EAN·UCC System

The Global Language of Business ®

EAN.UCC XML Architectural Guide Version 1.0

July 2001

Table of Contents

XML Architectural Standards

Document History

Document Version:	***Version 1.0***
Document Issue Date	***July 2001***

Document Change History Log

Date of Change	*Version*	*Reason for Change*	*Summary of Change*

1. Introduction

This document is intended to provide an easy, readable description of the EAN.UCC architecture of the XML schema system. It will also quickly orient you to a brief understanding of how to create and use the XML schema language.

1.1 Overview

There are four main concepts behind the architecture of version 1.0 of the EAN.UCC XML Standards of July 2001:

– Message layering
– Unique identifiers
– Reusable documents and
– Context sensitive message definitions.

Within the EAN.UCC XML standards, the core methodology for constructing core components is that the core is based on abstractions that are extensible.

The extensible part, or the extensions, will support specific business requirements, e.g. in the context of industries, country and legislative needs. Extensions are built upon the abstractions and create new extended components. The extension specializes the core by constraining the neutral core to assume a specific business practice.

1.1.1 Extensions in UML

Schemas begin as new or previously created classes, attributes, and interfaces represented as UML attributes and classes. The classes and attributes are shareable across the business processes and make up the basic building blocks of the EAN.UCC Candidate XML Schemas. The element tags and their values that make-up the EAN.UCC XML Instance Documents are created from the transformation of the UML model components into named tags and attributes.

Each UML class and attribute of a class is associated with a type of information. This information can be simple or complex. It is stored in a UML package. Reusable types can be found in different packages.

1.1.1.1 Common Core Components

A **Common Core Component** specifies just the elements and their attributes, which are common across all industries and all business processes. Core Components are a modeling of data structures. They are the basic building blocks that are commonly used across all the business processes and their hierarchies. These are pre-defined, pre-registered and pre-instantiated in the UML Repository and contain the basic units of functionality required for a business process document to be modeled and implemented. They are defined and depicted in the UML models, user scenarios, diagrams, and conventions that make up the UML EAN.UCC Business Process Class Models and their relations.

Examples of Core Components are PartyIdentification, Measurement, and Quantity in UML or PartyIdentificationType, MeasurementType, and QuantityType when transformed into the XSD schema.

1.2 In A Nutshell: A Business Process Document

A Business Process Document is a named group of related core components that provide all the information necessary to complete a definition of a particular business process. These named units of information or messages contain the data elements that are transferred between two or more systems.

A document for a particular business process is composed of a core portion, which is extended to meet additional requirements for the extension. The extension could be provided for a specific industry or a specific location or need. FMCG is an extension for the Fast Moving Consumer Goods. Simpl-eb is an extension for the simple electronic business. The extensions are implemented in the namespace proxies (see, fmcg.xsd and simpl-eb.xsd) and the files that are included in these proxies. The files contain the XML extensions of the abstract types found in the commo0nly used files.

Individual document schemas could be created from business class models representing each industry or geographic need. However, the end result would be that there would be many schemas that contain the same information with slight variations on some classes and attributes. By using schema extension mechanisms, similar to object-oriented sub-classing, the Core Business Process Schemas, are extended to create context specific Business Process Schemas with the appropriate elements to support a particular usage context. An extension mechanism is necessary to extend the abstract common core components to provide the variations that will adequately support industry and business usage.

1.3 Other Related Documents

The EAN.UCC Candidate XML Schemas are based upon the Business Requirements Documents and the Business Message Standards and on the models found in this document. As an additional aid, a file describing how to use the schemas is included as Appendix B of this document. The schemas themselves are found in printed form in Appendix C with a link to the actual xsd schemas. Finally a Reference to the schemas is located in Appendix D.

2. Implementation Guidelines

The guidelines were used in the construction of the EAN.UCC schemas. For a successful implementation of these schemas, users should understand these guidelines. Developers of schemas for rapid development of business documents should follow the guidelines described here. This will ensure that EAN.UCC XML conventions are met in the new schemas still to be created.

2.1 Schema Language

The World Wide Web Consortium (W3C) released XML Schema as a W3C Recommendation in three parts: Part 0: Primer, Part 1: Structures, Part 2: Datatypes on May 2, 2001. The EAN.UCC schemas conform to and are implemented using this Recommendation.

2.2 Implementation Software

The EAN.UCC schemas have been implemented and tested using the following software:

Schema Quality Checker (SQC), v1.0.17, now checks Schemas compatible with the May 2 W3C XML Schema Recommendation specification from IBM alphaWorks.

XML Schema Validator (XSV) version: XSV 1.195/1.97 of 2001/06/09, an XML Schema Validator from University of Edinburgh/W3C.

2.3 Messaging Protocols

The EAN.UCC version 1.0 schemas have been implemented for use with EDIINT/AS2 protocol from the Internet Engineering Task Force (IETF).

2.4 Guidelines and Conventions

The following guidelines and conventions were followed in implementing the business process models as Candidate XML Schemas.

2.4.1 Abstract Classes and Types

Conforming to the XSD language, abstract classes and types are used to derive concrete classes but do not by themselves implement elements.

Abstract classes are used to represent core documents, which by the definitions set forth by the Business Modeling Groups (BMGs) are not implementable as stand-alone modules.

