12 Subcommittee with primary responsibility for maintenance.

8.2.2. A **Document Member** is defined by the following:

8.2.2.1. **DocumentSlot**.

8.2.2.1.1. MUST be the same as the SlotName in the specified Template.

8.2.2.2. **ModuleSlotXMLName**.

8.2.2.2.1. MUST represent the physical/syntactic name assigned to the Slot within this Document.

8.2.2.2.2. It MUST be unique across all Slots within the Document.

8.2.2.2.3. It MAY be the same as a Slot in a different Document.

8.2.2.2.4. It MUST comply with the naming conventions in X12.7.

8.2.2.3. **ModuleName**.

8.2.2.3.1. MUST indicate the name of the Module designated to fill the Slot, as defined in the CICA Architecture.

8.2.2.4. **ModuleUsageName**.

8.2.2.4.1. MUST represent the logical/semantic name assigned to the Document Member that reflects the Module’s usage within the Slot in the Document.

8.2.2.4.2. MUST be unique across all Modules within the Document.

8.2.2.4.3. MAY be the same as a Module in a different Document.

8.2.2.5. **ModuleXMLUsageName**

8.2.2.5.1. MUST represent the physical/syntactic name assigned to the Document Member that reflects the Module’s usage within the Slot in the Document; it corresponds to ModuleUsageName, above.

8.2.2.5.2. MUST be unique across all Modules within the Document.

8.2.2.5.3. MAY be the same as a Module in a different Document.

8.2.2.5.4. MUST comply with the naming conventions in X12.7.

8.2.2.5.5. MAY have a “default value” mechanically generated by the CICA Database Tool, which can be overridden.

8.2.2.6. **ContextCategoryValueList**.

8.2.2.6.1. MUST provide a list of ebXML context that apply to the selection of this Module to occupy the Slot. These could potentially be used by a rules engine to programmatically determine which Module is appropriate for a Document that addresses given business process. At this time, no such engine has been implemented; hence this entry is purely informational.

8.2.2.7. **ReqDesignator**.

8.2.2.7.1. MUST indicate whether or not the Module is required (mandatory), or optional. This value overrides the corresponding value defined for the Slot in the Template.

8.2.2.7.2. MUST be either R for Required, or O for Optional.

8.2.2.7.3. MAY be the same as the defined for the Slot in the Template, or may be a “hardening” of it, i.e. Optional in the Template, and Required in the Document.

8.2.2.7.4. MUST be be compatible with the **MinOccurs** and **MaxOccurs** values for this member.

8.2.2.8. **MinOccurs**.

8.2.2.8.1. MUST indicate the minimum number of times this Module may repeat in this Slot of the Document. If the Module is in a Slot is in the Detail area, this represents the minimum number of times the Module may repeat within <u>each</u> occurrence/iteration of the entire Detail area. This value overrides the corresponding value defined for the Slot in the Template.

8.2.2.8.2. MUST be equal to the corresponding value for the Slot in the associated Template, or a “hardening” of it - it must be equal to or greater than the value in the Template, i.e. 1 in the Template, and 2 in the Document.

8.2.2.8.3. It MUST be a positive integer, i.e. equal to or greater than zero.

8.2.2.8.4. It MUST be compatible with the **ReqDesignator**, i.e. if Optional, **MinOccurs** should be zero, if Required, **MinOccurs** should be one or greater.

8.2.2.9. **MaxOccurs**.

8.2.2.9.1. MUST indicate the maximum number of times this Module may repeat in this Slot of the Document. If the Modules is in a Slot is in the Detail area, this represents the maximum number of times the Module may repeat within <u>each</u> occurrence/iteration of the entire Detail area. This value overrides the corresponding value defined for the Slot in the Template.

8.2.2.9.2. MUST be equal to the corresponding value for the Slot in the associated Template, or a “hardening” of it, i.e. unbounded in the Template, and 10 in the Document.

