|  |  |
| --- | --- |
| Symbol projektu:  Model Xml Schema | |
| Nazwa projektu:  **Model obiektowy Xml Schema** | |
| Tytuł dokumentu: | **Dokumentacja modelu** |
| Nr wersji: | **1.1** |
| Data utworzenia: | **22.06.2024** |
| Data ostatniej aktualizacji: | **07.07.2024** |
| Osoba odpowiedzialna: | **Jarosław Kuchta (JK)** |
| Autorzy: | **Jarosław Kuchta (JK)** |

Historia dokumentu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wersja | Opis modyfikacji | Autor | Data otwarcia | Data zamknięcia |
| 1.0 | Ekstrakcja z ModelDocumentation | JK | 22.06.2024 | 4.07.2024 |
| 1.1 | Duże przyspieszenie po doświadczeniu z OpenXmlLib | JK | 4.07.2024 | 7.07.2024 |

**Spis treści**

[1. Wprowadzenie 2](#_Toc171240562)

[2. Pliki XSD i ich zawartość 2](#_Toc171240563)

[2.1. Struktura pliku 2](#_Toc171240564)

[2.2. Model obiektowy schematu XSD 4](#_Toc171240565)

[2.2.1. Typy proste 6](#_Toc171240566)

[2.2.2. Typy złożone 8](#_Toc171240567)

[2.2.3. Grupy elementów 11](#_Toc171240568)

[2.2.4. Grupy atrybutów 12](#_Toc171240569)

[2.2.5. Elementy globalne 12](#_Toc171240570)

[2.2.6. Atrybuty globalne 12](#_Toc171240571)

[2.2.7. Adnotacje (pomijane) 13](#_Toc171240572)

[3. Model danych aplikacji 13](#_Toc171240573)

[3.1. Pliki i przestrzenie nazw 13](#_Toc171240574)

[3.2. Elementy deklarowane w przestrzeni nazw 14](#_Toc171240575)

[3.2.1. Typy 15](#_Toc171240576)

[3.2.2. Typy proste 15](#_Toc171240577)

[3.2.3. Typy złożone 18](#_Toc171240578)

[3.2.4. Atrybuty i grupy atrybutów 19](#_Toc171240579)

[3.2.5. Partykuły, elementy i grupy elementów 21](#_Toc171240580)

[4. Kontekst danych 25](#_Toc171240581)

[5. Parsowanie modelu schematu 26](#_Toc171240582)

[6. Wyniki 27](#_Toc171240583)

# Wprowadzenie

Projekt Model Xml Schema reprezentuje model obiektowy schematu OpenXml pobrany z plików XSD i zapisany w bazie danych MS Access.

Projekt składa się z dwóch modułów:

* ModelXmlSchema – moduł biblioteczny definiujący model obiektowy plików XSD.
* ModelXmlSchema App – aplikacji konsolowej do parsowania plików i zapisywania modelu w bazie danych.

Projekt czyta i parsuje pliki XSD zebrane w katalogu d:\VS\Docs\OpenXml\Schemas i zapisuje model schematu w bazie danych MS Access o nazwie XmlSchema.accdb położonej w ścieżce „d:\VS\Docs\OpenXml”. Ten sposób magazynowania danych został wybrany ze względu na dużą łatwość edycji rekordów bazy danych. Do odwzorowania modelu klas encyjnych w model relacyjnej bazy danych jest wykorzystywany framework Microsoft.­EntityFramework.Core.

# Pliki XSD i ich zawartość

Dokumenty pakietu Office (a w szczególności dokumenty programu Word) są w swojej istocie pakietami ZIP składającymi się z plików XML.

Schemat standardu OpenXml zapisany w plikach z rozszerzeniem XSD zawiera definicje elementów i atrybutów plików XML. Do definiowania elementów są wykorzystywane typy złożone (o nazwach zaczynających się od „CT\_”), a do definiowania atrybutów – typy proste (o nazwach zaczynających się od „ST\_”).

Przykład

<xsd:complexType name=„CT\_Color”>

<xsd:attribute name=„val” type=„ST\_HexColor” use=„required”/>

<xsd:attribute name=„themeColor” type=„ST\_ThemeColor” use=„optional”/>

<xsd:attribute name=„themeTint” type=„ST\_UcharHexNumber” use=„optional”/>

<xsd:attribute name=„themeShade” type=„ST\_UcharHexNumber” use=„optional”/>

</xsd:complexType>

Przetwarzane pliki XSD zebrano w jednym katalogu (tutaj: „D:\VS\Docs\OpenXml\Schemas”) i je sparsowano do bazy danych programu Access (plik Models.accdb).

## Struktura pliku

Przykładowy początek pliku XSD jest następujący:

<?xml version=„1.0” encoding=„utf-8”?>

<xsd:schema targetNamespace=http://purl.oclc.org/ooxml/drawingml/chart

xmlns=„http://purl.oclc.org/ooxml/drawingml/chart”

xmlns:xsd=„http://www.w3.org/2001/XMLSchema”

xmlns:a=„http://purl.oclc.org/ooxml/drawingml/main”

xmlns:r=„http://purl.oclc.org/ooxml/officeDocument/relationships”

xmlns:cdr=„http://purl.oclc.org/ooxml/drawingml/chartDrawing”

xmlns:s=„http://purl.oclc.org/ooxml/officeDocument/sharedTypes”

elementFormDefault=„qualified”

attributeFormDefault=„unqualified”

blockDefault=„#all”>

<xsd:import namespace=„http://purl.oclc.org/ooxml/officeDocument/relationships”  
 schemaLocation=„shared-relationshipReference.xsd”/>

<xsd:import namespace=„http://purl.oclc.org/ooxml/drawingml/main”   
 schemaLocation=„dml-main.xsd”/>

<xsd:import namespace=„http://purl.oclc.org/ooxml/drawingml/chartDrawing”  
 schemaLocation=„dml-chartDrawing.xsd”/>

<xsd:import namespace=„http://purl.oclc.org/ooxml/officeDocument/sharedTypes”  
 schemaLocation=„shared-commonSimpleTypes.xsd”/>

Element xsd:schema ma następujące atrybuty:

* targetNamespace – *docelową przestrzeń nazw*, czyli przestrzeń nazw, w której będą umieszczane elementy deklaracji,
* elementFormDefault – domyślną formę elementów. Wartość qualified oznacza, że elementy z docelowej przestrzeni nazw muszą być kwalifikowane przy użyciu prefiksu przestrzeni nazw),
* attributeFormDefault – domyślną formę atrybutów. Wartość unqualified oznacza, że atrybuty z docelowej przestrzeni nazw nie muszą być kwalifikowane przy użyciu prefiksu przestrzeni nazw.
* blockDefault – domyślną wartość atrybutu bloku dla elementów elementowych i typów złożonych w docelowej przestrzeni nazw. Atrybut bloku uniemożliwia użycie typu złożonego (lub elementu), który ma określony typ pochodny zamiast tego typu złożonego. Wartość #all zapobiega wszelkim pochodnym typom złożonym.

Na początku element xsd:schema deklaruje *importowane przestrzenie nazw*. Każdy element xsd:import ma dwa atrybuty:

* namespace – nazwę długą importowanej przestrzeni nazw w formie URL,
* schemaLocation – nazwę pliku, który deklaruje importowaną przestrzeń nazw.

Te deklaracje powinny być powiązane z deklaracjami prefiksów xmlns w elemencie xsd:schema.

Liczba przestrzeni nazw może być różna od liczby plików XSD, bowiem wiele plików odwołuje się do standardowych przestrzeni nazw XML i XSD. Są trzy takie przestrzenie:

* xml http://www.w3.org/XML/1998/namespace
* xsd http://www.w3.org/2001/XMLSchema
* inkml http://www.w3.org/2003/InkML

Długie nazwy przestrzeni nazw (URL) deklarowane w OpenXml zaczynają się od:

* http://purl.oclc.org/ooxml,
* http://schemas.microsoft.com,
* http://schemas.openxmlformats.org,
* urn:schemas-microsoft-com:office.

Nie ma jednego, ustalonego standardu dla nazw przestrzeni w OpenXml. Przy wczytywaniu plików XSD z różnych źródeł trzeba czasami zmieniać URL przestrzeni nazw dla dopasowania się do innych plików. W przeciwnym wypadku mogą się pojawić podwójne definicje tych samych elementów.

## Model obiektowy schematu XSD

Wszystkie elementy zawarte w pliku XSD opierają się na jednej, podstawowej klasie XmlSchemaObject (rys. 2). W hierarchii dziedziczenia najbardziej rozbudowana jest gałąź XmlSchemaAnnotated. Ta klasa abstrakcyjna reprezentuje wszystkie obiekty XmlSchema, które mogą mieć adnotacje. Jednocześnie obiekty tych klas tworzą główną strukturę schematu.