2.4.2 Concrete Classes and Elements

Concrete classes and elements may be derived from abstract classes and types by extending them to form concrete representations. An example of this process is extensions of abstract classes and types to form fast moving consumer goods (fmcg) specific types and elements. fmcg specific elements are implemented in this way and can be found in the XML instance file.

2.4.3 Complex Types and Groups

New EAN.UCC types can be derived from a group, simpleType or a complexType. A derived complexType has as its effective content model the content model of the base type plus the content model specified in the type derivation.

2.4.4 Naming and Capitalization Conventions

2.4.4.1 Elements and Attributes

All element and attribute names begin with a lower case. All words that follow in the name are capitalized, or in other words, elements and attributes appear in "camel case". For example,

- globalLocationNumber
- uniqueCreatorIdentification

2.4.4.2 Constants

Constants such as those that appear in the enumerated lists appear as all capitals. Words are separated via an underscore (_). For example:

- MARKET_GROUP
- DUNS

2.4.4.3 Complex Types and Groups

Complex types have each word begin with a capital. For example:

- PartyIdentificationType
- MeasurementType

Groups use the same naming convention as complex types.

2.4.4.4 Namespaces

Namespaces should use all lower case letters.

2.4.4.5 File Names

File names should use the same naming convention as complex types.

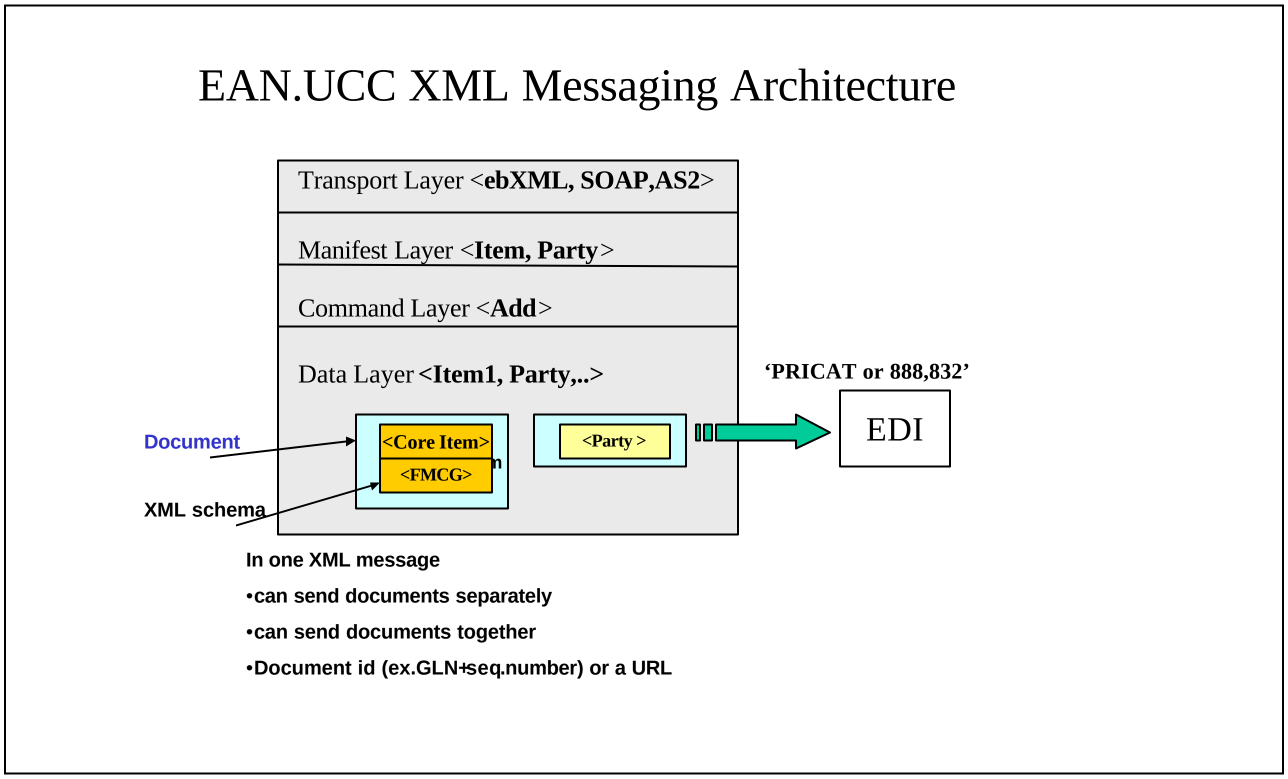
3. Namespaces

3.1 The Architectural Layers and targetNamespaces

The overall architecture principle of the EAN.UCC schema system is the separation into layering of functions. The concept of layering is implemented using XSD Namespaces. Namespaces allow for modularity in the overall schema design and in the separation of layers. Modularity allows for extensions to the core business documents, and in that way multiple products, industries and geographic needs can be supported.