8.2.2.9.3. MUST be either unbounded or a postive integer, and equal to or greater than the **MinOccurs** value.

APPENDIX A: CORE COMPONENTS CONTEXT CATEGORIES

In keeping with ASC X12's goal to align with the ebXML Core Components work, the following table and descriptive text are reproduced from Section 6.2.2 of the UN/CEFACT – ebXML Core Components Technical Specification, Part 1 (8 February 2002, Version 1.8). The UN/CEFACT – ebXML Core Components Technical Specification is copyrighted by UN/CEFACT and these excerpts are reproduced with that body's permission.

Note: The URL for the full document is

http://www.unece.org/cefact/ebxml/ebXML_CCTS_Part1_V1-9.pdf

A comprehensive list of values must be specified for each context category. The ebXML CC specification has identified one or more available sources for each category. X12 plans to identify an "X12 selection" for the context categories that have multiple resources.

6.2.2 Approved Context Categories

Table 6-4 contains the eight approved *Context Categorie*s.

[C32] When describing a specific *Business Contex*t, a set of values will be assigned to the business situation being formally described.

[C33] Applied *Business Context* will be from the list of approved context categories.

Table 6-4. Approved Context Categories

Business Process	The business process as described using the ebXML Catalogue of Common Business Processes as extended by the user.
Product Classification	Factors influencing semantics that are the result of the goods or services being exchanged, handled, or paid for, etc. (e.g. the buying of consulting services as opposed to materials)
Industry Classification	Semantic influences related to the industry or industries of the trading partners (e.g., product identification schemes used in different industries).
Geopolitical	Geographical factors that influence business semantics (e.g., the structure of an address).
Official Constraints	Legal and governmental influences on semantics (e.g. hazardous materials information required by law when shipping goods).
Business Process Role	The actors conducting a particular business process, as identified in the Catalogue of Common Business Processes.
Supporting Role	Semantic influences related to non-partner roles (e.g., data required by a third-party shipper in an order response going from seller to buyer.)
System Capabilities	This context category exists to capture the limitations of systems (e.g. an existing back office can only support an address in a certain form).

6.2.2.1 Business Process Context

In describing a business situation, generally the most important aspect of that situation is the business activity being conducted. *Business Process Context* provides a way to unambiguously identify the business activity. To ensure consistency with business process activities, it is important to use a common point of reference. The definitive point of reference for international standards is the UN/CEFACT *Catalogue of Common Business Processes*

[C34] Assigned *Business Process Contexts* shall be from the standard hierarchical classification: provided as part of the UN/CEFACT *Catalogue of Common Business Processes.*

[C35] *Business Process Context* values may be expressed as a single business process at any level, or may be expressed as a set of business processes at any level.

[C36] *Business Process Context* values may be taken from extensions to the business processes described in the *Catalogue of Common Business Processes* as provided for in that document.

[C37] When business process extensions are used, they shall include full information for each value sufficient to unambiguously identify which extension is providing the value used.

6.2.2.2 Product Classification Context

The Product Classification Context describes those aspects of a business situation related to the goods or services being exchanged by, or otherwise manipulated, or concerned, in the business process. Recognised code lists exist that provide authoritative sources of product classification contexts.

[C38] A single value or set of values may be used in a *Product Classification Context.*

[C39] If a hierarchical system of values is used for *Product Classification Contex*t, then these values may be at any level of the hierarchy.

[C40] If more than one classification system is being employed, an additional value specifying which classification scheme has supplied the values used shall be conveyed.

[C41] Product classification context code values shall be taken from recognised code lists to include:

- *Universal Standard Product and Service Specification* (UNSPSC)
  - Custodian: Electronic Commerce Code Management Association (ECCMA)
- *Standard International Trade Classification* (SITC Rev .3)
  - Custodian: United Nations Statistics Division (UNSD)
- *Harmonised Commodity Description and Coding System* (HS)
  - Custodian: World Trade Organization (WTO)
- *Classification Of the purposes of non Profit Institutions serving households* (COPI)
  - Custodian: UNSD (This provides a mapping between the first three.)

6.2.2.3 Industry Classification Context

The Industry Classification Context provides a description of the industry or sub-industry in which the business process takes place.

[C42] An *Industry Classification Context* may contain a single value or set of values at any appropriate level of the value hierarchy.

