* * * D R A F T * * *

ASC X12 Reference Model for XML Design Rules

Version 0.4

TABLE OF CONTENTS

FOREWORD

This paper was motivated by the action item that X12's Communications and Controls subcommittee (X12C) took at the August 2001 XML Summit to develop "draft design rules for ASC X12 XML Business Document development". Acting on that action item, X12C's EDI Architecture Task Group (X12C/TG3) determined that XML design rules could not be developed in a vacuum, without a basis for determining which XML features to use and how to use them. Thus the group also set about developing a philosophical foundation and putting forth some general design principals. This Reference Model covers those topics in addition to a preliminary set of design rules.

The approach discussed herein is intended to be the foundation for X12's future XML development. It is consistent with the decisions of X12's Steering Committee to develop its XML work within the ebXML framework. We expect it to undergo further refinement as the work progresses from its current status as a Task Group Reference Model to a full X12 standard.

Contributors to this project included:

- ***List of names to be completed***.

1.0 INTRODUCTION

The Extensible Markup Language (XML), developed by the World Wide Web Consortium (W3C), is a specification designed for Web documents that enables the definition, transmission, validation, and interpretation of data between applications and between organizations. It is a freely available and widely transportable approach to well-controlled data interchange that is open and accessible to the business community. The technology itself allows the design of languages that suit particular needs and harmonious integration into a general infrastructure that is extensible enough to meet requirements and adaptable enough to incorporate emerging new technologies.

The extensibility of XML is the main advantage of this technology as well as its main disadvantage. The ability to create custom-tailored markup languages can lead to a proliferation of languages within business entities. This may not be critical in simple browser-to-web-server solutions, but in business-to-business exchanges, it is very undesirable and costly. The development of document definition methodologies and XML design rules is of paramount importance to stem the flow of divergent XML solutions and ensure smart and efficient use of technology resources.

Much work has been done in the document definition and core components arena by ebXML, ANSI ASC X12, and UN/CEFACT Work Groups. Every effort has been made to build on that foundation. The XML design rules presented in this Reference Model are based on design decisions reached through a process of issue identification, presentation of examples, and evaluation of the pros and cons of each available action according to W3C approved specifications. They provide a set of best practices that define the creation of XML representations of standard business messages.

1.1 Target Audience

The X12 XML initiative is targeted at every sector of the business community, from international conglomerate to small and medium sized enterprises (SME) engaged in business-to-business (B2B), business-to-consumer (B2C) and application-to-application (A2A) trade. With that audience in mind, the X12 XML initiative is committed to developing and delivering products that will be used by all trading partners interested in maximizing XML interoperability within and across trading partner communities.

The motivation to develop common standards for document interchange is to enable independent business entities to communicate with minimal additional cost and effort across a wide range of business opportunities. One way organizations can gain advantages of interoperability is by establishing a common set of “good” XML and XML Schema guidelines. The current W3C XML specifications were created to satisfy a very wide range of diverse applications and this is why there may be no single set of “good” guidelines on how best to apply XML technology.

Although this document is created by X12 for its own use, it seeks a wider audience. While standards developers comprise most of the people who attend X12 standards development meetings, the majority of implementers may never participate in development of standards. SME are virtually not represented at standards development meetings but their needs can be served by products resulting from those efforts. Industries or associations who chose not to participate within the X12 environment can nevertheless follow these guidelines and position themselves to meet interoperability demands of the next generation of e-business standards.

Design rule decisions presented here are intended to balance the needs of all users of the standards. What seems like an advantageous decision from one viewpoint can be disadvantageous from another, but the intent was to produce guidelines to serve the common good.

1.2 High Level Design Principles

The following overall principles govern this design.

- Alignment with other standards efforts - We shall align with other standards efforts where possible and appropriate. Such efforts include but are not limited to UN/CEFACT and OASIS ongoing ebXML work, World Wide Web Consortium, and OASIS UBL.

- Simplicity - We shall keep components, interactions, use of features, choices, etc. to a reasonable minimum.

- Prescriptiveness - This means that, for example, schemas shall be as specific as possible for their particular intended usage, and not generalized. When applied to schemas, this leads to more schemas, each with fewer optional elements and with fairly tight validations. This means that schemas actually used by anyone (rather than template schemas for starting points) would tend to be analogous to an Implementation Guide of a transaction set rather than the full standard definition of the transaction set.

- Randomness - When applied to processing an electronic business document, this means that when the document is being processed there are a limited number of variations that may occur in the data. It is related to optionality and prescriptiveness. We shall keep randomness to a practical minimum. NOTE: This provides a good philosophical basis for disallowing things like substitution groups and the "ANY" content model when designing document schemas.

1.3 Background

The Extensible Markup Language (XML) history and ebXML Business Process and Core Components have been part of the development that has brought us to where we are today. Appendix G contains a more detailed review of each of these.

2.0 SCOPE

2.1 Introduction

This Reference Model addresses the semantic and syntactic representation of data assembled into business messages. The semantic representation defines an overall architectural model and refines the model to an abstract level of detail sufficient to guide the message development process. The syntactic representation utilizes features of the target syntax, while imposing semantic-to-syntax mapping rules and syntax constraints intended to simplify the task of interfacing business messages to business information systems and processes.

The large-scale structure of this architecture has five discrete levels of granularity. Each level builds on the levels below it in manners particular to their differing natures. The five levels are:

Template
Module
Assembly
Block
Component

The first two levels, Template and Module, provide features that promote interoperability between national cross-industry standards and proprietary user communities. The remaining three levels, Assembly, Block, and Component have characteristics expressly designed around a rational semantic model for granularity. Specifications of optionality and repetition are supported for all levels with the exception of the Template level. Special attention has been paid to the differing needs of senders and receivers in expressing the use of optionality and repetition required by their particular business practices.

The five-level structure of this architecture is designed to provide useful granularity, while at the same time preserving a useful semantic clarity.

Design rules come in two basic forms:

- Syntactic, and
- Semantic

An example of a syntactic design rule in X12 would be the basic data types, i.e. alphanumeric, date, etc. An example of a semantic design rule in X12 would be the general prohibition against duplication. These two aspects of design cannot stand alone. The existing X12 design rules are a direct outgrowth of the particular X12 syntax and the history that created it.

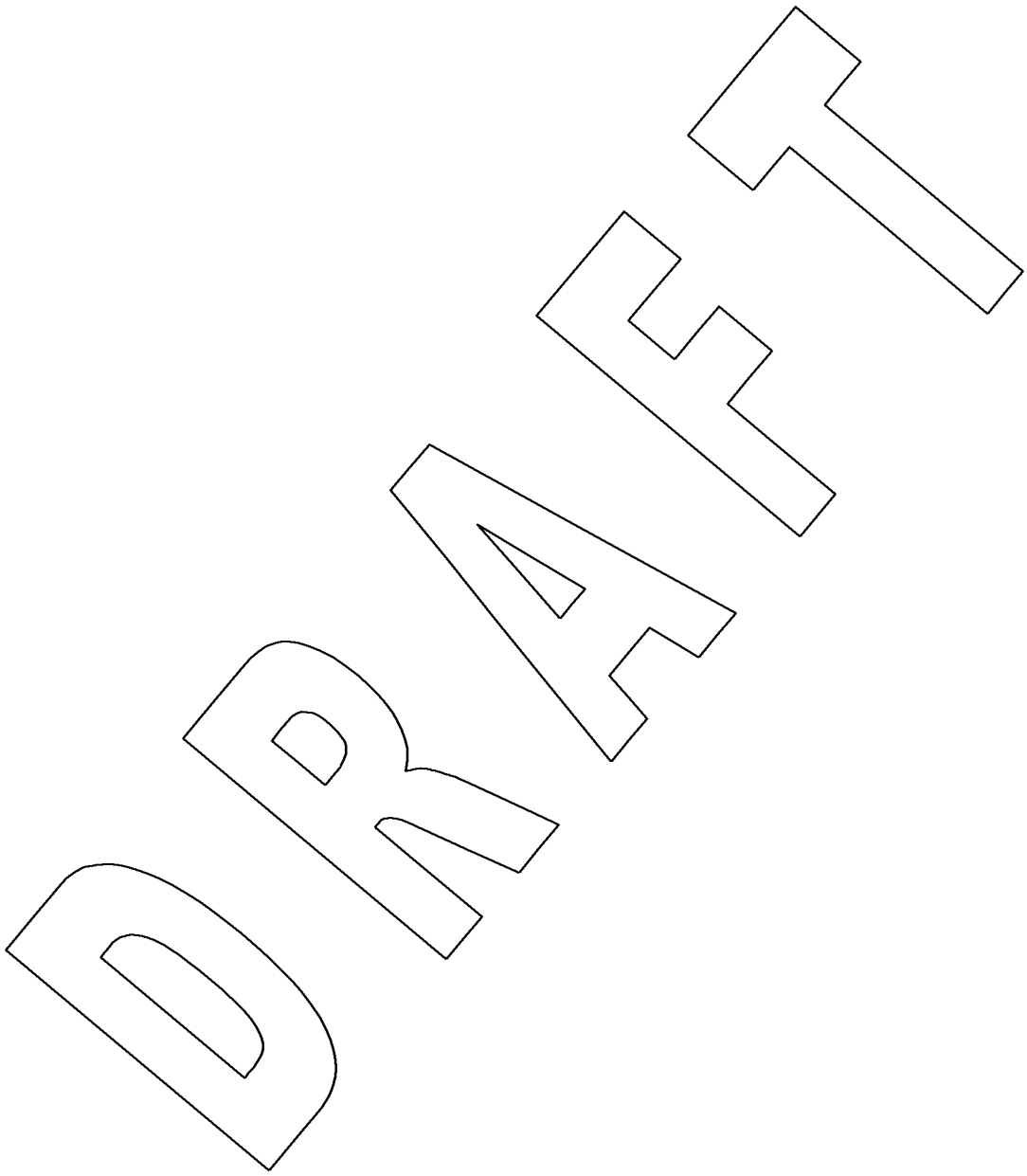
For the ASC X12 XML Reference Model, a semantic design approach has been selected, breaking the EDI lexicon into units for re-use. This approach has some pitfalls that result from a decomposition of EDI issues using only syntax as a guide.

2.2 Support for Proprietary Efforts

A primary requirement for this effort has been to meet a need first expressed at the first XML Summit in August 2001. This was a desire for non-X12 participants to contribute and make use of X12 work but in a manner that didn't require an all-or-nothing commitment to either the X12 process or X12 conclusions in every detail. The top two layers, Template and Modules, directly support this need. An external entity, corporation, organization, or individual can contribute fully-constructed Modules that fit into a Template.

The presumption is that differences in detail within a module might reflect unique business needs of the contributor.

The level of conformance applied to these contributions would be two-fold. First, does it meet the function and purpose expressed for a particular Slot in a Template? Second, does it conform to the purely syntactic design rules established? A "cross-industry usefulness" test would not be applied. A "duplication of existing item" test would not be applied. Adherence to the X12 philosophical structuring of the bottom three layers (Assembly, Block, Component) would not be required.



3.0 RESOURCES

The following documents provided resources for this document.

- http://www.xfront.com/ - XML Schema Best Practices as maintained by Roger L. Costello
- http://www.ibiblio.org/xml/ - Café Con Leche
- http://www.w3.org/XML – XML Schema Specifications
- http://xml.coverpages.org/sgmlnew.html – Archive of Robin Cover's XML Cover Pages at OASIS
- http://www.ietf.org/rfc/rfc2119.txt?number=2119 – Internet Engineering Task Force Request for Comments 2119
- http://www.tibco.com/products/extensibility/resources/index_best.htm – Tibco's XML Resources Center Best Practices
- http://www.ebxml.org – ebXML Project
- http://www.ebtwg.org – UN/CEFACT electronic business temporary work group
- Ducket, Jon, Oliver Grffin, Stephen Mohr, Francis Norton, Ian Stokes-Rees, Kevin Williams, Kurt Cagle, Nikola Ozu, and Jeni Tennison; *Professional XML Schemas;* Wrox Press, Birmingham UK, 2001
- Dodds, Leigh, "Designing Schemas for Business to Business E-Commerce", http://www.xml.com/lpt/a/2000/06/xmleurope/ecommerce.html
- Gregory, Arofan T. "XML schema design for business-to-business e-commerce", XML Europe Conference, 2000
- http://www.ebxml.org, Core Components Overview Vestion 1.05, May 10, 2001.
- http://quickplace.hq.navy.mil/QuickPlace/navyxml/Main.nsf/057A71D114B95B0D85256AF5006CAD86/1921E59CBABDEE2D85256AFB00605CB3, Initial DON XML Developer's Guide, October 29, 2001

4.0 HIGH LEVEL DESIGN ARCHITECTURE

4.1 The Vision -- An Analogy

Imagine a horizontal bar containing a set of seven (7) wheels, each having eight (8) surfaces. As the wheels are rotated on the bar, different surfaces are revealed [as illustrated in Figure 1], and on each of the eight surfaces is a different word.

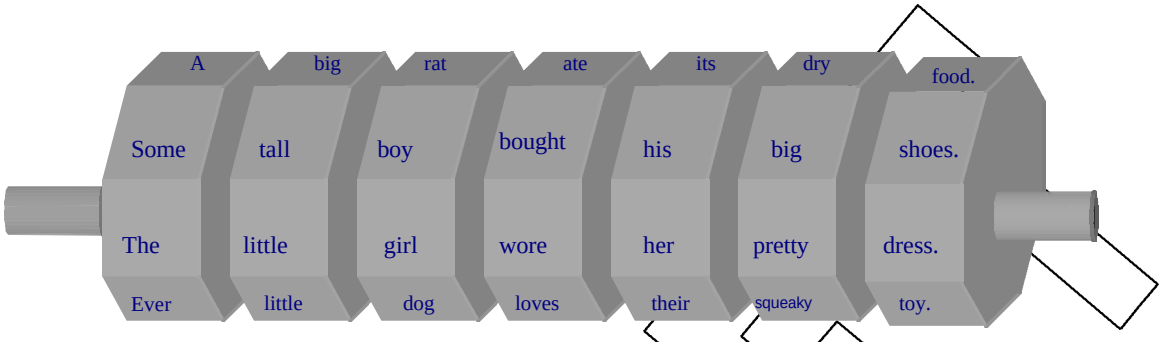


Figure 1

Each wheel rotates independently, thus the number of potential combinations grows exponentially with each additional wheel. With each possible combination of block surfaces, a complete and meaningful sentence is constructed.

This works because of grammar, the very thing that sentence structure is about. The logical placement of the blocks [noun phrase + verb phrase], with each block representing a part of speech [article, adjective, noun, verb, etc.], enables each combination to yield a meaningful sentence – some rather silly, but still valid.

The electronic messaging problem is analogous to that of natural language. We have a basic business need that is common, Invoicing, but some of the details are different due to product differences, which in turn results in packaging differences, which ultimately results in shipping differences, etc.

This analogy illustrates some concepts which, when employed within electronic business message design, provides a powerful solution to coping with differences, while at the same time retaining the conciseness needed to direct implementation. Further, this approach also provides solutions for coping with external organizations requiring a transition between proprietary approaches to that of a fully compliant standards solution.

Business documents are much more constrained than natural language communications and must only contain enough information to communicate *who, what, when, where* and *why.*

- W*ho* answers which parties participate in the business transaction and the actors involved in the exchange.
- *What* answers the primary subject or purpose of the message.
- *When* answers timing details.
- *Where* answers location details.
- *Why* is typically answered by the message type itself, along with accompanying reference information.

4.2 Context Inspired Component Architecture -- modularly flexible Smart Messages

Overview

The Context Inspired Component Architecture, "CICA", is based on the results on many years of critical analysis within the EDI/E-Business standardization efforts. This architecture leverages the best ideas to date in E-Business development, and applies a few semantic rules and adds some levels of abstraction.

The architecture of CICA includes five (5) layers, as illustrated in Figure 2.

The **Template** is conceptually the highest layer to the architecture, providing a "how used" description of the contents of a complete business document.

The **Module** is physically separate from the Template, but associated with the template on a Context case basis. In other words, the Module is loosely associated with the Template, and only bound with the Template when a predetermined condition is met, Context. Modules answer, at the document level, *Who, What, When, Where* or *Why*. Modules are made up of one or more Assemblies and/or Blocks.

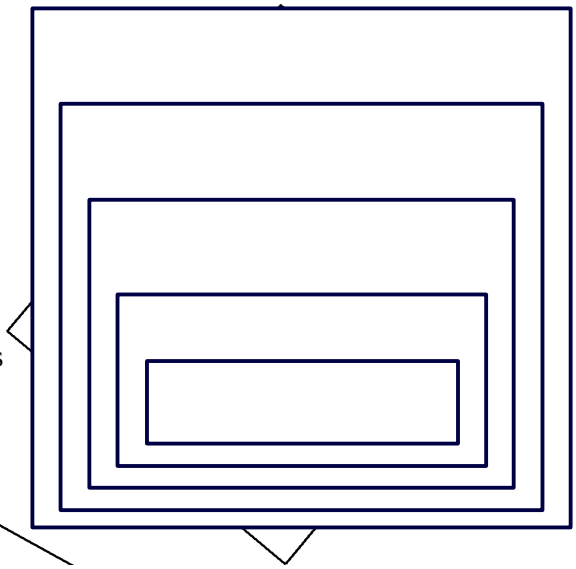


Figure 2

Assemblies are reusable aggregations of Blocks.