The layered message architecture of EAN.UCC is as follows:

3.1.1 EAN.UCC EC Messaging Architecture


EAN.UCC XML Messaging Architecture
Transport Layer <ebXML, SOAP,AS2>
Manifest Layer <Item, Party>
Command Layer <Add>
Data Layer <Item1, Party,..>
'PRICAT or 888,832'
EDI
Document
<Core Item>
<FMCG>
<Party >
XML schema
In one XML message
•can send documents separately
•can send documents together
•Document id (ex.GLN+seq.number) or a URL

Note: Support for Manifest layer will be provided in a future release.

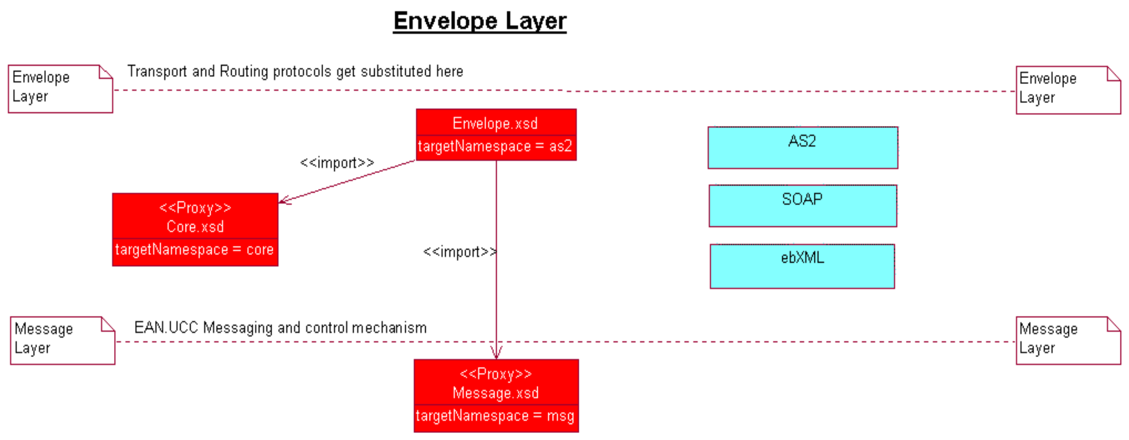
3.1.2 The Layers

How the layers are implemented using XSD, is shown next, but first we show the legend for the architectural diagrams.

Legend

🟨	Schema file
🟥	Schema File with namespace root
🟦	Transport and routing protocols
- - - - - - - - -	Supported in a future release
🗋	Comments

