Open Applications Group - Standards Document

OAGIS 9 Naming and Design Rules Standard

Effective Date: September 30, 2005

With Adherence to the:
UN/CEFACT XML Naming and Design Rules
Draft 1.2_8 sep September 2005

Authors:
Garret Minakawa – Oracle Corporation
Satish Ramanathan – MRO Software
Michael Rowell - OAGi

Reviewers:
David Connelly – OAGi
Steffen Fohn, ADP
Kurt Kanaskie – Lucent Technologies
Michelle Vidanes – STAR
Joe Zhou – Xtensible Solutions

Document Number: 060315-v.7

NOTICE

The information contained in this document is subject to change without notice.

The material in this document is published by the Open Applications Group, Inc. for evaluation. Publication of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, OPEN APPLICATIONS GROUP, INC. MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANT ABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Open Applications Group, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

For more information, contact:

Open Applications Group, Inc.
P.O. Box 4897
Marietta, Georgia 30061 USA
Telephone: +1 678 715 7588
Internet: http://www.openapplications.org

Table of Contents

OAGIS 9 Naming and Design Rules Standard

Abstract

The Open Applications Group Integration Specification (OAGIS) provides a canonical business language for vertical industries. Individual organizations and entire supply chains may further extend the specification in ways that meet their own unique needs. It is important for OAGi to define the naming, design rules and guidelines used for OAGIS in such a manner that these organizations may follow them for their extension.

This specification provides a means to identify, capture and maximize the re-use of business information expressed as XML Schema components within OAGIS and OAGIS extensions in order to support information interoperability across integrated environments.

Thank you to all who have contributed to the design, construction, and reviewing of the document. If we have missed anyone in our credits, we apologize to you.

This document will continue to grow as more details are added and updated.

1.0 INTRODUCTION

This "*OAGi – OAGIS 9.0 XML Naming and Design Rules Standard* (OAGIS NDR Standard)," defines the naming, design rules and guidelines that were applied by OAGi when developing the XML Schema instantiation of OAGIS 9.0. Since OAGIS 9.0 employs standards from other organizations this document defines how those standards are used and incorporated in OAGIS.

1.1 Support for UN/CEFACT Standards

OAGi supports UN/CEFACT standards where they exist and apply to OAGi standards. In terms of this document, the *OAGIS NDR Standard* the *UN/CEFACT ATG2 Naming and Design Rules (NDR)* applies.

As such this document will make numerous references to the UN/CEFACT NDR document.

1.2 Scope and Focus

This OAGIS NDR Standard can be employed wherever extensions to OAGIS 9.0 are to be made. They may also be employed in the design of other XML schema for defining the content of information exchange.

1.3 OAGi Approach

OAGi uses a unique approach to standards from most other organizations. OAGi works with other standards organizations both horizontal and vertical in nature. In doing this OAGi avoids the not-invented here syndrome that most organizations fall into.

Additionally OAGi focuses on being technology sensitive but not technology specific. This means that OAGIS can be used equally well with either Service Oriented Architecture (SOA) environment (ebXML or Web Services) or Message-Oriented-Middleware (MOM).

Having eleven years experience defining content to enable integrations in a SOA or SOA like environment provides OAGi the experience and expertise simply not available in other organization. Add to this eleven years worth of content that any other organization would have to build. OAGi has the experience and content needed for integrating business applications, today.

1.4 Terminology and Notation

The key words, "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in Internet Engineering Task Force (IETF) Request For Comments (RFC) 2119.[1] Where ever xsd: appears it is references to constructs from W3C XML schema specification. Where ever ccts: appears it is references to constructs from CCTS

The following are notations that are used throughout this document:

- Example – A representation of a definition or rule that are intended to be informative.
- [Note] – Explanatory information that is intended to be informative.
- [UN/CEFACT R n] – Denotes the identification of a rule that comes from the UN/CEFACT ATG2 NDR document that requires conformance.
- [OAGi R n] – Identifies a rule that is specified by this document that requires conformance.

  Where a UN/CEFACT rule exists a corresponding OAGi rule will be provided that references the UN/CEFACT rule and indicates OAGi's conformance. If OAGi does not comply with the UN/CEFACT rule the alternative that OAGi uses will be provided.

  [Note] Rules are normative. In order to ensure continuity across versions of the specification, rule numbers that are deleted will not be re-issued and any new rules will be assigned the next higher number regardless of the location.
- When defining rules the following annotations are used:
  - [ ] – Optional
  - < > - Variable
  - | - Choice
- `Courier` – All words in **`bolded courier`** font are values, objects or keywords.

[1] *Key words for use in RFCs to Indicate Requirement Levels*. Internet Engineering Task Force, Request for Comments: 2119, March, 1997, http://www.ietf.org/rfc/rfc2119.txt

1.5 Related Documents

UN/CEFACT Core Components Technical Specification, Part 8 of the ebXML Framework Version 2.01

UN/CEFACT XML Naming and Design Rules, Draft 1.12_14jul 15 July 2005

W3C XML Schema Part 1: Structures. W3C Recommendation, 2 May 2001

W3C XML Schema Part 2: Datatypes. W3C Recommendation, 2 May 2001

1.6 Guiding Principles

The guiding principles for this document extend the guiding principles defined in the UN/CEFACT NDR Guiding Principles section [2] by adding the following as the basis for all the design rules contained in this document.

- Conformance to the UN/CEFACT NDR document, where practically possible.
    - Where it is not practically possible to conform to the UN/CEFACT NDR, this document provides rules to define a more practical solution.
- Ensure the practical application of XML Schema in OAGIS such that it is implementable today.
- Simplify the use of OAGIS by further defining the naming and design rules used.

1.7 Conformance

Applications will be considered to be in full conformance with this technical standard if they comply with the content of the normative sections, rules and definitions.

[OAGi R 1]
[UN/CEFACT R 1] Applications will be considered to be in full conformance with this technical specification if they comply with the content of sections, Rules and definitions. [2]
OAGi adopts this rule with editorial changes only.
Applications will be considered to be in full conformance with this technical standard if they comply with the content of Appendix, Naming and Design Rules List.

2.0 OAGIS XML CONSTRUCTS

[2] UN/CEFACT XML Naming and Design Rules, Draft 1.2 8 September 2005

This section defines rules related to XML constructs that OAGIS uses. These rules include:

- Relationship to other standards
- Naming and Modeling Rules
- Reusability Scheme
- Modularity Model
- Namespace Scheme
- Versioning Scheme

2.1 Relationship to other standards

As indicated earlier OAGIS 9.0 includes references and makes use of other standards. This is accomplished in such a way that these other standards provide OAGIS users the greatest level of reuse of existing standards, while also minimizing the impact of these standards on OAGIS itself. The following is a list of the standards included in OAGIS 9.0:

- W3C - URI/URL
- W3C - XML Schema 1.0 Part 1
- W3C - XSL Schema 1.0 Part 2
- W3C - XML Style Language
- W3C - XML Path Language (XPath) Version 1.0
- ISO - ISO11179-5 Specification and standardization of data elements -- Part 5: Naming and identification principles for data elements
- ISO - ISO1500-5 Core Components Technical Specification – Also known as UN/CEFACT Core Component Technical Specification - CCTS
- ISO - ISO4217 - Currency Codes
- ISO - ISO639 - Language Codes
- MIME Media Type Code
- UNECE Unit Code
- UN/CEFACT ATG2 Naming and Design Rules – NDR

- UN/CEFACT Harmonized Core Components – TBG17
- ISO - ISO20022 (UNIFI Financial Standard) – IST Harmonization
- Object Management Group (OMG) – Unified Modeling Language (UML)

2.1.1 XML Core Technologies

OAGi has determined that the World Wide Web Consortium (W3C) XML schema definition (XSD) language is the generally accepted schema language. Therefore, all OAGi content specifications are expressed in XSD. All references to XML schema will be as XSD Schema or OAGIS XSD Schema or OAGIS XML Schema.

[OAGi R 2]

[UN/CEFACT R 2] All UN/CEFACT XSD Schema design rules MUST be based on the W3C *XML Schema Recommendations: XML Schema Part 1: Structures* and *XML Schema Part2: DataTypes*.[2]

OAGi adopts this rule with editorial changes only.

All OAGi XSD Schema or OAGIS Overlay Schema design rules MUST be based on the W3C *XML Schema Recommendations: XML Schema Part 1: Structures* and *XML Schema Part2: DataTypes*.

The W3C is the recognized source for XML specifications. W3C specifications may hold various states or status. Only W3C specifications with a status of recommended are guaranteed by the W3C to be stable.

[OAGi R 3]

[UN/CEFACT R 3] All UN/CEFACT XSD Schema and UN/CEFACT conformant XML instance documents MUST be based on the W3C suite of technical specifications holding recommendation status.

OAGi adopts this rule with editorial changes only.

All OAGi XSD Schema and OAGIS Overlay Schema and corresponding conformant XML instance documents MUST be based on the W3C suite of technical specifications holding recommendation status.

In order to maintain a consistent form in all of the OAGIS XSD Schemas, each needs to use a standard structure for all content. This standard structure is contained in *Schema File Structure* appendix in this document.

[OAGi R 4]

[UN/CEFACT R 4] UN/CEFACT XSD Schema MUST follow the standard structure defined in Appendix B.

OAGi further constrains this rule.

All OAGi XSD Schemas MUST follow the standard structure defined in the *Schema File Structure* appendix in this document. OAGIS Overlay schemas used to extend OAGIS MUST use this same structure.

2.1.2 Core Component Technical Specifications - CCTS

OAGi's implementation of the Core Component Technical Specification (CCTS) conforms with the approach described in UN/CEFACT NDR section *5.2 Relationship to the CCTS*. [2]

This means that the OAGIS 9.0 uses CCTS to represent the context neutral and context specific building blocks. A context neutral core component is "a building block for the creation of a semantically correct and meaningful information exchange package. It contains only the information pieces necessary to describe a specific concept." These neutral core components are then instantiated as context specific components for message assembly and model harmonization. These context specific components are defined as Business Information Entities (BIEs).

From this the design rules are coupled with CCTS in that:

- The message assembly is represented as a **`xsd:complexType`**.definion and element declaration in an XSD Schema. The element declaration is based on **`xsd:complexType`** that represents the document level ABIE. A global element appears in and is designated as the root element of a conformant XML instance.

- An ABIE is defined as a **`xsd:compleType`**.

- Depending upon the type of association an Association Business Information Entiity (ASBIE) will be declared as either a global element, if the ASBIE represents a composition, or as a local element when the ASBIE is not a composition, within the **`xsd:complexType`** representing the ABIE. The ASBIE element itself is based on the **`xsd:compleType`** of the associated ABIE. In this way the content model of the associated ABIE is represented in the XSD Schema instantiation.

  Note:

  Per CCTS, an ABIE can contain other ABIEs in ever higher levels of aggregation. When an ABIE contains another ABIE , this is accomplished by using an ASBIE. Where the ASBIE is the linking mechanism that shows hierarchical relationships between the ABIE constructs. When an ASBIE is used it referred to as the associating ABIE and the ABIE that it represents as the associated ABIE.

- A Basic Business Information Entity (BBIE) is declared as a local element or a local attribute within the xsd:complexType representing the parent ABIE. The BBIE is based on a qualified or unqualified data type (DT).

- A data type (DT) is defined as either a **`xsd:complexType`** or **`xsd:simpleType`**. DT's are based on Core Component Type **`xsd:complexType`** from the CCT schema module. These data types can be unqualified (no additional restrictions above those imposed by the CCT type) or qualified (additional restrictions above those imposed by the CCT type). XSD built-in data types will be used whenever the facets of the built-in data type are equivalent to the CCT supplementary components for that data type.

> **Note:**
>
> Data Types are not derived from the CCT complex types using **`xsd:restriction.`** Whereas all CCTs are defined as complex types with attributes representing their supplementary components, in several cases we leverage built-in **`xsd:simpleType`** whose facets correspond to the supplementary components.

2.1.3 UN/CEFACT ATG2 Naming and Design Rules – NDR

This document embraces and extends the UN/CEFACT Naming and Design Rules (NDR) document by identifying the how OAGIS 9.0 uses the UN/CEFACT NDR and other standards. This standard is provided for others to follow so as to consistently extend OAGIS in their own Overlay extensions. As well as, enabling tools vendors to design and code their applications to take advantage of OAGIS to share information in an open manner.

2.1.4 UN/CEFACT Harmonized Core Components – TBG17

OAGi has committed to use the Harmonized Core Components as they are approved by UN/CEFACT TBG 17. OAGIS 9.0 incorporates Core Components approved from TBG 17, as well as those that are proposed. OAGi incorporates approved components into OAGIS Components by making use of them directly as provided or by using them as a basis of an extended OAGIS ABIE. OAGi also provides those that are considered by TBG 17 to be unstable such that they maybe used by organizations looking to extend OAGIS Components.

At the time of publication for OAGIS 9.0 the list of TBG 17 Core Components is below along with an indication of the Core Components used by OAGIS 9.0.