Obiekt XmlSchema może zawierać:

* XmlSchemaSimpleType – typy proste,
* XmlSchemaComplexType – typy złożone,
* XmlSchemaGroup – grupy elementów, które mogą być wykorzystywane w całości przy definicji typów złożonych,
* XmlSchemaAttributeGroup – grupy atrybutów, które mogą być wykorzystywane w całości przy definicji typów złożonych,
* XmlSchemaElement – elementy globalne, które przypisują typy złożone do elementów XML,
* XmlSchemaAttribute – atrybuty globalne, które przypisują typy proste do atrybutów XML.

Rys. 1.Hierarchia dziedziczenia klas XmlSchema

### Typy proste

Typy proste określają sposób interpretacji łańcuchów tekstowych – wartości atrybutów i zawartości tekstowej elementów. Reprezentowane są przez obiekty klasy XmlSimpleType. Ich znaczenie precyzuje ich zawartość, która może być typu:

* XmlSimpleTypeRestriction – określa ograniczenie wartości
* XmlSimpleTypeUnion – określa typy składowe typu unijnego,
* XmlSimpleTypeList – określa typy składowe typu listowego.

#### Ograniczenia wartości

Ograniczenia wartości typu prostego określa zawartość typu XsdSchema­SimpleTypeRestriction. Obiekt taki zazwyczaj odwołuje się do typu bazowego i może na niego nakładać ograniczenia, np. co do długości:

<xsd:simpleType name=„ST\_LongHexNumber”>

<xsd:restriction base=„xsd:hexBinary”>

<xsd:length value=„4”/>

</xsd:restriction>

</xsd:simpleType>

Ograniczenia typu prostego mają fasetki (ang. facets), które mogą być różnych typów (zależnie od typu bazowego):

* Dla typu xsd:hexBinary:
* XmlSchemaLengthFacet – określa długość ciągu w bajtach,
* Dla typu xsd:string i xsd:NCName:
* XmlSchemaMaxLengthFacet – określa maksymalną długość w znakach,
* XmlSchemaMinLengthFacet – określa minimalną długość w znakach,
* Dla typu xsd:string:
* XmlSchemaPatternFacet – określa wzorzec (wyrażenie regularne) łańcucha,
* XmlSchemaEnumerationFacet – zazwyczaj jest ich kilka, wyliczają dopuszczalne wartości łańcucha,
* XmlSchemaWhiteSpaceFacet – określa sposób traktowania białych znaków (spacji). Dopuszczalne wartości to preserve (zachowuj), replace (zamieniaj na spacje), collapse (znaki nowego wiersza, tabulatory, spacje, powroty karetki są zastępowane spacjami, spacje wiodące i końcowe są usuwane, a wiele spacji jest redukowanych do jednej spacji). Nie używane w przetwarzanych plikach XSD,
* Dla typów liczbowych:
* XmlSchemaMaxInclusiveFacet – określa maksymalną dozwoloną wartość,
* XmlSchemaMaxExclusiveFacet -– określa maksymalną niedozwoloną wartość
* XmlSchemaMinInclusiveFacet – określa minimalną dozwoloną wartość,
* XmlSchemaMinExclusiveFacet – określa minimalną niedozwoloną wartość,
* XmlSchemaFractionDigitsFacet – określa liczbę cyfr dziesiętnych (nie używane),
* XmlSchemaTotalDigitsFacet – określa całkowitą liczbę cyfr (nie używane).

Przykład ograniczenia długości

<xsd:simpleType name=„ST\_XmlName”>

<xsd:restriction base=„xsd:NCName”>

<xsd:minLength value=„1”/>

<xsd:maxLength value=„255”/>

</xsd:restriction>

</xsd:simpleType>

Przykład wzorca

<xsd:simpleType name=„ST\_TextScalePercent”>

<xsd:restriction base=„xsd:string”>

<xsd:pattern value=„0\*(600|([0-5]?[0-9]?[0-9]))%”/>

</xsd:restriction>

</xsd:simpleType>

Przykład wyliczenia wartości

<xsd:simpleType name=„ST\_Em”>

<xsd:restriction base=„xsd:string”>

<xsd:enumeration value=„none”/>

<xsd:enumeration value=„dot”/>

<xsd:enumeration value=„comma”/>

<xsd:enumeration value=„circle”/>

<xsd:enumeration value=„underDot”/>

</xsd:restriction>

</xsd:simpleType>

Przykład wartości minimalnej i maksymalnej

<xsd:simpleType name=„ST\_Integer255”>

<xsd:restriction base=„xsd:integer”>

<xsd:minInclusive value=„1”/>

<xsd:maxInclusive value=„255”/>

</xsd:restriction>

</xsd:simpleType>

Przykład wartości wykluczonych

<xsd:simpleType name=„ST\_FixedAngle”>

<xsd:restriction base=„ST\_Angle”>

<xsd:minExclusive value=„-5400000”/>

<xsd:maxExclusive value=„5400000”/>

</xsd:restriction>

</xsd:simpleType>

Przykład wartości wyliczanych

<xsd:simpleType name=„ST\_LayoutTarget”>

<xsd:restriction base=„xsd:string”>

<xsd:enumeration value=„inner”/>

<xsd:enumeration value=„outer”/>

</xsd:restriction>

</xsd:simpleType>

#### Unie

Unie to są typy proste, które mogą być interpretowane alternatywnie, a jako zawartość w schemacie mają obiekty typu XmlSchemaSimpleTypeUnion, np.:

<xsd:simpleType name=„ST\_Coordinate”>

<xsd:union memberTypes=„ST\_CoordinateUnqualified s:ST\_UniversalMeasure”/>

</xsd:simpleType>

#### Anonimowe składowe unii

W niektórych uniach jako składowe występują typy proste bez nazwy. Przykład:

<xsd:simpleType name=„ST\_DoubleOrAutomatic”>

<xsd:union memberTypes=„xsd:double”>

<xsd:simpleType>

<xsd:restriction base=„xsd:string”>

<xsd:enumeration value=„auto”/>

</xsd:restriction>

</xsd:simpleType>

</xsd:union>

</xsd:simpleType>

#### Listy

Typy listowe to typy proste, które określają taki sposób interpretacji łańcucha znaków jakby to miały być listy wartości. W schemacie te typy mają zawartość typu XsdSchemaSimpleTypeList, która określa typ elementów, np.:

<xsd:simpleType name=„ST\_Sqref”>

<xsd:list itemType=„ST\_Ref”/>

</xsd:simpleType>

### Typy złożone

Typy złożone określają sposób interpretacji elementów Xml. Definicje typów złożonych zawierają w sobie deklaracje atrybutów oraz ewentualnie struktury elementów składowych.

#### Atrybuty

Deklaracje atrybutów są reprezentowane przez obiekty typu XmlSchemaAttribute, które określają nazwę atrybutu XML, nazwę typu prostego, do którego atrybut się odnosi, i sposób użycia atrybutu.

Przykład

<xsd:complexType name=„CT\_Boolean”>

<xsd:attribute name=„val” type=„xsd:boolean” use=„optional” default=„true”/>

</xsd:complexType>

#### Odwołania do atrybutów globalnych

Jeśli XmlSchemaAttribute nie ma nazwy, to powinien mieć określoną właściwość RefName, to znaczy, że odwołuje się do atrybutu globalnego zdefiniowanego w innej (lub tej samej) przestrzeni nazw.

Przykład

<xsd:complexType name=„CT\_RelId”>

<xsd:attribute ref=„r:id” use=„required”/>

</xsd:complexType>

Deklaracje atrybutów występują nie tylko w typach złożonych, ale też w grupach atrybutów definiowanych na poziomie głównym schematu oraz na poziomie globalnym schematu.

#### Odwołania do grup atrybutów

W definicji typu złożonego obok atrybutów mogą występować referencje do grup atrybutów definiowanych na poziomie głównym schematu. Reprezentowane są przez obiekty typu XmlSchemaAttributeGroupRef, które przez właściwość RefName podają nazwę kwalifikowaną grupy atrybutów zdefiniowanej w innej (lub tej samej) przestrzeni nazw:

Przykład

<xsd:complexType name=„CT\_SVGBlip”>

<xsd:attributeGroup ref=„a:AG\_Blob”/>

</xsd:complexType>

#### Elementy i inne partykuły

Elementy XML występujące w typach złożonych są zorganizowane w hierarchiczną strukturę *partykuł* (ang. particles). Jest kilka typów partykuł. Wszystkie one są pochodne od abstrakcyjnego typu XmlSchemaParticle. Element XML jest reprezentowany przez obiekt typu XmlSchemaElement, która jest podtypem XmlSchemaParticle.