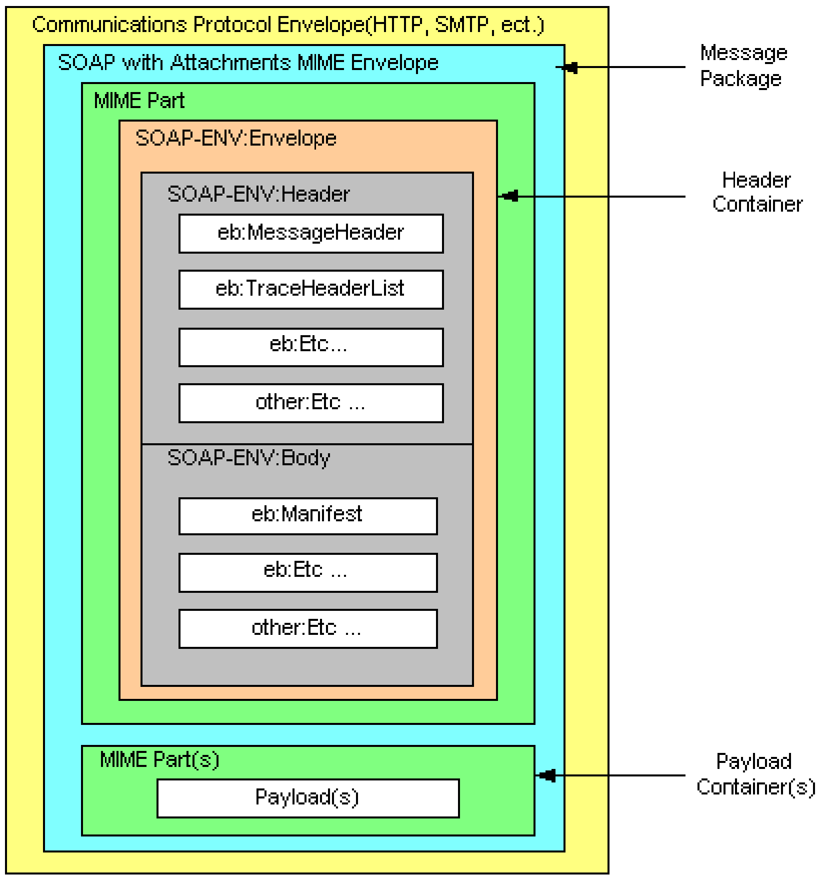
3.1.2.1 Envelope Layer



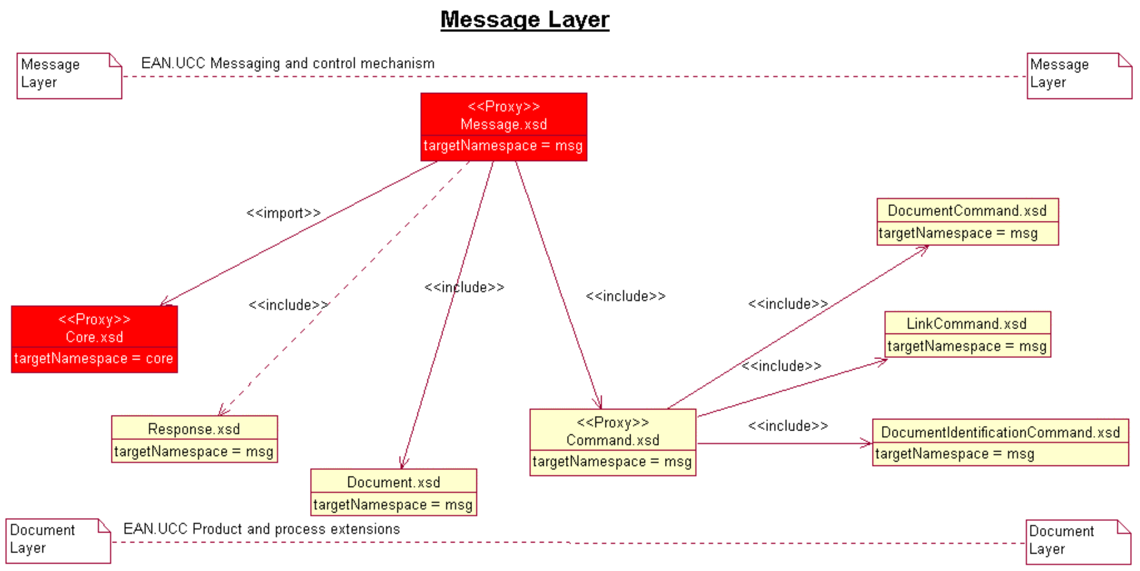
The envelope layer contains the message header. The targetNamespace for this layer is as2, since AS2 is the current routing and transport mechanism supported. In the future other protocols will be supported. The Message Header currently supports the AS2 protocol. When the new protocols are implemented, the as2 namespace will be completely replaced by the namespace, soap and ebXML.

Messaging Architecture - Future Support (ebXML)

In a future release, the ebXML Transport and Routing protocol shown below will be supported.