Table 2-1 List of TBG 17 Core Components

TBG 17 Core Components	Used in OAGIS
AllowanceCharge	X

Authorization	X
PaymentAuthorization	X
Calculation	X
Communication	X
Contact	X
Dimension	X
CurrencyExchange	X
HazardousMaterial	X
Location	X
PaymentTerms	X
Period	X
Person	X
Price	X
TemperatureRange	X
Status	X
Tax	X
Preference	X
Temperature	X
Project	X
CountrySubDivision	
Country	
Range	X
GeographicalCoordinate	
Address	
Account	
BusinessProfile	
Card	
Charge	
CompletedWork	
Condition	
Consignment	
Construction	
Contract	
DangerousGoods	
DeliveryTerms	
Document	
Event	
ExaminationResult	
GoodsDescription	
GoodsItem	
Guarantee	
Instructions	
Metrics	
Organization	
Party	
PartyMeans	
Payment	

Penalty
Process
ProductItem
Qualification
Registration
Route
Service
Staff
TaxCategory
TechnicalCapability
TradeTerms
WorkCapability

2.1.5 ISO20022 (UNIFI Financial Standard) – IST Harmonization

ISO20022 – IST Harmonization is a joint initiative of OAGi, IFX, SWIFT and TWIST. The initiatives purpose is to define a standard set of interactions between corporations and banks and to capture these standards in a repository that can be found at www.iso20022.org. At the time of publication for OAGIS 9.0, this repository consists of two XML Schema standards:

- CoreCreditTransferInitiation ($pain.001.001.01.xsd) – corporate to bank payment initiation message (credit transfer)
- PaymentInitiationStatus ($pain.002.001.01.xsd).- bank to corporate payment initiation status message.

OAGi incorporates these IST standards into OAGIS 9.0 by providing Nouns and Business Object Documents (BODs) that make use of these schema documents by directly importing and using the component definitions of the IST group. OAGi codifies equivalent BODs, Nouns, and Components in the OAGIS library providing a consistent approach to reuse of these standards througout OAGIS 9.0.

2.1.6 OMG UML

OAGi uses UML to model OAGIS content and business interactions. This is done in accordance to the UN/CEFACT UMM. OAGi uses UML Class diagrams to model the content. UML Sequence and Collaboration Diagrams are used to model the business interactions in the OAGIS Scenarios. The Sequence and Collaboration Diagrams can then be used as the basis for UML Activity Diagrams that fully capture the actual implementation.

The Sequence and Collaboration diagrams are provided as part of the documentation for OAGIS. It is the responsibility of the implementers to use these as the basis of the Activity Diagram to capture the resulting integrated system. As the detail of each specific integration is unique.

This documentation is added as of the OAGIS 9.0.1 release.

2.2 Naming and Modeling Rules

OAGIS XML Schema are derived from CCT, CCTS, and UMM process modeling and data analysis. The OAGIS library contains conformant CCT and CCTS dictionary entry names as well as truncated XML element names that are conformant with the naming constraint rules that follow. The qualified XPath ties the information to its standardized semantics as described by the underlying CCTS, while the XML element or attributes names are a truncation that reflects the hierarchy inherent in the XML construct. This implies that a part of the fully qualified XPath will represent the CCTS dictionary entry name of the corresponding ABIE, BBIE, ASBIE or DT.

[OAGi R 5]

[UN/CEFACT R 5] Each element or attribute XML name MUST have one and only one fully qualified XPath (FQXP).

OAGi adopts this rule without modification.

For example: `Communication/Address/StreetName`

The official language for OAGi is English. Therefore, all official XML constructs are published by OAGi in English. XML development work may occur in other languages; however submissions for inclusion in the OAGIS library must be in English. Other language translations of OAGi publications are at the discretion of the users.

[OAGi R 6]

[UN/CEFACT R 6] Element, attribute and type names MUST be composed of words in the English language, using the primary English spellings provided in the Oxford English Dictionary.

OAGi adopts this rule without modification.

Lower Camel Case capitalizes the first character of each word except the first word and compounds the name (i.e. removes all white space). Upper Camel Case capitalizes the first character of each word and compounds the name. OAGi uses Lower Camel Case (LCC) for naming attributes and Upper Camel Case (UCC) for naming elements and types.

[OAGi R 7]

[UN/CEFACT R 7] Lower camel case (LCC) MUST be used for naming attributes.

OAGi adopts this rule without modification.

Example of an attribute: `<xsd:attribute name="unitCode" …>`

[OAGi R 8]

[UN/CEFACT R 8] Upper camel case (UCC) MUST be used for naming elements and types.

OAGi adopts this rule without modification.

Example of an element: `<xsd:element name="LanguageCode">`

Example of a type: `<xsd:complexType name="CodeType">`

[OAGi R 9]

[UN/CEFACT R 9] Element, attribute and type names MUST be in a singular form unless the concept itself is plural.

OAGi adopts this rule without modification.

Example of Singular and Plural concept forms:

Singular – Allowed: `<xsd:element name="GoodsQuantity" …>`

Plural – Not Allowed: `<xsd:element name="ItemsQuantity" …>`

[OAGi R 10]

[UN/CEFACT R 10] Element, attribute and type names MUST be drawn from the following set: `a – z` and `A – Z`.

OAGi adopts this rule without modification.

Example of Non-Letter Characters – Not Allowed

```
<xsd:element name="LanguageCode8" …>
```

XML 1.0 specifically prohibits the use of certain reserved characters in XML tag names. These include periods, spaces, and other separators.

[OAGi R 11]

[UN/CEFACT R 11] XML element, attribute and type names constructed from dictionary entry names MUST NOT include periods, spaces, or other separators; or characters not allowed by W3C XML 1.0 for XML names.

OAGi adopts this rule without modification.

Example of Spaces in Name – Not Allowed

```
<xsd:element name="Customized_ Language. Code:8" …>
```

[OAGi R 12]

[UN/CEFACT R 12] XML element, attribute and type names MUST NOT use acronyms, abbreviations, or other word truncations except those included in the UN/CEFACT controlled vocabulary or listed in Appendix C.

OAGi relaxes this rule.

XML element, attribute and type names MUST NOT use acronyms, abbreviations, or other word truncations except those included in the UN/CEFACT controlled vocabulary, listed in Appendix C of the UN/CEFACT NDR document or in the *Appendix E - OAGi Accepted Acronyms and Abbreviations* in this document.

[OAGi R 13]

[UN/CEFACT R 13] The acronyms and abbreviations listed in Appendix C MUST always be used.

OAGi adopts this rule with editorial changes only..

The acronyms and abbreviations listed in Appendix OAGi Acronyms and Abbreviations MUST always be used.

[OAGi R 14]

[UN/CEFACT R 14] Acronyms and abbreviations at the beginning of an attribute declaration MUST appear in all lower case. All other acronyms and abbreviation usage in an attribute declaration must appear in upper case.

OAGi adopts this rule without modification.

[UN/CEFACT R 15] Acronyms MUST appear in all upper case for all element declarations and type definitions.

OAGi adopts this rule without modification.

Example Acronyms and Abbreviations

ID is an allowed abbreviation: `<xsd:element name="ID">`

Cd is not an approved abbreviation : `<xsd:element name="ReasonCd">`

2.2.1 Module Naming

In order to ease implementation it is critical that the name of the schema modules be consistent across platforms. For this reason OAGi uses the same Upper Camel Case naming convention described above for the name of schema modules. For example a

Purchase Order schema is name PurchaseOrder. This avoids using white space that may be represented differently on different systems.

[OAGi R 15] Upper camel case (UCC) MUST be used to name schema modules.

2.3 Reusability Scheme

OAGi like UN/CEFACT is committed to an object based approach for its process models and core component implementation as supported by both UMM and CCTS. A type based approach for XML management provides the closest alignment with the process modeling methodology in UMM. Type information is now accessible when processing XML instance documents. Post schema validation infoset (PSVI) capabilities are emerging that support this approach. For example “data-binding” software that compiles schema into ready-to-use object classes that are capable of manipulating XML data based on their types and structure.

The most significant issue to a type based approach is the risk of developing an inconsistent element vocabulary where elements are declared locally and allowed to be reused without regard to semantic clarity and consistency across types.

In order to avoid this OAGi and UN/CEFACT recommend creating a consistent element vocabulary such that when an element is bound to a type that binding persists across the namespace in which the binding is defined. The result of this is that every element is uniquely named. As a result of this requirement OAGIS 9.0 uses a primarily all global element i.e. Garden of Eden XML Schema Design Pattern.

While it is possible to accomplish this using the Garden of Eden XML Schema Design Pattern, which indicates that all elements are defined globally with globally defined types. Or by using the Ventian Blind XML Schema Design Pattern, which indicates all elements other than the root element is defined locally using globally defined types. Neither of these design patterns communicates the information captured in the Model that the schemas are based upon.

To address these requirements OAGi and the UN/CEFACT recommend using the Hybrid XML Schema Design Pattern but do not make it a requirement. While enforcing the requirement that the element names be unique within the given namespace whether they are declared locally or globally.

The Hybrid XML Schema Design provides benefits over a pure type based approach. Most significantly it increases reusability of a library of content both at a modeling and XML Schema level. For more information about the Hybrid XML Schema Design Pattern please see the *Hybrid XML Schema Desigin Pattern – Position Paper* from the Open Applications Group.

The key principles of the “hybrid approach” are:

1. Global types and elements are used to represent reusable constructs that have sufficient semantics independent of the context in which they are used.

2. Local types and elements are used to represent constructs that are only meaningful within a specific context.

3. All classes are expressed as complexTypes in the XML Schema.

4. All attributes of a class are declared as local xsd:element within the corresponding xsd:complexType.

5. Classes associated through aggregation (e.g. Party, BuyerParty in figure 1 below) are globally declared as an xsd:element and referenced in the aggregating element.

6. Classes associated through composition (e.g. PurchaseOrderHeader and PurchaseOrderLine in figure 1) are locally declared as xsd:element elements within the xsd:complexType of the PurchaseOrder. A Composition ASBIE is defined as a specialized type of ASBIE that represents a composition relationship between the associating ABIE and the associated ABIE.

7. Generalization associations indicate classes that inherit the source class. This is represented in XML Schema using complexType derivation by extension.

Due to the advantages of the Hybrid XML Schema Design Pattern OAGIS will implement this design pattern in a future release of OAGIS. OAGIS is able to transition to the Hybrid XML Schema Design Pattern without affecting compatibility as descrbied in the Versioning section of this document.

[OAGi R 16]

For each ABIE, a named **`xsd:element`** MUST be globaly declared.

[OAGi R 17]

For each ABIE, a named `xsd:complexType` MUST be globaly declared.

[OAGi R 18]

For each attribute of an object class (BBIE) identified in an ABIE, a named **`xsd:element`** MUST be locally declared within the **`xsd:complexType`** representing that ABIE.

[OAGi R 19]

For each ASBIE whose **`ccts:AssociationType`** is Composition, a named **`xsd:element`** MUST be locally declared within the **`xsd:complexType`** representing the associating ABIE.

[OAGi R 20]

For each ASBIE whose **`ccts:AssociationType`** is not Composition, a **`xsd:element`** MUST be globally declared.

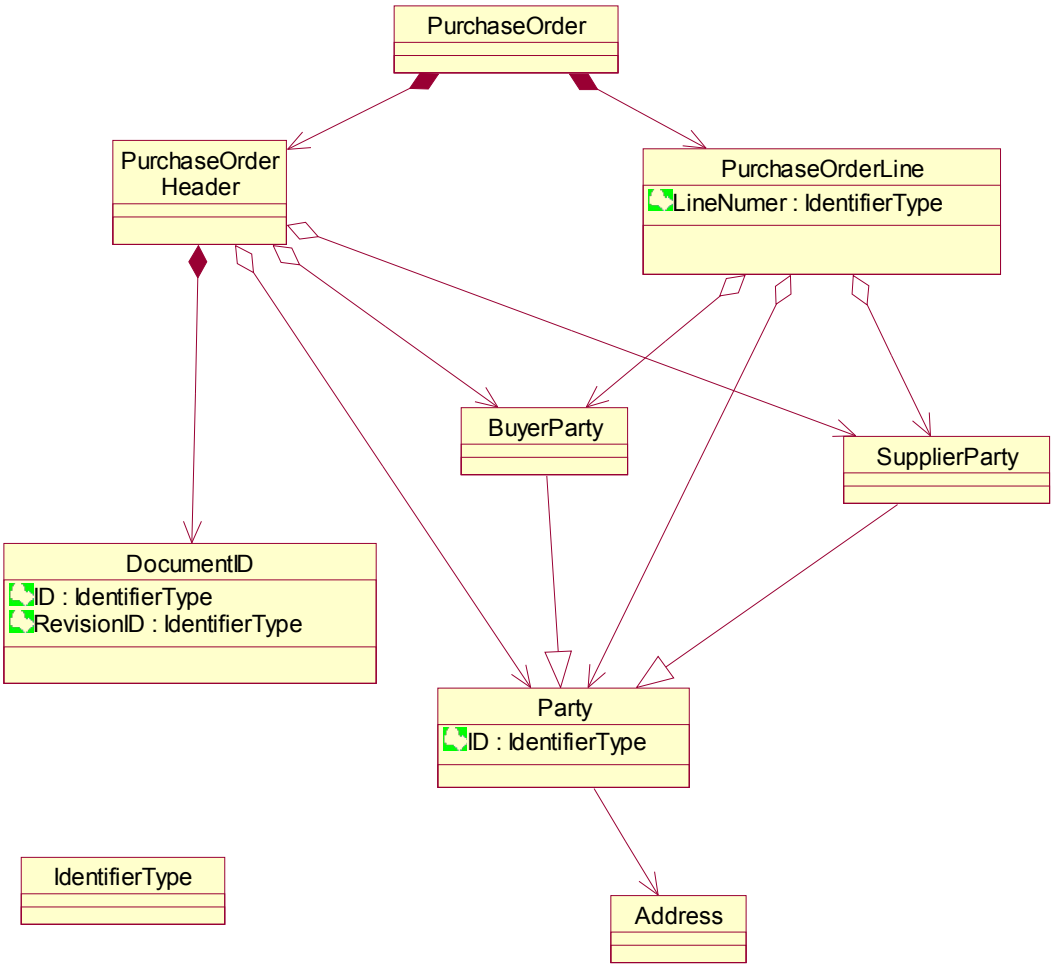


Figure 1 – UML Model of a PurchaseOrder

Figure 1 shows a UML representation of a very simple model of a PurchaseOrder. In this example the PurchaseOrder contains two composite ABIEs the PurchaseOrderHeader and the PurchaseOrderLine.