Typ XmlSchemaParticle ma cztery właściwości:

* MinOccurs – minimalna liczba wystąpień danej partykuły (liczba całkowita),
* MinOccursString – minimalna liczba wystąpień danej partykuły (łańcuch),
* MaxOccurs – maksymalna liczba wystąpień danej partykuły (liczba),
* MaxOccursString – maksymalna liczba wystąpień danej partykuły (łańcuch), unboundedoznacza brak ograniczenia.

Oprócz elementu wyróżnia się następujące partykuły:

* XmlSchemaAny – reprezentuje dowolny element lub inną zawartość,
* XmlSchemaGroupRef – reprezentuje odwołanie do zdefiniowanej grupy elementów.
* XmlSchemaAll – reprezentuje grupę elementów, które mogą pojawiać się w dowolnej kolejności. Każdy element może wystąpić raz, ale nie musi.
* XmlSchemaSequence – reprezentuje grupę elementów, które muszą pojawić się w określonej sekwencji.
* XmlSchemaChoice – reprezentuje grupę elementów do wyboru. Tylko jeden z podanych elementów może się pojawić.

Przykład sekwencji

<xsd:complexType name=„CT\_FontCollection”>

<xsd:sequence>

<xsd:element name=„latin” type=„CT\_TextFont” minOccurs=„1” maxOccurs=„1”/>

<xsd:element name=„ea” type=„CT\_TextFont” minOccurs=„1” maxOccurs=„1”/>

<xsd:element name=„cs” type=„CT\_TextFont” minOccurs=„1” maxOccurs=„1”/>

<xsd:element name=„font” type=„CT\_SupplementalFont” minOccurs=„0” maxOccurs=„unbounded”/>

<xsd:element name=„extLst” type=„CT\_OfficeArtExtensionList” minOccurs=„0” maxOccurs=„1”/>

</xsd:sequence>

</xsd:complexType>

Partykuła sekwencji występuje nawet, gdy typ złożony zawiera tylko jeden element.

Przykład partykuły Choice

<xsd:complexType name=„CT\_AnimationElementChoice”>

<xsd:choice minOccurs=„1” maxOccurs=„1”>

<xsd:element name=„dgm” type=„CT\_AnimationDgmElement”/>

<xsd:element name=„chart” type=„CT\_AnimationChartElement”/>

</xsd:choice>

</xsd:complexType>

W partykule wyboru powinno być zadeklarowanych wiele elementów (lub innych partykuł) składowych.

Przykład partykuły All

<xsd:complexType name=„CT\_DocPartPr”>

<xsd:all>

<xsd:element name=„name” type=„CT\_DocPartName” minOccurs=„1”/>

<xsd:element name=„style” type=„CT\_String” minOccurs=„0”/>

<xsd:element name=„category” type=„CT\_DocPartCategory” minOccurs=„0”/>

<xsd:element name=„types” type=„CT\_DocPartTypes” minOccurs=„0”/>

<xsd:element name=„behaviors” type=„CT\_DocPartBehaviors” minOccurs=„0”/>

<xsd:element name=„description” type=„CT\_String” minOccurs=„0”/>

<xsd:element name=„guid” type=„CT\_Guid” minOccurs=„0”/>

</xsd:all>

</xsd:complexType>

Partykuła Any występuje w ramach partykuły grupowej, najczęściej sekwencji.

Przykład partykuły Any

<xsd:complexType name=„CT\_GraphicalObjectData”>

<xsd:sequence>

<xsd:any minOccurs=„0” maxOccurs=„unbounded” processContents=„strict”/>

</xsd:sequence>

<xsd:attribute name=„uri” type=„xsd:token” use=„required”/>

</xsd:complexType>

Partykuła Any ma dwie właściwości:

* Namespace– określa przestrzenie nazw zawierające elementy, które mogą być używane. Możliwości są następujące:
* ##any – dozwolone są elementy z dowolnej przestrzeni nazw (jest to ustawienie domyślne),
* ##other – mogą być obecne elementy z dowolnej przestrzeni nazw, która nie jest przestrzenią nazw elementu nadrzędnego,
* ##local - elementy muszą pochodzić z lokalnej przestrzeni nazw,
* ##targetNamespace - mogą być obecne elementy z przestrzeni nazw elementu nadrzędnego,
* lista odwołań do przestrzeni nazw poprzez ich URI, — mogą być obecne elementy z rozdzielanej spacjami listy przestrzeni nazw. Mogą też wystąpić słowa kluczowe ##targetNamespace i ##local.
* ProcessContents – określa sposób walidacji elementów przez procesor XML. Możliwości są następujące:
* strict – procesor XML musi uzyskać schemat dla wymaganych przestrzeni nazw i zweryfikować elementy (jest to ustawienie domyślne),
* lax – tak samo jak strict, ale jeśli nie można uzyskać schematu, nie są zgłaszane żadne błędy,
* skip – procesor XML nie próbuje sprawdzać poprawności żadnych elementów z określonych przestrzeni nazw.

#### Elementy

Elementy są reprezentowane przez obiekty typu XmlSchemaElement, która pochodzi od typu XmlSchemaParticle i sama ma właściwości, np.:

* Name – nazwa elementu definiowanego lub nazwa elementu, do którego dany element się odwołuje,
* RefName – alternatywne odwołanie do elementu globalnego,
* ElementType – nazwa typu elementu.

Inne właściwości są pomijane przy parsowaniu.

Przykład

W poniższym przykładzie trzy elementy mają przypisania typów, które są zadeklarowane w tej samej przestrzeni nazw, a dwa do innej.

<xsd:complexType name=„CT\_TrendlineLbl”>

<xsd:sequence>

<xsd:element name=„layout” type=„CT\_Layout” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„tx” type=„CT\_Tx” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„numFmt” type=„CT\_NumFmt” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„spPr” type=„a:CT\_ShapeProperties” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„txPr” type=„a:CT\_TextBody” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„extLst” type=„CT\_ExtensionList” minOccurs=„0” maxOccurs=„1”/>

</xsd:sequence>

</xsd:complexType>

#### Odwołania do elementów globalnych

Elementy, które nie mają właściwości Name, powinny mieć właściwość RefName, przez którą odwołują się do elementów globalnych, zdefiniowanych na poziomie przestrzeni nazw.

Przykład

<xsd:complexType name=„CT\_TextMath”>

<xsd:sequence>

<xsd:element ref=„m:oMath”/>

</xsd:sequence>

</xsd:complexType>

#### Odwołania do grup elementów

W schemacie mogą być grupy elementów, do których można się odwoływać w różnych miejscach. Typ odwołania do grupy XmlSchemaGroupRef jest pochodny od typu XmlSchemaParticle i ma właściwość:

* RefName: string – nazwa grupy, do której odwołuje się dana referencja.

Przykład

<xsd:complexType name=„CT\_Drawing”>

<xsd:sequence>

<xsd:group ref=„EG\_Anchor” minOccurs=„0” maxOccurs=„unbounded”/>

</xsd:sequence>

</xsd:complexType>

### Grupy elementów

W schemacie można definiować grupy elementów, do których można się odwoływać w typach złożonych lub innych grupach elementów. Grupy są reprezentowane przez obiekty typu XmlSchemaGroup, które określają grupę partykuł.

Przykład

<xsd:group name=„EG\_DLblShared”>

<xsd:sequence>

<xsd:element name=„numFmt” type=„CT\_NumFmt” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„spPr” type=„a:CT\_ShapeProperties” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„txPr” type=„a:CT\_TextBody” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„dLblPos” type=„CT\_DLblPos” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„showLegendKey” type=„CT\_Boolean” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„showVal” type=„CT\_Boolean” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„showCatName” type=„CT\_Boolean” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„showSerName” type=„CT\_Boolean” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„showPercent” type=„CT\_Boolean” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„showBubbleSize” type=„CT\_Boolean” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„separator” type=„xsd:string” minOccurs=„0” maxOccurs=„1”/>

</xsd:sequence>

</xsd:group>

Do zdefiniowanej grupy można się odwołać przez element typu XmlSchemaGroupRef.