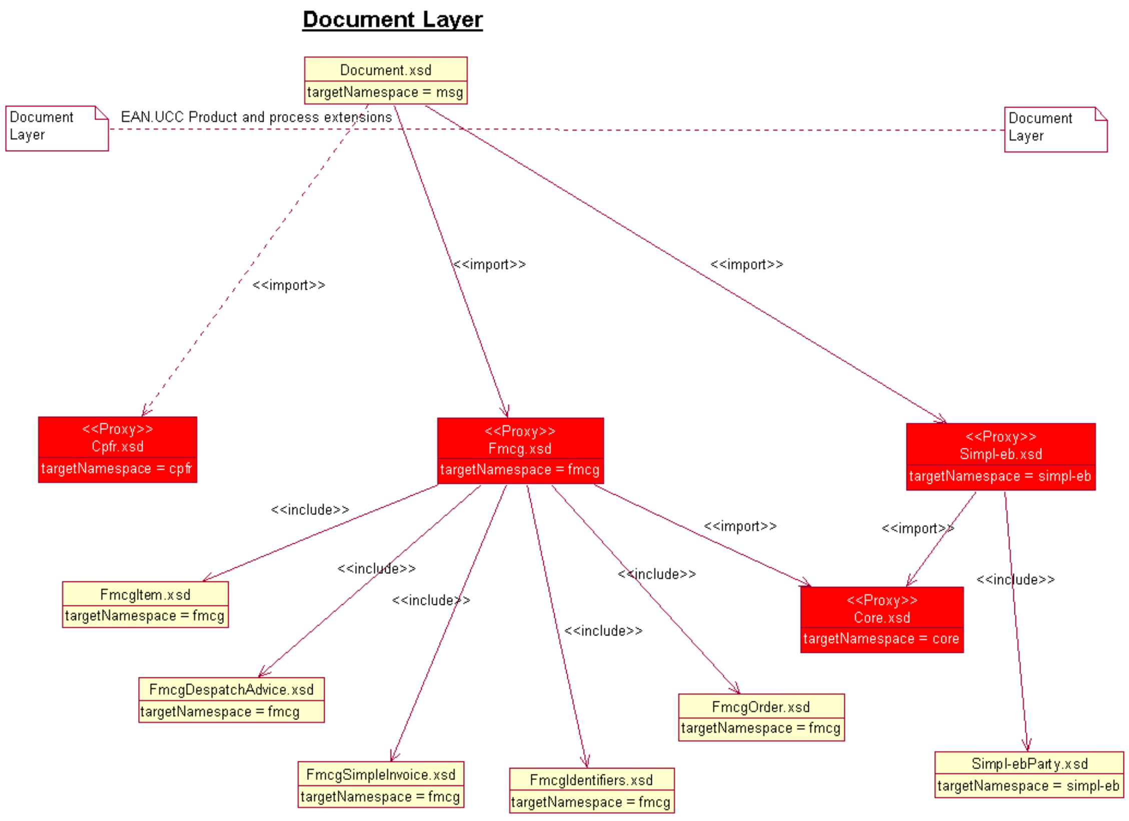


3.1.2.2 Message Layer



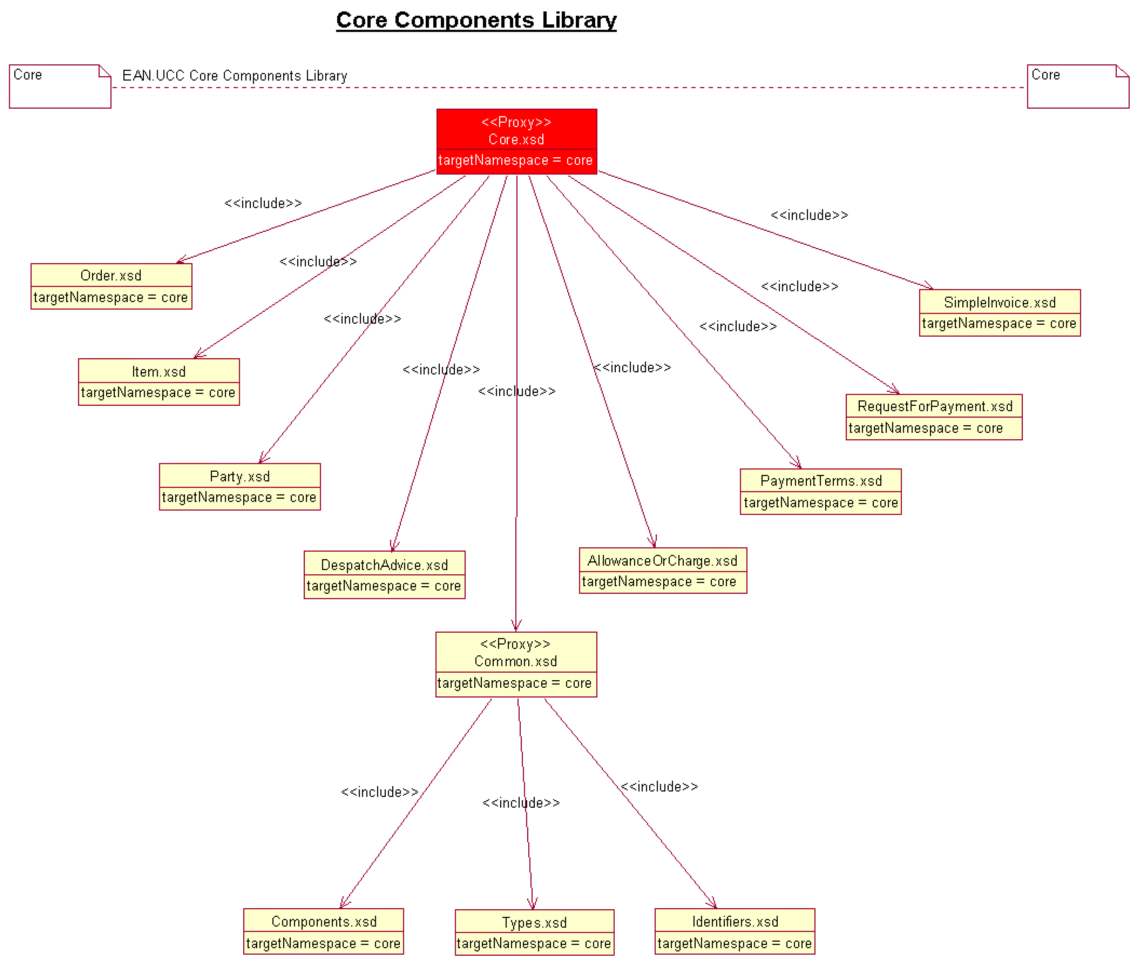
The Message Layer supports the EAN.UCC Message Architecture. This Layer contains the transaction, command, and the interface to the Document Layer. It also will contain the response in the future. The targetNamespace for this layer is msg, and supports the messaging architecture.

3.1.2.3 Document Layer



The Document Layer supports the extensions and currently contains the fast moving consumer goods (fmcg) and the simple electronic business (simpl-eb) extensions. In the future it will also contain the Collaborative Planning Forecasting and Replenishment (cpfr) extensions. There are currently 2 supported targetNamespaces in this layer: fmcg and simpl-eb. Document.xsd imports the documents in the vertical extensions.

3.1.3 Core Components Library



The Core Components Library provides the reusable library of base definitions or commonly used components for each of the Layers. These common components are imported into each of the three layers above and are the foundational basic building blocks. The targetNamespace for this layer is core, and supports the core and complex components.

The elements and attributes defined in core can be used by multiple XML components. The reusable components, which occur commonly across all the business documents, reside in the core namespace. Abstract types, which are not implementable because they have not yet been extended into concrete representations, reside in the core namespace.

3.2 The Benefit of Namespaces

Namespaces resolve the problem of preventing naming collisions. Therefore the abstractions and their derived concrete representations may have the same name, however, since they belong to 2 different namespaces, core and fmcg, for example, there are no collisions as a result. Software validators used to parse the schemas for XML file validation recognize the tags and attribute names and resolve them. Thus the collisions are avoided.

Since our design consists of schema modules which taken together constitute a working schema different layers containing schema modules at each layer, the tag and attributes names have a scope that extends outside the local scope, or outside the .xsd file in which they are created.