Blocks are reusable, semantically limited units of information. Blocks specify a Party, Resource, Event, Location or Condition.

The **Component** is the fifth and final layer.
Data elements are defined within Components, and Components are placed within Blocks. Components specify either identity information or characteristics for the given Block.

The terminology in Figure 2 can be related to the analogy in Section 4.1. The set of seven wheels in Figure 1 corresponds to a template, each wheel corresponds to a module slot, and each surface of the wheel corresponds to a module.

The highest level of the CICA architecture is the Business Document Template, "Template". The Template is divided into a header, detail and summary, and within each section are one or more Module Slots, "Slots". The Slots are a description of the abstract purpose of the slot, and act as placeholders for details, which are specified in Modules. For each Slot, there are one or more Modules, the use of which is determined by a specific pre-established context. For example, consider Figure 3, which depicts a Template with three (3) slots in the header, and two (2) slots in the detail. If the cube shaped Slot in the detail section of the Template represents Line Item, there are three (3) sector specific Modules which fit

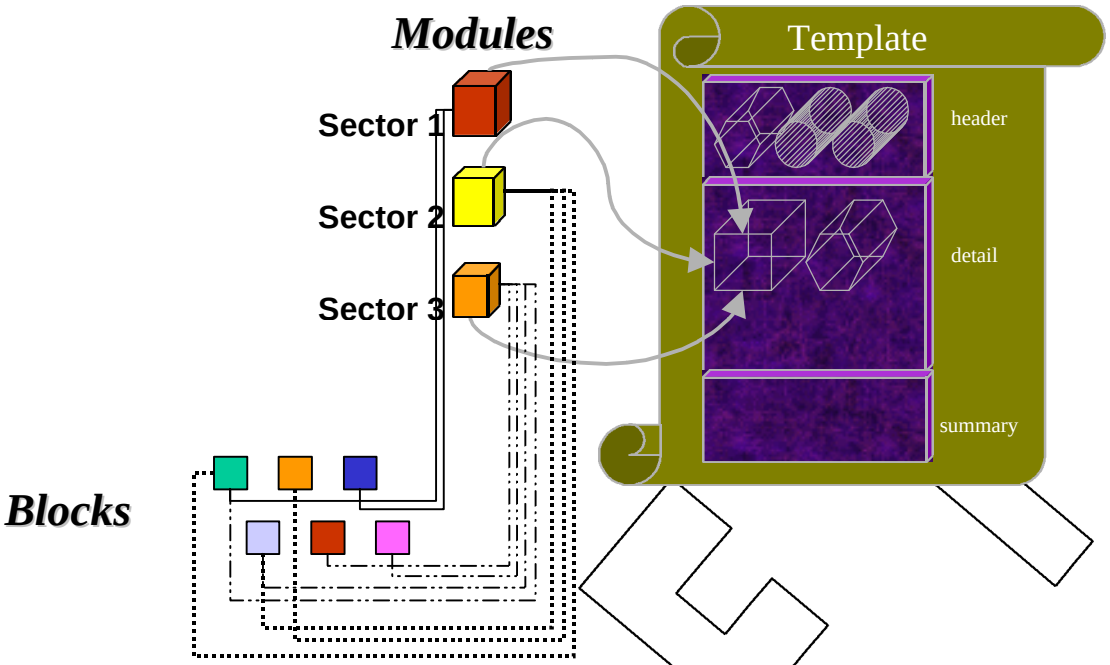


Figure 3

into the Slot, each of which is used according to the context which specifies its use. For each Slot, one or more Modules are created in order to fulfill the purpose specified by the Slot. Each Module is either associated with the Slot as either the default Module, or with a specific usage Context. In order to generate a Business Document, Context is specified and the requisite links are drawn upon to compile the Context Specific Business Document.

4.3 Relationship between Vision and the CICA Architecture.

The vision, presented in Section 4.1 and Figure 1, is recast to present the key CICA constructs.

Each wheel diagram represents a single Template, Figure 4.

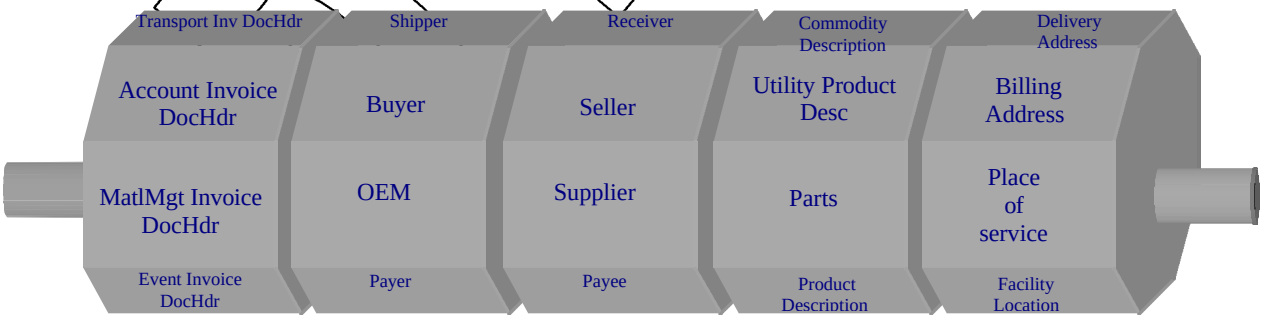


Figure 4

A Template is determined based on business process circumstance triggering a unique situation. For example, in Invoicing, there are two distinct trigger events that result in entirely different arrangements and organizations of information – event based and account based. Event based Invoices are sent in response to a trigger event [Order, Shipment, etc.] Account based Invoices are sent at regular time intervals, with no specific precursor event. Therefore, a separate Template is used for each.

Each Wheel in the diagram is a template slot, which is an abstract specification for the Slot purpose. For example, one Slot might be for the "Buyer", even though various industries might have different terms used in place of "Buyer". Thus, each wheel surface represents a Module, which is where each industry specification for the Slot is specified with a Module. The Module contains all of the details required for that Context specific use of the Slot.

The power of this architecture is two fold. First, the modular flexibility provides structured flexibility, maintaining stability at the context specific level. The second is the underlying layers of semantics, which provide for levels of agreement. For example, it may not be possible to agree to the details of how to specify product, but it is possible to agree that this is the place where product must be specified. The layers provide a means to achieve agreement and harmonization that are practical.

4.4 Templates

Overview

Templates are the highest level construct in the architecture, and play a critical role in accomplishing modularly flexible messages. Modularly flexible messages are an important innovation in that the resulting Business Document is semantically concise, yet the Template provides a mechanism through Module substitution for flexibility. The result is to accomplish both of which would otherwise be considered opposing objectives – flexibility and semantically concise.

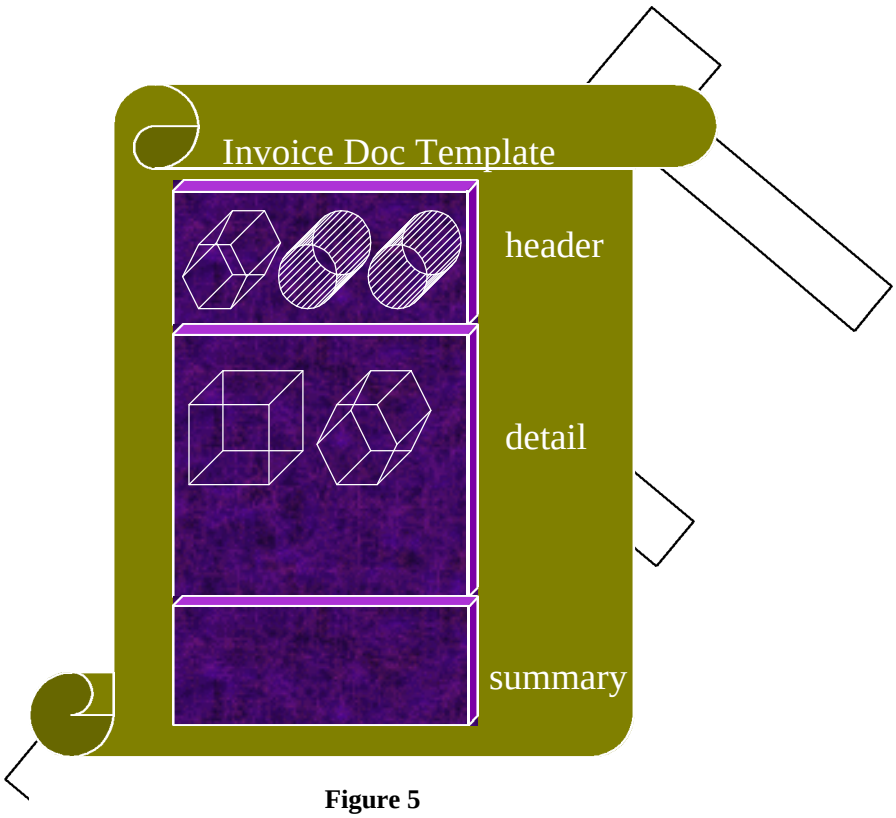


Figure 5

Templates are established for each business process specific use of a message. As in example, there are two fundamental procurement models, event and time based. Event based procurement involves a buyer and seller, where the buyer places a specific Order with the seller, the seller delivers in accordance with the Order, and the buyer is Invoiced in accordance with an event. Examples of this include catalog orders, trips to the store, traditional healthcare plans, etc. In contrast, time based procurement involves an arrangement whereby the buyer and seller have a pre-existing relationship, the goods/services are continuously available and used as desired, and invoicing occurs according to a time schedule. This procurement style includes any statement/time based invoicing methods, specifically: utilities, credit card, hotel stay, etc.

Contents

Shown in Figure 5 is a Template. A Template is divided into three logical areas, “Area”: header, detail and summary. These subdivisions have semantic significance in that header information applies to the entire Business Document and specifies the business context and parties to the business exchange, the

detail contains the subject of the message, and the summary contains summarized information about the detail [use of this section is generally discouraged].

Within each Area are zero or more Slots. Slots, depicted in Figure 6 by various 3D wire frame shapes, identify in abstract terms the logical composition of the message at the business purpose level. Slots are absolutely specific in term of the logical business purpose that they identify. The Slots are abstract in that they use a generic term, such as Seller, although various industries/sectors might use Supplier or Provider. The abstraction is in harmonized terms, generally recognized terms independent of industry or sector specific terminology. This aids in the reuse of the Templates, which are developed around Business Process requirements.

Slots do not contain contents or data elements. They serve as a logical break between the purpose of information and the detailed context specific contents.

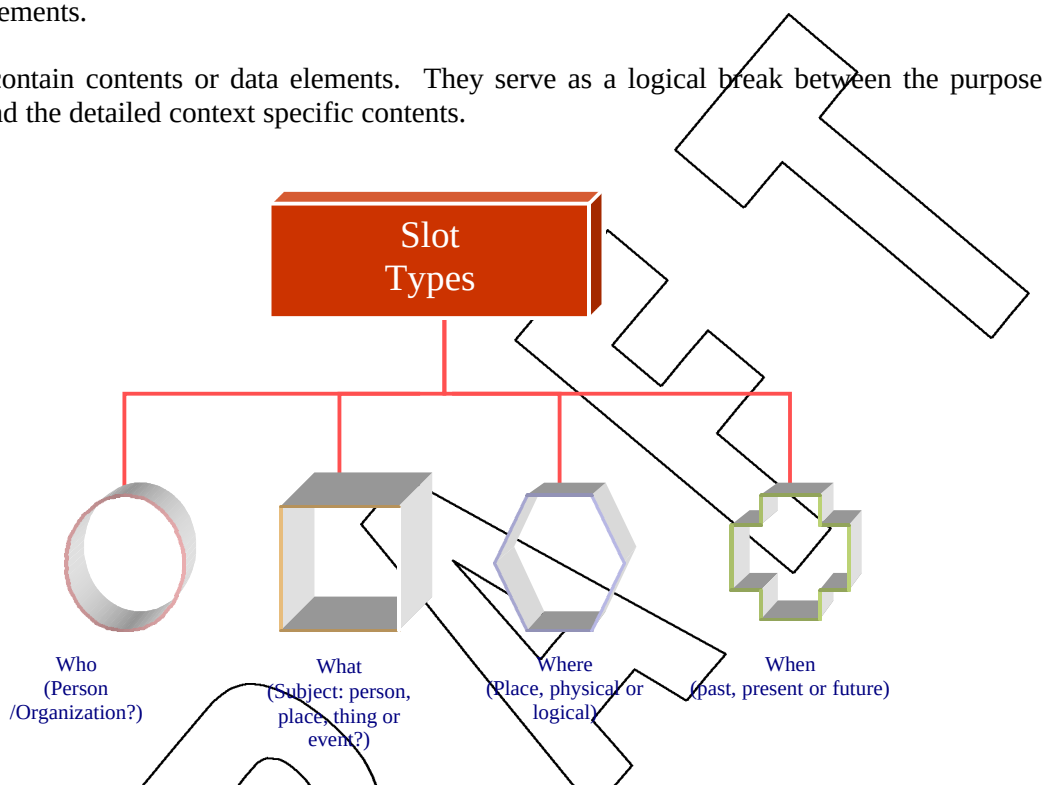


Figure 6

Slots, like Modules, are designed around the Business Document need to express the Who, What, When and Where, [as shown in Figure 6], which when combined detail a Business Document. Each Slot specifies only one of the Who, What, When or Where, of the Business Process.

4.5 Modules

Overview

Modules specify details in accordance with the abstract business purpose identified by the Slots in the Templates. A Module is required for each Slot identified in the Template, although it is possible that a single Module can fill more than one Slot in a Template. It is expected that some of the Slots will have only one or a small number possible Modules, such as Buyer or Seller. In other cases, there could potentially be many different modules, based on perhaps business sectors.

Figure 3, shown earlier, illustrates a situation where multiple Modules are associated with a single Slot. On the left are a set of Modules which can be plugged into the module slot. Each module in the example has some commonality – shown by the shared red filled box. This commonality in some compositions is not a requirement of peer modules, but what is certain is that there is different composition. Therefore, amongst various industry sectors there are differences in information requirements for modules, e.g. line item. The links, shown with arrows, are established for a context. What is meant by context is a specific business circumstance that unambiguously links the specific Module to the Module Slot in the Template.

At the Template level, for each of the Slots, context specific links are made between Slots and Modules. Modules can be reused many times across Templates, whether they are:

- Peer Templates: Templates which serve the same general business function, such as Invoice.
- Same Business Process: Templates used within the same business process. It is expected that a single Module could appear in multiple or all Templates used in the business process.
- Same Sector: Modules which are sector specific, such as ones specifying the sector's product/service, could be used in a variety of business processes in which sector members participate.

Types

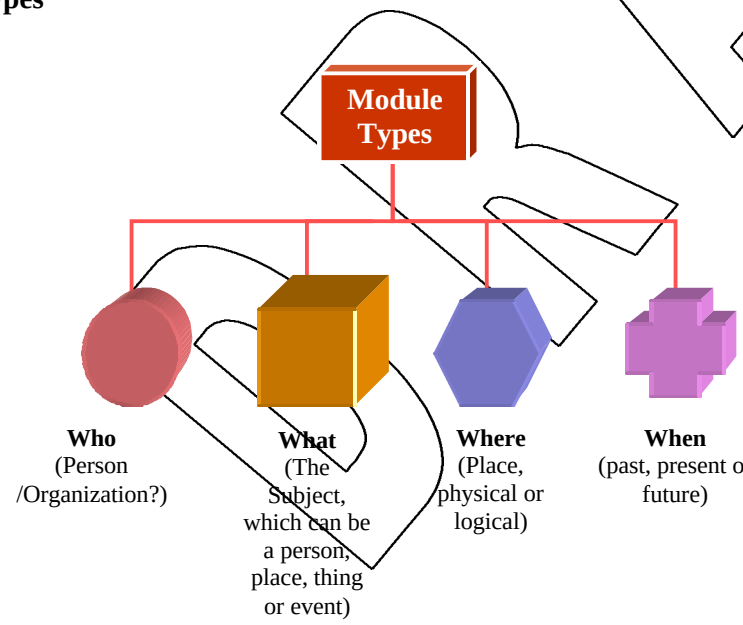


Figure 7

Modules, like Slots, are formed around semantically motivated boundaries. Grammatically speaking, like Slots, Modules specify either a Who, What, When, Where or Why, as illustrated in Figure 7. The Slot identifies the need for a module in terms of the Purpose, or business usage, which is identified in abstract terms. The Module supplies a set of details responding to the prescribed purpose, the Slot.

Business documents also need to explicitly specify the relationships among their components, to reflect the appropriate structure of

those components during assembly. Knowing how the pieces fit together in the overall structure encourages reuse of the components in other documents or processes. In some cases the structure will be simple, but where documents represent a large volume of different items, or multiple references (e.g. a ship notice containing items requested in separate purchase orders), the structure can easily become more complex.

Contents

Modules are made up of reusable constructs, which are either Assemblies or Blocks. A Module can be constructed from a single Block, or a set of Blocks and Assemblies.