Przykład

<xsd:group name=„Group\_DLbl”>

<xsd:sequence>

<xsd:element name=„layout” type=„CT\_Layout” minOccurs=„0” maxOccurs=„1”/>

<xsd:element name=„tx” type=„CT\_Tx” minOccurs=„0” maxOccurs=„1”/>

<xsd:group ref=„EG\_DLblShared” minOccurs=„1” maxOccurs=„1”/>

</xsd:sequence>

</xsd:group>

### Grupy atrybutów

Podobnie w schemacie można definiować grupy atrybutów, do których można się odwoływać w typach złożonych lub innych grupach atrybutów. Grupy atrybutów są reprezentowane przez obiekty typu XmlSchemaAttributeGroup.

Przykład

<xsd:attributeGroup name=„AG\_Locking”>

<xsd:attribute name=„noGrp” type=„xsd:boolean” use=„optional” default=„false”/>

<xsd:attribute name=„noSelect” type=„xsd:boolean” use=„optional” default=„false”/>

<xsd:attribute name=„noRot” type=„xsd:boolean” use=„optional” default=„false”/>

<xsd:attribute name=„noChangeAspect” type=„xsd:boolean” use=„optional” default=„false”/>

<xsd:attribute name=„noMove” type=„xsd:boolean” use=„optional” default=„false”/>

<xsd:attribute name=„noResize” type=„xsd:boolean” use=„optional” default=„false”/>

<xsd:attribute name=„noEditPoints” type=„xsd:boolean” use=„optional” default=„false”/>

<xsd:attribute name=„noAdjustHandles” type=„xsd:boolean” use=„optional” default=„false”/>

<xsd:attribute name=„noChangeArrowheads” type=„xsd:boolean” use=„optional” default=„false”/>

<xsd:attribute name=„noChangeShapeType” type=„xsd:boolean” use=„optional” default=„false”/>

</xsd:attributeGroup>

Do zdefiniowanej grupy można się odwołać przez element typu XmlSchemaAttributeGroupRef.

Przykład

<xsd:complexType name=„CT\_PictureLocking”>

<xsd:sequence>

<xsd:element name=„extLst” type=„CT\_OfficeArtExtensionList” minOccurs=„0” maxOccurs=„1”/>

</xsd:sequence>

<xsd:attributeGroup ref=„AG\_Locking”/>

<xsd:attribute name=„noCrop” type=„xsd:boolean” use=„optional” default=„false”/>

</xsd:complexType>

### Elementy globalne

Elementy globalne są definiowane na poziomie głównym schematu. Odwołują się one do zdefiniowanych typów złożonych.

Przykład

<xsd:element name=„footnotes” type=„CT\_Footnotes”/>

Do elementów globalnych można, ale nie trzeba się odwoływać w schemacie.

### Atrybuty globalne

Podobnie atrybuty globalne są definiowane na poziomie głównym schematu. Odwołują się one do zdefiniowanych typów prostych.

Przykłady

<xsd:attribute name=„id” type=„ST\_RelationshipId”/>

<xsd:attribute name=„embed” type=„ST\_RelationshipId”/>

<xsd:attribute name=„link” type=„ST\_RelationshipId”/>

<xsd:attribute name=„dm” type=„ST\_RelationshipId” default=„„/>

<xsd:attribute name=„lo” type=„ST\_RelationshipId” default=„„/>

<xsd:attribute name=„qs” type=„ST\_RelationshipId” default=„„/>

<xsd:attribute name=„cs” type=„ST\_RelationshipId” default=„„/>

<xsd:attribute name=„blip” type=„ST\_RelationshipId” default=„„/>

<xsd:attribute name=„pict” type=„ST\_RelationshipId”/>

<xsd:attribute name=„href” type=„ST\_RelationshipId”/>

<xsd:attribute name=„topLeft” type=„ST\_RelationshipId”/>

<xsd:attribute name=„topRight” type=„ST\_RelationshipId”/>

<xsd:attribute name=„bottomLeft” type=„ST\_RelationshipId”/>

<xsd:attribute name=„bottomRight” type=„ST\_RelationshipId”/>

W odróżnieniu od elementów globalnych, do atrybutów globalnych trzeba się odwoływać w definicji typów złożonych (lub grup atrybutów), aby mogły być wykorzystane.

Przykład

<xsd:attributeGroup name=„AG\_Blob”>

<xsd:attribute ref=„r:embed” use=„optional” default=„„/>

<xsd:attribute ref=„r:link” use=„optional” default=„„/>

</xsd:attributeGroup>

...

<xsd:complexType name=„CT\_Blip”>

…

<xsd:attributeGroup ref=„AG\_Blob”/>

<xsd:attribute name=„cstate” type=„ST\_BlipCompression” use=„optional” default=„none”/>

</xsd:complexType>

### Adnotacje (pomijane)

Obiekty deklarowane w pliku XSD należące do klasy XmlSchemaAnnotated mają właściwość Annotation zawierającą opis obiektu.

# Model danych aplikacji

Obiekty XmlSchema odczytywane z plików XSD są zapisywane w relacyjnej bazie danych. W związku z tym w bibliotece ModelXmlSchema zdefiniowano klasy encyjne (z wykorzystaniem frameworku EntityFramework.Core), które reprezentują dane dopasowane do relacyjnej bazy danych.

## Pliki i przestrzenie nazw

Informacje o plikach schematu są przechowywane w tabeli SchemaFiles. Każdą encję reprezentuje klasa SchemaFile o właściwościach:

* Id: int {autonum, PK} – identyfikator encji,
* FileName: string – nazwa pliku (bez ścieżki i rozszerzenia),
* TargetNamespaceId: int {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw definiowanej w tym pliku,
* TargetNamespace? TargetNamespace – właściwość nawigacyjna do przestrzeni nazw definiowanej w tym pliku,
* UsedNamespaces: ICollection<UsedNamespace> – kolekcja przestrzeni nazw używanych w tym pliku.

Przestrzenie nazw są przechowywane w tabeli Namespaces. Każdą encję reprezentuje klasa Namespace o właściwościach podstawowych:

* Id: int {autonum, PK} – identyfikator encji,
* Url: string – URL przestrzeni nazw (długa nazwa),
* Prefix: string[10] – prefiks przestrzeni nazw powszechnie używany w schematach.

W każdym pliku schematu mogą być wykorzystywane inne przestrzenie nazw. Do rejestracji używanych przestrzeni nazw służy tabela UsedNamespaces. Encja UsedNamespace reprezentuje relację wiele-wiele pomiędzy plikami schematów i przestrzeniami nazw i ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* FileId: int {FK „SchemaFiles”} – identyfikator pliku schematu,
* NamespaceId: int {FK „Namespaces”} – identyfikator przestrzeni nazw,
* Prefix: string[10] – prefiks przestrzeni nazw używany w danym pliku.
* SchemaNamespace: Namespace – właściwość nawigacyjna prowadząca do przestrzeni nazw

Relacje między tabelami SchemaFiles, Namespaces i UsedNamespaces przedstawiono na rys. 2.

Prefiksy używane do odwołania się do tej samej przestrzeni nazw mogą być różne w różnych plikach. Zostały one ujednolicone w zebranych plikach XSD tak, aby mogły zostać jednoznacznie wpisane do tabeli Namespaces. Posłużyła to tego celu klasa pomocnicza NamespacePrefix, która została wykorzystana do tymczasowego przechowywania wyników kwerendy na tabelach SchemaFiles i UsedNamespaces.

Ponieważ nie wszystkie przestrzenie nazw zdefiniowane w plikach XSD są wykorzystywane jawnie w innych plikach, więc po przepisaniu prefiksów z tabeli UsedNamespaces do Namespaces wiele encji przestrzeni nazw nie miało ustalonych prefiksów. Ponieważ jednak prefiksy są wyświetlane przy przeglądaniu bazy danych, więc brakujące prefiksy zostały ustalone arbitralnie.

A screenshot of a computer

Description automatically generated

Rys. 2. Relacje między plikami a przestrzeniami nazw

## Elementy deklarowane w przestrzeni nazw

W przestrzeni nazw deklarowane są:

* Typy (proste i złożone),
* Atrybuty i elementy globalne,
* Grupy atrybutów i grupy elementów.

### Typy

W modelu danych ModelXmlSchema typy proste i złożone są przechowywane we wspólnej tablicy Types. Encja TypeDef reprezentuje klasę bazową dla klas SimpleType i ComplexType.

Encja TypeDef ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* Name: string {required} – nazwa typu,
* OwnerNamespaceId: int {FK „Namespaces”} – identyfikator przestrzeni nazw, w której jest zdefiniowany dany typ,
* IsComplex: bool {dbType „bit”} – określa, czy typ jest złożony (true) czy prosty (false).
* BaseTypeId: int? {FK „Types”} – identyfikator typu bazowego (opcjonalny),
* OwnerNamespace: Namespace – właściwość nawigacyjna do przestrzeni nazw, w której jest zdefiniowany dany typ,
* BaseType: TypeDef? – właściwość nawigacyjna do typu bazowego (opcjonalna),
* FullName: string => OwnerNamespace.Prefix + „:” + Name – pełna nazwa typu wyliczana wraz z prefiksem przestrzeni nazw.