The PurchaseOrderHeader has an additional composite association DocumentID and aggregations to the Party objects BuyerParty, SellerParty, and Party.

The DocumentIDType is further defined by the ID, RevisionID, and VariationID each of which are defined by the CCTS DataType IdentifierType.

The PurchaseOrderLine is defined by a LineNumber that is defined by the CCTS DataType IdentifierType and aggregations to the Party objects BuyerParty, SellerParty.

By applying the rules for the Hybrid XML Schema Design Pattern to the UML Model in Figure 1 results in the sample XML schema code provided in Figure 2. In this schema code sample it is possible to identify the Objects ABIEs and the Composite ABIEs from the sematic context of the Purchase Order.

Composite associations are realized by using XML Schema local elements. The associations to other objects are realized by referencing the global elements for the given object. Further more the classes are realized by using XML Schema xsd:complexType and/or xsd:simpleType.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.openapplications.org"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.openapplications.org"
elementFormDefault="qualified" attributeFormDefault="unqualified">
      <xsd:element name="PurchaseOrder" type="PurchaseOrderType"/>
      <xsd:complexType name="PurchaseOrderType">
            <xsd:sequence>
                  <xsd:element name="PurchaseOrderHeader"
type="PuchaseOrderHeaderType"/>
                  <xsd:element name="PurchaseOrderLine"
type="PurchaseOrderLineType"/>
            </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="PuchaseOrderHeaderType">
            <xsd:sequence>
                  <xsd:element name="DocumentID" type="DocumentIDType"/>
                  <xsd:element ref="BuyerParty"/>
                  <xsd:element ref="SupplierParty"/>
                  <xsd:element ref="Party"/>
            </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="PurchaseOrderLineType">
            <xsd:sequence>
                  <xsd:element name="LineNumber" type="IdentifierType"/>
                  <xsd:element ref="BuyerParty"/>
                  <xsd:element ref="SupplierParty"/>
                  <xsd:element ref="Party"/>
            </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="SupplierParty" type="SupplierPartyType"/>
      <xsd:complexType name="SupplierPartyType">
            <xsd:complexContent>
                  <xsd:extension base="PartyType"/>
            </xsd:complexContent>
      </xsd:complexType>
      <xsd:element name="BuyerParty" type="BuyerPartyType"/>
      <xsd:complexType name="BuyerPartyType">
            <xsd:complexContent>
```