Placing a Block or Assembly into a Module gives it a semantic purpose. Modules can be complex enough to require the use of multiple Blocks and Assemblies, although the primary purpose is singular. For example, within Vital Records exists the need to specify a party who has died, a decedent. The decedent is the Module, as shown in Figure 8, but peripheral to the decedent are the birth/parents, last address, spouse/marriage, and adoption/parents, etc. These are descriptive details about the decedent involving parties, locations, events, etc.

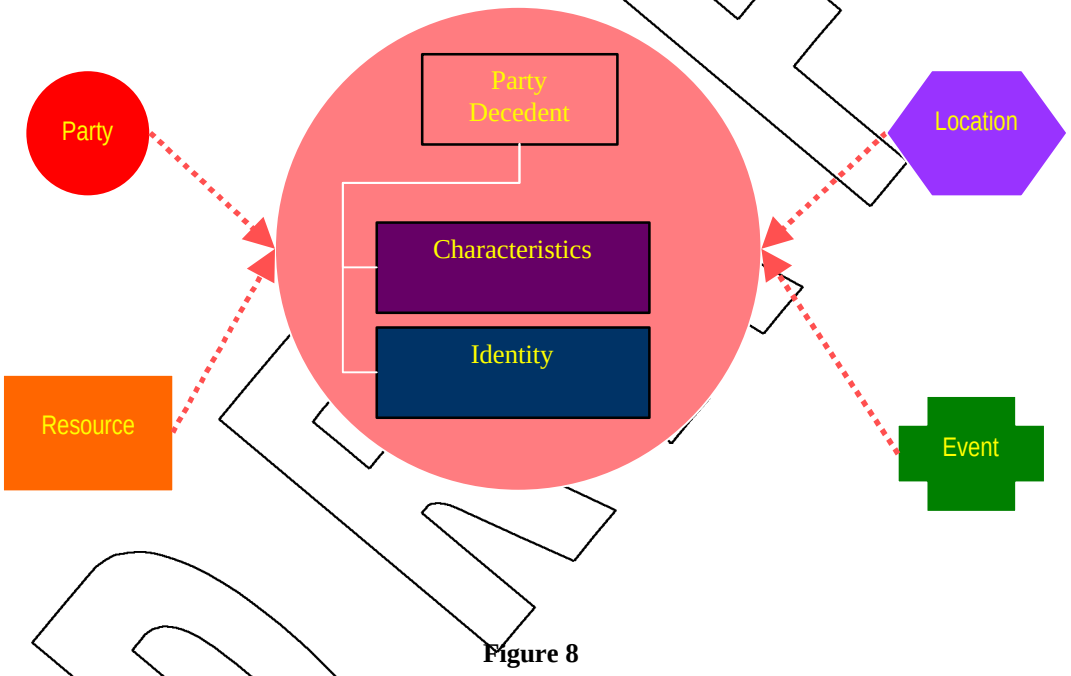


Figure 8

4.6 Assemblies

Overview

Assemblies are a construct used to create reusable groupings of Blocks. Like Blocks, they are independent of usage and fit between Modules and Blocks. Blocks, which are detailed in the following section, are semantically limited to specify a single Party, Location/Place, Resource, Event or Purpose. Various groupings of these constructs can be very convenient to construct for purposes of reuse and applying structure. For example, party + location are commonly used constructs and an Assembly is a convenient means for managing this reuse.

Assemblies also play a role in providing structure.

Types

Assemblies will typically have a primary type. In the example above using party + location, the purpose of the assembly is to specify a Party of a type that has a location. While location information is supplied, the primary purpose is to specify the Party.

Contents

Blocks are the only contents available to place within assemblies.

4.7 Blocks

Overview

Blocks are constructs, created to specify a single Party, Resource, Event, Location or Purpose. Blocks are concise units, in that they specify in detail and with all that applies to the Identification and Characteristics of the object being specified.

Types

Blocks specify a single noun, either a Party, Location, Resource, Event or Purpose [as shown in Figure 9]. Blocks have Identity information about the noun they are specifying and may include Characteristics. Anything less is not a Block; anything more must be an Assembly or Module.

The single noun is a critical element of this architecture's granularity. All Blocks are comprised of a single noun, therefore, semantically Blocks carry noun information and are predictable in terms of completeness. This granularity assures that peer Blocks are semantic equivalents. This is a foundation required to achieve modular flexibility.

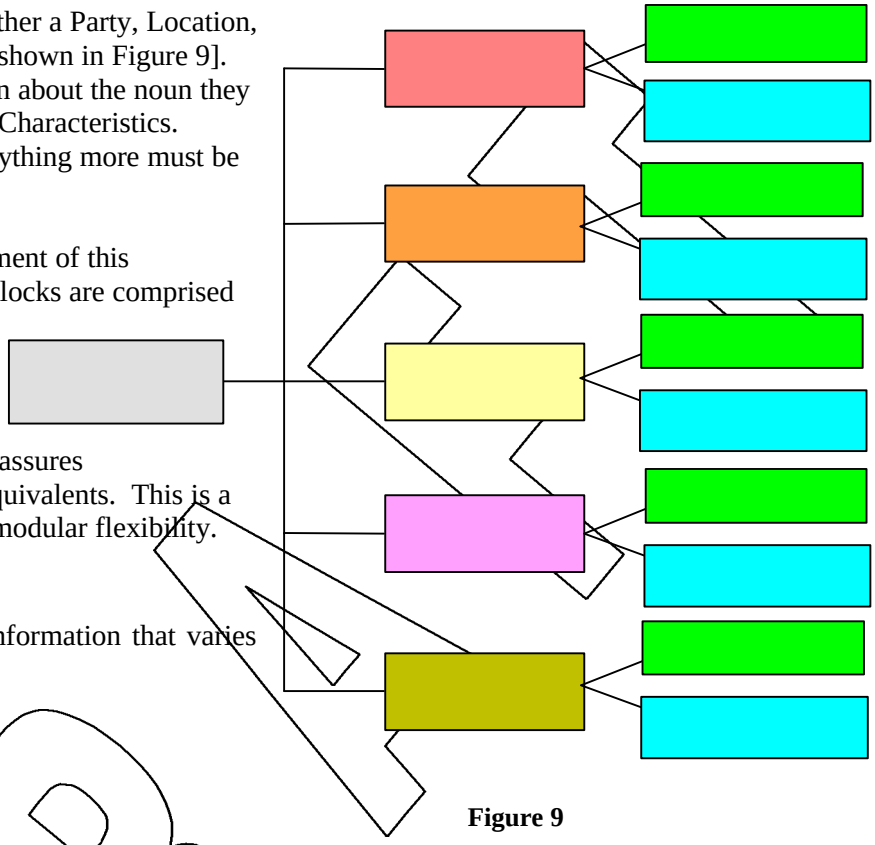


Figure 9

Contents

Each Block contains Identity information that varies depending on the type of Block

4.8 Components

Overview

Components are the lowest level contained within this architecture. Components, like Blocks, are formed based on the need for a physical arrangement of information. For example, given two types of Parties, an individual and an organization, the identity information required for the two types of Parties is significantly different, therefore the components used to specify identity are different. The need for different components results in the need for separate Blocks.

Types

Components are used to specify one of two types of information, Identity or Characteristics. Identity information is going to vary based on the Block type. The details required to identify a person are dramatically different from those details used to identify an event. Characteristics provide descriptive information, such as physical or demographic details. Typically, characteristics are a unit of measure or are one of a finite list. Some examples of characteristics are height, weight, hair color, class of service, property feature or quality, etc.,

Contents

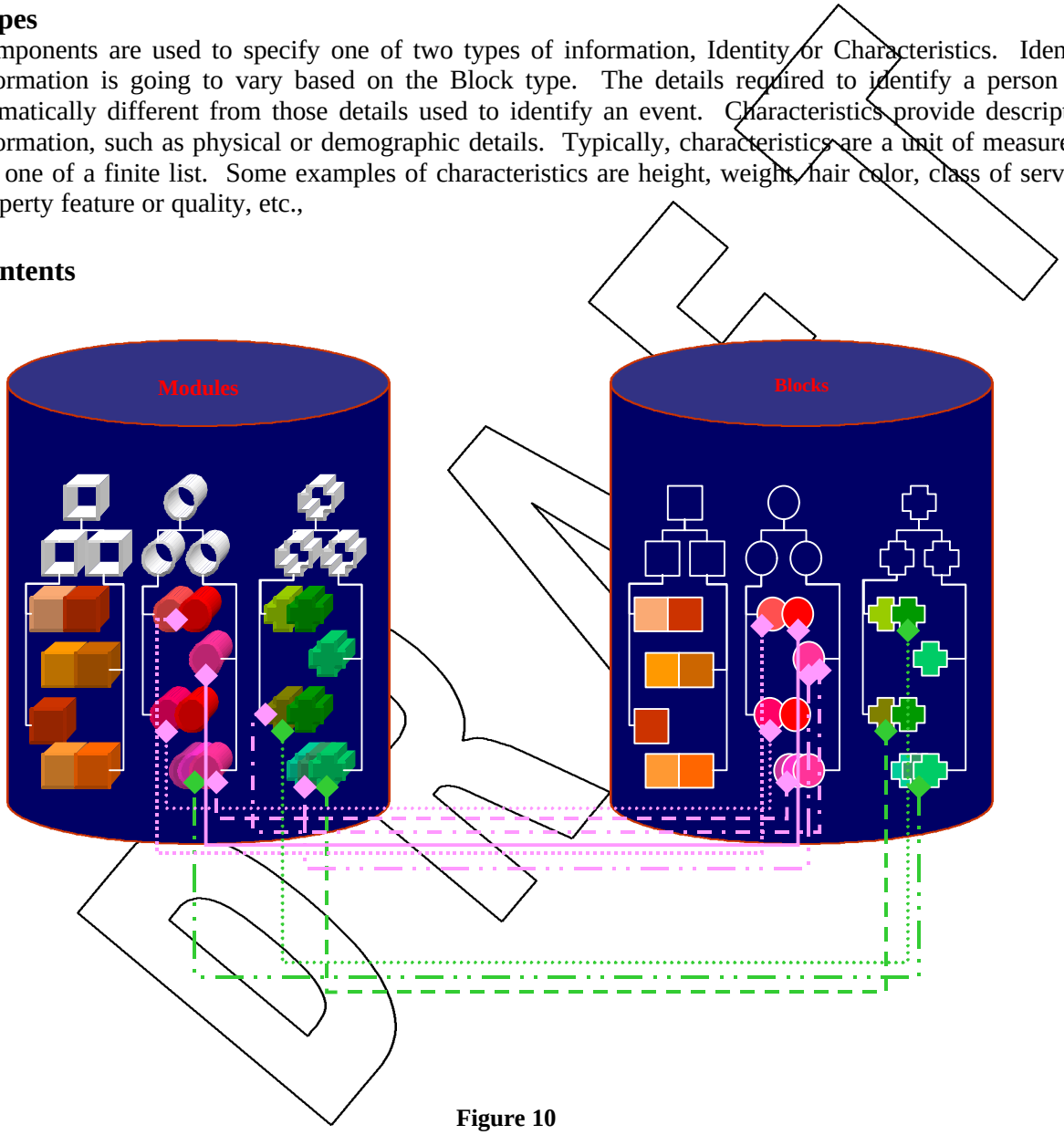


Figure 10

5.0 STRUCTURING

Given the number of industries, organizations and business processes that are involved in making eBusiness standards – there is no shortage to the complexity. In this environment, even making the determination that two things are the same is not as straightforward as it sounds. And when they are not the same, how many ways can they be related? And what conclusions and knowledge can be drawn from structural relationships?

This effort relies heavily on a strong semantic foundation for all decisions. Integral with the strong semantic foundation is the need for quantifiable indicators for making decisions, including the ability to quantify precisely the ways in which two things might be considered related and at what point they are to be considered the same.

From these tests, rules can be formulated to reinforce these conclusions.

5.1 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 effect 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:

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.

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. For example, a case requiring a single number, Part Number, to communicate the desired item, is not equal to another case that requires the Part Number, Catalog Number, and Page Number to convey the desired item. The use of Part Number in the two cases provides different levels of qualification to the recipient. It cannot be true that Part Number = Part Number + Catalog Number + Page Number. These are two different levels of specificity of information, even though their users are accustomed to calling them both Part Number.

FUNCTION: Purpose or how used. When comparing two information structures, occasionally it will be observed that there is a common purpose – which causes some to expect to treat them as the same. In the Form example above, Part Name and Person Name are compared. They are common in that both are designed to specify the Name; they have a shared purpose. Many examples of this exist in eBusiness; Product being one of the more obvious examples. In this

case, many business documents need to specify a product, and with the multitude of industries involved the details required to specify that product [goods or service] vary dramatically. For example, Part ID/Number is used to specify THE item being referenced. But in some industries other item identification schemes are used, such as catalog number, part name, UPC, etc. These are all used in the same purpose or function.

5.2 Detailed Structure Rules

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

5.2.1 Condition 1:

FORM = YES
FIT = YES
FUNCTION = YES

When all three test 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.

5.2.2 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.

5.2.3 Condition 3:

FORM = YES
FIT = NO
FUNCTION = NO

This case is very common in EDI today and is basically the case in question. 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.

5.2.4 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.

5.2.5 Condition 5:

FORM = YES
FIT = NO
FUNCTION = YES

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

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

GM 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 ... 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.

5.2.6 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.

5.2.7 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.

5.3 Preliminary Block Structures

Applying these rules and the desire to illustrate the concepts presented in Section 4 has lead to an initial set of Block constructs that are at a level where we are accustomed to operating in the EDI world. The usage-independent nature of Blocks makes them inherently cross industry.

Blocks contain two types of information, Identity and Characteristics. Identity information is used to specify the unique, instance identity of the Block. The content is dependent on the type of Block. This will be examined in more detail in a subsequent section. Characteristic information is descriptive information, which is typically in one of two forms, pick-list or value plus unit of measure. Examples include: length, width, height, weight, eye color, temperature, etc.

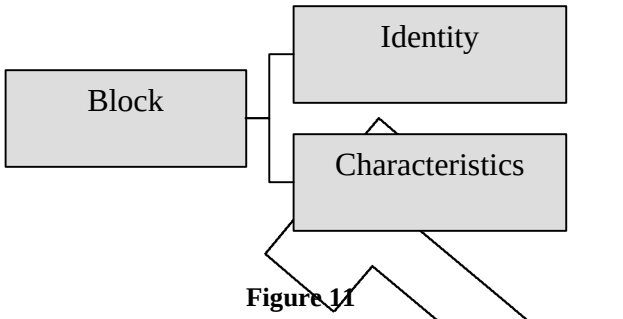


Figure 11

5.4 Party Blocks

The party answers a single *who* question. Parties in a process and message can be individuals or organizations, or combinations of the two. In some cases the parties are also actors. For example, many purchasing applications need only buyer and seller organizations as actors, optionally identifying contact persons. With other processes, the party becomes the subject of the message, e.g. health care, education, or law enforcement. In these latter cases, the data represented in the process and subsequent messages become more detailed. The detail manifests itself in one of two ways, first with characteristic details [height, weight, eye color, etc.] conveyed at the Block level and secondly, details that need to be associated with the party but are not intrinsic to the party. For example, other parties, events, locations, etc. might need to be associated with a base party block in order to construct a complex structure. This is done in an abstract manner with Assemblies and a context specific manner with Modules. The key point is that these complex needs, beyond those of Characteristics, are accomplished with other blocks.

This approach allows Blocks to focus on what is directly attributable to a Block, usage independence.

The fundamental difference between Role Player and Subject parties is that Role Players tend to have Identification information, but not to have Characteristics. Therefore, any Party can be a Role Player.

Based on the set of structure rules detailed in Section 5.2, the top level breakdown for Party is depicted in Figure 12. Differences in the Identity and Characteristics ranges specifically prescribe this breakdown. Identity information for an Organization includes a Name and an Organization ID number. However, there is a fundamental difference between Corporations/ Businesses and Regulatory Organizations, leading to a further breakdown subordinate to that of Organization. The Name probably doesn't vary, but the organization might have a number of ID numbers depending on context. However, they are all ID numbers that are suitable and appropriate for the identification of an organization.