Pełna nazwa typu jest stosowana do rejestrowania typów w słowniku typów przypisanym do każdej przestrzeni nazw. Tak więc encja Namespace ma dodatkowe właściwości reprezentujące zdefiniowane w niej typy. Są to:

* Types: ICollection<TypeDef> – właściwość nawigacyjna reprezentująca kolekcję typów (prostych i złożonych) zdefiniowanych w tej przestrzeni nazw,
* TypesDictionary: Dictionary<string, TypeDef> {notMapped} – słownik typów zdefiniowanych w tej przestrzeni nazw (do szybkiego wyszukiwania). Kluczem jest pełna nazwa typu.

### Typy proste

Typy proste są zapisywane w tabeli Types, a reprezentowane przez encję SimpleType, która jest pochodna od klasy TypeDef i ma sama następujące właściwości:

* Length: int – określenie długości dla typu bazowego hexBinary (w bajtach),
* MinLength: int – ograniczenie minimalnej długości łańcucha lub nazwy (w znakach),
* MaxLength: int – ograniczenie maksymalnej długości łańcucha lub nazwy (w znakach),
* MinInclusive: string – minimalna dozwolona wartość dla typu liczbowego,
* MaxInclusive: string – maksymalna dozwolona wartość dla typu liczbowego,
* MaxExclusive: string – maksymalna niedozwolona wartość dla typu liczbowego,
* MinExclusive: string – minimalna niedozwolona wartość dla typu liczbowego,
* HasPatterns: bool {dbType „bit”} – określa, czy typ prosty ma przypisane jakieś wzorce,
* IsEnum: bool {dbType „bit”} – określa, czy typ prosty jest wyliczany (ma przypisane jakieś składowe wyliczane),
* IsUnion: bool {dbType „bit”} – określa, czy typ prosty jest unią (ma przypisane jakieś składowe unii),
* IsList: bool {dbType „bit”} – określa, czy typ prosty jest listą (ma przypisane jakieś składowe unii),
* Patterns: ICollection<Pattern> – właściwość nawigacyjna do kolekcji wzorców,
* PatternsDictionary: Dictionary<string, Pattern> {notMapped} – słownik do szybkiego wyszukiwania wzorców (kluczem jest wartość wzorca),
* EnumValues: ICollection<EnumValue> – właściwość nawigacyjna do kolekcji wartości wyliczanych,
* EnumValuesDictionary: Dictionary<string, EnumValue > {notMapped} – słownik do szybkiego wyszukiwania wartości wyliczanych (kluczem jest nazwa wartości),
* UnionMembers: ICollection<UnionMember> – właściwość nawigacyjna do kolekcji składowych unii,
* UnionMembersDictionary: Dictionary<string, UnionMember> {notMapped} – słownik do szybkiego wyszukiwania składowych unii (kluczem jest pełna nazwa typu składowego),
* ListItems: ICollection<ListItem> – właściwość nawigacyjna do kolekcji składowych listy,
* ListItemsDictionary: Dictionary<string, ListItem> {notMapped} – słownik do szybkiego wyszukiwania składowych listy (kluczem jest pełna nazwa typu składowego).

Wartości dozwolone (minimalne i maksymalne) muszą być typu string, bo dla każdego typu bazowego mogą mieć różny zakres i format.

Flagi HasPattern, IsEnum, IsUnion i IsList służą do zwiększenia efektywności wyszukiwania typów prostych o szczególnych właściwościach (nie trzeba przeprowadzać kwerendy po innych tabelach bazy danych).

Uwaga

*W czasie implementacji rozwiązania rozważano zdefiniowanie oddzielnych encji typu wyliczeniowego, unii i listy. Umożliwiłoby to zmniejszenie obciążenia pamięci operacyjnej. Rozwiązanie to zostało odrzucone ze względu na odwołania do typów w atrybutach, elementach i innych typach. Odwołania takie mogą się pojawić przed definicją danego typu. Typ taki jest wówczas rejestrowany jako typ prosty, a jego szczegóły są ustalane później, gdy parser napotka definicję typu prostego w przestrzeni nazw. Nie wymaga to wówczas zmiany klasy encji, co byłoby problematyczne przy użyciu EntityFramework.Core.*

#### Wzorce (wyrażenia regularne)

Wzorce są zapisywane w osobnej tabeli Patterns. Chociaż w analizowanych plikach XSD nie stwierdzono różnych wzorców przypisanych do tego samego typu prostego, to teoretycznie istnieje taka możliwość, dlatego zastosowano osobną tabelę. Encja Pattern ma następujące właściwości.

* Id: int {autonum, PK} – identyfikator encji,
* OwnerTypeID: int {FK „Types”} – identyfikator typu prostego, do którego jest przypisany wzorzec,
* Value: string – tekst wzorca (wyrażenie regularne)
* OwnerType: SimpleType – właściwość nawigacyjna do typu prostego, do którego jest przypisany wzorzec.

#### Wartości wyliczane

Wartości typu wyliczeniowego są zapisywane w tabeli EnumValues. Encja EnumValue ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* TypeId: int {FK „Types”} – identyfikator typu prostego, dla którego podano tę wartość,
* Name: string – łańcuch znaków, który jest rozpoznawalny jako dana wartość,
* Value: int? – numer kolejny wartości (od zera).

Właściwość Name wbrew pozorom nie musi być nazwą. Może być łańcuchem pustym (ale nie null). Przykładowe rozpoznawalne łańcuchy są następujące:

* -+
* 00
* 1
* 1024,768
* 1024x768
* 1pic
* 35mm
* Aa

Typ wyliczeniowy najczęściej jako bazowy ma typ string, ale też często występuje token, a w pojedynczych przypadkach ST\_String i byte.

#### Składowe unii

Składowe unii są zapisywane w tabeli UnionMembers. Encja UnionMember ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* OwnerTypeId: int {FK „Types”} – identyfikator typu prostego, do którego należy ta składowa,
* MemberTypeId: int {FK „Types”} – identyfikator typu składowej (który może być zdefiniowany w tej samej lub innej przestrzeni nazw),
* OwnerType: SimpleType – właściwość nawigacyjna do typu prostego, do którego należy ta składowa
* MemberType: TypeDef – właściwość nawigacyjna do typu składowej. Najczęściej jest to typ prosty, ale model tego nie wymusza.

#### Anonimowe składowe unii

Jeśli przy definicji unii wystąpi typ prosty bez nazwy, to jest mu przypisywana automatycznie nazwa „\_anon\_*N*”, gdzie *N* jest kolejnym numerem typu anonimowego.

Przykład

<xsd:simpleType name=„ST\_DoubleOrAutomatic”>

<xsd:union memberTypes=„xsd:double”>

<xsd:simpleType> <!- temu typowi jest przypisywana nazwa "anon\_1" -->

<xsd:restriction base=„xsd:string”>

<xsd:enumeration value=„auto”/>

</xsd:restriction>

</xsd:simpleType>

</xsd:union>

</xsd:simpleType>

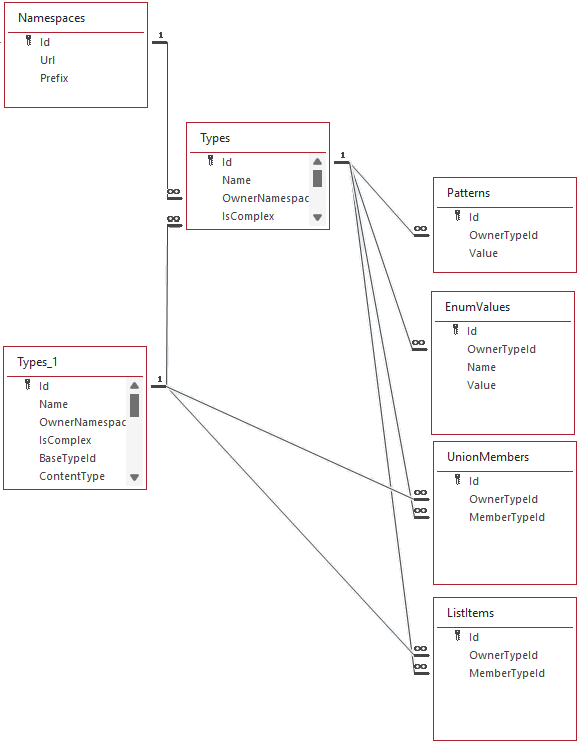
#### Elementy listy

Elementy składowe listy są zapisywane w tabeli ListItems. Encja ListItem ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* OwnerTypeId: int {FK „Types”} – identyfikator typu prostego, do którego należy ta składowa,
* MemberTypeId: int {FK „Types”} – identyfikator typu składowej (który może być zdefiniowany w tej samej lub innej przestrzeni nazw),
* OwnerType: SimpleType – właściwość nawigacyjna do typu prostego, do którego należy ta składowa
* MemberType: TypeDef – właściwość nawigacyjna do typu składowej. Najczęściej jest to typ prosty, ale model tego nie wymusza.