```
                        <xsd:extension base="PartyType"/>
                  </xsd:complexContent>
            </xsd:complexType>
            <xsd:element name="Party" type="PartyType"/>
            <xsd:complexType name="PartyType">
                  <xsd:sequence>
                        <xsd:element name="ID" type="IdentifierType"/>
                        <xsd:element ref="Address"/>
                  </xsd:sequence>
            </xsd:complexType>
            <xsd:element name="Address" type="AddressType"/>
            <xsd:complexType name="AddressType">
                  <xsd:sequence/>
            </xsd:complexType>
            <xsd:complexType name="DocumentIDType">
                  <xsd:sequence>
                        <xsd:element name="ID" type="IdentifierType"/>
                        <xsd:element name="RevisionID" type="IdentifierType"
minOccurs="0"/>
                  </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="IdentifierType">
                  <xsd:simpleContent>
                        <xsd:extension base="xsd:normalizedString"/>
                  </xsd:simpleContent>
            </xsd:complexType>
</xsd:schema>
```

Figure 2 – XSD Schema Definition of a Purchase Order.

2.4 Modularity Model

Modules can be defined unique in their functionality, or represent splitting of larger schema files for performance of manageability. A modularity model provides an efficient and effective mechanism for importing components as needed rather than dealing with complex, multi-focused schema.

2.4.1 UN/CEFACT Modularity Model

UN/CEFACT has defined several types of schema modules that support this approach. Figure 2-1 shows the CEFACT modularity model. The schema modules are categorized into message assembly and external schema. The message assembly modules include a root schema and internal schema modules that reside in the same namespace as the root schema. The external schema modules consist of a set of reusable schema for ABIEs, unqualified data types, qualified data types, and code lists. Each of these schema modules reside in their own namespace. Dependencies exist as shown in the figure between the various modules. It is important to note that the modularity model has been designed such that there are no circular includes or imports.

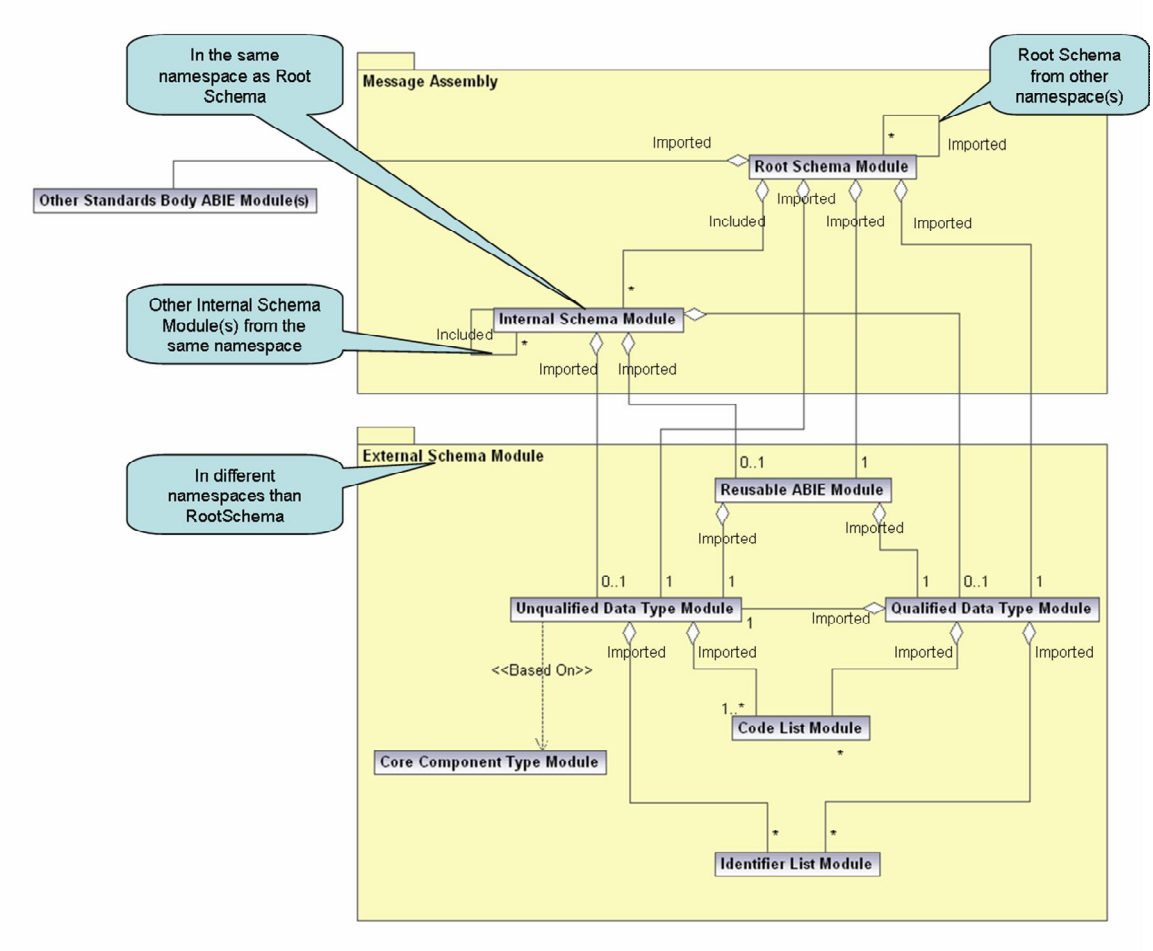


Figure 3 - UN/CEFACT XSD Schema Modularity Scheme

Note: Figure 3 is an OAGi depiction of the UN/CEFACT NDR Schema Modularity Scheme figure 5-5.

2.4.2 OAGIS Schema Modularity

In OAGIS 9.0, OAGi introduces the concept of Developer BODs and Standalone BODs. Each serves a different purpose. The Developer BODs are intended to maintain the schema modularity and the ability to reuse existing components as need without redefining them. This is the same principle expressed in both OAGIS 8.0 and in the UN/CEFACT NDR Schema Modularity. The Standalone BODs are intended to enable implementations. Many tools available today have difficulty working with schemas that modularize the content into different schema files. For these reasons OAGi provides both the Developer and Standalone BODs that have the same content. The Standalone BODs contain everything that a given BOD uses from the OAGIS 9.0 namespace that it uses. The Developer BODs include the other schemas to obtain the common components that are needed.

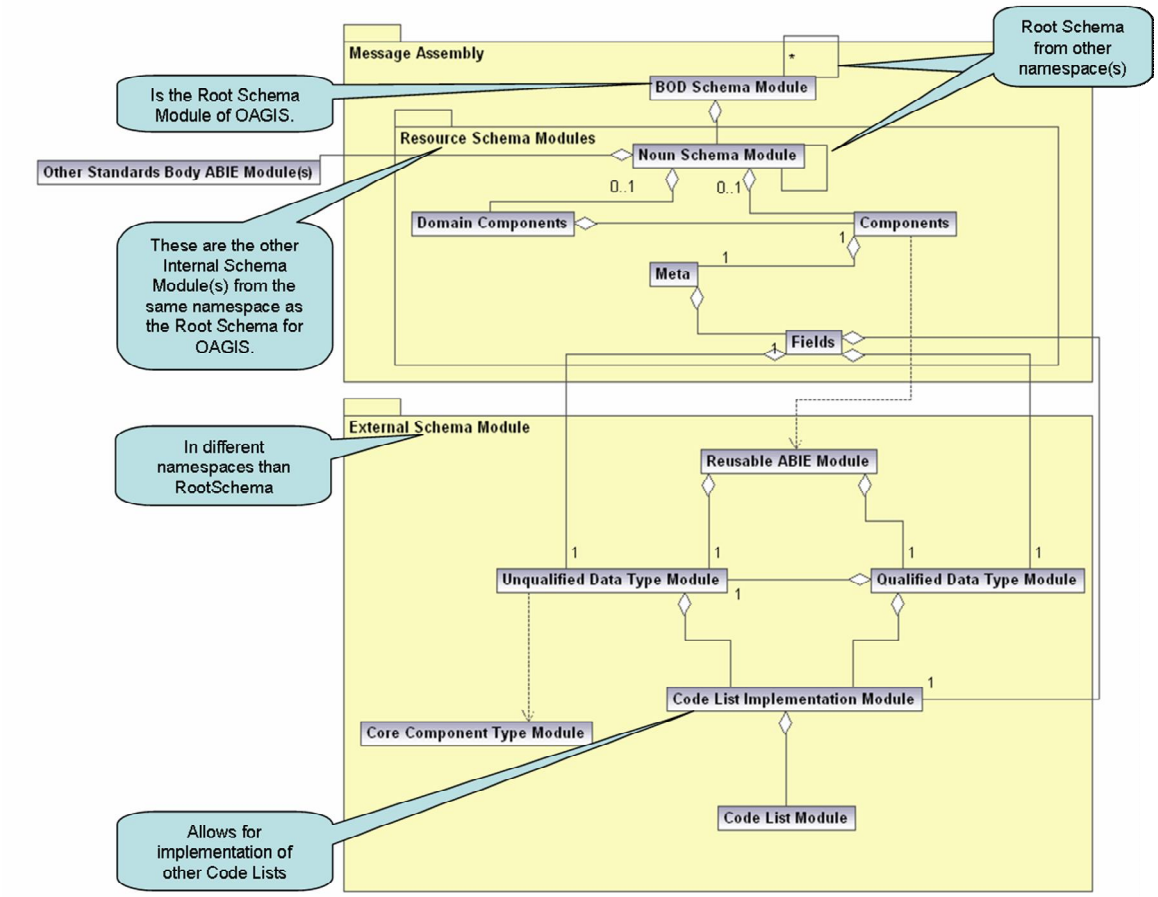


Figure 4 - OAGIS Developer XSD Schema Modularity Scheme

The Developer BODs are what OAGi uses to develop the BODs and should be used by those interested in extending OAGIS using an Overlay. The Developer BODs should also be used for those that have tools that are XML Schema compliant enough to utilize the modular nature of XML Schema that is necessary to achieve the modularity scheme recommended by UN/CEFACT and a model driven approach to XML Schema.

The Standalone BODs are used only in deploying an implementation. Only if the tools and applications in the implementation are not XML Schema compliant enough to utilize the modular nature of XML Schema.

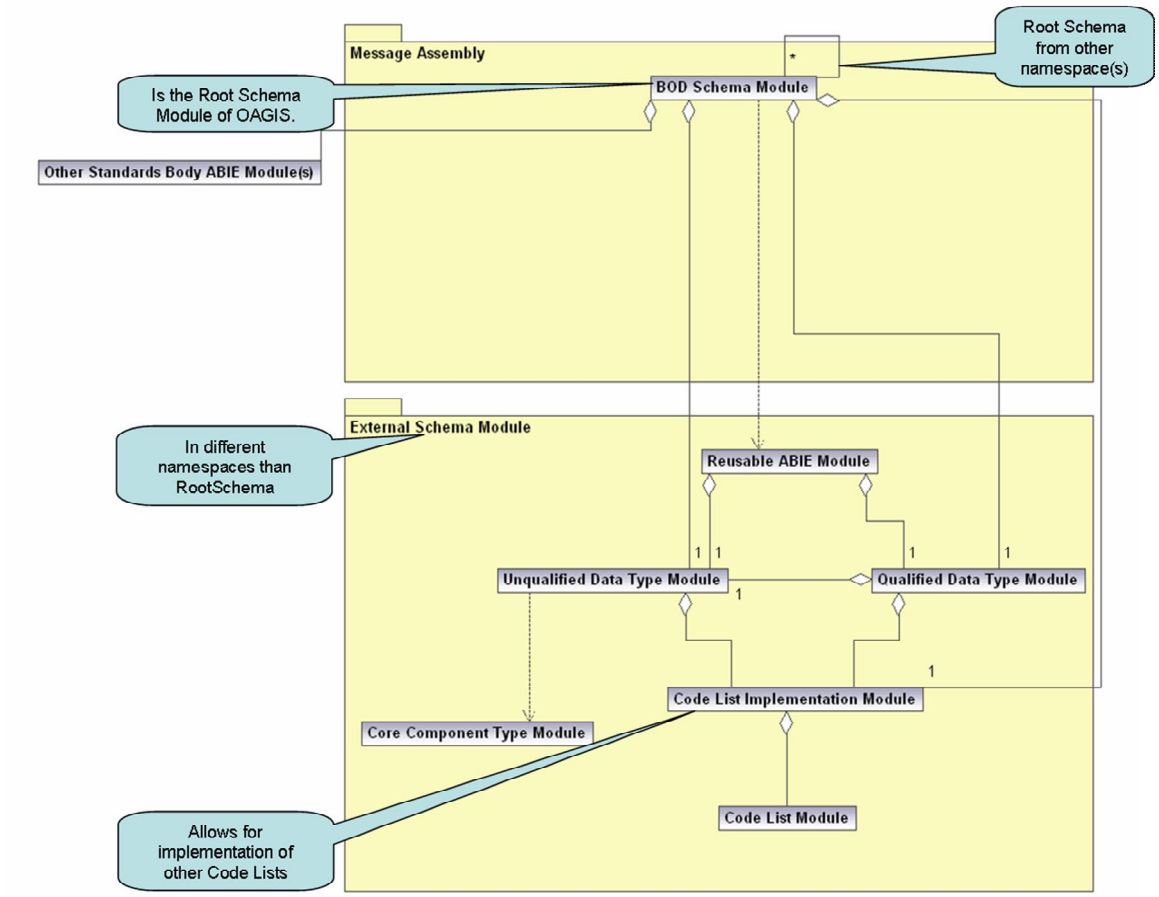


Figure 5 – OAGIS Standalone XSD Schema Modularity Scheme

In both the Developer or the Standalone BODs the relationship to the schema modules identified by UN/CEFACT are the same.

- The BOD schema module plays the role of the UN/CEFACT root schema module. It always includes any internal schemas residing in its namespace. It may import root schemas from other namespaces as well as reusable schemas from other standards bodies.

- The OAGIS Noun, Components, Fields and Meta schema modules play the role of the UN/CEFACT internal schema modules. The Fields schema module imports the unqualified data type, and qualified data type. The Components schema module imports the reusable ABIE schema modules.

- The core component type schema modules are provided as references to the different external schema modules. Each in their own namespaces

637 The difference in the Developer and the Standalone BODs comes down to the presence of
638 the OAGIS Noun, Components, Fields, and Meta files or the Internal Schema modules.
639 This difference is depicted graphically in Figures 4 and 5.

640 Each of which is compliant with the UN/CEFACT NDR since the Internal Schema Modules
641 may have zero to unbounded includes.

Note:

In order to be consistently understood, the remainder of this document will use the following schema module names and tokens.

Table 1 Schema Module and Token

Schema Module Name	Token
RootSchema	rsm
CCTS/CCT	cct
UN/CEFACT Reusable Aggregate Business Information Entity	ram
UN/CEFACT Unqualified Data Type	udt
UN/CEFACT Qualified Data Type	qdt
CodeList	clm
Identifier List	ids
Open Applications Group Integration Standard	oa
OAGIS BODs	bod
OAGIS Components	oac
OAGIS Fields	oaf
OAGIS Nouns	oan
OAGIS Code Lists	oacl

642

643 **Note:** OAGIS uses the names of the schema module using upper camel case as the
644 names of the XML Schema files. All of OAGIS is defined in a single OAGIS namespace
645 other than the OAGIS Code List assembly, which is defined in a second namespace.

646 2.4.3 BOD – Root Schema

647 OAGi incorporates the modularity model as described above. There are over four hundred
648 OAGIS BOD root schema in OAGIS 9.0, each of which express a separate business

function. Add to this the vertical extension of OAGIS that exist by the different vertical industry groups and there are many more BODs defined than just those defined by OAGIS itself.

[OAGi R 21]

[UN/CEFACT R 16] A root schema MUST be created for each unique business information exchange.

OAGi adopts this rule with editorial changes only.

A BOD, root schema, MUST be created for each unique business information exchange.

The modularity approach enables the reuse of an individual BOD with out having to import the entire OAGIS BOD library. Additionally, a BOD schema can include individual modules without having to include the entire OAGIS library. This is applies both within the OAGIS and for Overlays of OAGIS. Each BOD defines its own dependencies. A BOD root schema should not duplicate reusable XML constructs contained in other schema; instead it should reuse existing constructs where they exist. Specifically, BOD root schema will import or include other schema modules to maximize reuse through **`xsd:include`** or **`xsd:import`** as appropriate.

[OAGi R 22]

[UN/CEFACT R 17] A root schema MUST NOT replicate reusable constructs available in schema modules capable of being referenced through **`xsd:include`** or **`xsd:import`**.

OAGi adopts this rule with editorial changes only.

A Developer BOD, root schema, MUST NOT replicate reusable constructs available in schema modules capable of being referenced through **`xsd:include`** or **`xsd:import`**. A Standalone BOD must reference reusable constructs only through **`xsd:import.`**

Schema modules used by the BOD schema are treated as either internal or external schema modules so that correct namespace decisions are made.

[OAGi R 23]

[UN/CEFACT R 18] UN/CEFACT XSD schema modules MUST either be treated as external schema modules or as internal schema modules of the root schema.

OAGi adopts this rule with editorial changes only.

The schema modules MUST be treated as external schema modules or as internal schema modules of any OAGi or OAGi Overlay BOD schema module.

OAGIS BOD modules include the corresponding Noun schema module which defines the reusable constructs needed. This is done as indicated above through either the use of an **`xsd:include`** or **`xsd:import`**.

[OAGi R 24]

OAGi BOD root schema modules MUST be named <VerbName><NounName>.

Where:

<VerbName> is the name of the Verb used by the BOD.

<NounName> is the name of the Noun used by the BOD.

[OAGi R 25] OAGi BOD root schema modules MUST include the Noun schema module that is indicated in the BOD Name. This is done by use of use of an **`xsd:include`** or **`xsd:import`**.

[OAGi R 26] The BOD module also defines the BOD root element that is the same as the name of the BOD. For example ProcessPurchaseOrder identifies the Verb Process and the Noun PurchaseOrder are used in this BOD.

The BOD root element makes use of a type that is named the same as the BOD with a postfix of Type. This BODNameType is a **`xsd:complexType`** and is based on the **`oa:BusinessObjectDocumentType`**, which it extends by adding a DataArea element.

The DataArea element uses a **`xsd:complexType`** named BODNameDataAreaType. The DataAreaType binds the Verb and the Noun.

The Verb identifies the intended processing that is to occur as a result of the BOD. The Noun identifies the object plus object attribute, action plus object or qualifier(s) plus object data that the process is to use. The object may also be considered a document as is the case in a PurchaseOrder.

[OAGi R 27]

OAGi BOD root schema module MUST define a root element that is named <VerbName><NounName>, this is also known as the <BODName>.

[OAGi R 28]

OAGi BOD root element MUST be define by a type that is named the same as the BOD root element name post fixed with the word “Type” of the form <BODName>Type.

Where:

<BODType> = <VerbName><NounName>Type

[OAGi R 29]

Each OAGi <BODType> MUST be based on the oa:BusinessObjectDocumentType defined by OAGi.

[OAGi R 30]

The <BODType> MUST extend the oa:BusinessObjectDocumentType by adding a local DataArea element.

[OAGi R 31]

OAGI BOD DataArea element must be defined by a type named <BODName>DataAreaType. This type must bind the Verb and Noun indicated in the <BODName>, by referencing them in an **`xsd:sequence`**.

2.4.4 Noun, Components, Fields, Meta - Internal Schema

Not all ABIEs will be applicable at a world-wide level. Some may be limited to a specific business function, vertical industry need, or to certain information exchange. Nor have all ABIE's needed been addressed by TBG17 at this time. Even after TBG17 is complete there are always new requirements for business that will require ABIEs that are not in the UN/CEFACT Core Components.

These ABIEs that are not part of the TBG17 Core Components are to be implemented in an internal schema module rather than in the reusable ABIE module. The UN/FACT NDR indicates that a schema may have zero or more internal modules. These internal schema modules will reside in the same namespace as their parent root schema. Being in the same namespace as the root schema they use an xsd:include to incorporate these internal schema modules. The modularity approach ensures that logical associations exist between root and internal schema modules and that individual modules can be reused to maximum extent possible.

The OAGIS Component library has always been designed with this in mind. The OAGIS Nouns, Components, Fields, Meta and CodeList schema modules play the role of the internal schema modules. These modules exist within the same namespace as the root schema modules the BOD schema module. In the case of an Overlay the schema may point to the corresponding OAGIS schema module in order to reuse existing constructs.

[OAGi R 32]

[UN/CEFACT R 19] All UN/CEFACT internal schema modules MUST be in the same namespace as their corresponding **`rsm:RootSchema`**.

OAGi adopts this rule with editorial changes only.

All internal schema modules (**`Nouns, Components, Fields, Meta`** modules) MUST be in the same namespace as their corresponding BOD root schema module.

OAGIS internal schema modules will identify the type of content in which they contain. For example Components module contains Components or ABIEs that maybe used across

many different BODs. Further more the location of these internal schema modules within the OAGIS repository further identify the scope in which they are used. For example:

The common Component schema module are located in: Resources/Components/Common/ along with the other common schema modules Meta, Fields and CodeLists.

- The financial Components schema module are located in: Resources/Components/Financial/.
- The operational Components schema module are located in: Resources/Components/Operational/.
- Similarly, all of the Nouns can be found in Resources/Nouns/.

[OAGi R 33]

[UN/CEFACT R 20] Each UN/CEFACT internal schema module MUST be named <ParentRootSchemaModuleName><InternalSchemaModuleFunction> Schema Module.

OAGi adopts the intent of this rule but modifies the actual implementation.

Each internal schema module MUST be named one of the following depending upon the modules function.

- The module containing the Noun MUST be named the same as the global element representing the Noun. Where the Noun identifies the object plus object attribute, action plus object or qualifier(s) plus object data that the process is to use. The object may also be considered a document.
- The module containing reusable Components MUST be named Components and depending upon the scope in which the components are applicable may be placed in an appropriate location.
- The module containing reusable Fields MUST be named Fields.
- The module containing constructs that are used for the design of the BOD Architecture MUST be named Meta.
- The module containing references to existing CodeLists that are external schema modules or define new CodeLists or extensions to existing CodeLists are to be named CodeLists.

2.4.5 External Schema

These schemas are identified as external because they reside in a different namespace from the BOD root schema. The BOD or internal schemas may import one or more of these external schema modules. The UN/CEFACT NDR has identified the need for the following external schema modules:

- Core Component Type

- Unqualified Data Type
- Qualified Data Type
- Reusable ABIE
- Code List
- Identifier List
- Other Standards Body ABIE module

2.4.5.1 Core Component Type Schema Module

The UN/CEFACT NDR requires that a schema module exists to represents the normative form of the CCTs from CCTS. This schema in turn is the basis of the UDT schema module. However, it is never to be imported directly into any schema module.

[OAGi R 34]
[UN/CEFACT R 21] A Core Component Type schema module MUST be created.
OAGi adopts this rule without modification.

The Core Component Type schema module will have a standard name that uniquely differentiates it from other schema modules.

OAGi implements this name different from the UN/CEFACT NDR because of the issue of consistently referencing files names with white spaces. Please see section *2.2.1 Module Naming*.

[OAGi R 35]
[UN/CEFACT R 22] The **`cct:CoreComponentType`** schema module MUST be named "UN/CEFACT Core Component Type Schema Module".
OAGi adopts the intent of this rule but modifies the actual implementation.
The **`cct:CoreComponentType`** schema module MUST be named Core Component Type Schema Module and be contained in the **`CoreComponentTypes.xsd`** file.

2.4.5.2 Unqualified Data Type Schema Module