3.3 The Namespace Universal Resource Indicator and Prefix

There are 2 parts to using namespaces:

They are identified with Uniform Resource Identifiers (URIs)

The namespace is identified via a name prefix that qualifies the namespace.

3.3.1 The Universal Resource Indicator

The XML Namespace is a collection of names, identified by a Universal Resource Indicator (URI) which is used in XML documents as element types and attribute names. In our set of schemas for the fmcg business processes, the URI is:

http://www.uc-council.org/smp/schemas

3.3.2 The Prefix

XML namespaces may have qualified names, which contain a namespace prefix followed by a single colon, and a local name. A targetNamespace declaration enables us to distinguish between definitions and declarations from different vocabularies by identifying them. The entities within that target namespace are explicitly prefixed or are implicitly namespaced by default. A prefix is associated with the schema's target namespace and used to refer to types and elements defined and declared within the schema.

The prefixes for the core and fmcg namespaces in the EAN UCC XML Standards are:

- as2
- core
- fmcg
- msg
- simpl-eb.

3.3.3 Localizing and Exposing Namespaces

In the EAN UCC XML Standards resultant XML instance document, the core: namespace is localized or hidden in the XML file. This is because exposing this namespace does not provide any added value or any significant information to the XML instance document. By hiding the core: prefix, the XML file is easier to read and understand.

On the other hand, exposing the fmcg and simpl-eb namespace indicates that the tags prefixed belong to the specialized case, that of fast moving consumer goods or simple electronic business. This is needed in the XML instance file to signal extended information. Special instructions may be required to process this information, and because of the prefix qualification, those tags are readily identifiable.

The elementFormDefault=”unqualified” rule in the schema hides or localizes the namespaces, and the elementFormDefault=”qualified” exposes them in the instance documents. Qualification can occur upon attributes or elements. Attributes or elements can be qualified when they are declared globally or because the **attributeFormDefault** or **elementFormDefault** is set to **qualified.**

4. Extensions

Within the EAN UCC XML Standards, the central part making up the predefined core or complex components cannot be changed (unless it is determined that they must be changed at some release by the Business Managers). These are the abstractions.

Extensions will support the specialization requirements within products, industries, country needs, usage, business practices and legacy technology requirements. Extensions build upon the abstractions and are concrete. The abstractions create new components, by extensions to them. This extension specializes the core by forcing it to assume a particular business practice.

4.1 The Business Requirements

4.1.1 Support Multiple Industries and Countries

We have a business need to support multiple industry and countries needs without fundamentally changing the core of our ebusiness standards. A large percentage of our business process nucleus does not change across geography, industry and usage needs. A smaller percentage does change across geography, industry and usage needs.

4.1.2 The Need for Speed

The Rapid Process Development methodology resolves the need for speed by allowing reusability across business processes. In this way, we will meet time-to-market needs of our members and at the same time allow for the flexibility to maintain their diverse business representation needs. For example, the core components of the fmcg business processes are recycled in the CPFR set of schemas allowing for a Rapid Process Development by reusing common components across both sets.

4.2 UML Depiction of Extensions

Schemas start out from new or previously created classes, attributes, and interfaces and are represented as UML attributes and classes. These are shareable and are the basic building blocks for building schemas. Element tags and their values that make-up the XML Instance Document are created from the transformation of named tags and attributes in the UML model.

4.2.1 Reusable Types Model

Each UML attribute of a class is associated with a type of information. This information can be a simple one or a compound one. It is stored in a UML package. Reusable types can be found in different packages.

A Common Core Component specifies just the elements and their attributes, which are common across all industries and all business processes. Core Components are a modeling of data structures. They are the basic building blocks that are commonly used across all the business processes and their hierarchies. Examples of Core Components are PartyIdentificationType, MeasurementType, and QuantityType.

These are pre-defined, pre-registered and pre-instantiated in the UML Repository and contain the basic units of functionality required for a business process document to be modeled and implemented. They are defined and depicted in the UML models, user scenarios, diagrams, and conventions that make up the UML EAN.UCC Standard Business Process Class Models and their relations.

4.3 Core and Extensions in XSD

4.3.1 The XSD Data Types

The XML Schema Definition Language already provides a basic collection of simple data types. The parser validates the inserting of correct data in the XML instance document according to the XSD data types.

4.3.2 XSD Libraries of Reusable Components

The EAN UCC XML Standard contains libraries of more complex commonly used components created from the simple XSD data types. These are EAN UCC created business representations, for example, represent global location numbers, measurements, and global trade items.

Basic building block core component XML instantiations are produced from the data structures and built-in data types provided by the W3C XML Schema Definition (XSD) Language Recommendation of May 2001. They are created in XML by transcribing the UML Class Models according to the rules captured in the EAN UCC UML to XML Design document. Strict adherence to those rules is essential for the transformation from UML to XML to take place. The transformation may be done manually or automatically.

4.3.2.1 The XSD Core Components Library

The core components library is composed of the following files.