#### Relacje w ramach typów prostych

Reasumując, relacje między tabelami w ramach definicji typów prostych są przedstawione na rys. 3. Ze względu na specyfikę diagramu relacji programu MS Access tabela Types jest reprezentowana przez dwa obiekty graficzne: Types i Types\_1 (nie ma znaczenia, czy linia relacji prowadzi do obiektu Types czy Types\_1, bo oba reprezentują tę samą tabelę).



Rys. 3. Relacje między tabelami w ramach typów prostych

### Typy złożone

Typy złożone są rejestrowane w tabeli Types, a reprezentowane przez encję ComplexType, która jest pochodna od klasy TypeDef i ma sama następujące właściwości:

* ContentType: ContentType {dbType „byte”} – typ zawartości (TextOnly, Empty, ElementOnly, Mixed),
* HasAttributes: bool {dbType „bit”} – czy typ ma atrybuty (dla przyspieszenia wyszukiwania),
* ParticleId: int? {FK „Particles”} – identyfikator partycji opisującej zawartość elementu wyrażającego typ (opcjonalny),
* Attributes: ICollection<AttributeBase> – właściwość nawigacyjna do kolekcji atrybutów (zawsze istnieje, ale kolekcja może być pusta),
* AttributesDictionary: Dictionary<string, AttributeBase> {notMapped} – słownik atrybutów zdefiniowanych w danym typie (kluczem jest pełna nazwa atrybutu),
* Particle: Particle? – właściwość nawigacyjna do partycji opisującej zawartość elementy wyrażającego typ (opcjonalna).

### Atrybuty i grupy atrybutów

Atrybuty są reprezentowane przez abstrakcyjną klasę bazową AttributeBase oraz klasy potomne: AttributeDef, AttributeRef i AttributeGroupRef.

Encja AttributeBase ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* Type: AttributeType {dbType „byte”} – określenie typu konkretnego encji,
* OwnerTypeId: int? {FK „Types”} – identyfikator typu, w którym jest zdefiniowany dany atrybut,
* OwnerGroupId: int? {FK „AttributeGroup”} – identyfikator grupy atrybutów, w którym jest zdefiniowany dany atrybut,
* OwnerNamespaceId: int? {FK „Namespaces”} – identyfikator przestrzeni nazw, w którym jest zdefiniowany dany atrybut (dotyczy atrybutów globalnych),
* Name: string {required} – nazwa atrybutu,
* Use: AttributeUse? – określa sposób użycia atrybutu (dopuszczalne: None, Optional, Prohibited, Required).
* OwnerType: ComplexType? – właściwość nawigacyjna do typu złożonego, w którym jest zdefiniowany dany atrybut,
* OwnerGroup: AttributeGroup? – właściwość nawigacyjna do grupy atrybutów, w której jest zdefiniowany dany atrybut,
* OwnerNamespace: Namespace? – właściwość nawigacyjna do przestrzeni nazw, w której jest zdefiniowany dany atrybut (dotyczy atrybutów globalnych),
* FullName: string {derived}– pełna nazwa atrybutu, wyliczana wraz z prefiksem przestrzeni nazw.

Właściwości reprezentujące klucze obce: OwnerTypeId, OwnerGroupId i OwnerNamespaceId (oraz odpowiadające im właściwości nawigacyjne) są opcjonalne, ale dokładnie jedna z nich musi wystąpić.

Uwaga

*W czasie implementacji rozważano wystąpienie jednej właściwości klucza obcego (np.* OwnerId*) i rozróżnianie typów obiektów właścicielskich, ale w EntityFramework.Core jest to trudne do realizacji.*

#### Definicja atrybutu

Encja AttributeDef, pochodna od AttributeBase, ma następujące właściwości:

* ValueTypeId: int {FK „Types”} – identyfikator typu wartości,
* DefaultValue: string – domyślna wartość atrybutu (w zapisie tekstowym),
* IsFixed: bool – określa, czy wartość domyślna atrybutu jest wartością ustaloną (nie używane w analizowanych plikach XSD),
* ValueType: TypeDef – właściwość nawigacyjna do typu wartości.

#### Definicja odwołania do atrybutu

Encja AttributeRef, pochodna od AttributeBase, reprezentuje odwołanie danego atrybutu do definicji atrybutu globalnego. Ma następujące właściwości:

* RefAttributeId: int {FK „Attributes”} – identyfikator atrybutu globalnego, do którego odwołuje się dany atrybut,
* RefAttribute: AttributeBase – właściwość nawigacyjna do atrybutu globalnego.

#### Odwołanie do grupy atrybutów

Encja AttributeGroupRef, pochodna od AttributeBase, reprezentuje odwołanie danego atrybutu do grupy atrybutów. Ma następujące właściwości:

* GroupId: int {FK „AttributeGroups”} – identyfikator grupy atrybutów, do której odwołuje się dany atrybut,
* RefGroup: AttributeGroup – właściwość nawigacyjna do grupy atrybutów.

#### Atrybuty globalne

Atrybuty globalne są zdefiniowane bezpośrednio w przestrzeni nazw. Mają ustawioną właściwość OwnerNamespaceId (i odpowiadającą OwnerNamespace).

Atrybuty globalne są rejestrowane w encji Namespace poprzez następujące właściwości:

* Attributes: List<AttributeBase> – właściwość nawigacyjna reprezentująca kolekcję atrybutów globalnych zdefiniowanych w tej przestrzeni nazw,
* AttributesDictionary: Dictionary<string, AttributeBase > {notMapped} – słownik atrybutów globalnych zdefiniowanych w tej przestrzeni nazw (do szybkiego wyszukiwania). Kluczem jest pełna nazwa atrybutu.

#### Grupy atrybutów

Grupy atrybutów, podobnie jak atrybuty globalne, są zdefiniowane bezpośrednio w przestrzeni nazw. Są zapisywane w osobnej tabeli, AttributeGroups. Encja AttributeGroup *nie pochodzi* od klasy AttributeBase. Ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* OwnerNamespaceId: int {FK „Namespaces”} – identyfikator przestrzeni nazw, w którym jest zdefiniowana dana grupa,
* Name: string {required} – nazwa grupy,
* Attributes: List<AttributeBase> – właściwość nawigacyjna do kolekcji atrybutów zdefiniowanych w danej grupie,
* AttributesDictionary: Dictionary<string, AttributeBase> {notMapped} – słownik atrybutów zdefiniowanych w danej grupie (kluczem jest pełna nazwa atrybutu),
* OwnerNamespace: Namespace – właściwość nawigacyjna do przestrzeni nazw, w której jest zdefiniowany dana grupa,
* FullName: string {derived}– pełna nazwa grupy, wyliczana wraz z prefiksem przestrzeni nazw.

Uwaga

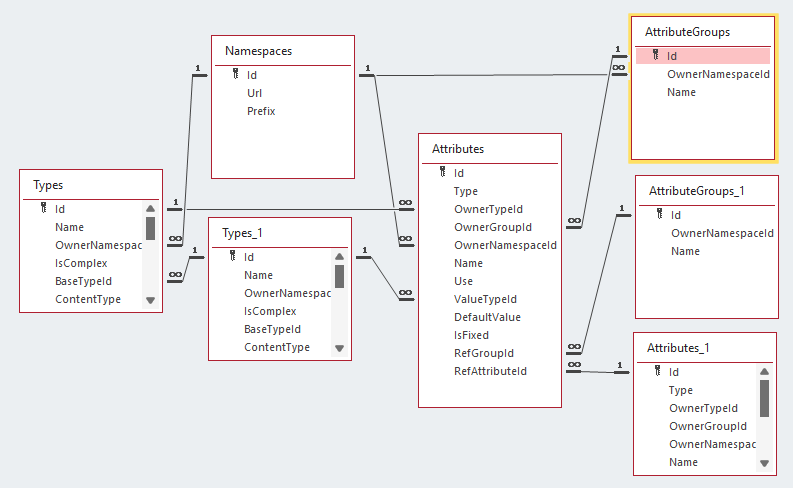
*W czasie implementacji rozważano zdefiniowanie klasy* AttributeGroup *jako pochodnej od* AttributeBase*. Odrzucono to rozwiązanie, gdyż grupa atrybutów nie może być zdefiniowana nigdzie indziej jak tylko na poziomie przestrzeni nazw.*

Grupy atrybutów są rejestrowane w encji Namespace poprzez następujące właściwości:

* AttributeGroups: List<AttributeGroup> – właściwość nawigacyjna reprezentująca kolekcję grup atrybutów zdefiniowanych w tej przestrzeni nazw,
* AttributeGroupsDictionary: Dictionary<string, AttributeGroup> {notMapped} – słownik grup atrybutów zdefiniowanych w tej przestrzeni nazw (do szybkiego wyszukiwania). Kluczem jest pełna nazwa grupy.