A schema module is required to represent the normative form of the data types for each CCT as expressed in the CCTS meta model. These data types are based on the XSD constructs from the CCT schema module but where possible represent the builtin xsd:simpleType instead of their parent CCT xsd:complexType. Because of this the

unqualified data type schema module does not import the CCT schema module. The unqualified data types are so named because they contain no additional restriction on their source CCTs other than those define in CCTS and the agreed upon best practices. An unqualified data type is defined for all approved CCTS primary and secondary representation terms.

[OAGi R 36]

[UN/CEFACT R 23] An Unqualified Data Type schema module MUST be created.

OAGi adopts this rule without modification.

The unqualified data type schema module must have a standard name that uniquely differentiates it from other schema modules.

OAGi implements this name different from the UN/CEFACT NDR because of the issue of consistently referencing files names with white spaces. Please see section *2.2.1 Module Naming*.

[OAGi R 37]

[UN/CEFACT R 24] The **`udt:UnqualifiedDataType`** schema module MUST be named "UN/CEFACT Unqualified Data Type Schema Module".

OAGi adopts the intent of this rule but modifies the actual implementation.

The **`udt:UnqualifiedDataType`** schema module MUST be named "Unqualified Data Type Schema Module" and be contained in the **`UnqualifiedDataTypes.xsd`** file.

2.4.5.3 Qualified Data Type Schema Module

As data types are reused for different BIEs, restrictions on the data type may be applied. These restricted data types are referred to as qualified data types. These qualified data types will be defined in a separate qualified data type schema module. This qualified data type module will import the Unqualified Data Type Schema Module.

[OAGi R 38]

[UN/CEFACT R 25] A Qualified Data Type schema module MUST be created.

OAGi adopts this rule without modification.

The qualified data type schema module will have a standard name that uniquely differentiates it from other schema modules.

OAGi implements this name different from the UN/CEFACT NDR because of the issue of consistently referencing files names with white spaces. Please see section *2.2.1 Module Naming*.

[OAGi R 39]

852 [UN/CEFACT R 26] The **qdt:QualifiedDataType** schema module MUST be named
853 "UN/CEFACT Qualified Data Type Schema Module".

854 OAGi adopts the intent of this rule but modifies the actual implementation.

855 The **qdt:QualifiedDataType** schema module MUST be named "Qualified Data Type
856 Schema Module" and be contained in the **QualifiedDataTypes.xsd** file.

857 2.4.5.4 Reusable Aggregate Business Information Entity 858 Schema Module

859 A single reusable aggregate business information entity schema module is required.
860 This schema module contains a type definition for every reusable ABIE in the
861 UN/CEFACT Core Component Library. This module may be segmented into additional
862 modules in the future, if deemed necessary. This single reusable schema module may
863 be compresses for runtime performance considerations if necessary. In this case
864 compression means that a run time of the schema module would be created that
865 contains a subset of the ABIEs. This subset would consist only of the ABIEs necessary
866 to support the specific root schema being validated.

867 [OAGi R 40]

868 [UN/CEFACT R 27] A Reusable Aggregate Business Information Entity schema module
869 MUST be created.

870 OAGi adopts this rule without modification.

871 The reusable aggregate business information entity schema module will have a
872 standard name that uniquely differentiates it from other schema modules.

873 OAGi implements this name different from the UN/CEFACT NDR because of the issue
874 of consistently referencing files names with white spaces. Please see section *2.2.1*
875 *Module Naming*.

876 [OAGi R 41]

877 [UN/CEFACT R 28] The **ram:ReusableAggregateBusinessInformationEntity**
878 schema module MUST be named "UN/CEFACT Reusable Aggregate Business Information
879 Entity Schema Module".

880 OAGi adopts the intent of this rule but modifies the actual implementation.

881 The **ram:ReusableAggregateBusinessInformationEntity** schema module MUST
882 be named Reusable Aggregate Business Information Entity Schema Module" and contained
883 in a **ReusableAggregateBusinessInformationEntity.xsd** file.

2.4.5.5 Code List Schema Modules

When a code list is required or used, reusable code list schema modules will be created to minimize the impact of code list changes on BOD and other reusable schema modules. Each reusable code list schema module will contain enumerated values for the codes and code values.

[OAGi R 42]

[UN/CEFACT R 29] Reusable Code List schema modules MUST be created to convey code list enumerations.

OAGi adopts this rule without modification.

Code list schema modules must have a standard name that uniquely differentiates it from other schema modules.

[OAGi R 43]

[UN/CEFACT R 30] The name of each clm:CodeList schema module MUST be of the form: **<Code List Agency Identifier | Code List Agency Name><Code List Identification Identifier | Code List Name> - Code List Schema Module**

Where:

- Code List Agency Identifier = Identifies the agency that maintains the code list
- Code List Agency Name = Agency that maintains the code list
- Code List Identification Identifier = Identifies a list of the respective corresponding codes
- Code List Name = The name of the code list as assigned by the agency that maintains the code list

OAGi adopts this rule without modification.

2.4.5.6 Identifier List Schema Module

The UN/CEFACT NDR indicates where run time validation is required for an identifier scheme. A separate identifier list schema module will be created to minimize the impact of identifier list changes on root and other schemas.

Since this is an implementation specific choice OAGi does not include an identifier list schema module.

Identifiers by their nature are considered an infinite list of values, where a given value identifies a corresponding object. In many implementations each party involved has their own identifier for an object. Cross-referencing identifiers can be implemented

using an Identifier List schema module, but since this is a run time activity an XML instance or data base look up table may be a better fit.

[OAGi R 44]

[UN/CEFACT R 31] An Identifier List schema module MUST be created to convey enumeration values for each identifier list that requires run time validation.

OAGi relaxes this rule.

For those run time environments that require identifier cross reference validation one of the following SHOULD BE used to convey the enumerated values for each identifier list:

- An Identifier List schema module MAY BE used or
- A XML Instance identifier cross reference MAY BE used or
- A cross reference database MAY BE used.

If the identifier list schema modules are used, it must have a standard name that uniquely differentiates it from other schema modules.

[OAGi R 45]

[UN/CEFACT R 32] The name of each **`ids:IdentifierList`** schema module MUST be of the form: <Identifier Schema Agency Identifier | Identifier Schema Agency Name><Identifier Schema Identifier | Identifier Schema Name> - Identifier List Schema Module

Where:

- Identifier Scheme Agency Identifier = The identification of the agency that maintains the identification scheme
- Identifier Scheme Agency Name = Agency that maintains the identifier list
- Identifier Scheme Identifier = The identification of the identification scheme
- Identification Scheme Name = Name as assigned by the agency that maintains the identifier list

OAGi adopts this rule without modification.

2.4.5.7 Other Standards Body Aggregate Business Information Entity Schema Modules

The UN/CEFACT NDR indicates that other standards bodies ABIE modules contain reusable constructs created by standards bodies other than UN/CEFACT and made publicly available. UN/CEFACT will only import other Standards Body ABIE modules that are in strict conformance to the requirements of the CCTS and the UN/CEFACT NDR.

OAGIS is intended to be fully conformant to the UN/CEFACT NDR. The differences described in this document facilitate integration by others..

[OAGi R 46]

[UN/CEFACT R 33] Imported schema modules MUST be fully conformant with the *UN/CEFACT XML Naming and Design Technical* Specification and the *Core Components Technical Specification*.

OAGi relaxes this rule.

Imported schema modules to OAGIS SHOULD be fully conformant with the *OAGi OAGIS Naming and Design Rules Technical Standard*, the *UN/CEFACT XML Naming and Design Technical Specification* and the *Core Components Technical Specification*.

An example of a standard that OAGIS imports that does not follow the standards indicated is the ISO 20022 –Financial Payment Harmonization. This standard defines payment transactions between corporations and banks. By relaxing this rule OAGIS can be used in the banking industry.

2.5 Namespace Scheme

As defined by the W3C, XML namespaces provide a means of qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references. This enables interoperability and consistency in the XML artifacts for an extensive library of reusable types and schema modules. The reusability methodology used by OAGi maximizes the reuse of defined named types, globally declared elements and locally defined attributes within the types. In addition, the modularity approach of multiple reusable schema modules further enables the maximum amount of reuse possible. These are expressed in the relationships between the various BOD, internal and external schema modules identified earlier in this document.

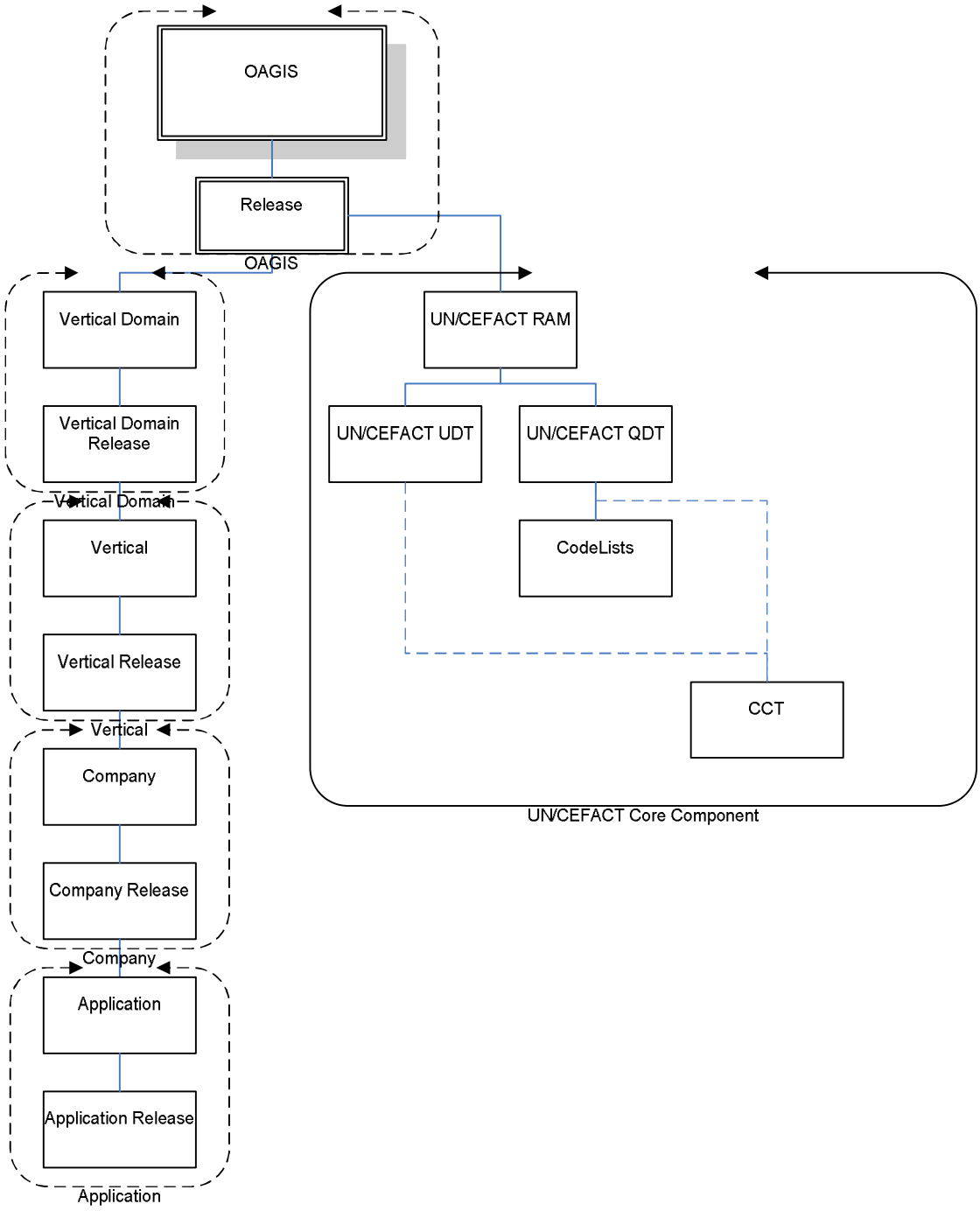
2.5.1 OAGIS Namespace Scheme

The namespace scheme used by OAGIS must allow for the relationships necessary to support the OAGi Modularity Scheme and for the incorporation of other standards namespaces such as the namespace scheme from UN/CEFACT.

The namespace scheme must also support being extended by the adoption of vertical industry groups like the Automotive Industry Action Group (AIAG) to incorporate their schema modules.

In addition to vertical standards organizations the namespace scheme must support a hierarchy of namespaces within a larger vertical area. For example in automotive there are several vertical groups that focus on certain aspects or geographic regions of automotive.

983 Figure 7 shows the approach used the OAGIS namespace structure.



984

985 **Figure 7 OAGi Namespace Scheme**

2.5.2 Declaring Namespace

Best practice indicates that every schema module should be declared in a namespace. Further more the UN/CEFACT NDR declares that internal schemas must be declared in the same namespace as the root schemas or BOD schemas.

[OAGi R 47]

[UN/CEFACT R 34] Every UN/CEFACT defined or imported schema module MUST have a namespace declared, using the **`xsd:targetNamespace`** attribute.

OAGi adopts this rule with editorial changes only.

Every defined or imported schema module MUST have a namespace declared, using the **`xsd:targetNamespace`** attribute.

2.5.3 Namespace Persistence

Namespaces are used to further qualify elements, attributes and types so that they maybe uniquely identified. The name of an element, attribute and type are further defined by the namespace in which it belongs. An element named X is different from an element named X in a second namespace. Furthermore a namespace should identify the maintainer, the standard and the version of that standard. For example the OAGIS namespace identifies the Open Applications Group, **`http://www.openapplications.org`**; the name of the standard **`oagis`**; and the version of the standard **`9`**. Adding these together define the OAGIS 9.0 namespace as: **`http://www.openapplications.org/oagis/9`**

A schema is interdependent upon the schemas that it includes or imports. All of the internal schemas must affect the versioning of the root schemas. Conversely, imported schema must effect the version of the root schema..

[OAGi R 48]

[UN/CEFACT R 35] Every version of a defined or imported schema module other than internal schema modules MUST have its own unique namespace.

OAGi adopts this rule without modification.

All of OAGIS is defined in the single namespace for the given release. As such each BOD is defined in this namespace and all of the internal schema modules (Components, Fields, Meta, and Nouns) are also defined in this single namespace and included.

Furthermore, OAGi follows a modular approach that allows for additions to be made such that the additions maintain backward compatibility. By doing this minor releases of OAGIS maybe defined using the same namespace, where the major release version is identified in the namespace, but not the minor release indicators. This is described further in the *Versioning* section below and the *OAGi Versioning Standard*.

A given namespace is associated with each major release and does not require a new namespace for each minor release. This further enables the reuse of existing code and localizes changes to the context of extensions.

[OAGi R 49]

[UN/CEFACT R 36] UN/CEFACT published namespace declarations or contents MUST never be changed unless such change does not break backward compatibility.

OAGi adopts this rule without modification.

[OAGi R 50]

All extensions to OAGIS SHOULD use either an Overlay of OAGIS or a UserArea extension of OAGIS.

Other forms of modifications to the OAGIS specification result in derivative versions that defeat the fundamental intent of a standard.

2.5.4 Namespace Uniform Resource Identifiers

OAGi recommends that namespaces be resolvable to a persistent location to find more information about the schema being defined. Uniform Resource Indicators (URIs) are used to identify a namespace. Valid URIs include: Uniform Resource Locators (URLs) and Uniform Resource Names (URNs). After reviewing the two options OAGi determined: URLs are resolvable and are as persistent as the organizations that maintain the schemas; URNs are not resolvable and identify a name for a given standard that is typically associated with the organization that maintains the standard. When that name changes the URN and URL change; Therefore, URNs and URLs were determined to be equally persistent. Since URLs are resolvable, OAGi choose to use URLs.

[OAGi R 51]

[UN/CEFACT R 37] UN/CEFACT namespaces MUST be defined as Uniform Resource Names

OAGi adopts the intent of this rule but modifies the actual implementation.

OAGi namespaces MUST be defined as Uniform Resource Locators

In order to ensure consistency, each OAGi namespace will have the same general structure. The following is an example of this structure:

<URL>\<Standard>\<Major Release>

Where :

<URL> = the URL of the Open Applications Group, http://www.openapplications.org.

<Standard> = the name of the standard being defined. In the case of OAGIS, it is **oagis**

<Major Release> = the major release of the standard that is being defined. In the case of OAGIS 9.0, it is **9**.

OAGi does not change the namespace either a draft or a standard releases. Instead, OAGi uses the schemaLocation to point to the appropriate repository. Again, this ensures the maximum amount of reusability of object classes for implementations that may have started prototyping work with draft releases.

[OAGi R 52]

[UN/CEFACT R 38] The names for namespaces MUST have the following structure while the schema is at draft status:

urn:un:unece:uncefact:<schematype>:draft:<name>:<major>

Where :

- schematype = a token identifying the type of schema module: **data | process | codelist | identifierlist | documentation**
- name = the name of the module (using upper camel case)
- major = the major version number. Sequentially assigned, first release starting with the number 1.

OAGi relaxes this rule.

[OAGi R 53]

[UN/CEFACT R 39] The namespace names for schemas holding specification status MUST be of the form:

urn:un:unece:uncefact:<schematype>:standard:<name>:<major>

Where :