[C43] The *Industry Classification Context* value hierarchy must be identified.

[C44] *Industry Classification Context* code values shall be taken from recognised code lists to include:

- *International Standard Industrial Classification* (ISIC) -- Custodian: UNSD
- *Universal Standard Product and Service Specification* (UNSPSC) Top-level Segment [digits 1 and 2] used to define industry. --Custodian: ECCMA

[Note] There are many other industry classification schemes that may be used for *Industry Classification Contex*t.

6.2.2.4 Geopolitical Context

Geopolitical Contexts allow description of those aspects of the business context that are related to region, nationality, or geographically based cultural factors.

[C45] *Geopolitical Context* shall consist of appropriate continent, economic region, country, and region identifiers.

[C46] *Geopolitical Regional Classification* may associate one or more values with any business message or component. are related to region, nationality, or geographically based cultural factors. country, and region identifiers. any business message or component.

[C47] *Geopolitical Regional Classification* shall employ the following hierarchical structure:

```
Global
  [Continent]
    [Economic Region]
      [Country] - ISO 3166.1
        [Region] - ISO 3166.2
```

[C48] At any level of the *Geopolitical Regional Classification* hierarchy, a value may be a single value, a named aggregate, or cross-border value.

[C49] *Geopolitical Regional Classification* hierarchy values shall structured as follows:

- **Single Value:** A single value indicating a single continent, economic region, country, or region, depending on position within the hierarchy.
- **Named Aggregate:** A related group of values (which may themselves be single values, named aggregates, or cross-border pairs of values), which have been related and assigned a name. A named aggregate contains at least two values.
- **Cross-Border:** One or more pairs of values, designated *T*o, *Fro*m, or *Bi- directional,* indicating the direction of cross-border context. Values may be named aggregates or single values.

[Example] The following example shows an extract of the basic, single-value hierarchy of recommended values, based on the common ISO 3166.1 *Country Codes.* (The value at the top of any hierarchy is always understood to be *Globa*l.)

```
Europe
  Eastern Europe
    AL – ALBANIA
    AM – ARMENIA
```

[C50] Points in the *Geopolitical Regional Classification* hierarchy shall be specified by the use of the node value, or by the full or partial path.

[C51] The full path of the *Geopolitical Regional Classification* hierarchy must be used to understand the hierarchy when complex constructs are employed.

[C52] A single-point specification is understood to inherit all of the properties of the single-value hierarchy except where otherwise specified.

[C53] *Geopolitical Values* will be taken from ISO 3166.1 and 3166.2

6.2.2.5 Official Constraints Context

The Official Constraints Context category describes those aspects of the business situation that result from legal or regulatory requirements and similar official categories. This category contains two distinct parts:

- Regulatory and Legislative. These are normally unilateral in nature and include such things as customs.
- Conventions and Treaties. These are normally bi- or multilateral agreements and as such are different from regulatory and legislative constraints.

[C54] The *Official Constraints Context* will consist of at least two values:

- Identification of the legal or other classification used to identify the context values.
- Identification of the official constraint itself. These values may represent a hierarchical structure depending on the official constraints system being referenced.

Because there is no known global classification of all *Official Constraints Contexts* as used here, any implementation must provide a set of recognised official constraints classifications for use within the appropriate *Core Components* Registry implementation.

[C55] Individual *Core Component* implementations shall register used official constraint classification schemes with the appropriate supporting *Core Components* Registry implementation.

6.2.2.6 Business Process Role Context

The Business Process Role Context describes those aspects of a business situation that are specific to an actor or actors within the business process. Its values are taken from the set of Role values provided by the Catalogue of Common Business Processes. A Business Process Role Context is specified by using a value or set of values from this source.

[C56] *Business Process Role Context* values shall be taken from an approved list provided by the business process model library being employed.

[C57] The UN/CEFACT *Catalogue of Common Business Processes* shall be the definitive source of *Business Process Role Context* values for all UN/CEFACT *Business Information Entitie*s.

6.2.2.7 Supporting Role Context

The Supporting Role Context identifies those parties that are not active participants in the business process being conducted but who are interested in it. A Supporting Role Context is specified with a value or set of values from a standard classification.