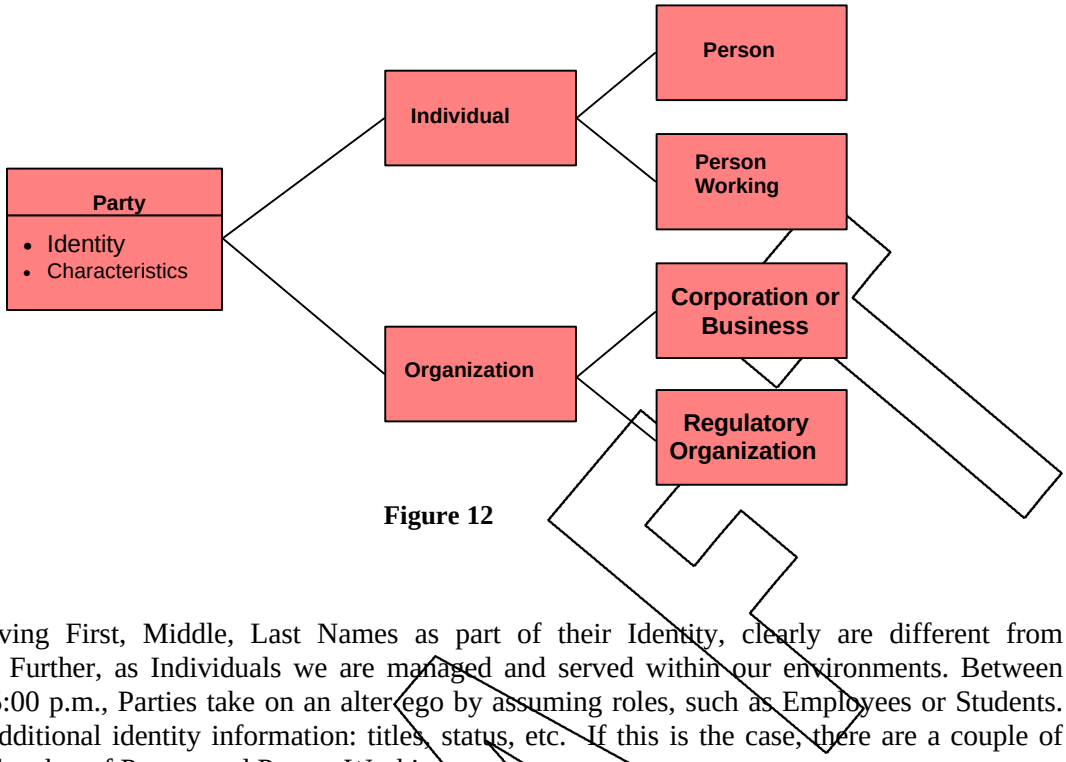


Figure 12

Individuals, having First, Middle, Last Names as part of their Identity, clearly are different from Organizations. Further, as Individuals we are managed and served within our environments. Between 9:00 a.m. and 5:00 p.m., Parties take on an alter ego by assuming roles, such as Employees or Students. This calls for additional identity information: titles, status, etc. If this is the case, there are a couple of individual Blocks, that of Person and Person Working.

5.5 Resource

Economic resources answer the *what* questions in a business document. As Figure 13 shows, resources break down into products and financial instruments. Products are the goods and services of value

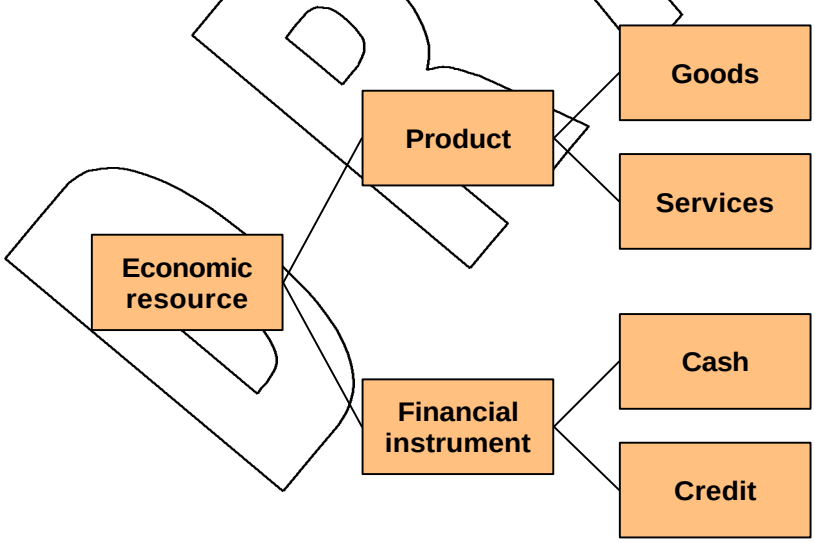


Figure 13

generated by companies for their customers, while the financial instruments – various forms of cash or credit – are the means by which the customers pay for those goods and services.

5.6 Events

Events answer the *when* question in business documents and are easily spotted with the telltale date and time details. As shown in Figure 14, preliminary thoughts are that there are two primary types of Events, basic events and experiences. Basic events include the Event Identity [which of course includes a Date/Time]. Basic Event examples include Birth, Incorporation, Shipment, events which are immutable – they happened. Experiences cover the type of specialized Event which are mutable and tend to have durations (certificates, level of attainment, status), and time periods such as in licensing. Further definition is still needed for capturing histories, such as audit trails or shipping/receiving histories.

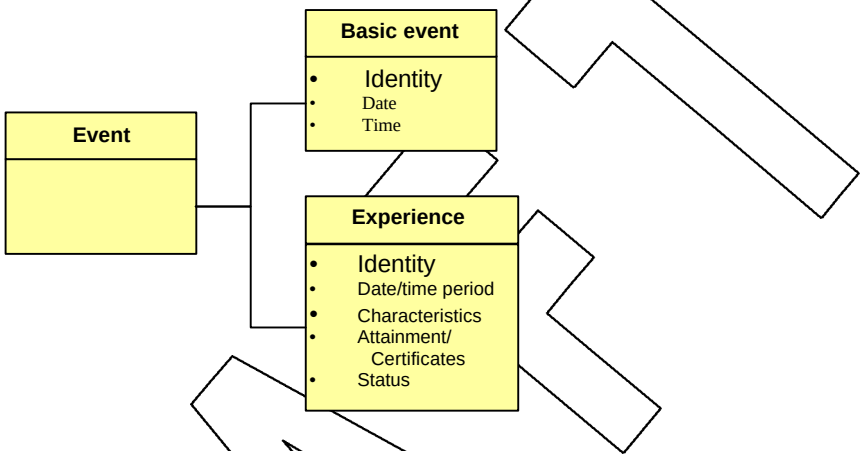


Figure 14

5.7 Location

The location represents the answer to the *where* question in a business process (Figure 15). Locations are either physical or electronic, but each provides as part of its identity a precise and unique address. Physical locations can be represented either in geography by latitude and longitude readings, or by postal and delivery addresses. Electronic addresses in order to be unique often need to follow standard schemes, such as Uniform Resource Indicators or ITU international telephone number conventions.

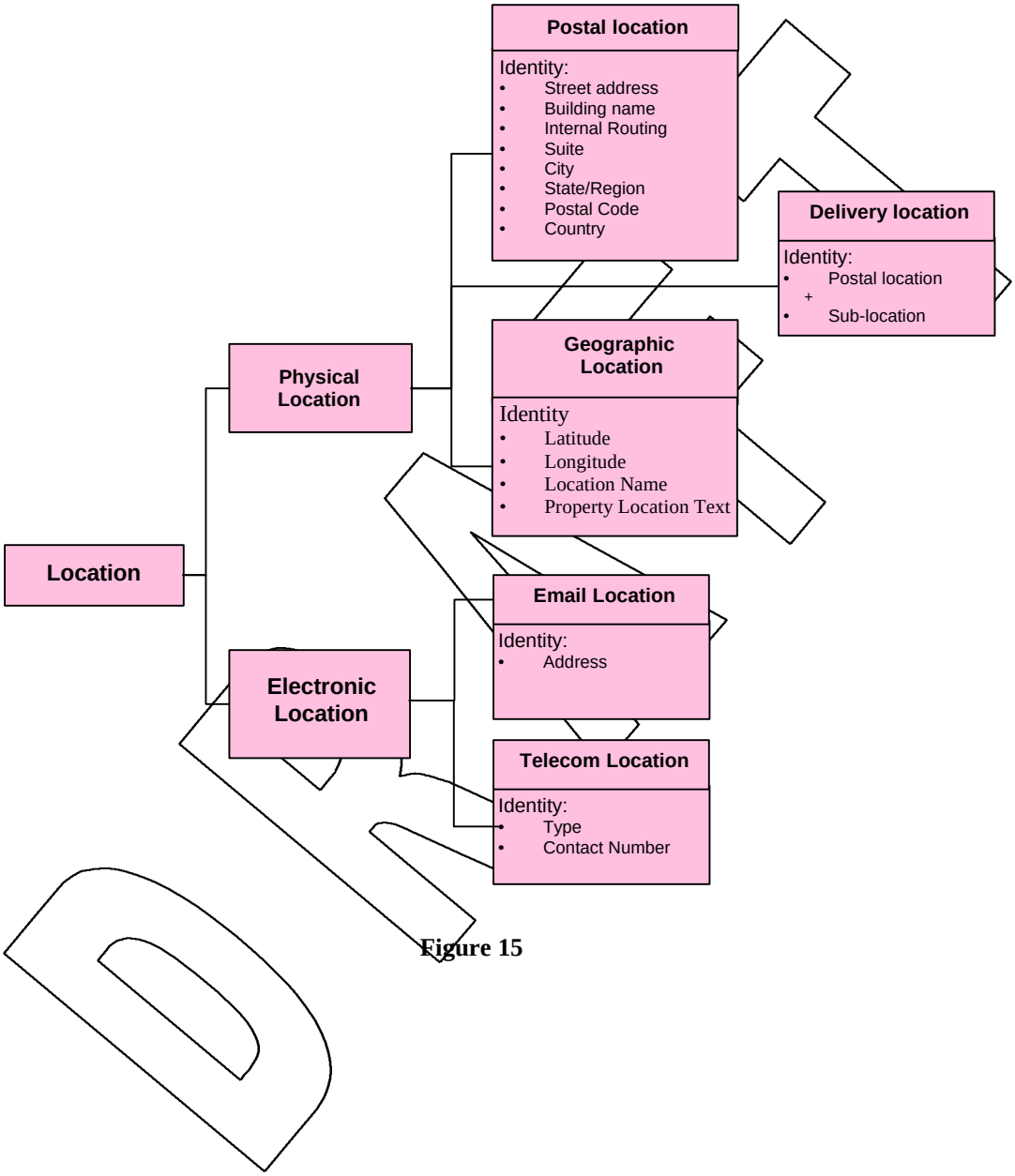


Figure 15

6.0 OTHER DESIGN ISSUES

6.1 What Constitutes a "Bullet" Document?

Concept Defined
A document sent from one person or organization to one or more persons or organizations containing a single instance of a primary subject including supporting details or data.

Example

Discussion

A "single primary subject" does not imply there can only be one line item in a single business document.

The current X12 TS 837 for health care claims allows information for more than one patient to be submitted in a single transmission or a single instance of a transaction set. Applying our definition of a document would require six documents for six patients, a document for each, that could contain multiple supporting details for that patient.

6.2 Default Override

Concept Defined
In order to specifiy a delivery of a line item, you must say what it is and to where it will be delivered. If XML maps those two things at each line item, it is simply syntax conversion. Using "default" requires (1) sorting capability, and (2) knowledge of doing comparisions to determine if the detail matches the default. If XML were to require that advanced processing capability and knowledge, simple off-the-shelf tools will not handle it, resulting in a situation that precludes bringing on board the SMEs.

Depending on your specific concerns this might be thought of as "duplication of data problem", "default and overriding data", or perhaps "table 1 & table 2 semantics".

Example
X12 Practice:
1) The X12 "Semantics in EDI" paper states the premise like this (paraphrasing here in a semi-code-like fashion)
1-A) If some data XXX appears in both table-1 and table-2, The data XXX is considered default values for all iterations table-2.

1-B) if the data YYY appears in an iteration of table-2, The data YYY overrides the earlier data XXX.

2) Many X12 docs have multiple sections that carry "structurally-like" but "semantically-different" data. What makes this worse is that in situations where the two "hunks" of data might be the same, a need (perceived or otherwise) for data compression leads to gyrations in either the message construction (read: weird loop or HL) or usage (read: gotta explain it in the IG). All so you don't have to send/specify the values twice.

Discussion

- Problems:

This is understood well by the X12 community, perhaps too well. This "fact" of the overriding defaults is not consistently pointed out in our semantic notes for particular Transaction Sets, and thus often must appear in implementation guides. And if the references to it are not called out well, a recipient can misinterpret the intent of the sender.

A related issue revolves around duplication of structure (and data values) in messages (and their instances). We have discussed this as a "multiple" roles issue. For instance, in a health care claim there is always a subscriber and a patient. Groups of segments are provided in the 837 for both, and the HIPAA guides (and other IG's) describe what values to send when both are the same person.

Straw Man Proposal:
Suppose we introduce specific "semantic attributes" to positively indicate in the instance data stream the situations/conditions described above. Not wanting to color things too much, I'll give the attributes silly names. I think good final names for these attributes are terrifically important.

My thinking is that these attributes might only appear on what we have been calling "modules"/"blocks", I fear using them in a finer-grain manner may introduce as many problems as we solve.

Examples:

1) A typical "table1 is default"-"table-2 overrides" example

1-A) To indicate that something is a "default" we have an attribute for modules as in:

```
<ShipTo  gork="default">  --ship data-- </ShipTo>
```

1-B) Later in a "table 2 iteration" (not limited to this, but to keep discussion simple) we have additional overriding shipping info:

```
<LineItem>
     <ShipTo  gork="override">  --ship data-- </ShipTo>
    --line item data--
</LineItem>
```

2) A simple "Same As" or "Also Is" example

2-A) A module of "subscriber" info stating it is also the "patient"

```
<Subscriber  woof="Patient">  --party info-- </Subscriber>
```

-or-

2-B) A module of "subscriber" followed by a module of "patient"

```
<Subscriber>  --party info-- </Subscriber>
<Patient  woof="subscriber">  --info?--  </Patient>
```

3) Complex or "deep hierarchy" document twist on things. We might need an attribute to make sure we link the right "pairs" default/override or same-as/also-is modules. I propose here a "serialization" mechanism, knowing full well this kind of think might make some folks freak.

3-A) Variation of 1-A/1-B

```
<ShipTo  gork="default"  blat="001">  --ship data-- </ShipTo>
...
<LineItem>
     <ShipTo  gork="override" blat="001">  --ship data-- </ShipTo>
    --line item data--
</LineItem>
```

3-B Variation of 2-B

<Subscriber blat="001"> --party info-- </Subscriber>
<Patient woof="subscriber" blat="001"> --info?-- </Patient>

Conclusions:
By Expressly stating the individual semantics being expressed in the instance document, we are able to avoid "implicit" relationships that now appear irregularly in semantic notes and IG's.
I also think use of attributes here is perfect, as we are conveying "semantic relationships" in a way that is outside of the "data content". I am unsure this morning if the two concepts (Default/Override in example 1 and SameAs/AlsoIs in example 2) should use the same attribute ("gork" & "woof" in the examples) in practice.

- DEFAULTs

OK, this was a little difficult to read, so let me start with stating what I understand. You propose that we use attributes to explicitly specify that something is a "default". And, that this occurs at the 'block' level, only.
You are proposing that we specify attributes, at the block level, which explicitly declare its contents to include "defaults" [or do you mean then entire block is a default]. Does each piece of info, which is a default, have a default attribute designation?
How will this effect mandatory versus optional? Semantically speaking, the information is mandatory. But, if a default block is used, then the info in the subsequent blocks is optional. So, the default block must be mandatory! Is this what you had in mind?
I think this default concept works.
On the other one, the sameAs/alsoIs, I'm not sure about this one. This one is more difficult because of the mandatory/optional stuff. If the Subscriber = Patient, then you shouldn't have to supply some patient details, but otherwise these are mandatory. How do we handle this? Could we have mutually exclusive sections, one for when the subscriber is the patient, and one for when the subscriber is not?
This is a pervasive problem that can have far reaching consequences. It is important that we give some serious thought to how to go about solving this.

Decision

Rules

6.3 Two Roles for Same Instance Information: Explicit vs Referential Content

Concept Defined

Many business documents have data structures that repeat. Sometimes the identical data structures can contain identical content as well. Examples include *ship to/bill to*, *subscriber/patient*, *manufacturer/vendor*, etc. For these cases it's quite reasonable to consider whether specifying a way to eliminate repeating data (*referential content, implied content, or inferred content*) is better than just repeating the data (*explicit content*) where applicable. The following example illustrates an instance of this situation.

Example

Explicit Content

```
<HealthCareClaim>
  <Subscriber>
    <IdentificationCode>1</IdentificationCode>
    <Name>Santa Clause</Name>
    <Address>North Pole</Address>
    <WorkPhone>555-555-9627</WorkPhone>
  </Subscriber>
  <Patient>
    <IdentificationCode>1</IdentificationCode>
    <Name>Santa Clause</Name>
    <Address>North Pole</Address>
    <WorkPhone>555-555-9627</WorkPhone>
    <EmergencyContact>Mrs. Clause</EmergencyContact>
    <EmergencyPhone>555-555-9628</EmergencyPhone>
  </Patient>
  <ReasonForVisit>Chimney Smoke Inhalation</ReasonForVisit>
  <Total>73.48</Total>
</HealthCareClaim>
```

Referenced Content

```
<HealthCareClaim>
  <Subscriber>
    <IdentificationCode>1</IdentificationCode>
    <Name>Santa Clause</Name>
    <Address>North Pole</Address>
    <WorkPhone>555-555-9627</WorkPhone>
  </Subscriber>
  <Patient>
    <PatientSameAsSubscriber>true</PatientSameAsSubscriber>
    <EmergencyContact>Mrs. Clause</EmergencyContact>
    <EmergencyPhone>555-555-9628</EmergencyPhone>
  </Patient>
  <ReasonForVisit>Chimney Smoke Inhalation</ReasonForVisit>
  <Total>73.48</Total>
</HealthCareClaim>
```

Discussion

Arguments for the Referential Approach

1. Smaller XML instance documents
    a. Requires less bandwidth
    b. Requires less storage space
2. Consistent with a referential approach to data structures that some developers are comfortable with.