- schematype = a token identifying the type of schema module: **data | process | codelist | identifierlist | documentation**
- name = the name of the module (using upper camel case)
- major = the major version number. Sequentially assigned, first release starting with the number 1.

OAGi relaxes this rule.

[OAGi R 54]

Each OAGi namespace MUST have the following structure:

<URL>/<standard>/<major>/[<overlayname>|<substandardname>]/[<overlay major>|<substandardmajor>]]

For example OAGIS and OAGIS Overlays use the following:
http://www.openapplications.org/oagis/<major>/[<overlayname>]/[<overlaymajor>]

Where :

- <URL> = the URL of the Open Applications Group, **http://www.openapplications.org**
- <standard> = the name of the standard being defined. In the case of OAGIS, it is **oagis**
- <major> = the major release of the standard that is being defined. In the case of OAGIS 9.0, it is **9**.
- overlayname = name of the overlay, this is typically the name of the organization and or project of the overlay. The overlayname MAY be of the form **organization/project**.
- substandardname = identifies a sub portion of the standard for example the OAGIS implementation of the codelist.
- overlaymajor = the major version number of the overlay, sequentially assigned, first release starting with the number 1.
- Substandardmajor = the major version number of the substandard, sequentially assigned, first release starting with the number 1.

For example:

http://www.openapplications.org/oagis/9/aiag/ivi/1

2.5.5 Namespace Constraint

In order to be consistently defined OAGi namespaces like OAGIS must be created and assigned by OAGi. Likewise any extension namespaces must be created and assigned by the organization that is extending OAGIS, or their agents.

[OAGi R 55]

[UN/CEFACT R 40] UN/CEFACT namespaces MUST only contain UN/CEFACT developed schema modules.

OAGi adopts this rule with editorial changes only.

OAGi namespaces MUST only contain OAGi developed schema modules.

[OAGi R 56]

OAGi extensions must be made in a namespace that reflects the name of the organization that is responsible for the extensions being made to OAGIS. These schema modules MUST only contain content developed by these organization or their agents.

2.5.6 Schema Namespace Tokens

Each namespace used by OAGi will have its own namespace token. This token is used as an alias when referencing the namespace in element, and type names. The list of these token is provided in Table 2-2 earlier in this document.

2.6 Schema Location

Schema locations are required to be in the form of a URI scheme. Since the purpose of the schema location is to provide a reference point in which to obtain access to a schema definition, it must be resolvable. Therefore, most schema locations are URLs, which are the resolvable form of a URI.

During deployment the schema definitions referenced by the schema location may need to reside in many different places. It is not practical to provide an Internet address in this URL for all implementations to resolve at runtime. Especially considering that many of these implementations are critical to the operations of the organizations that use them, where there are millions of exchanges of information an hour not to mention in a day.

In order to facilitate this, the schema locations provided by OAGi in OAGIS are normative and relative referenced schema locations for the XML schema references. The initial reference to the defining XSD in the XML instance must provide the persistent location to find the root or BOD schema. The remaining references within the schema set use normative and relative reference URLs. This allows OAGIS to be deployed via an Internet, Intranet, locally on the machine, or in a database repository. This also supports Unix, Windows, or Mainframe based servers implementations. Using any other form for the URI in the schema location limits the possibilities for implementation by the end user.

[OAGi R 57]

[UN/CEFACT R 41] The general structure for schema location MUST be:
http://www.unece.org/uncefact/<schematype>/<name>_<major>.<minor>.[<revision>]_[<status>].xsd

Where:

- schematype = a token identifying the type of schema module: **data | process | codelist | identifierlist | documentation**
- name = the name of the module (using upper camel case)
- major = the major version number, sequentially assigned, first release starting with the number 1.
- minor = the minor version number within a major release, sequentially assigned, first release starting with the number 0.
- revision = sequentially assigned alphanumeric character for each revision of a minor release. Only applicable where status = draft.

- status = the status of the schema as: **`draft | standard`**

OAGi relaxes this rule.

[OAGi R 58]

The schema location in an XML instance document when referring to an OAGIS BOD MUST be of the form:

`<URL>/oagis/<major>.<minor>/BODs/<schemaform>/<name>.xsd`.

Where:

- URL = is the URL of the location that will resolve the reference to XSD file. This maybe the Open Applications Group Web site or it may be an implementation specific URL where OAGIS is stored.
- major = the major version number, sequentially assigned, first release starting with the number 1.
- minor = the minor version number within a major release, sequentially assigned, first release starting with the number 0.
- schemaform = the form of the schema: **`developer | standalone`**
- name = the name of the BOD or root schema.

[OAGi R 59]

The schema location in an XML instance document when referring to an OAGIS Overlay BOD MUST be of the form:

`<URL>/oagis/<major>>.<minor>/<overlayname>/<overlaymajor>.<overlayminor>/BODs/[<schemaform>]/<name>.xsd`.

Where:

- URL = is the URL of the location that will resolve the reference to XSD file. This maybe the Open Applications Group Web site or it may be an implementation specific URL where OAGIS and the Overlay is stored.
- major = the major version number of OAGIS, sequentially assigned, first release starting with the number 1.
- minor = the minor version number within a major release, sequentially assigned, first release starting with the number 0.
- overlayname = name of the overlay, this is typically the name of the organization and or project of the overlay. The overlayname MAY be of the form **`organization\project`**.
- overlaymajor = the major version number of the overlay, sequentially assigned, first release starting with the number 1.
- overlayminor = the minor version number within a major release of the overlay, sequentially assigned, first release starting with the number 0.

- schemaform = the form of the schema: **`developer | standalone`**
- name = the name of the BOD or root schema.

[OAGi R 60]

[UN/CEFACT R 42] Each **`xsd:schemaLocation`** attribute MUST contain a persistent and resolvable URL.

OAGi adopts the intent of this rule but modifies that actual implementation.

Each **`xsd:schemaLocation`** attribute in an XML Instance MUST contain a persistent and resolvable URL.

[OAGi R 61]

Each **`xsd:schemaLocation`** attribute in an XSD document SHOULD use relative reference paths that are normative.

[OAGi R 62]

[UN/CEFACT R 43] Each **`xsd:schemaLocation`** attribute declaration URL MUST contain an absolute path.

OAGi adopts the intent of this rule but modifies that actual implementation.

Each **`xsd:schemaLocation`** attribute declaration in an XML instance document MUST contain an absolute path.

2.7 Versioning

The one constant in the world is change. This is never more evident than in today's business world, where needs and requirements are constantly changing. The best practices for dealing with these changes require flexiblity while identifying when the changes affect compatibility.

Instance of BODs are said to be compatible if they can be validated by both the source and destination Schemas. This is further defined below.

The OAGi versioning schema embraces compatibility as an enabling factor for implementation. It is critical to capture what has changed between each version or release. It is also important to identify what are major changes and what are minor changes. OAGi uses compatibility as the deciding factor as to what is a major release versus a minor release. In other words if, the changes break compatibility it is a major release. Likewise, if the changes are simply adding new optional content that does not break compatibility then the release is a minor release.

OAGi uses this distinction in the namespaces as well. Since a major release by its nature breaks compatibility then the namespace reflect that and enforce the incompatibility. Also the change to the namespace further breaks compatibility. Since minor releases do not break compatibility, and changing the namespace would break compatibility only to reference the minor change, OAGi does not change the namespace for any minor releases.

The *OAGi Versioning Policy* describes the OAGi approach to versioning in more detail.

2.7.1 Version Compatibility

There are two types of version compatibility: backward compatibility and forward compatibility. XML.com article by David Orchard December 03, 2003 describes these as follows:

> "Backwards compatibility means that a new version of a receiver can be rolled out so it does not break existing senders. This means that a sender can send an old version of a message to a receiver that understands the new version and still have the message successfully processed.
>
> Forwards compatibility means that an older version of a receiver can consume newer messages and not break. Of course the older version will not implement any new behavior, but a sender can send a newer version of a message and still have the message successfully processed.
>
> In other words, backwards compatibility means that existing senders can use services that have been updated, and forwards compatibility means that newer senders can continue to use existing services.
>
> Forwards-compatible changes typically involve adding optional element(s) and/or attribute(s). The costs associated with introducing changes that are not backwards- or forwards-compatible are often very high, typically requiring deployed software to be updated to accommodate the newer version."

A key point from the excerpt above is that the cost of incompatible changes is often high due to the need to modify deployed solutions.

2.7.2 Major Versions

A major version in an OAGi schema module constitutes non-backward compatible changes, as described above. These changes major consist of, but not limited to:

- Changing element, type, and attribute names

- Changing the structures so as to break polymorphic processing capabilities
- Deleting or adding mandatory elements or attributes
- Removing or changing values in enumerations.

Major release numbers are indicated in the namespace declaration as defined previously declared.

[OAGi R 63]

[UN/CEFACT R 44] Every schema major version namespace declaration MUST have the URI of: **`urn:un:unece:uncefact:<schematype>:<status>:<name>:<major>`**

OAGi adopts the intent of this rule but modifies the actual implementation.

Every schema major version namespace declaration MUST have a URI of the form: **`http://www.openapplications.org/<standard>/<major>/[<overlayname>|<substandardname>]/[<overlaymajor>|<substandardmajor>`**>]

[OAGi R 64]

[UN/CEFACT R 45] Every UN/CEFACT XSD Schema and schema module major version number MUST be a sequentially assigned incremental integer greater then zero.

OAGi adopts this rule with editorial changes only.

Every XSD Schema and schema module major version number MUST be a sequentially assigned incremental integer greater than zero.

2.7.3 Minor Versions

Within a major release of an OAGi schema module there can be a series of minor releases that are all compatible. All minor releases are compatible as long as they are contained within a single major release. This allows the user to determine what releases are compatible and which can be used together. Minor versions incremented when compatible changes occur. These may consist of but are not limited to the following:

- Adding optional elements or attributes
- Adding values to enumerations

[OAGi R 65]

[UN/CEFACT R 46] Minor versioning MUST be limited to declaring new optional XSD constructs, extending existing XSD constructs and refinements of an optional nature.

OAGi adopts this rule without modification.

Minor version numbers are NOT reflected in the namespace declaration because changing the namespace breaks compatibility. Anytime a namespace is changed the code that process that namespace must also change to address the new namespace whether the content changed or not.

[OAGi R 66]

[UN/CEFACT R 47] Every UN/CEFACT XSD Schema minor version MUST have the URI of: **`urn:un:unece:uncefact:cc:schema:<name>:<major>`**

OAGi relaxes this rule.

[OAGi R 67]

Every minor version MUST use the same namespace as the major version to which it is associated.

Like major versions, minor versions numbers should be based on a logical progression to ensure the understanding of the approach and guarantee consistency in representation. The minor version number is a sequentially assigned incremental integer greater than zero..

Minor versions changes are not allowed to break compatibility with previous versions as described earlier in this document.

[OAGi R 68]

[UN/CEFACT R 48] For UN/CEFACT minor version changes, the name of the schema construct MUST NOT change.

OAGi adopts this rule with editorial changes only.

For OAGi minor version changes, the name of the schema construct MUST NOT change.

[OAGi R 69]

[UN/CEFACT R 49] Changes in minor versions MUST NOT break semantic compatibility with prior versions.

OAGi adopts this rule without modification.

For a given namespace, the parent major release and subsequent minor releases create a relationship. In OAGIS each minor release utilizes the same namespace as the parent major release. The first minor release must incorporate the parent major release, and each subsequent release must incorporate the previous minor release.

[OAGi R 70]

[UN/CEFACT R 50] UN/CEFACT minor version schema MUST incorporate all XML constructs from the immediately preceding major or minor version schema.

OAGi adopts this rule with editorial changes only.

Minor version schema MUST incorporate all XML constructs from the immediately preceding major or minor version schema.

3.0 GENERAL XML SCHEMA CONVENTIONS

XML Schema includes many different concepts. Some are generally supported by tools and some are not or only partially supported by tools. The *OAGi Practical Guide to XML Schema* provides a detailed review of the constructs that should be implemented in order to practically claim support for XML Schema. This section identifies the rules associated with these constructs as they are used within OAGi and particular OAGIS 9.0.

3.1 Schema Construct

XML Schema includes many different constructs. OAGIS uses those constructs that are considered consistently implemented by tools.

[OAGi R 71]

[UN/CEFACT R 51] The **`xsd:elementFormDefault`** attribute MUST be declared and its value set to **`"qualified"`**.

OAGi adopts this rule without modification.

[OAGi R 72]

[UN/CEFACT R 52] The **`xsd:attributeFormDefault`** attribute MUST be declared and its value set to **`"unqualified"`**.

OAGi adopts this rule without modification.

[OAGi R 73]

[UN/CEFACT R 53] The "**`xsd`**" prefix MUST be used in all cases when referring to http://www.w3.org/2001/XMLSchema as follows:
`xmlns:xsd=http://www.w3.org/2001/XMLSchema`

OAGi adopts this rule with editorial changes only.

The "**`xsd`**" prefix MUST be used in all cases when referring to http://www.w3.org/2001/XMLSchema as follows:
`xmlns:xsd="http://www.w3.org/2001/XMLSchema"`

An example of these rules:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.openapplications.org/oagis/9"
targetNamespace="http://www.openapplications.org/oagis/9"
```

```
elementFormDefault="qualified"
attributeFormDefault="unqualified">
```