[C58] *Supporting Role Context* values shall be taken from the UN/EDIFACT *Code List for DE 3035 Party Roles.*

[Note] Users are cautioned that duplication exists in the current version of the required code list. UN/CEFACT will review this code list to clarify duplicates and identify non- *Supporting Role Context* values.

6.2.2.8 System Capabilities Context

This category identifies a system, a class of systems or standard in the business situation. The System Capabilities Context requires a least one pair of values: an identification of the classification scheme being used and a value from that scheme. A valid System Capabilities Context may include more than one such pair of values.

[C59] *Systems Capabilities Context* values shall consist of pairs of values. Each pair shall be comprised of an identification of the referenced classification scheme and the value(s) being employed.

[Note] There is no known classification of all types of information systems and standards. It is recommended that a mechanism for the registration of system and standard names be provided by the ebXML registry, as valid values for the *System Capabilities Context.*

Table 6-1 Permissible Representation Terms

Representation Term	**Definition**	**Links to *Core Component Type***
Amount	A number of monetary units specified in a currency where the unit of currency is explicit or implied.	Amount. Type
Code	A character string (letters, figures or symbols) that for brevity and / or language independence may be used to represent or replace a definitive value or text of an attribute. Codes usually are maintained in code lists per attribute type (e.g. colour).	Code. Type
Date	A day within a particular calendar year (ISO 8601).	Date Time. Type
Date Time	A particular point in the progression of time (ISO 8601).	Date Time. Type
Graphic	A diagram, graph, mathematical curves, or similar representation	Graphic. Type
Identifier	A character string used to establish the identity of, and distinguish uniquely, one instance of an object within an identification scheme from all other objects within the same scheme. [Note: Type shall not be used when a person or an object is identified by its name. In this case the *Representation Term* "Name" shall be used.]	Identifier. Type
Indicator	A list of two, and only two, values that indicate a condition such as on / off; true / false etc. (synonym: "Boolean").	Indicator. Type
Measure	A numeric value determined by measuring an object. Measures are specified with a unit of measure. The applicable unit of measure is taken from UN / ECE Rec. 20.	Measure. Type
Name	A word or phrase that constitutes the distinctive designation of a person, place, thing or concept.	Text. Type
Percent	A rate expressed in hundredths between two values that have the same unit of measure.	Numeric. Type
Picture	A visual representation of a person, object, or scene	Picture. Type
Quantity	A number of non-monetary units. It is associated with the indication of objects. Quantities need to be specified with a unit of quantity.	Quantity. Type
Rate	A quantity or amount measured with respect to another measured quantity or amount, or a fixed or appropriate charge, cost or value e.g. US Dollars per hour, US Dollars per Euro, kilometre per litre, etc.	Numeric. Type
Text	A character string generally in the form of words of a language.	Text. Type
Time	The time within a (not specified) day (ISO 8601).	Date Time. Type
Value	Numeric information that is assigned or is determined by calculation, counting or sequencing. It does not require a unit of quantity or a unit of measure	Numeric. Type

In addition to permissible representation terms for *Core Components*, there are also permissible representation terms for *Aggregate Core Components* and *Core Component Types*. Table 6-2 contains the permissible representation terms that apply to *Aggregate Core Components* or *Core Component Types*.

[C31] The *Representation Term* for *Aggregate Core Components* or *Core Component Types* shall be one of the list of permissible *Aggregate Core components* or *Core Component Type Representation Terms*

Table 6-2 Permissible Representation Terms for Aggregate Core Components or Core Component Types

Representation Term	**Definition**	**Links to *Core Component Type***
Details	The expression of the aggregation of *Core Components* to indicate higher levelled information entities	Not Applicable
Type	The expression of the aggregation of *Core Components* to indicate the aggregation of lower levelled information entities to become *Core Component Types*. All *Core Component Types* shall use this *Representation Term*	Not Applicable
Content	The actual content of an information entity. *Content* is the first information entity in a *Core Component Type*	Used with the *Content Components* of *Core Component Types*