Arguments for the Explicit Approach

1. Easier to express as an XML schema design rule.
2. Easier to to apply as an XML schema design rule. Schema standard working groups will set standards faster and be more confident in their decisions.
3. The data structure requirements of a business document can be expressed exclusively in the associated XML schema. Additional documentation is required for the referential approach.
4. Instance documents are clearer (arguably).
5. Easier for companies to implement.

   a. Slightly lower learning curve.
   b. Lower development, integration, and testing costs.
6. Lower costs to bring new trading partners on-line.

Notes

1. Some have suggested that their on-line purchase experiences validate the referential approach. Many B2C e-commerce sites (like Amazon.com) require bill-to and ship-to information. These sites often require that the user enter bill-to information and allow the user to simply click on a "Same as Bill-To" check box rather than enter duplicate information in the Bill-To fields (if applicable). This case really doesn't apply to the rule under discussion since the driving factor for the user interface design (web page) is user convenience which does not necessarily suggest a corresponding data structure on the web server.
2. If one takes the referential approach, would the reference be *required* if the data matches? In terms of our example, if the subscriber data and reference data match, *must* the patient be referenced? Is it acceptable for the patient data to be explicity expressed (i.e. duplicated)?
3. Are there cases where one would need to *know* that the patient *is* the subscriber?
4. This rule is related to but independent of the use of identification codes. For example, the XML schema may require subscriber and patient identification codes and not require any of the demographic information. Considerations about this type of data structure are not affected by the rule under discussion.
5. People's time is money.
6. Delayed ROI is money.

Decision

Rules

7.0 METADATA AND ORGANIZATION

7.1 Storage of Templates

In general, the storage of Templates must accommodate a few types of classifications and a logical structure ranging from general to specific which differentiates amongst like documents. For example, consider the following discussion of procurement/invoicing.

> Consider the event based procurement scenario. Principally, there are two possible events that trigger the invoice, Shipment or Order Complete.
>
> In the Shipment triggered invoice, there is a common ship method [carrier and details], and common point of shipment origin [tax authority], therefore part of the document level context information should include the Carrier and Tax details.
>
> Order triggered invoice covers potentially multiple shipments, multiple ship from/ship to destinations, etc. Therefore, tax and carrier/delivery information must be determined at the line item level. These differences are reflected with the use of different Templates.
>
> Further, for data compression reasons, historically EDI message design would want a third invoice style, which is really a combination of the two previous invoices. In this scenario, "default" information would be specified in the header, and then the same details could appear at each line item. When the details appear at the line item level, for that line item the default information is overridden by the line item level detail. This style requires significant processing knowledge, based on contents, and a latter section will comment on continued support.

This example details the requirement for three different Templates, all Invoices, all used in Event based Procurement, and all with subtle differences. Clearly, there are a number of classifications that users might be interested in using on this set of documents: procurement process general, event based procurement, finance related, etc. All of these classifications are flat; you are a member or you are not a member.

In addition to these flat classifications is the need to logically organize the various like Templates, such as the family of Invoices, in accordance with their similarities and differences. This requirement is to facilitate reuse and to assist implementers in making proper Template selection.

7.2 Storage of Modules

Modules, like Templates, have two fundamental storage/retrieval requirements. Basic classification is analogous to set theory; you are a member or you are not a member. Storage for this is flat.

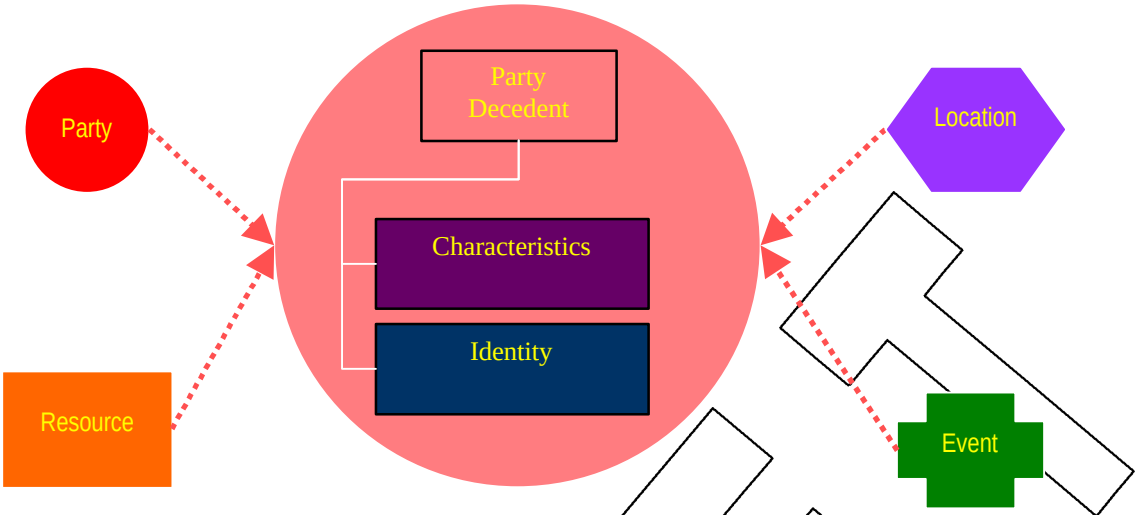


Figure 15

In addition to this flat classification is the need to logically organize the various like Modules, such that the family of like Modules are organized in accordance with their similarities and differences. This requirement is to facilitate reuse and to assist implementers in making proper Template selection.

8.0 XML SYNTAX DESIGN

8.1 General

8.1.1 Scope and purpose

This section addresses XML syntax design issues that are common to both the design of XML messages (instance documents) and schemas describing those messages.

8.1.2 Rules

8.1.3 General Rules

8.1.4 Versioning

8.1.5 Internationalization Features

8.1.6 Industry Domain Context

8.1.7 Software processing considerations

8.2 Messages

8.2.1 Scope of purpose

This section addresses XML syntax design issues relevant to the design of XML messages (instance documents).

8.2.2 Naming Conventions

8.2.3 Standardization Conventions

8.2.4 Instance Document Conventions

8.2.5 Absence of Data and Related Considerations

Absence of data - If an element or attribute does not occur in an instance document, no semantics shall be interpreted from it, i.e. no default values shall be assumed. Nothing can be inferred other than that the creator of the document did not include the element or attribute in the document.

Spaces - Spaces sent as values for string type elements or attributes shall be interpreted as spaces. Leading and trailing spaces should be removed, but are assumed to be significant if they appear.
NOTE: We have yet to achieve consensus on this recommendation. We may revise it pending discussion of the schema whiteSpace facet (preserve, replace, or collapse).

Zeroes - A zero appearing in a numeric type element or attribute shall be interpreted as a zero value.

Nullability - In certain cases, it may be desirable to convey that an element has no value (a null value) rather than indicating that it has a value of spaces or that it is not present in a document. In these cases, the originator of the instance document should convey explicitly that an element is null using the null type (e.g. xsi:null="true"), rather than using zero, spaces, or an empty element.
NOTE: We have yet to achieve consensus on this recommendation.

8.2.6 Comments

8.2.7 Elements vs. Attributes

Description: Often it is possible to model a data item as a child element or an attribute.

Benefits of Using Elements

- They are more extensible because attributes can later be added to them without affecting a processing application.
- They can contain other elements. For example, if you want to express a textual description using XHTML tags, this is not possible if description is an attribute.
- They can be repeated. An element may only appear once now, but later you may wish to extend it to appear multiple times.
- You have more control over the rules of their appearance. For example, you can say that a product can either have a number or a productCode child. This is not possible for attributes.
- They can be used in substitution groups (if substitution groups are used).
- They can use type substitution to substitute derived types in the instance (if type substitution is used).
- Their order is significant, while the order of attributes is not. Obviously, this is only an advantage if you care about the order.
- When the values are lengthy, elements tend to be more readable than attributes.

Disadvantages of Using Elements

- Elements require start and end tags, so are therefore more verbose.
- No type checking is possible when using elements with DTDs (if DTDs are supported).
- Default values are not possible with DTDs (if DTDs are supported).

Benefits of Using Attributes

- They are less verbose.
- If you plan to validate using DTDs as well as schemas, you can perform some minimal type checking on attribute values. For example, color can be constrained to a certain set of values. Element character data content cannot be validated using DTDs.
- Attributes can be added to the instance by specifying default values. Elements cannot (they must appear to receive a default value)

Disadvantages of Using Attributes

- Attributes may not be extended by adding children, whereas a complex element may be extended by adding additional child elements or attributes.
- If attributes are to be used in addition to elements for conveying business data, rules are required for specifying when a specific data item shall be an element or an attribute.

Recommendation: Use elements for data that will be produced or consumed by a business application, and attributes for metadata.

NOTE: There is at least one significant, continuing dissenting opinion to this consensus. An alternate position paper is being prepared by Bob Miller.

8.2.8 Namespaces

XML schemas allow for instance documents that have zero, one or many namespaces. The namespace of an instance document is specified as a "target namespace" of the schema document.

Benefits of Using No Namespace

- It is simpler: there are fewer design decisions to be made, and instance documents are more readable.
- DTDs do not mix well with namespaces, so if DTD validation is planned, use of namespaces will complicate it.
- Allows for use of "chameleon" design. In other words, when a schema that has no targetNamespace is included in another schema, the components within the included schema taken on the same namespace as the including schema - therefore, they are "chameleons".

Disadvantages of Using No Namespace

- Most XML processors cache schema components for validation by namespaces. If no namespace is used, there will be no caching. Processing is therefore much less efficient without namespaces.
- Most current XML schema designers are using namespaces, so not using them will go against convention.
- More work is required to avoid result name collision, i.e. if there is an element in the included schema that has the same name as an element in the including schema, an error will result.

Benefits of Using One Namespace

- The vocabulary of an instance document is immediately recognizable.
- One namespace declaration does not significantly complicate an instance document.

Disadvantages Using One Namespace

- The size of a single namespace for the whole of X12/XML may be rather large, even when a particular instance document uses a limited number of components from the namespace. Processing efficiency is reduced if a single, large namespace is used.

Benefits of Using Multiple Namespaces

- Namespaces can be used to categorize components.
- Helps to avoid name collision.
- It is easy to distinguish "core components" from extensions.

Disadvantages of Using Multiple Namespaces

- Multiple namespaces lead to a more complex design.

Recommendation
Use a tiered, hierarchical approach to namespaces. One core namespace shall include components to all functional X12 subcommittees. Each functional subcommittee (or other logical grouping) shall have a unique namespace that imports the common namespace. All instance document schemas related to the subcommittee (or other logical grouping) shall use that subcommittee namespace.

8.2.9 Communication Intregrity - envelope, security, Header information -- To Be Completed

8.2.10 Processing Instructions

Description: Processing instructions can be used to pass information to the processing application.
Benefits:
Risks: Processing instructions usually contain information that should normally be included in the document as XML.
Recommendation: Do not use processing instructions in either the schema document or the instance.

8.3 Schema

The purpose of an XML schema is to define the allowable content and structure of an XML instance document. Based on the message design philosophy from Section 4.0 of this document, a notional X12/XML message or instance document was created and is located in Annex B of this document. An accompanying XML schema for the notional X12/XML message needs to define the allowable content and structure in terms of templates, slots, modules, assemblies, blocks, and components as defined in Section 4.0. Annex C contains a notional X12/XML schema for the notional X12/XML message located in Annex B.

8.3.1 Scope and purpose

This section addresses XML syntax design issues that are relevant to the design of schemas describing XML messages (instance documents, or business documents).

8.3.2 Schema Considerations for Namespaces, Nullability and Related Issues

String type - An empty string type element or attribute satisfies mandatory constraints in XML schema (elements with minOccurs of 1 or mandatory attributes). Therefore, elements or attributes with a type of string that are defined as mandatory shall be defined with a minimum length requirement of 1. To be considered: Require a pattern of at least one non-space character for such required elements or attributes. To satisfy the requirement for a string element or attribute, XML schema considers any Unicode character to be valid. One space in a string element or attribute is considered valid.

Nullability - An element shall not be marked as nullable if it is mandatory, i.e. minOccurs is one. Conversely any element defined with minOccurs of zero shall be nullable.

NOTE: We have yet to achieve consensus on this recommendation. We may revise it pending discussion of the schema whiteSpace facet (preserve, replace, or collapse).

8.3.3 Content Models

- **Use of Mixed Content**
  Description: Elements with mixed content are allowed to have both child elements and textual content.
  Benefits: Mixed content is useful for textual descriptions, which may or may not contain markup to indicate emphasis, formatting, etc.
  Risks: The textual content of mixed elements cannot be validated or constrained to any particular data type.
  Recommendation: Do not allow mixed types since they are inappropriate for usage in documents designed solely for data exchange.

- **Wildcards**
  Description: XML Schema allows wildcards to be specified in content models (using <any>) and attribute declarations (using <anyAttribute>).
  Benefits: Wildcards allow a content model (or attribute list) to be highly flexible, making them more extensible.
  Risks: Wildcards can sometimes allow invalid data (e.g. a product with two sizes when only one is allowed), so they should generally be used only for elements in other namespaces.
  Recommendation: Disallow use of wildcards.

- **Abstract Types**
  Description: Abstract types allow use of complex types in such a way that a single element name can be used to represent various types in an XML document instance. Abstract types are complex types that act as “templates” that cannot be directly used in an XML document instance. In order to use an abstract type, a derived type must be used to represent the abstract type in an XML document instance.

  For example, consider the following abstract type:

```
<xsd:complexType name=”AddressType” abstract=”true”>
    <xsd:sequence>
        <xsd:element name="StreetName1" type="xsd:string" />
        <xsd:element name="StreetName2" type="xsd:string"  minOccurs=”0” />
        <xsd:element name="City" type="xsd:string" />
    </xsd:sequence>
</xsd:complexType>
```

  For example purposes, the above complex type contains all of the basic minimal information that is contained in an address. Additional information will be necessary beyond this information, exactly what information is needed depends on the country that the address represents. For instance, in a United States address, the additional information would be State and Zip Code. However, in a Canadian address, the additional information would be Province and Postal Code. Each of these would constitute a new complex type that is derived from the *Address* complex type shown above. A declaration for a United States address may look as follows:

```
<xsd:complexType name=”UnitedStatesAddressType”>
    <xsd:complexContent>
        <xsd:extension base="AddressType">
            <xsd:sequence>
                <xsd:element name="State" type="StateCodeType" />
                <xsd:element name="ZipCode" type="ZipCodeType" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

NOTE: The declarations of types *StateCodeType* and *ZipCodeType* are not shown in this example.

In the above example, the *AddressType* complex type has been extended with the two additional elements that comprise a United States address. Along similar lines, a declaration for a Canadian address may look as follows:

```
<xsd:complexType name=”CanadaAddressType”>
    <xsd:complexContent>
        <xsd:extension base="AddressType">
            <xsd:sequence>
                <xsd:element name="Province" type="ProvinceCodeType" />
                <xsd:element name="PostalCode" type="PostalCodeType" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

If a complex type is declared to be abstract, an element can be declared to be of that complex type; however, in an XML instance document that declaration is overridden. Therefore, we can define an element type be of type *AddressType,* as shown below:

```
<xsd:element name="EmployeeInformation" maxOccurs=”unbounded”>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="EmployeeName" type="xsd:string" />
            <xsd:element name="EmployeeAddress" type="AddressType" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

However, in an XML document instance the *EmployeeAddress* element is not actually of type *AddressType;* rather, it is either of type *UnitedStatesAddressType* or *CanadaAddressType*. Therefore, the abstract type acts as a sort of “template” upon which other types are based.

The following is an example of the above content model in an XML instance document - note the use of the *xsi:type* attribute:

```
<EmployeeInformation>
    <EmployeeName>John Smith</EmployeeName>
    <EmployeeAddress  xsi:type="UnitedStatesAddressType">
```

```
        <StreetName1>2 Main St.</StreetName1>
        <StreetName2>Apt. 12</StreetName2>
        <City>Anytown</City>
        <State>MS</State>
        <ZipCode>55145</ZipCode>
    </EmployeeAddress>
</EmployeeInformation>
<EmployeeInformation>
    <EmployeeName>Mary Francis</EmployeeName>
    <EmployeeAddress  xsi:type="CanadaAddressType">
        <StreetName1>10 White Way</StreetName1>
        <City>Thunder Bay</City>
        <Province>Ontario</Province >
        <PostalCode>M1A 3X9</PostalCode>
    </EmployeeAddress>
</EmployeeInformation>
```

In the first *EmployeeInformation* content model above, the employee is a United States employee. Therefore, the *xsi:type* value indicates that the address is of type *UnitedStatesAddressType*. In the second *EmployeeInformation* content model above, the employee is a Canada employee. Therefore, the *xsi:type* value indicates that the address is of type *CanadaAddressType*. Yet, the same element named *EmployeeAddress* was used in both cases, i.e. we did not have to define an element named *UnitedStatesEmployeeAddress* and one named *CanadaEmployeeAddress*.

Benefits: Extensibility - other schemas can use the abstract type as the basis for derived types. Consider the following schema excerpt that assumes that the AddressType declaration is in a schema file called Employees.xsd:

```
<xsd:include schemaLocation="Employees.xsd">
<xsd:complexType name="UKAddressType">
    <xsd:complexContent>
        <xsd:extension base="AddressType">
            <xsd:sequence>
                <xsd:element name="PostCode"
                             type="PostCodeType" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

The following is an example of the above content model in an XML instance document:

```
<EmployeeInformation>
    <EmployeeName>Mary Poppins</EmployeeName>
    <EmployeeAddress  xsi:type="UKAddressType">
        <StreetName1>Arden House</StreetName1>
        <StreetName2>1102 Warwick Road</StreetName2>
        <City>Birmingham</City>
        <PostCode>B27 6BH</PostCode>
    </EmployeeAddress>
</EmployeeInformation>
```

As shown above, the base complex type AddressType has been extended in a schema other than the one in which it is declared, to create yet another address type.