3.1.1 Constraints on Schema Construction

[OAGi R 74]

[UN/CEFACT R 54] The **xsi** prefix SHALL be used where appropriate for referencing **xsd:schemaLocation** and **xsd:noNamespaceLocation** attributes in instance documents.

OAGi further constrains this rule.

The **xsi** prefix SHALL only be used where appropriate for referencing **xsd:schemaLocation** and **xsd:noNamespaceLocation** attributes in instance documents.

[OAGi R 75]

[UN/CEFACT R 55] **xsd:appInfo** MUST NOT be used.

OAGi adopts this rule without modification.

[OAGi R 76]

[UN/CEFACT R 56] **xsd:notation** MUST NOT be used.

OAGi adopts this rule without modification.

[OAGi R 77]

[UN/CEFACT R 57] **xsd:wildcard** MUST NOT be used.

OAGi adopts this rule without modification.

OAGIS uses xsd:any to enable UserArea extensions. This extension allows additional elements to be added to an instance document without making any modifications to the XML Schema.

[OAGi R 78]

[UN/CEFACT R 58] **xsd:any** element MUST NOT be used.

OAGi relaxes this rule.

xsd:any element MUST NOT be used with the one exception of the UserArea within OAGIS.

[OAGi R 79]

[UN/CEFACT R 59] **`xsd:any`** attribute MUST NOT be used.

OAGi adopts this rule without modification.

[OAGi R 80]

[UN/CEFACT R 60] Mixed content MUST NOT be used (excluding documentation).

OAGi adopts this rule without modification.

OAGIS 9.0 does not use substitutionGroups within the core Schemas. However substitutionGroups are used to enable Overlay extensions.

[OAGi R 81]

[UN/CEFACT R 61] **`xsd:substitutionGroup`** MUST NOT be used.

OAGi relaxes this rule.

xsd:substitutionGroup SHOULD only be used as an extension mechanism to extended the original definition in order to provide additional contextual requirements.

[OAGi R 82]

[UN/CEFACT R 62] **`xsd:ID/IDREF`** MUST NOT be used.

OAGi adopts this rule without modification.

[OAGi R 83]

[UN/CEFACT R 63] **`xsd:key/xsd:keyref`** MUST be used for information association.

OAGi adopts this rule without modification..

[OAGi R 84]

[UN/CEFACT R 64] The absence of a construct or data MUST NOT carry meaning.

OAGi adopts this rule without modification.

3.2 Attribute and Element Declarations

OAGi makes use of both elements and attributes. Primarily elements are used because they are extensible and attributes are not. Attributes are used for the simple qualification of an element. Many of the attributes used by OAGIS come directly from the CCTS. While CCTS does not require the use of attributes, the UN/CEFACT NDR does.

3.2.1 Attributes

3.2.1.1 Usage of Attributes

User declared attributes can be used to convey supplementary components of core component types. The intent of the attributes as used by OAGi is to qualify the associated elements. Built-in **`xsd:attributes`** will be used as described in this document. User declared attributes can represent different types of values. The values may be variable information or can be based on code lists.

[OAGi R 85]

[UN/CEFACT R 65] User declared attributes MUST only be used to convey core component type (CCT) supplementary component information.

OAGi relaxes this rule.

[OAGi R 86]

[UN/CEFACT R 66] An attribute of a supplementary component with variable information MUST be based on the appropriate built-in XSD data type.

OAGi adopts the intent of this rule but modifies the actual implementation.

An attribute with variable information MUST be based on the appropriate built-in XSD data type.

[OAGi R 87]

[UN/CEFACT R 67] An attribute of a supplementary component which represents codes MUST be based on the **`xsd:simpleType`** of the appropriate code list.

OAGi adopts this rule without modification.

[OAGi R 88]

[UN/CEFACT R 68] An attribute of a supplementary component which represents identifiers MUST be based on the **`xsd:simpleType`** of the appropriate identifier scheme.

OAGi relaxes this rule.

3.2.1.2 Constraints on Attribute Declarations

The absence of an element in an XML instance does not have meaning. It may indicate the information is unknown or not applicable, or the element may be absent for some other reason. XML Schema does provide a construct where an element may be

transferred with no content, but still use its attributes and carry semantic meaning. This is possible by using the nillable attribute.

[OAGi R 89]

[UN/CEFACT R 69] The **`xsd:nillable`** attribute MUST NOT be used.

OAGi adopts this rule without modification.

3.2.2 Elements

Elements are declared for document level message assembly, following the Core Component approach. Elements are generally used by the Business Object Documents (BODs), although, they may be used for lower level message assembly to communicate information about components including field level information.

3.2.2.1 Element Declaration

[OAGi R 90]

[UN/CEFACT R 70] Empty elements MUST NOT be used.

OAGi adopts this rule without modification.

[OAGi R 91]

[UN/CEFACT R 71] Every BBIE leaf element delcaration MUST be of the **`udt:UnqualifiedDataType`** or **`qdt:QualifiedDataType`** that represents the The **`xsd:type`** of each leaf element declaration MUST be of the data type of its source business information entity (BBIE) **`ccts:DataType`**

OAGi adopts this rule without modification.

3.2.2.2 Constraints on Element Declarations

[OAGi R 92]

[UN/CEFACT R 72] The xsd:all element MUST NOT be used.

OAGi adopts this rule without modification.

[OAGi R 93]

All elements MUST be declared using named types.

3.3 Type Definitions

In order to maximize reusability all elements must be declared using named types. This allows the type definitions to be reused across multiple elements and to be extended where appropriate.

[OAGi R 94]

[UN/CEFACT R 73] All type definitions MUST be named.

OAGi adopts this rule without modification.

[OAGi R 95]

[UN/CEFACT R 74] Data type definitions MUST NOT duplicate the functionality of an existing data type definition.

OAGi adopts this rule without modification.

3.3.1 Simple Type Definitions

OAGIS uses Core Component Technical Specification (CCT) for all end level elements and attributes where they can be applied. This is done by using the representations identified in the UDT and QDT data types for the basis of these OAGIS defined types. In doing this OAGIS 9.0 does not use any XML Schema simpleTypes directly.

The OAGIS representations of the CCT, UDT, QDT, and CodeList are required to define the representation by using the XML Schema simpleTypes, so that they satisfy the business requirements. ComplexTypes are only used when a simpleType does not satisfy these business requirements.

OAGIS also uses simple types to define the intermediary types for code lists that are based on the appropriate code list simple type.

Simple Type in the Unqualified Data Type Schema Module

```
<xsd:simpleType name="DateTimeType">
  <xsd:restriction base="xsd:dateTime"/>
</xsd:simpleType>
```

3.3.2 Complex Type Definitions

User defined complex types may be used when XML Schema built-in simple types do not satisfy the business requirements or when an aggregate business information entity (ABIE) must be defined.