#### Relacje dotyczące atrybutów

Relacje atrybutów, grup atrybutów, typów i przestrzeni nazw są przedstawione na poniższym rysunku.



Rys. 4. Relacje atrybutów, grup atrybutów, typów i przestrzeni nazw

### Partykuły, elementy i grupy elementów

Wszystkie partykuły są rejestrowane we wspólnej tabeli Particles. Podstawą hierarchii partykuł jest klasa abstrakcyjna Particle. Klasa ta ma potomne klasy:

* Any – reprezentuje deklarację XmlSchemaAny,
* Element – reprezentuje deklarację XmlSchemaElement,
* ElementGroupRef – reprezentuje deklarację XmlSchemaGroupRef,
* PartitionGroup – abstrakcyjna klasa grupująca inne partycje. Ma klasy potomne:
* All – reprezentuje deklarację XmlSchemaAll,
* Sequence – reprezentuje deklarację XmlSchemaSequence,
* Choice – reprezentuje deklarację XmlSchemaChoice,

Klasa Partition ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* ParticleType: ParticleType {dbType „byte”}– konkretny typ partykuły (Any, Element, GroupRef, All, Choice, Sequence),
* OwnerNamespaceId: int? {FK „Namespaces”} – identyfikator przestrzeni nazw, w której jest zadeklarowana dana partykuła (dotyczy partykuł zdefiniowanych na poziomie schematu – grup elementów i elementów globalnych),
* OwnerGroupId: int? {FK „ElementGroups”} – identyfikator grupy elementów, w której jest zadeklarowana dana partykuła (dotyczy partykuł zdefiniowanych w grupie elementów),
* OwnerTypeId: int? {FK „Types”} – identyfikator typu złożonego, w którym jest zadeklarowana dana partykuła (dotyczy partykuł zdefiniowanych w typie złożonym),
* OwnerParticleId: int? {FK „Particles”} – identyfikator partykuły nadrzędnej, w której jest zadeklarowana dana partykuła (dotyczy partykuł zagnieżdżonych),
* OrdNum: int? – numer porządkowy partykuły zadeklarowanej w ramach innej partykuły (liczony od 1),
* MinOccurs: int? – minimalna liczba wystąpień danej partykuły,
* MaxOccurs: int? – maksymalna liczba wystąpień danej partykuły (wartość MaxInt oznacza nieograniczoną liczbę),
* OwnerNamespace: Namespace? – właściwość nawigacyjna do przestrzeni nazw, w której jest zadeklarowana dana partykuła,
* OwnerGroup: ElementGroup? – właściwość nawigacyjna do grupy elementów, w której jest zadeklarowana dana partykuła,
* OwnerType: TypeDef? – właściwość nawigacyjna do typu, w którym jest zadeklarowana dana partykuła,
* OwnerParticle: ParticleGroup? – właściwość nawigacyjna do grupy elementów, w której jest zadeklarowana dana partykuła,
* HostingNamespace: Namespace {derived} – nadrzędna przestrzeń nazw, w której jest zadeklarowana dana partykuła (obliczana na podstawie innych obiektów nadrzędnych).

#### Partykuła Any

Partykuła Any dopuszcza dowolny element (również te spoza schematu). Klasa Any odzwierciedla deklarację XmlSchemaAny deklarując dwie właściwości:

* Namespace: string? – opcjonalne określenie przestrzeni nazw, których elementy można używać w tym miejscu. Domyślnie to ##any,
* ProcessContents: ContentProcessing? – opcjonalne określenie sposobu przetwarzania zawartości. Domyślnie Strict.

Partykuła Any jest tylko rejestrowana. Żadne dalsze przetwarzanie nie jest stosowane.

#### Partykuły grupowe (Sequence, Choice, All)

Encja ParticleGroup ma właściwości grupujące inne partycje:

* Items: List<Particle> – właściwość nawigacyjna do elementów składowych,
* ItemsDictionary<int, Particle> – słownik elementów składowych. Kluczem jest numer porządkowy elementu w deklaracji.

Encje konkretne (Sequence, Choice, All) nie mają żadnych specjalnych właściwości.

#### Elementy

Elementy są reprezentowane przez encję Element, która pochodzi od typu Particle i sama ma następujące właściwości:

* RefElementId: int? {FK „Particle”} – identyfikator elementu globalnego, do którego odwołuje się dany element,
* Name?: string – nazwa elementu definiowanego lub nazwa elementu, do którego dany element się odwołuje,
* RefTypeId: int? {FK „Types”} – identyfikator typu, do którego odwołuje się dany element,
* RefElement: Element? – właściwość nawigacyjna do elementu, do którego odwołuje się dany element,
* RefType: ComplexType? – właściwość nawigacyjna do typu złożonego, do którego odwołuje się dany element,
* FullName: string {derived} – pełna nazwa elementu (obliczana).

#### Odwołania do grup elementów

Encja odwołania do grupy, ElementGroupRef jest pochodna od Particle i ma dwie właściwości:

* RefGroupId: int {FK „ElementGroups”} – identyfikator grupy elementów, do której odwołuje się dana partycja,
* RefGroup: ElementGroup – właściwość nawigacyjna do grupy elementów, do której odwołuje się dana partycja.

#### Elementy globalne

Elementy globalne są definiowane na poziomie przestrzeni nazw. Odwołują się one do zdefiniowanych typów złożonych. Encja elementu globalnego ma ustawioną właściwość OwnerNamespaceId (i odpowiadającą OwnerNamespace).

Elementy globalne są rejestrowane w encji Namespace poprzez następujące właściwości:

* Elements: List<Element> – właściwość nawigacyjna reprezentująca kolekcję elementów globalnych zdefiniowanych w tej przestrzeni nazw,
* ElementsDictionary: Dictionary<string, Element > {notMapped} – słownik elementów globalnych zdefiniowanych w tej przestrzeni nazw (do szybkiego wyszukiwania). Kluczem jest pełna nazwa elementu.

#### Grupy elementów

Grupy elementów, podobnie jak elementy globalne, są zdefiniowane bezpośrednio w przestrzeni nazw. Są zapisywane w osobnej tabeli, ElementGroups. Encja ElementGroup ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* OwnerNamespaceId: int {FK „Namespaces”} – identyfikator przestrzeni nazw, w którym jest zdefiniowana dana grupa,
* Name: string {required} – nazwa grupy,
* ParticleId: int? – identyfikator partycji, która definiuje zawartość grupy,
* Namespace: Namespace – właściwość nawigacyjna do przestrzeni nazw, w którym jest zdefiniowana dana grupa,
* Particle?: ParticleGroup – właściwość nawigacyjna do partycji, która definiuje zawartość grupy (musi to być partycja grupowa),
* FullName: string {derived}– pełna nazwa grupy, wyliczana wraz z prefiksem przestrzeni nazw.

Uwaga

*Grupa elementów w oryginalnej nomenklaturze XmlSchema jest nazywana po prostu grupą. W modelu danych encji stosuje się nazwę* ElementGroup*, aby upodobnić ją do nazwy* AttributeGroup*.*

Grupy elementów są rejestrowane w encji Namespace poprzez następujące właściwości:

* ElementGroups: List<ElementGroup> – właściwość nawigacyjna reprezentująca kolekcję grup elementów zdefiniowanych w tej przestrzeni nazw,
* ElementGroupsDictionary: Dictionary<string, ElementGroup> {notMapped} – słownik grup elementów zdefiniowanych w tej przestrzeni nazw (do szybkiego wyszukiwania). Kluczem jest pełna nazwa grupy.

#### Relacje dotyczące partykuł

Relacje partycji i grup elementów są przedstawione na poniższym rysunku.