- *Using a base and derived types allows certain updates to take place in only one location. For instance, if in the previous examples it were determined that all City elements should be a maximum of 30 characters, that update would need to be made only in the AddressType definition, and would then propagate to all complex types that were derived from AddressType.*
- *This technique allows a single element name to be associated with more than one type in an XML instance document. This allows derived types to be added to a schema as needed, while element names remain the same. Therefore, in the above example, a processing application can simply recognize an element named "EmployeeAddress", rather than elements named "UnitedStatesEmployeeAddress" or "CanadaEmployeeAddress". If more address types were added in the future, it would require less update to a processing application because it would need to be updated only to recognize a new type, not a new element and a new type.*
- *This technique allows specification of the minimum amount of information required for a content model. For instance, the above example ensured that, at a minimum, the following elements would be included in any address:*
    - *StreetName1*
    - *StreetName2*
    - *City*

Risks: It is possible that a processing application *(such as a data translation product)* may not be able to easily handle this technique. That is, a processing application may be need to be configured to recognize an element named *EmployeeAddress* as always having a single, static type *(such as UnitedStatesAddressType)* rather than a type that can vary depending on the XML document instance. A processing application may not be able to be configured to recognize and refer to the *xsi:type* value for the type of the element that it is currently processing.

Recommendation: Abstract types MUST NOT be used because they contribute to a degree of uncertainty about what an XML document instance will look like, i.e. they contribute to randomness.

- **Use of Groups**

Description: XML Schema allows fragments of content models to be named and referenced from multiple complex types. It is also possible to create attribute groups that can be reused in multiple complex types.
Benefits: Use of groups promotes reuse.
Risks: Occasionally, too much reuse can complicate maintenance.
Recommendation: Use both named model groups and attribute groups for anything that is likely to be reused in disparate data types. For similar data types, type derivation may be a better way to use content model fragments and attributes.

- **Substitution Groups**
  Description: XML Schema allows for elements to substitute for other elements by defining substitution groups. An element can be declared to be a substitute for another element, the "head" element, allowing the new element to appear anywhere the head element may appear.
  Benefits
  - Substitution groups result in flexible, extensible types.
  - They can simplify content models, by specifying only the "head" element in the content model and using substitution to allow all the possibilities.

  Risks: Excessive flexibility. Another schema author can significantly alter a type by declaring substitution elements.
  Recommendation: Avoid substitution groups.

- **Group Redefinition**
  Description: XML Schema allows a schema author to redefine the types or groups of another schema document.
  Benefits: Redefinition is useful for making small changes to an existing schema document.
  Risks
  - Because the redefined components replace the original components, they can have adverse effects on other components defined in the original schema document.
  - Redefinition is underspecified in the XML Schema recommendation, and it is likely that different processors treat redefinitions slightly differently.

  Recommendation: Do not use redefinition.

8.3.4 Types

- **Anonymous vs. Named Types**
  Description: XML Schema allows for types (simple and complex) to be named (and defined globally) or anonymous (and defined locally).
  Benefits of Named Types
  - Named types may be defined once and used many times. This encourages reuse and consistency, simplifies maintenance, and reduces the size of schemas.
  - Named types can also make the schema document more readable, when the type definitions are complex.
  - Named types can be redefined and have other types derived from them. This increases their flexibility and extensibility.

  Benefits of Anonymous Types
  - They are slightly less verbose.
  - They can be more readable when they are relatively simple. It is sometimes desirable to have the definition of the type right there with the element or attribute declaration.

  Recommendation: Always use named types.

- **Built-In Simple Types**
  Description: XML Schema has 44 built-in data types, covering numbers, strings, dates and times, XML 1.0 types such as NMTOKENS and ID, boolean, anyURI, and other common types. These types have specific lexical formats, e.g. a date must be CCYY-MM-DD.

Benefits

- Using the built-in types increases interoperability with other XML applications.
- Values of built-in types are automatically validated by the processor, e.g. a date cannot be April 31.

Risks: The built-in types may not have the lexical formats that you have traditionally used.

Recommendation: Use only XML Schema built-in data types. Further, we shall use a subset of the full types, with that subset to be defined.

- **Type Redefinition**

  Description: XML Schema allows a schema author to redefine the types or groups of another schema document.

  Benefits: Redefinition is useful for making small changes to an existing schema document.

  Risks

  - Because the redefined components replace the original components, they can have adverse effects on other components defined in the original schema document.
  - Redefinition is underspecified in the XML Schema recommendation, and it is likely that different processors treat redefinitions slightly differently.

  Recommendation: Do not use redefinition.

- **Type Derivation**

  Description: XML Schema allows a type to be derived from another type (its base type), either by extension or restriction. Extension adds attributes, and adds elements to the end of the content model of the base type. Restriction limits a base type to a more restrictive set of valid values.

  Benefits

  - Restriction allows more refined data types to be created which allows stricter validation in specific cases.
  - Extension allows the base type to be used with additional extensions, which encourages reuse.

  Risks: Derived types can be used for type substitution (see "Type Substitution"). If type substitution is not to be allowed, the base complex type should have the block attribute specified.

  Recommendation: Allow type derivation.

- **Type Substitution**

  Description: Type substitution allows for the use of derived types in an instance document. If an element is declared to be of a base type, the element may appear in the instance having any type that is derived from the base type. To do this, it must use the xsi:type attribute to identify the derived type to which it conforms.

  Benefits: Type substitution allows an element to have one of several types in an instance document. For example, a generic address type can be created, with extensions for specific countries, e.g. UKAddressType, USAddressType, etc. The address element can then appear in the instance using whichever of these types is appropriate.

  Risks

- Can lead to problems in processing by applications when a type specified in an instance document overrides the type specified in a schema.
- If you do not intend to allow flexibility of the type of an element, you should not allow type substitution.

Recommendation: Disallow type substitution.

8.3.5 Local vs. Global Declarations

Description: Elements and attributes can be either declared globally or locally. Globally declared elements and attributes appear at the top level of the schema (with xsd:schema as their parent). Locally declared elements and attributes are declared entirely within a complex type.

Benefits of Global Declarations

- They can be reused in many complex types.
- A globally declared element can be the root element of the instance document for validation purposes (a locally declared element cannot.)
- Global element declarations can participate in substitution groups; local element declarations cannot.

Benefits of Local Declarations

There can be many locally declared elements with the same name but different types and/or different default or fixed values. For example, it is possible to have a "title" element that is a child of "person", which has the valid values "Mr.", "Mrs." and "Ms.". Another element named "title" that is a child of "book" can have free-form text. Because global element declarations are unique by name, there can only be one globally declared element named "title".

Recommendation: Declare elements and attributes locally, except for the root element.

8.3.6 Use of Default/Fixed Values

Description: XML Schema allows fixed or default values to be specified for elements and attributes.

Benefits: Additional information can be added to the instance without requiring the instance author to specify it.

Risks: When a schema is not present, the default or fixed value cannot be filled in.

Recommendation: Disallow use of default and fixed values.

8.3.7 Keys and Uniqueness

Description: Sometimes it is desirable to associate information within an XML document with other information in the document. For instance, consider a case where invoices need to be associated with customers. In a relational database, there would normally be a table that stored customer information, and another table that stored invoices. A relationship between the two tables (i.e. the invoices and the customers to which they are associated) would be represented through the use of keys that are used to “join” the two tables during processing. The keys may represent a customer ID that is unique to each customer, and the key in the invoice table can be referred to as a foreign key that is associated with a primary key in the customer table. This approach is desirable because it avoids duplication of customer information (which is relatively static) for each invoice (whose information in general is relatively dynamic, i.e. multiple invoices can be generated for a given customer in a short period of time). In the examples that

follow, the customer information will be referred to as “static”, while the invoice information will be referred to as “dynamic”.

It is often necessary to represent such relationships in an XML document, and there are several different ways to do this. One way is to mix the dynamic information in with the static information, as follows:

```
<CustomerInvoice>
    <Customer CustomerID=”A157”>
        <CustomerInfo>
          <Name>Bob Smith</Name>
          <Address>2 Main St.</Address>
          <City>Anytown</City>
          <State>MS</State>
          <ZipCode>55145</ZipCode>
              …more details…
        </CustomerInfo>
        <InvoiceInfo>
          <InvoiceNumber>14587</InvoiceNumber>
          <OrderDate>2001-11-01</OrderDate>
          <ShipDate>2001-11-02</ShipDate>
              …more details…
       </InvoiceInfo>
       <InvoiceInfo>
          <InvoiceNumber>15879</InvoiceNumber>
          <OrderDate>2001-12-01</OrderDate>
          <ShipDate>2001-12-02</ShipDate>
              …more details…
       </InvoiceInfo>
   </Customer>
</CustomerInvoice>
```

Although the above approach is possible, it is not advisable for several reasons:

- It is unconventional in that it mixes together information, i.e. customers and invoices are really two separate entities, and it is more convention from a relational standpoint to represent them separately
- It is an inefficient structure for an XML processor: it will cause an XML processor to work much harder then necessary, thereby increasing processing time

Regarding the second point above: consider a situation in which you would like to report on all invoices for a given order date. In order to access a set of invoices for a customer, an XML processor would have to first access the <CustomerInvoice> element, a <Customer> element, then an <InvoiceInfo> element just to arrive at the invoice information. Then, it would have to move back up several levels to be able to access the next <Customer> element, etc. This will cause the XML processor to work much harder then necessary, thereby increasing processing time. Also, what if a customer had no invoices, either for the given order date or not at all? The XML processor would still have to access all the necessary levels, which would be a waste of processing time.

A better approach is to separate the static information from the dynamic information, and associate *(i.e. link)* the dynamic information with its corresponding static information through the use of one of several possible mechanisms. Consider the following new structure:

```
<Customers>
    <Customer CustomerID="A157">
        <Name>Bob Smith</Name>
          <Address>2 Main St.</Address>
          <City>Anytown</City>
          <State>MS</State>
          <ZipCode>55145</ZipCode>
          …more details…
    </Customer>
    …more customers…
</Customers>
<Invoices>
    <Invoice InvoiceID="R245"  InvoiceCustomerID="A157">
        <InvoiceNumber>14587</InvoiceNumber>
          <OrderDate>2001-11-01</OrderDate>
          <ShipDate>2001-11-02</ShipDate>
          …more details…
    </Invoice>
    <Invoice InvoiceID="R459"  InvoiceCustomerID="A157">
          <InvoiceNumber>15879</InvoiceNumber>
          <OrderDate>2001-12-01</OrderDate>
          <ShipDate>2001-12-02</ShipDate>
          …more details…
    </Invoice>
    …more invoices…
</Invoices>
```

With this approach, ID values are used to link invoices with their corresponding customers. Therefore, it is clear that the two invoices shown above are both associated with the customer whose ID is A157. This approach nicely separates the customer and invoice entities, allowing for more efficient processing. For instance, in order to report on all invoices for a given order date with the above approach, an XML processor would simply have to access the <Invoices> element, then each <Invoice> element. If a customer had no invoices at all, or none for the given order date, the XML processor can determine this much more efficiently than in the first approach shown above.

The next issue is: how should the linking be performed? There are several possible techniques:

ID/IDREF technique
KEY/KEYREF technique
XLink/XPointer technique

Each of these techniques is discussed further below. The concept of enforcement of uniqueness among information is then discussed as a separate but related topic.

ID/IDREF

This concept originated with DTD's, and is also used in XML Schema. In this technique, an ID value is used (as shown in the second approach above) by an XML processor to associate information within an XML document. This allows information to be separated within an XML document (as with the customer and invoice information above), yet still be associated during processing. The following is an example of a DTD declaration excerpt for the customer and invoice information in the second approach shown above:

```
<!ELEMENT Customers (Customer)*

<!ELEMENT Customer (Name, Address, City, State, ZipCode)>

<!ELEMENT Name (#PCDATA)>

…declarations for Address, City, State, ZipCode…
<!ATTLIST Customer CustomerID ID #REQUIRED>

<!ELEMENT Invoices (Invoice)*

<!ELEMENT Invoice (InvoiceNumber, OrderDate, ShipDate…)>

<!ELEMENT InvoiceNumber (#PCDATA)>

…declarations for OrderDate, ShipDate…

<!ATTLIST Invoice InvoiceID ID #REQUIRED>

<!ATTLIST Invoice InvoiceCustomerID IDREF #REQUIRED>
```

The above declaration means that an XML processor must validate an IDREF value in an XML document to ensure that there is a corresponding ID value. In other words, the following XML document excerpt must yield an error from an XML processor:

```
<Customers>
    <Customer CustomerID="A157">
        <Name>Bob Smith</Name>
          <Address>2 Main St.</Address>
          <City>Anytown</City>
          <State>MS</State>
          <ZipCode>55145</ZipCode>
…more details…
    </Customer>
</Customers>
<Invoices>
    <Invoice InvoiceID="R245"  InvoiceCustomerID="A157">
        <InvoiceNumber>14587</InvoiceNumber>
          <OrderDate>2001-11-01</OrderDate>
          <ShipDate>2001-11-02</ShipDate>
…more details…
    </Invoice>
    <Invoice InvoiceID="R653"  InvoiceCustomerID="A159">
          <InvoiceNumber>15879</InvoiceNumber>
```

```
        <OrderDate>2001-12-01</OrderDate>
        <ShipDate>2001-12-02</ShipDate>
…more details…
    </Invoice>
</Invoices>
```

In the above example, there is no customer whose ID is A159. Thus, this example would yield an error from an XML processor. It should be noted that with this technique, an XML processor cannot link together associated items. Rather, it can only verify that there is a corresponding ID value in an XML document instance for a given IDREF value. For example, considering the DTD and example shown above: if there were an additional ID attribute declared in the DTD (perhaps completely unrelated to customer information), and that ID attribute coincidentally happened to contain a value of A159, the XML processor would not yield an error because the requirement of a matching ID value is fulfilled. However, a processing application could enforce links between associated IDREF and ID values during its processing of the XML document instance, i.e. the onus would be on the processing application rather than the XML processor.

Benefits of ID/IDREF Technique:

- It allows information in an XML document instance to be linked during processing by a processing application
- It ensures validation of the associations by an XML processor — i.e. that there is a corresponding ID value for an IDREF value — without defining extra processing (i.e. it is "built in" to an XML processor).

Risks:

- It does not allow links between entities in an XML document instance to be recognized by an XML processor
- An ID value must be unique within an XML document. This means that in the above example, there could never be the same ID value for a customer and an invoice. This requirement is not realistic, as the ID values for two different entities may not only be of the same structure but may also have the same values in certain cases.
- An ID value must begin with a letter and cannot contain whitespace or non-alphanumeric characters (except for underscore). This means that the ID values shown above could not be 157 or 159. Also note that fact that each invoice has both an invoice number (which is numeric) and an invoice ID. This requirement is not realistic, as there may be many cases in which ID values begin with numbers, for instance, social security numbers. This would prohibit such values from being used as ID values.

KEY/KEYREF

This concept originated with XML Schema. Unlike the ID/IDREF technique, this technique allows links between entities in an XML document instance to be recognized by an XML processor. It also allows ID values to be repeated within XML documents without yielding an error from an XML processor (as with the uniqueness technique, discussed below). Additionally, it adds the requirement that the element or attribute specified in the field element of a constraint declaration must always appear in an XML document instance.

Using the XML instance document example shown previously, the following declarations stipulate that there must be a corresponding *CustomerID* value for each *InvoiceCustomerID* value:

```
<xs:key name="CustomerKey">
    <xs:selector xpath="Customers/Customer"/>
    <xs:field xpath="@CustomerID"/>
</xs:key>

<xs:keyref name="InvoiceCustomerID" refer="CustomerKey" >
    <xs:selector xpath="Invoices/Invoice"/>
    <xs:field xpath="@InvoiceID"/>
</xs:key>
```

The selector elements in the above declaration specify the range of elements to which the key and key references apply—in this case, all Customer and Invoice elements respectively. The field element specifies the element or attribute within those ranges that must have a unique value—in this case, the CustomerID and InvoiceID attributes. The name and refer attributes in the key reference declaration relate the foreign key (InvoiceCustomerID) to the primary key (CustomerID) through a reference to the CustomerKey key declaration.