OAGIS uses complex types to define:

1. The OAGIS 9.0 representation of the UN/CEFACT artifiacts for:
    - ACC in the ReusableAggregateCoreComponent.xsd and
    - ABIEs in the ReusableAggregateBusinessInformationEntity.xsd
    - CodeLists
2. To define the OAGIS intermediary types for:
    - CodeLists
    - ABIEs
3. To define OAGIS specific:
    - Components or ABIEs
    - Nouns
    - Verbs
    - BODs
    - Base types in which the above inherit from.

Complex type of an object class AccountType:

```
<xsd:complexType name="AccountType">
      <xsd:sequence>
            <xsd:element name="ID" type="udt:IdentifierType"
minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="Text" type="udt:TextType" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element name="Code" type="udt:CodeType" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element name="DateTime" type="udt:DateTimeType"
minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="Status" type="rcm:StatusType"
minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="Country" type="rcm:CountryType"
minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="Person" type="rcm:PersonType"
minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="Organization"
type="rcm:OrganizationType" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
</xsd:complexType>
```

3.4 Use of Extension and Restriction

IOAGIS uses an object model where the base concepts are identified and reused where appropriate.

Looking at the features of XML Schema OAGi made the conscious decision to use derivation by extension and not to use derivation by restriction for complex types. This was based on discussions with W3C and an in depth understanding of how each works.

3.4.1 Derivation by Extension

OAGIS is able to inherit through derivation by extension from base types as needed. For example in business level application integration there are several base communications that are document based like a PurchaseOrder, and an Invoice. At the simple level each has a concept of a header and a line or details. Beyond this the headers have a document identifier and a timestamp for the document. The line or details each have a line number identifier.

OAGIS reuses these definitions as a form of inheritance to avoid redundancy.

[OAGi R 96]

[UN/CEFACT R 75] xsd:extension MUST only be used in the **`cct:CoreComponentType`** schema module and the **`udt:UnqualifiedDataType`** schema module. When used it MUST only extend a built-in XSD datatype.

OAGi relaxes this rule.

`xsd:extension` is used to extend the content of an existing type to meet the needs of further requirements for a given object or composite object.

OAGIS Overlays make use of derivation by extension in order to extend a given OAGIS object type to meet the new requirements identified.

3.4.2 Derivation by Restriction

Derivation by restriction is only used for simple types in the OAGIS representations of the UN/CEFACT UDT, QDT, and Code Lists. These may include OAGIS or OAGIS Overlay code lists as needed.

Derivation by restriction for complex types is considered by many to be broken. In that it simply makes a copy of the original type and begins to remove content. Additionally, XML Schema does not allow derivation by restriction across namespaces.

For derivation by restriction to be practical, changes to the core Schema specifications are required or tool vendors must manage the changes more efficiently.

[OAGi R 97]

[UN/CEFACT R 76] When **xsd:restriction** is applied to a **xsd:simpleType** or **xsd:complexType** the derived construct MUST use a different name.

OAGi further constrains this rule.

When **xsd:restriction** is applied to a **xsd:simpleType** the derived construct MUST use a different name. **xsd:restriction** MUST only be applied to a **xsd:simpleType**.

3.5 Annotation

OAGi uses the xsd:annotation to provide documentation per the UN/CEFACT NDR description of documentation. This is used in OAGIS 9.0 in the CCT, UDT and QDT modules. The other schema modules in OAGIS 9.0 capture a description of the element or type and its intended use.

Future releases of OAGIS all schema modules will make use of the UN/CEFACT documentation as described in UN?CEFACT NDR 6.5.1 Documentation.

[OAGi R 98]

[UN/CEFACT R 77] Each UN/CEFACT defined or declared construct MUST use the **xsd:annotation** element for required CCTS documentation.

OAGi adopts the intent of this rule but modifies the actual implementation.

Each defined or declared construct MUST use the **xsd:annotation** element for documentation.

[OAGi R 99]

Each **xsd:annotation** MUST use the **xsd:documentation** element for documentation.

[OAGi R 100]

Each **xsd:documentation** MUST use the **source** attribute with the following value: “http://www.openapplications.org/oagis”

For example:

```
<xsd:complexType name="SenderType">
    <xsd:annotation>
      <xsd:documentation
source="http://www.openapplications.org/oagis/9">
         Identifies the sender of the given BOD instance.
      </xsd:documentation>
    </xsd:annotation>
</xsd:complexType>
```

4.0 SCHEMA MODULES

OAGIS 9.0 includes various schema modules all of which serve different roles in building OAGIS content. These schema modules are consistent with the schema modules defined by the UN/CEFACT NDR as describe previousily in this document.

4.1 BOD

The BOD schema serves as the container for all schema content that is required to fulfill an exchange of business information. The BOD schema is defined in the OAGIS 9.0 namepace – http:\\www.openapplications.org\oagis\9.

The Developer BOD schema modules include references to the internal schema modules (Nouns, Components, Fields, and Meta) as needed. It may also import external schemas modules as needed, as is the case for Overlay BODs that extend an existing OAGIS BOD. The Standalone BODs schema modules include directly in the schema module all of the artifacts required for the given BOD for the exchange of business information.

4.1.1 Schema Construct

Each Developer BOD schema must be defined in a standard format in order to ensure consistency and ease of use. The format is shown in Figure 8.

```
<?xml version="1.0" encoding="utf-8"?>
<!--

** OAGIS® Revision: 9.0 **
** Date: 08 April 2005 **
** Copyright 1998-2005, All Rights Reserved **

This is an OAGIS® BOD XML Schema (XSD) Definition.

License information for this file is provided in the file **2005 OAGi
License Agreement.txt** that is provided with this download package.

For support, more information, or to report implementation bugs, please
contact the Open Applications Group at xml@openapplications.org.

    XML Schema

   Name: \OAGIS\9.0\BODs\Developer\NameOfTheBOD.xsd
-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.openapplications.org/oagis/9"
```

```
targetNamespace="http://www.openapplications.org/oagis/9"
elementFormDefault="qualified" attributeFormDefault="unqualified">

      <xsd:include schemaLocation="..See Nouns.. "/>

      <xsd:element name="BODRootElement" type="BODRootElementType">
      </xsd:element>

      <xsd:complexType name="BODRootElementType">
            <xsd:complexContent>
                  <xsd:extension base="BusinessObjectDocumentType">
                        <xsd:sequence>
                              <xsd:element name="DataArea"
type="BODRootElementDataAreaType">

                              </xsd:element>
                        </xsd:sequence>
                  </xsd:extension>
            </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="BODRootElementDataAreaType">
            <xsd:sequence>
                  <xsd:element ref="Verb"/>
                  <xsd:element ref="Noun" maxOccurs="unbounded"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:schema>
```

Figure 8 - Structure of the Developer BOD Schema Module

Each Standalone BOD schema module that is a part of OAGIS 9.0 is generated by an application available from the Open Applications Group from the Developer BOD schema module. The resulting schema module has a standard structure that is followed. This structure is shown in Figure 9.

```
 <?xml version="1.0" encoding="utf-8"?>
<!--

** OAGIS® Revision: 9.0 **
** Date: 08 April 2005 **
** Copyright 1998-2005, All Rights Reserved **

This is an OAGIS® BOD XML Schema (XSD) Definition.

License information for this file is provided in the file **2005 OAGi
License Agreement.txt** that is provided with this download package.

For support, more information, or to report implementation bugs, please
contact the Open Applications Group at xml@openapplications.org.

    XML Schema

   Name: \OAGIS\9.0\BODs\Standalone\NameOfTheBOD.xsd
```

```
-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.openapplications.org/oagis/9"
targetNamespace="http://www.openapplications.org/oagis/9"
elementFormDefault="qualified" attributeFormDefault="unqualified">

      <xsd:import ...of all namespaces used by the Standalone BOD.../>

      <xsd:attributeGroup …of all attributeGroups used…/>

      <xsd:complexType …of all complexTypes used…/>

      <xsd:element …of all elements used…/>

      <xsd:group …of all groups used…/>

      xsd:simpleType …of all simpleTypes used…/>

</xsd:schema>
```

Figure 9 - Structure of the Standalone BOD Schema Module

4.1.2 Namespace Scheme

All BODs published in OAGIS 9.0 use the OAGIS 9.0 namespace, http://www.openapplications.org/oagis/9. Future releases of OAGIS may include additional namespaces to identify the different domains that OAGIS covers.

[OAGi R 101]

[UN/CEFACT R 77] The root schema module MUST be represented by a unique token.

OAGi relaxes this rule.

[OAGi R 102]

A BOD schema module MUST be defined in the OAGIS Namespace in the case of OAGIS. In the case of an Overlay of OAGIS the BOD schema module must be defined in a different namespace that corresponds to the Overlay.

5.0 OAGIS 9.0 ARCHITECTURE

5.1 Design Considerations for OAGIS 9.0

5.1.1 Address Non-Determinism

Non-determinism can roughly be defined as a situation where, upon encountering an element in an instance document, it is ambiguous which path was taken in the schema document.

Ninety percent of the instances of OAGIS non-determinism occur with how earlier versions of OAGIS segments were represented, due mostly to limitations of XML DTDs. A deeper explanation of this problem's basis in type theory is beyond the scope of this document. Suffice it to say that element non-determinism has been a thorn in the side of many OAGIS users.

5.1.1.1 The Non-Determinism Problem in a Nutshell

In prior versions of OAGIS, fields that relied on segments were named based on the ***intended type*** of a field (e.g., "DateTime"), not based on the ***actual name*** of the thing being described (e.g., "NeedDelivery"). What would have been the natural name of the field was instead buried in a "qualifier" attribute. So, instead of modeling the NeedDelivery field of a PurchaseOrderLine as

```
<PurchaseOrderLine>

        <NeedDelivery>   </ NeedDelivery>

</PurchaseOrderLine>
```

it was modeled as

```
<PurchaseOrderLine>

    <DateTime qualifier="NeedDelivery">  </DateTime>

</PurchaseOrderLine>
```

This was one of the few ways that DTDs could impose the needed DateTime structure on the NeedDeliveryBy field, so that parsers could do some (minimal) checking of the content.

The problem arose when more than one field of type DateTime was needed in a given element model (e.g., more than one DateTime child of a PurchaseOrderLine):

```
<PurchaseOrderLine>

    <DateTime qualifier="NeedDelivery">  </DateTime>

    <DateTime qualifier="PromisedDelivery">  </DateTime>

</PurchaseOrderLine>
```

The non-determinism exists because there are two different DateTime elements in the content of the PurchaseOrderLine . When the parser sees this and can't distinguish one from the other, it raises this as a warning. Furthermore, since the parse cannot distinguish one from the other, there is no way for it to require that, e.g., a NeedDelivery is required and a PromisedDelivery is optional.

The outcome of this is that, prior OAGIS 8.0, OAGIS designers were limited in what they could express in a given element, and XML parsers were limited in what structural integrity they could enforced.

5.1.2 Addressing the Non-Determinism

The problem is addressed by promoting the qualifier's value to being (part of) the element's name, e.g.,

<NeedDelivery> </ NeedDelivery>

and by defining the element's model (type).

<element name="NeedDelivery" type="DateTime"> </element>

Now, rather than naming elements according to their types, elements are named according to their primary meaning, purpose, or function. Thus, there will no longer be an Amount(Extended)(T). Instead, the element will be named something like a required "TotalPrice" of type "Amount."[3] Furthermore, there can also be an optional "AdditionalCost" of type "Amount."

With XML Schema's relatively advanced type system, the context of the TotalPrice element and the binding, in the schema, of TotalPrice to the type Amount is all that are needed for a validating parser to validate that the content of a TotalPrice element is indeed an Amount and fits all of the criteria to be a legal Amount. Parsers can not only distinguish between a TotalPrice and an AdditionalCost, but can enforce that the former is required and the latter is optional.

[3] In all prior OAGIS releases, the practice of shortening field and segment names resulted in names that were less meaningful that their full equivalents, and often resulted in names that were inconsistently abbreviated. OAGIS 8.0 instead uses the long names that have long been associated with each element, as documented in Appendices C and D. For example AMOUNT(ESTFREIGHT)(T) in previous releases of OAGIS now uses the intended names, e.g., EstimatedFreightCharge.

APPENDIX A – OAGI ACCEPTED ACRONYMS AND ABBREVIATIONS

Acronyms

BOD	Business Object Document
BOM	Bill of Material
DUNS	Data Universal Numbering System
EFT	Electronic Funds Transfer
GL	General Ledger
HR	Human Resources
HTML	Hyper Text Markup Language
SCE	Supply Chain Execution
UOM	Unit of Measure
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WIP	Work In Process

Abbreviations

Class	Classification
Doc	Document
Enum	Enumeration
ID	Identifier
Ind	Indicator
Max	Maximum
Min	Minimum
Ship	Shipment
Sync	Synchronize

Non-Oxford

ABC Classification
Subentity
Subline