1. Common.xsd (including):
   - 1.1 Types.xsd
   - 1.2 Components.xsd
   - 1.3 Identifiers.xsd
2. The rest are complex:
   - 2.1 AllowanceOrCharge.xsd
   - 2.2 Payment Terms.xsd
   - 2.3 DespatchAdvice.xsd
   - 2.4 Item.xsd
   - 2.5 Order.xsd
   - 2.6 Party.xsd
   - 2.7 RequestForPayment.xsd
   - 2.8 SimpleInvoice.xsd

The library contains the core component business processes and supplements (AllowanceOrCharge.xsd and PaymentTerms.xsd) that make-up the EAN UCC system. Once built and tested, these common complex core components are used to implement XML instance documents by extending them.

4.3.2.2 Common.xsd

The Common.xsd file contains the more atomic components, which can be used with other low-level and higher-level tasks. These are core component building blocks that are reused to build ever more complex routines.

Common.xsd is imported in each layer of the schema architecture. It is imported in the Envelope Layer; the Message Layer; the Extensions Layer.

The features of the common core components in Common.xsd are that they are used as a library of elements and types in Components.xsd. These have no cardinality and no optionality until they are referenced and implemented. They have no namespace of their own and are not identified as being required or not required. Common core components have no specific product, service or process information. The flexible non-target namespace approach would represent this architecture best.

The Benefits of Core Components are that once the information is identified, it may be used, reused, and repurposed. It may be used as units (elements and element cases) or in a modular manner.

4.3.2.3 XSD Complex Core Components

There are even more complex core components, which are the more complex representations of business processes that form the common denominator of a business document. These are common denominators because from that core complex component, no attributes or elements can be removed. This is the basis for the core business processes.

Reusable object-oriented core components provide a method for creating more complicated functional units from the predefined set of functions that basic common core components perform at the simplest level of electronic business process functionality.

In the EAN UCC System, the core business processes form the next level of hierarchy. An example of this is the Core Order, Core Item or Core Party. These are instantiated in XML in the order.xsd, item.xsd or party.xsd files. The core functionality of the business process includes its most common denominator functionality. This functionality may not be removed from core nor is core enhanced under any circumstances. The basic functionality of that business process is captured in its UML model and in its schema.

A Core Business Process, be it a representation of AllowanceOrCharge or DespatchAdvice, can never be implemented. All its internal structures are abstract. No elements or attributes can be derived from them, unless the abstractions are extended for a specialized purpose.

The core business processes and supplements are implemented in the following schema files:

- AllowanceOrCharges.xsd (supplement)
- Item.xsd
- Order.xsd
- Party.xsd
- PaymentTerms.xsd (supplement)
- DespatchAdvice.xsd
- SimpleInvoice.xsd

4.3.3 XSD Extensions

Finally, these complex core components are extended to produce the varied, flexible business extensions which extend the EAN UCC **basic building blocks** to accommodate different industries and geographic regions. Implementation level specific, unambiguous components can be created from the core. They are made unambiguous in the instance document by prefixes.

The extensions methodology can be described analogous to an XML tree. The components can be packaged as a sequence of components forming that XML tree. Each node on the tree has its required component, and each leg of the tree has its rules, which prescribe what the next linked in component will be in order to maintain a valid tree. Strict adherence to these rules will determine that the sequence of components is valid. In this way a pattern is established. If A->B is a valid combination and B->C is a valid combination then A->B->C is a valid combination.

4.4 Working Schemas From Multiple Schema Modules

The design of the extensions methodology as implemented in the EAN UCC system uses the XSD capabilities of including files and importing namespaces.

4.4.1 Composing Schemas from Included Files

The XSD provides the capability to compose schemas from other schemas or from schema modules. As already mentioned, this gives the EAN UCC system the ability to have new types based on existing types. Multiple included documents can be contained in one including document. All included documents must reference a common namespace.

4.4.2 Composing Schemas from Imported Namespaces

It is possible to refer to types included in other schemas with a different target namespace enabling re-use of definitions and declarations. The import mechanism enables schema components from different target namespaces to be used together. Import enables schema validation of instance content defined across multiple namespaces.

4.4.3 Schema Component Design Characteristics

Core Schema Components with no targetNamespace are more flexible because they are able to take on the namespace of any schema that <include>s them. They can be reused by any schema. They can assume any semantics for the schema that <include>s them they can take on a new role and semantics. Any schema, regardless of the schema's target namespace can redefine them.

4.4.3.1 The W3C Namespace Approach

The namespace approach was implemented to allow for greater flexibility in the reuse of the common core components. The namespace approach uses proxies to allow schema modules to assume a specific namespace. The root of the namespace is implemented in the namespace proxy files. These files are:

- Message.xsd
- Core.xsd
- Envelope.xsd
- Fmcg.xsd
- Simpl-eb.xsd

In addition, there are proxy files that do not implement a namespace root, but instead include an already implemented namespace. These are:

- Command.xsd
- Common.xsd

5. The Message Architecture

The envelope and message layers make up the message architecture part of the schema. They give the capability of packaging the message in an envelope, routing and transporting it and defining the messaging information.

5.1 Message Set Overview

The Message Set is broken into:

- Core – core structures used through-out the message set
- Envelope (Transport Layer)
    - Including the Message Header for AS2
- Message (Body)
    - Transaction
    - Command
        - Including Document
        - Document Identifier
        - Link and Unlink

5.2 Architecture Principles

The principles of the architecture followed in the implementation of the EAN UCC UML and XML Standards development are the following.