It should be noted that the key declaration above could be used without a corresponding keyref declaration to enforce uniqueness among all CustomerID values for the specified range. As noted above, all CustomerID attributes must be present and contain a value. This is not a requirement in the uniqueness technique, which is discussed below.

Benefits of KEY/KEYREF Technique:

- It allows links between entities in an XML document instance to be recognized by a schema processor
- It allows ID values to be repeated within XML documents without yielding an error from a schema processor
- ID values do not have the format constraints that were imposed in the ID/IDREF technique; that is, an ID value may be of any datatype

Risks of KEY/KEYREF Technique:

- Constraint declaration names must be unique within an XML document instance, regardless of namespace - this applies for externally referenced schemas as well. This means that if an externally referenced schema contained a constraint declaration named "CustomerKey", even if the target namespace of the schema were different than target namespace for the Customer element in the XML document instance, an XML processor will yield an error.
- A schema processor may not detect an incorrect XPath expression in either the selector or field element of the constraint declaration. This can cause the constraint to not be enforced, resulting in potential violations of the key constraint.

XLink/XPointer

This technique utilizes two relatively new XML concepts to link entities within XML document instances. It allows links to be specified either within an XML the same document instance as the entities being linked (through use of a

"simple" link or "extended" link), or outside of it in a different XML document instance (through use of an "extended" link). Extended links can be very useful in cases where an XML document instance cannot be updated; they also allow linking information to be centralized in one place if required.

Simple Links

Simple links are unidirectional links, much like the HTML "A" element. Consider the following XML instance document example (same as an earlier example):

```
<Customers>
    <Customer CustomerID="A157">
        <Name>Bob Smith</Name>
          <Address>2 Main St.</Address>
          <City>Anytown</City>
          <State>MS</State>
          <ZipCode>55145</ZipCode>
          …more details…
    </Customer>
</Customers>
<Invoices>
    <Invoice InvoiceID="R245"  InvoiceCustomerID="A157">
        <InvoiceNumber>14587</InvoiceNumber>
          <OrderDate>2001-11-01</OrderDate>
          <ShipDate>2001-11-02</ShipDate>
          …more details…
    </Invoice>
    <Invoice InvoiceID="R653"  InvoiceCustomerID="A157">
          <InvoiceNumber>15879</InvoiceNumber>
          <OrderDate>2001-12-01</OrderDate>
          <ShipDate>2001-12-02</ShipDate>
          …more details…
    </Invoice>
</Invoices>
```

Instead of the above format for Invoices, the links between invoices and customers can be defined in XLink syntax using simple links along with XPointer constructs within each Invoice element as follows:

```
<Invoices>
    <Invoice InvoiceID="R245" >
        <InvoiceNumber>14587</InvoiceNumber>
          <OrderDate>2001-11-01</OrderDate>
          <ShipDate>2001-11-02</ShipDate>
          …more details…
          <InvoiceCustomer xlink:type="simple"  xlink:href="#A157" />
    </Invoice>
    <Invoice InvoiceID="R653">
          <InvoiceNumber>15879</InvoiceNumber>
          <OrderDate>2001-12-01</OrderDate>
          <ShipDate>2001-12-02</ShipDate>
          …more details…
        <InvoiceCustomer xlink:type="simple"  xlink:href="#A157" />
    </Invoice>
</Invoices>
```

The notation for the xlink:href attributes above is called an XPointer Bare Names notation. It simply means that a link is defined between the invoice in which the InvoiceCustomer element appears and the ID in the XML document instance whose value matches the value after the "#" sign in the xlink:href attribute (since nothing precedes the "#" sign, it means the current XML document instance). This means that an XLink-aware processor can associate both invoices above with the element in the XML document instance whose ID attribute value is A157, i.e. the customer whose ID is A157. It should be noted that with this technique, the ID values must be of type ID. This means they have the format constraints that were imposed in the ID/IDREF technique.

Extended Links

Extended links are more flexible than simple links in that they may be multi-directional and may also link external entities. This means that an extended link may exist in an XML document instance that is external to one or more (or all) of the entities it links. For example, the following content model, when added to the XML document instance shown previously, allows all link declarations to be centralized:

```
<xlink:extended role="Link Invoices to Customers"
                title="Links">
    <xlink:locator href="#A157"
                role="Customer"
                label="Customer A157">
    <xlink:locator href="#R245"
                role="Invoice"
                label="Invoice R245">
    <xlink:locator href="#R653"
                role="Invoice"
                label="Invoice R653">
    <xlink:arc  from="Invoice R245"
                to="Customer A157"
                   arcrole="Invoice Belongs To">
    <xlink:arc  from="Invoice R653"
                to="Customer A157"
                   arcrole="Invoice Belongs To">
</xlink:extended>
```

This content model is known as an extended link container. The InvoiceCustomerID attributes would also be removed from the XML instance document, as they would no longer be necessary. The xlink:locator elements in the above example specify the "locations" (in this case elements) that participate in the extended link. There is one locator element for each customer and invoice. The role attribute is mostly for semantic purposes, to describe the function of the location. The xlink:arc elements specify the actual links; that is, they define associations between two locations participating in an extended link. Each xlink:arc element specifies one association. In the example above, this would be an association between an invoice and a customer. It is important to note that the label attribute for each locator element is used as the identifier in the extended link, rather than the href attribute (as with simple links). The reason for this will become clear below.

It is also possible to specify extended links in an XML document instance that is external to one or more (or all) of the entities it links. Suppose the above XML document instance containing customer and invoice information were in a file called Invoices.XML. It would be possible to have the extended link declarations in a separate instance document; this is known as a linkbase. The above extended link container declaration would change in that the href attributes would need to begin with the name of the XML document instance file, followed by the ID value that is already listed. This would look as follows:

```
<xlink:extended role="Link Invoices to Customers"
                title="Links">
    <xlink:locator href="Invoices.xml#A157"
                role="Customer"
                label="Customer A157">
    <xlink:locator href="Invoices.xml#R245"
                role="Invoice"
                label="Invoice R245">
    <xlink:locator href="Invoices.xml#R653"
                role="Invoice"
                label="Invoice R653">
    <xlink:arc   from="Invoice R245"
                to="Customer A157"
                   arcrole="Invoice Belongs To">
    <xlink:arc   from="Invoice R653"
                to="Customer A157"
                   arcrole="Invoice Belongs To">
</xlink:extended>
```

This again uses the XPointer Bare Names notation to reference an ID value contained in the file called Invoices.XML. It should now be clear why the label attribute for each locator element is used as the identifier in the extended link, rather than the href attribute. If the file name for the XML instance document listed above were to change, not only would the href attributes require update, but all xlink:arc elements would as well. With this approach, the xlink:arc elements can remain static as the href attribute values change.

Benefits of XLink/XPointer Technique:

- It allows links between entities in an XML document instance to be recognized by a schema processor (although the schema processor must be XLink- and XPointer-aware)
- The use of XLink constructs allow the links to be given special handling in an XLink-aware processor. For instance, additional XLink constructs may be used to allow links to be highlighted for selection
- Extended links can be specified either in the same XML document instance as the entities that they link or outside of it in a different XML document instance

Risks of XLink/XPointer Technique:

This technique has several disadvantages:

- An ID value must be unique within an XML document. For more information, see ID/IDREF section above.
- An ID value must begin with a letter and cannot contain whitespace or non-alphanumeric characters (except for underscore). For more information, see ID/IDREF section above.
- Since the XLink and XPointer standards are both very new (XLink became a W3C Recommendation in June 2001 and XPointer is currently a Candidate Recommendation), there is currently very little XML processor support for them
- Use of extended links requires a fair amount of additional information to be specified for each entity that is being linked; e.g. xlink:locator elements, role attributes, xlink:arc elements, etc.

Recommendation

NOTE: The following recommendations are strictly preliminary. For more information, see "Special Notes" section below.

The KEY/KEYREF technique SHOULD be used to enforce links between entities in an XML document instance.

The uniqueness technique SHOULD be used to enforce uniqueness when the element or attribute specified in the field element is not mandatory. The KEY technique (without KEYREF) SHOULD be used to enforce uniqueness when the element or attribute specified in the field element is mandatory.

Extreme caution should be applied in each of the above techniques to ensure that the XPath expression that is specified is correct, so that the uniqueness constraint can be properly enforced.

It is also recommended that the following situation never be allowed:

- Uniqueness must be enforced AND
- Links are required AND
- The element or attribute specified in the field element is not mandatory

There is no technique that is available to handle the above situation, because in the KEY/KEYREF technique the element or attribute specified in the field element must appear in the XML instance document. For this reason, it is recommended that in all cases where links are required, the element or attribute specified in the field element be declared as mandatory.

Special attention should also be given to the fact that constraint declaration names must be unique within an XML document instance (see earlier for more information on this).

Once the XPointer and XLink standards become more mature, schemas MAY use the XLink/XPointer technique. However, special attention should be given to the restrictions placed upon ID values that are discussed in the XLink/XPointer section. If the XLink/XPointer technique used, it is recommended that the simple link technique be used first as XML processors may support this technique more readily than the extended link technique.

Special Notes

There have been several other recommendations made regarding Keys and Uniqueness which still must be considered. In summary, these are:

- There may not be a requirement for these techniques, because X12 does not currently have any syntax-level constraints for them. At the present time, most of the functionality for this is handled at the application level rather than the EDI interchange level.
- Establish a "key" datatype whose elements include an attribute that specifies a key name. Using the examples above, this would look as follows for linking invoices to customers:

```
... <Customer>
        <Key Name='Customer'>A157</Key>
... </Customer>
... <Invoice>
        <Key Name='InvoiceNumber'>14587</Key>
        <Key Name='Customer'>A157</Key>
... </Invoice>
```

8.3.8 Annotations and Notations

- **Annotations**

  Description: XML Schema allows schema components to be annotated using the <annotation> element. The annotation element can contain one or more <documentation> or <appinfo> elements that can themselves have any attributes and contain any text or child elements.

  Benefits

  - An annotation adds descriptive information that makes a schema component easier to understand.
  - Structured annotations are machine- as well as human-readable, allowing them to be used by applications or to generate specification guides.

  Risks: Excessively large or repetitive annotations actually decrease the readability of the schema document, and slow down validation.

  Recommendation: Use annotations for all type definitions. Structure the annotations according to a predefined standard [DEFINE STANDARD HERE; for example, a <documentation> can contain one <description>, etc.]. Do not use XML comments in schemas.

- **Notations**
  Description: Notations can be used to specify the type of a file (for example, a graphics image) that is related to an XML document via an external entity.
  Benefits: Notations can be useful for identifying the type of a file.
  Risks
  - There is no standardized way to process notations.
  - Generally, notations are unnecessary because the processing application already understands the type of a related file.

  Recommendation: Do not use notations.

- **Documentation**
  Description: Documentation is an important part of system maintenance, as it allows those that maintain programs in a system to understand how various parts of the programs function. Documentation also allows a record of changes to a program to be kept inside the program itself. Documentation is also an important part of XML Schema.

  In DTDs, comments were marked as follows:

  ```
  <!-- this is a comment  --!>
  ```

  While this is sufficient to allow the reader of a DTD to gain an understanding of its contents, it does not allow for machine processing of such comments. For instance, it would be advantageous if comments in an XML Schema could be processed by a stylesheet, perhaps to create a user manual.

  W3C Schema introduces a standard <*documentation*> element that can be used to enclose comments. This looks as follows:

  ```
  <documentation>this is a comment</documentation>
  ```

  The DTD-style comment technique is also supported in W3C Schema. The <documentation> element is a subelement of an element called <annotation>. The <annotation> element contains, in addition to the <documentation> element, an <appinfo> element that is discussed in Section 8.3.9. Processing Instructions from Schema level <APPINFO>. The <documentation> element can have two attributes:
  - A “source” attribute that contains a URL to a file containing supplemental information
  - An xml:lang attribute that specifies the language that the documentation is written in

  The idea of using the xml:lang attribute is that you can have several sets of documentation for the same section of a schema, each in a separate language. A stylesheet may be written to extract only the comments written in a given language, and it can use the xml:lang attribute to select the text it needs.

The <annotation> elements (and its subelements) may be placed at the beginning of a schema as well as within schema constructs. For instance, the following <documentation> element provides general notes for an element:

```
<xsd:element name="Book">
    <xsd:annotation>
        <xsd:documentation>
          The Book element contains basic information about books in the
          library, such as title, author, etc.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Title" type="xsd:string"/>
            <xsd:element name="Author" type="xsd:string"/>
            <xsd:element name="Date" type="xsd:string"/>
            <xsd:element name="ISBN" type="xsd:string"/>
            <xsd:element name="Publisher" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

An XSLT stylesheet may contain a template rule to access all <documentation> elements in a schema as follows:

```
<xsl:template  match="xsd:schema/xsd:annotation">
    <xsl:apply-templates  select=”xsd:documentation”>
</xsl:template>
```

Formatting rules can also be applied to the <documentation> elements to create (for example) a user manual

Benefits of Using <documentation> Element: Use of the <documentation> element to add comments to a schema rather than the DTD-based approach is advantageous because it allows the comments to be processed by a processing application or program such as a stylesheet. Once this is done, there is no limit to what can be done with the extracted comments.

Risks of Using <documentation> Element:

There are no risks to using this technique.

Recommendation: The *<documentation>* element SHOULD be used for comments. The DTD-based comment technique SHOULD NOT be used.

8.3.9 Processing instructions from Schema level <APPINFO>

Description: This section discusses the inclusion of supplemental instructions in an XML schema for the purposes of passing them onto a processing application. Once a processing application receives such instructions, it is anticipated that the application would execute them for some intended purpose, most likely related to an XML document. This section begins with a discussion of Processing Instructions, which originated in XML DTDs, then covers the equivalent for XML Schema, which is the <appinfo> element. The discussion of Processing Instructions is intended to provide a basis and history for the concept of supplemental instructions.

Processing Instructions: Processing instructions are an XML DTD feature that allow an XML parser to pass information to a processing application. The information that is passed is of no particular interest to the parser; however, it is most often a command or set of commands for a processing application. Processing instructions are indicated in DTDs by special tags of <? and ?>, and are of the following format:

```
<?target …instruction…?>
```

The instruction portion is merely a string literal that may include any valid character string except the "?>" string literal. The target portion is an indicator that identifies the application to which the instruction is directed. An example is as follows:

```
<?realaudio  version="5.0"  frequency="5.5kHz"  bitrate="16kbps" ?>
```

A XML parser may pass the above information to a processing application. It is then up to that processing application to locate an application called realaudio (or at least one identified by that string) and pass it the parameters shown above (alternatively, it may identify a realaudio application version 5.0 from among several possible versions that are available). The most important point is that an XML parser, if designed to do so, simply passes the processing instruction on to an application. What is done with the processing instruction from that point on is completely out of scope of the XML parser.

It should be noted that the XML prolog (the line that appears at the top of every XML document) is also itself a processing instruction:

```
<?xml version="1.0"?>
```

APPINFO Element

The <appinfo> element is the XML Schema equivalent of the processing instruction. The <appinfo> element is one of two types of annotation elements that can occur in a schema, the other being the <documentation> element. Both elements occur under the <annotation> element, which is discussed in Section 8.3.8. Like processing instructions, the <appinfo> element offers a place in which to provide additional information that can be passed to a processing application by an XML parser. An example is as follows:

```
<xsd:group name="VehicleYearGroup">
    <xsd:annotation>
        <xsd:appinfo>
            if (currentNode.firstChild != "2001 Vehicles")
                docParser.load(Vehicles2001List);
            else
                document.write("There are no vehicles yet available for this
                year.");
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:choice>
        </xsd:element name="2001 Vehicles" type="xsd:string" />
        </xsd:element name="2002 Vehicles" type="xsd:string" />
```

```
        </xsd:element name="2003 Vehicles" type="xsd:string" />
    </xsd:choice>
</xs:group>
```

The above example is associated with an application that presents to a user a list of vehicle years from which to choose, presumably to search for one or more vehicles from that year. It assumes that of the three choices of years (2001, 2002, and 2003), only 2001 vehicles are available for search. The script inside of the <xsd:appinfo> tags checks which of the three available "choice" values was selected by a user; if the first was selected *("2001 Vehicles")*, a program is invoked to display a list of 2001 vehicles to the user. Otherwise, a message is displayed stating that *"There are no vehicles yet available for this year."*. It should be noted that although there are more efficient ways of accomplishing this in an application *(perhaps by querying a database rather than hardcoding the choice in the script)*, it is for illustration purposes only. It should also be noted that although the processing instructions shown above are very "Java-like", they could be equally have been a URL reference to be passed to a processing application, a set of XSLT commands for manipulating an XML document, or any other type of information that a processing application might utilize.

Benefits of <appinfo> Element

The <appinfo> element can be very useful for passing processing commands or other types of supplemental information to a processing application.

Risks of Using <appinfo> Element

The use of the <appinfo> element is considered highly risky at this time, due to the immaturity of XML schema processors available. There is no guarantee that a given XML schema processor will properly pass the processing instructions to an application, or, if it does, that an application will be able to accept them or handle them properly.

Recommendation

The <appinfo> element MUST NOT be used.

9.0 SUMMARY OF PROPOSED DESIGN RULES

To Be Completed.

10.0 PROCESS AND MANAGEMENT CONSIDERATIONS

10.1 Management of Templates

The creation of new and reuse of existing Templates is a critical consideration to the community. A high-level guiding principal of this work is to base all work on semantics. Semantically duplicate or uses not being semantically equal would undermine this goal and introduce a level of chaos to this picture. Therefore, Templates are heavily managed and scrutinized.

Templates will be governed by specific subcommittees, who will have the primary responsibility to manage the set of peer Templates. The subcommittee, in accordance with documented business process driven requirements, manages the decision to create new peer Templates or to alter use of existing Templates. Technical assistance and scrutiny are provided by X12J, Technical Assessment, with assistance as required by X12C, Communication & Controls. Approval is required by Technical Assessment in addition to the entire X12 community.

10.2 Management of Modules

Modules are constructed out of a set of reusable Blocks and/or Assemblies, such that the Modules respond to the Purpose specified by the Slot. The Assemblies and Blocks are a pre-existing tool kit for building Modules. Therefore, autonomy for the construction of these Modules is reasonable. Any recognized group, which conforms to the Semantic requirements for fulfilling the Slot purpose, and which builds using the existing Blocks, can construct Modules which should be considered compliant. Therefore, Modules are built by subcommittees and are approved by subcommittees and Technical Assessment.

Annex A: Definitions

To be completed, and subject to review and revision.

Abstract elements and types	
Aggregate Information Entity	Defines a functional unit representation form that contains embedded information entities. An Aggregate Information Entity contains two or more Basic Information Entities or Aggregate Information Entities that together form a single business concept (e.g. postal address). Each Aggregate Information Entity has its own business semantic definition
Annotation	An annotation is information for human and/or mechanical consumers. The interpretation of such information is not defined in the XML Schema specifications
AnyAttribute	
AnyElement	
Assembly Information Entity	
Attribute	
Attribute Declaration	An attribute declaration is an association between a name and a simple type definition, together with occurrence information and (optionally) a default value. The association is either global, or local to its containing complex type definition. Attribute declarations contribute to validation as part of complex type definition validation, when their occurrence, defaults and type components are checked against an attribute information item with a matching name and namespace
Attribute Group	
Attribute Group Definition	An attribute group definition is an association between a name and a set of attribute declarations, enabling re-use of the same set in several complex type definitions
Basic Information Entity	Defines a component which contains data but which does not have embedded information entities. A Basic Information Entity is a singular concept that has a unique business semantic definition. A Basic Information Entity adds semantic meaning to a single datatype or a Core Component Type (CCT).
Block Information Entity	
Built-in Datatypes	**Built-in** datatypes are those which are defined in this specification, and can be either primitive or derived
Business Process Models	UML models that describe interoperable business processes that allow business partners to collaborate
Character set	
Complex Type	Complex types which allow elements in their content and may carry attributes
Complex type abstractness	

Complex Type Definition	A complex type definition is a set of attribute declarations and a content type, applicable to the attributes and children of an element information item respectively. The content type may require the children to contain neither element nor character information items (that is, to be empty), to be a string which belongs to a particular simple type or to contain a sequence of element information items which conforms to a particular model group, with or without character information items as well.
Complex type extension	
Complex type restriction	
ComponentInformationEntity	
Core Component	Generic term that covers Core Component Type, Aggregate Information Entity and Basic Information Entity
Core Component Type	Core Components that have no business meaning on their own. When they are reused in a business context, they become Basic Information Entities. For example, quantity on its own has no business meaning, whereas the quantity shipped does have business meaning. Core Component Types consist of one component that carries the actual value (value component) plus others that give extra definition to the value (supplementary component(s)). For example, the value component 12 has no meaning on its own, but 12 kilometres or 12 Euros do have meaning
Datatype	A **datatype** is a 3-tuple, consisting of a) a set of distinct values, called its value space, b) a set of lexical representations, called its lexical space, and c) a set of facets that characterize properties of the value space, individual values or lexical items.
Default attribute values	
Derived Data Types	**Derived** datatypes are those that are defined in terms of other datatypes. A datatype is said to be derived by **restriction** from another datatype when values for zero or more constraining facets are specified that serve to constrain its value space and/or its lexical space to a subset of those of its base type. Every datatype that is derived by **restriction** is defined in terms of an existing datatype, referred to as its **base type**. **base type**s can be either primitive or derived
Document Entity	The root element for an X12/XML message consisting of one or more Aggregate Information Entities
Element	
Element Declaration	An element declaration is an association of a name with a type definition, either simple or complex, an (optional) default value and a (possibly empty) set of identity-constraint definitions.
Empty content	

Empty element	
Enumeration	The practice of limiting the value space of an element or an attribute to a specific set of values
Facet	A **facet** is a single defining aspect of a value space. Generally speaking, each facet characterizes a value space along independent axes or dimensions
Fixed attribute values	
Globally defined attributes	
Globally defined elements	
Groups	
Lexical Space	A **lexical space** is the set of valid *literals* for a datatype
List types	
Locally defined attributes	
Locally defined elements	
Mixed Content	A combination of child elements and character data nested within an element
Model Group	A model group is a constraint in the form of a grammar fragment that applies to lists of element information items. It consists of a list of particles, i.e. element declarations, wildcards and model groups. There are three varieties of model group: • Sequence (the element information items match the particles in sequential order); • Conjunction (the element information items match the particles, in any order); • Disjunction (the element information items match one of the particles).
Model Group Definition	A model group definition is an association between a name and a model group, enabling re-use of the same model group in several complex type definitions
Module Information Entity	
Multipart keys	
Named Types	
Namesapces	An XML namespace is a collection of names identified by a URI reference which are used in XML documents as element types and attribute names
Notations	
Notation Declaration	A notation declaration is an association between a name and an identifier for a notation. For an attribute information item to be valid with respect to a NOTATION simple type definition, its value must have been declared with a notation declaration
Occurrence constraints	
Primitive Data Types	**Primitive** datatypes are those that are not defined in terms of other datatypes; they exist *ab initio*
Processing instructions	
Scoped keys	
Simple Type	Simple types cannot have element content and cannot carry attributes

Simple Type Definition	A simple type definition is a set of constraints on strings and information about the values they encode, applicable to the ·normalized value· of an attribute information item or of an element information item with no element children. Informally, it applies to the values of attributes and the text-only content of elements
SlotInformationEntity	
Substitution groups	
Target namespace	
TemplateEntity	
Type Derivation	
Type Redefinition	
Type Substitution	Allows a base type to be substituted by any derived type
Union types	
Uniqueness constraint	
User-derived Datatypes	**User-derived** datatypes are those derived datatypes that are defined by individual schema designers
Value Space	A **value space** is the set of values for a given datatype. Each value in the **value space** of a datatype is denoted by one or more literals in its lexical space.
Wildcard	A wildcard is a special kind of particle which matches element and attribute information items dependent on their namespace name, independently of their local names
XML Schema	An XML document that defines the allowable content of a class of XML documents. A class of documents refers to all possible permutations of structure in documents that will still confirm to the rules of the schema

Annex B: Notional X12/XML Message

To Be Provided

Annex C: Notional X12/XML Schema

To Be Provided

Annex D: A model of the message design process

To Be Provided

Annex E: A model of the schema design process

To Be Provided

Annex F: Use of modeling with XML development

To Be Provided

Annex G: Background

1.0 Background

The Extensible Markup Language (XML) was developed by the World Wide Web Consortium (W3C), the de facto standards body for the Internet and the World Wide Web. The first working draft paper on the concept of XML was published 14 November 1996. The original goal was, *"…to enable SGML to be served, received, and processed on the Web in the way that is now possible with HTML."* A primary design consideration was to design XML, *"…for ease of implementation, and for interoperability with both SGML and HTML."* Much of the original concept was applied to using XML as a means for graphical communication. The idea of its use for conducting EDI was applied later when the first studies were done on this subject in late 1997. Early work on XML/EDI was conducted both jointly and independently by ANSI ASC X12, UN/CEFACT, CommerceNet, and the XML/EDI Group as well as other organizations. The goals of XML/EDI as defined by the XML/EDI Group are:

- To deliver unambiguous and durable business transactions via electronic means
- Utilize existing systems and processes
- Protect the investment in traditional EC/EDI
- Provide a migration path to next generation XML/EDI systems
- Use existing business processes as implemented
- Facilitate direct interoperation in an open environment

In November 1999, work began on the ebXML project, a joint UN/CEFACT and OASIS initiative, whose mission was to provide an open XML-based infrastructure enabling the global use of electronic business information in an interoperable, secure, and consistent manner by all parties. The project concluded in May 2001 and delivered a modular suite of specifications that enable enterprises to conduct business over the Internet. The specifications address the following areas:

- Messaging Services
- Registries and Repositories
- Collaborative Protocol Profile
- Implementation, Interoperability, and Conformance
- Core Components and Business Process Models

The ebXML specifications are currently being transitioned to UN/CEFACT and OASIS for the purpose of developing global electronic business standards.

X12 began work on XML/EDI in 1998 with the creation of an ad-hoc XML work group that transitioned to X12C/TG3. X12C/TG3 in conjunction with CommerceNet produced a paper entitled "Preliminary Findings and Recommendations on the representation of X12 Data Elements and Structures in XML". In addition to this collaborative effort, X12C/TG3 produced a technical white paper providing additional information on using XML to represent business exchanges. In February 2000, the X12 Steering Committee chartered the X12 XML Task Group to develop recommendations for the Steering Committee in conjunction with the X12 subcommittees on XML. The resolutions approved by the Steering Committee in June/October 2000 were:

- The ANSI ASC X12 Steering Committee fully supports the continuation of the mission, goals, and efforts of ebXML. ASC X12 will pursue its XML development efforts within the framework defined by ebXML.
- X12 will develop accredited, cross-industry, XML business standards. All XML business standards and associated schema development work will be done in collaboration with UN/CEFACT Work Groups and shall be based on the UN/CEFACT business process/core component work.
- The X12 Steering Committee will petition ANSI for official recognition as an ANSI accredited XML business standards body
- X12C will function as the X12 XML technical experts with respect to all internal and external XML technical specifications including the development of XML design rules in conjunction with X12J
- The X12 Steering Committee shall task DISA to begin working with X12X TG4 WG2 to market X12's role in developing ANSI accredited XML business standards.
- The X12 Steering Committee shall task the Process Improvement Group (PIG) to include the need to recognize the requirement for an accelerated process for XML standards development as part of their work plan
- The X12 Steering Committee shall task the Process Improvement Group (PIG) to work with the Policies and Procedures Task Group (P&P) to provide expertise and assist the EWG/X12 on the Joint Development Task Group in the development of an aligned approval process that meets the needs of both organizations related to the development and maintenance of XML core components.

Every effort has been made to build on the experience and work done previously by ebXML, the UN/CEFACT Work Groups, CommerceNet, and ANSI ASC X12 in document definition methodologies and core components. The X12/XML design rules presented in this document are based on design decisions reached through a process of issue identification, presentation of examples, and evaluation of the pros and cons of each available action. They provide a set of syntax production rules that define the conversion of standardized, cross-industry business messages into XML documents.

2.0 Overview of ebXML Business Process and Core Components

The business process determines characteristics of the business document payload. For example, if the business process is Ordering then the order information must specify details about the order itself (payment, delivery, references to external business agreements, etc.). There are certain characteristics of the Order Document, which typically do not vary across industries, while other details (such as those required because of product type) will vary dramatically.

Business documents, by their very nature, communicate a semantically complete business thought: *who*, *what*, *when*, *where* and *why*. The *what* in electronic business terms is typically the product. It is widely recognized that products are goods or services. Goods are manufactured, shipped, stored, purchased, inspected, etc., by parties. Services are performed by parties, and may involve goods and/or parties. Parties can be either organizations or individuals, and can be associated with other parties and products. And these products have events associated with them, inspections, transportation, building, sale, etc.

This problem is addressed by a combination of structured information and the use of context. This structure uses a series of layers, designed to take into account commonality across industry business process. Further the structure is designed to support specialization based on the specific use of contexts. Context is the description of the environment within which use will occur. For

example, if one was to say that “someone was pounding on my car with a hammer”, the response is very different depending whether it is a repair shop or a neighbourhood youth. Context is what is used to direct interpretation.

A component is a ‘building block’ that contains pieces of business information, which go together because they are about a single concept. An example would be *bank account identification*, which consists of *account number* and *account name*.

Core components are components that appear in many different circumstances of business information and in many different areas of business. A core component is a common or “general” building block that basically can be used across several business sectors. It is therefore context free.

Re-use is the term given to the use of common core components when they are used for a specific business purpose. The purpose is defined by the combination of contexts in which that business purpose exists. Each context specific re-use of a common component is catalogued under a new business information name ‘that uses core component X’.

A domain component is specific to an individual industry area and is only used within that domain. It may be re-used by another domain if it is found to be appropriate and adequate for their use, and it then becomes a core or common component.

Components can be built together into aggregates.

As described above for components, aggregated components can be common components. These are generic and can be used across several business sectors. They can be re-used for a specific business purpose, defined by a combination of contexts. Each context specific re-use of a common aggregate component is catalogued under a new business information name ‘that uses core component X’.

There are also domain specific aggregated components.

Aggregates and components can be gathered into "document parts". These are useful assemblies which can individually satisfy a business process’s requirement for information, or which may be "sewn together" in a structured way to achieve the same. For example, the structured combination may be to satisfy a business process’s need for information presented in a particular way for efficiency of processing.

An individual document part and the "sewn together" parts, come at increasingly domain-specific and context-specific levels. They form documents or partial documents that satisfy a business process or a part of a business process.

Figure 16 illustrates how core components can be built into business documents by explicitly linking components with the ebXML Business Process Worksheets, and the underlying modelling approach. The top right-hand corner of the Figure comes from Figure 8.4-1 in the ebXML Business Process Overview document.

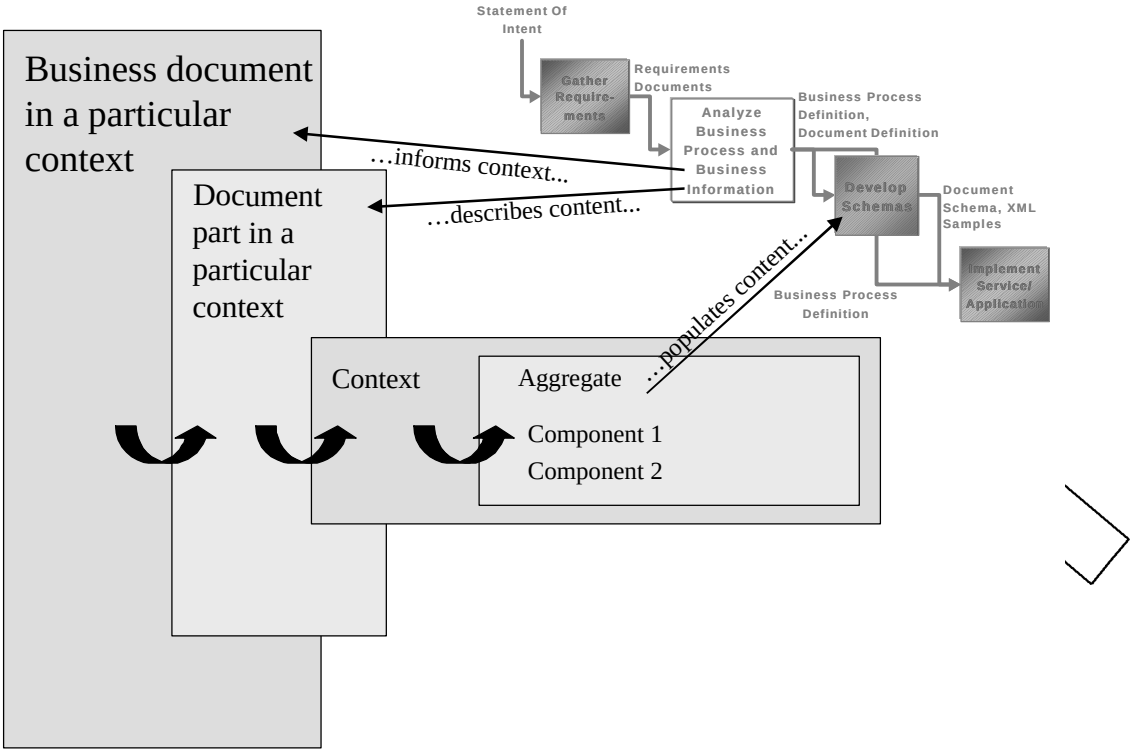


Figure 16

Note that in this instance document parts are pieces of business information required to satisfy a particular business process, from a specific contextual viewpoint.

3.0 Relationship to other XML Efforts

Since most other XML efforts lack an overriding semantic organization, many efforts have been directed to production of "bullet" messages. This effort is directly applicable by narrow definition of the business purpose underlying each Template. In particular, ebXML efforts have componentry definitions with instances that span several levels. The architecture proposed here provides a structured mechanism to impose a semantic discipline in this arena.

Several XML efforts have modeling as a primary tenet. Modeling may prove to be the best way to develop items at the top levels of this architecture (certainly Templates and Modules and possibly Assemblies). X12 feels that this architecture allows modeling to be used at high levels, where it is most effective.