5.2.1 XML Message Layer Separation

XML Message layer separation has certain benefits. It reduces the overall message set needed for the EAN.UCC System. There is no need to create new schemas supporting the command for each document type (No specific ItemAdd/ItemChange or PartyAdd/PartyDelete schema is required). It allows for reusable components.

Commands such as Add can be reused for various documents and likewise, documents can be used for various commands. Thus, it allows for new processes to be rapidly developed through reuse of existing message command sets. When adding business process functionality, all documents can reuse add, delete, or link, commands. This allows for new commands to be added quickly.

5.3 The Benefit of Layering

Message layers can be replaced with other messaging systems while not affecting the lower layers. For example, the transport layer could be updated with any message transport standard which is defined (i.e., ebXML, XML Transport, SOAP).

Different parties may create the different layers

Communication Service provider (VAN,...) creates transport layer

For example, the functionality service provider (an exchange, for instance) could create the command layer, while the Trading partner may create the business or document layer.

5.4 Transport and Communication Layer

At the time of this writing, there exist several messaging standards, in various stages of implementation, which may be used by the EAN UCC community. These are AS2, ebXML Transport and Routing and SOAP. There is also work in progress from the W3C XML Protocol Group on the specification of a Messaging protocol.

The communications layer defines tags to allow messages to get from a sender to a set of receivers. This layer is essential to provide transport information such as: sender, recipient, guaranteed message delivery info, message id. The EAN UCC Standard does not define the transport standard, rather it is adopted from the work of other standards bodies such as the W3C or ebXML. It is assumed that other organizations will define this standard (W3C, ebXML)

5.4.1 Envelope.xsd

The envelope is in the Envelope.xsd file and wraps the entire package being sent it one posting. CommunicationVersion is a required attribute of “envelope” tag, which describes the version of schema this XML representation was created with. It is used for message versioning and allows the receiver of the message to identify the message set used to create content. The messageHeader is referenced in the envelope.xsd schema. Currently, for the July 2001 release, only the AS2 transport is implemented. In the future, there will be messageHeaders created that will support other transport standards, for example, SOAP or ebXML Transport and Routing.

The AS2 Message Header wraps a series of fields related to message being sent:

userId – indicate the individual or machine user related to the message being sent.

password – optional field indicating the authentication string of the above user. If/when IETF/EDIINT is integrated, password could be replaced by digital signatures utilizing digital certificates where IETF/EDIINT is used.

representingParty – indicates the GLN that originated the message. This maybe different than the sender of the message in the case of a service provider or agent relationship.

to – Optional field indicating the GLN the message is to be sent to. Should always be UCCnet in 2.0.

from – Optional field specifying the GLN the message is from.

creationDate – optional field supplying the date and time the message was created.

messageId – identifier for the message.

5.4.2 Message.xsd

5.4.2.1 Body

The body wraps around one or more Transactions.

5.4.2.2 The Transaction

The transaction is a part of the Command Layer. The Message set allows for multiple commands to be sent in one message. This gives users the capability to have a group of commands all succeed or all fail. A transaction wrapped around multiple commands will facilitate this all or nothing.

The contents of the transaction tag are;

EntityIdentification – a global identifier related to the transaction that allows an acknowledgement to separately provide status.

The command layer.

5.4.3 Command.xsd

5.4.3.1 Command

The command layer provides the messaging information. The body is referenced in the envelope.xsd. Envelope.xsd wraps around both the messageHeader and the command layer. The command layer begins with the body tag. The body tag can provide for 0 to many commands. The commands are packaged within transactions and the transactions can occur 0 to many times.

The command layer defines what action is to be taken to process the incoming message. An example of the type of action that can take place is an Add, or, Delete.

The information that the command action acts upon is defined within Command layer. For example: If an item is to be added, the command will be Add while the structure within the Add tag will be an item definition.

The supported command list is:

documentCommand

documentIdentificationCommand

linkCommand

The actual command could be an attribute of the list above. Every individual command is wrapped in a "command" tag.

documentCommand

The documents data layer defines information for the document that the documentCommand is to operate upon. An example of a document is the Fmcg Item or Simpl-eb Party. Currently, there are 5 such documents. There are 2 types of Data commands one is the Document itself.

documentIdentificationCommand

The second is the pointer to document (GLN, GTIN, Unique Document Identifier). This is used to indicate a delete on the document referenced when the command is invoked. The Command will contain a qualifier indicating actual action and documents to act on. The Qualifier includes the Delete command whose action removes a particular document stored by the recipient. The entityIdentification element is the identifier of the command and the documentIdentificationList contains the list of document identifiers that this command is to act on.

link Command

The linkCommand establishes a relationship between two or more documents. It may link two entities, item and party, of the same type in a parent-child relationship. It may also link two entities of different types in an association, such as associating a user to a role.

The linkCommand may be used for linking and for unlinking. The elements and attributes used in this command are: the linkType which is an attribute indicating whether the command is linking or unlinking the entities together; the entityIdentification or the identification of the command; the documentIdentifier of the parent document to add the links to. A choice of a heirarchyList to establish parent child relationship between the parent and multiple children; and an associatedDocumentList establishes multiple document identifiers in a peer relationship. The child provides the document identification of child relationship.