A screenshot of a computer

Description automatically generated

Rys. 5. Relacje związane z partykułami i grupami elementów

# Kontekst danych

Kontekst danych reprezentuje klasa XmlSchemaDbContext. Klasa ta:

1. Deklaruje zbiory danych (DbSet) reprezentujące tabele.
2. Deklaruje kolekcję elementów (Elements) reprezentującą podzbiór zbioru danych Particles.
3. Deklaruje słowniki plików i przestrzeni nazw (FilesDictionary i NamespacesDictionary). Słowniki te umożliwiają wyszukiwanie odpowiednich encji bez odwoływania się do bazy danych.
4. W konstruktorze:
   1. Ustala nazwę pliku bazy danych przekazaną przez parametr.
   2. Upewnia się, że baza danych została utworzona,
   3. Ustala niektóre właściwości bazy danych programu MS Access korzystając z mechanizmu automatyzacji COM w metodzie SetupAccessDatabase.
5. W metodzie OnConfiguring włącza użycie mechanizmu Jet MS Access.
6. W metodzie OnModelCreating tworzy relacje między encjami.
7. W metodzie LoadFiles inicjuje słowniki plików FilesDictionary i przestrzeni nazw NamespacesDictionary. Do zbiorów danych SchemaFiles i Namespaces przypisuje procedury obsługi zdarzenia dodania encji, które dodają encje do odpowiedniego słownika.
8. W metodzie LoadNamespaces inicjuje słowniki typów, atrybutów i inne, które są zdefiniowane w poszczególnych encjach.

W tym modelu korzysta się z ładowania chętnego (eagerly loading) zamiast domyślnej metody ładowania leniwego (lazy loading). Encje są indeksowane lokalnie (w pamięci aplikacji) przy użyciu słowników. Minimalizuje to odwołania do bazy danych.

Metoda SetupAccessDatabase ułatwia przeglądanie utworzonej i wypełnionej bazy danych w programie MS Access. Tworzy kwerendy danych i tabele reprezentujące typy wyliczeniowe. Ponadto ustawia właściwości związane z kolumnami danych tak, że MS Access wyświetla odpowiednie nazwy zamiast wartości liczbowych.

Ponieważ EntityFramework.Core nie zapewnia takich możliwości, więc metoda SetupAccessDatabase wykorzystuje interfejs COM (Component Object Model) do automatyzacji programu MS Access. Mechanizm ten jest zaimplementowany w pakietach MicrosoftOffice.Interop.Access i MicrosoftOffice.Interop.Access.Dao, które korzystają z pakietu MicrosoftOfficeCore15. Ten ostatni pakiet jest ściągnięty w wersji dla .NetFramework (a nie dla .Net8.0, dla którego jest skompilowany cały projekt). Przy kompilacji jest wyświetlane ostrzeżenie, ale automatyzacja działa.

Jedyny problem następuje z zamknięciem interfejsu użytkownika programu MS Access. Jest to znany problem przy wykorzystywaniu mechanizmu COM zaimplementowanych w kodzie natywnym (C++) w programie w kodzie zarządzanym (C#). Dla rozwiązania tego programu na koniec metody SetupAccessDatabase jest wywoływana metoda KillMsAccess, która „zabija” wszystkie procesy o nazwie „MSACCESS”. Skutkiem ubocznym jest zamykanie tej instancji programu MS Access, która została wcześniej otwarta przez użytkownika.

# Parsowanie modelu schematu

Za parsowanie odpowiada klasa publiczna XmlSchemaParser zadeklarowana w module ModelXmlSchemaApp.

Klasa XmlSchemaParser ma główną metodą publiczną ParseSchemaFiles, dla której podaje się:

* sourceXsdPath: string – ścieżkę do katalogu zawierającego pliki XSD,
* dbFilename: string – pełną nazwę pliku z bazą danych MS Access.

Metoda ta:

1. Ładuje wszystkie pliki XSD z podanego katalogu do zbioru XmlSchemaSet i kompiluje ten zbiór.
2. Tworzy wspólny dla wszystkich innych metod obiekt dbContext klasy XmlSchemaDbContext.
3. Przegląda pliki ze zbioru XmlSchemaSet tworząc encje SchemaFile, SchemaNamespace i SchemaUsedNamespace i wypełniając odpowiednie tabele w bazie danych.
4. Ustala prefiksy przestrzeni nazw w tabeli SchemaNamespaces wykonując kwerendę grupującą na tabeli SchemaUsedNamespaces.
5. Parsuje schematy ze zbioru XmlSchemaSet.

W czasie parsowania elementów schematu program sprawdza, czy istnieją odpowiednie encje w słownikach wyszukując je po odpowiednich kluczach drugoplanowych. Jeśli encja nie istnieje, to ją tworzy i dodaje je do kontekstu danych.

Poprawne działanie programu jest zapewnione dzięki odpowiedniej aktualizacji właściwości nawigacyjnych i słowników, co jest zapewniane przez zdefiniowanie odpowiednich relacji w klasie XmlSchemaDbContext w metodzie OnModelCreating oraz w metodach LoadFiles i LoadNamespaces. Brak odpowiedniej aktualizacji danych w tych metodach powoduje, że program parsujący nie znajduje encji w pamięci operacyjnej i próbuje je dodawać w bazie danych, co kończy się wyjątkiem programu.

Na koniec program parsujący wyświetla statystykę:

* SchemaFilesTotal, SchemaFilesAdded;
* NamespacesTotal, NamespacesAdded, NamespacesUpdated;
* UsedNamespacesTotal, UsedNamespacesAdded;
* SimpleTypesTotal, SimpleTypesAdded, SimpleTypesUpdated;
* ComplexTypesTotal, ComplexTypesAdded, ComplexTypesUpdated;
* AttributeDefsTotal, AttributeDefsAdded, AttributeDefsUpdated;
* AttributeRefsTotal, AttributeRefsAdded, AttributeRefsUpdated;
* AttributeGroupsTotal, AttributeGroupsAdded;
* AttributeGroupRefsTotal, AttributeGroupRefsAdded;
* EnumValuesTotal, EnumValuesAdded;
* PatternsTotal, PatternsAdded;
* ListItemsTotal, ListItemsAdded, ListItemsUpdated;
* UnionMembersTotal, UnionMembersAdded, UnionMembersUpdated;
* ElementGroupsTotal, ElementGroupsAdded;
* ElementGroupRefsTotal, ElementGroupRefsAdded, ElementGroupRefsUpdated;
* ElementsTotal, ElementsAdded, ElementsUpdated;
* OtherParticlesTotal, OtherParticlesAdded, OtherParticlesUpdated;

Nazwy liczników mówią same za siebie. Przy dodawaniu encji mogą być one czasem dodatkowo aktualizowane po dodaniu. Stąd pojawiają się też niezerowe liczniki Updated nawet, jeśli początkowo baza danych jest pusta.

Partykuły inne to partykuły nie będące elementami ani odwołaniami do grup elementów.

# Wyniki i uwagi końcowe

W aplikacji przeanalizowano i sparsowano 74 poniższe pliki XSD:

1. dml-chart.xsd
2. dml-chartDrawing.xsd
3. dml-compatibility.xsd
4. dml-lockedCanvas.xsd
5. dml-main.xsd
6. dml-picture.xsd
7. dml-spreadsheetDrawing.xsd
8. dml-wordprocessingDrawing.xsd
9. ink-2010-main.xsd
10. office-2019-extlst.xsd
11. office-drawing-2010-chartDrawing.xsd
12. office-drawing-2010-diagram.xsd
13. office-drawing-2010-main.xsd
14. office-drawing-2010-picture.xsd
15. office-drawing-2012-chart.xsd
16. office-drawing-2012-chartStyle.xsd
17. office-drawing-2012-main.xsd
18. office-drawing-2012-timeslicer.xsd
19. office-drawing-2013-main-command.xsd
20. office-drawing-2014-chart.xsd
21. office-drawing-2014-chart-ac.xsd
22. office-drawing-2014-chartex.xsd
23. office-drawing-2014-main.xsd
24. office-drawing-2016-11-diagram.xsd
25. office-drawing-2016-11-main.xsd
26. office-drawing-2016-12-diagram.xsd
27. office-drawing-2016-ink.xsd
28. office-drawing-2016-SVG-main.xsd
29. office-drawing-2017-03-chart.xsd
30. office-drawing-2017-decorative.xsd
31. office-drawing-2017-model3d.xsd
32. office-drawing-2018-animation.xsd
33. office-drawing-2018-animation-model3d.xsd
34. office-drawing-2018-hyperlinkcolor.xsd
35. office-drawing-2018-sketchyshapes.xsd
36. office-drawing-2020-classificationShape.xsd
37. office-drawing-2021-livefeed.xsd
38. office-drawing-2021-scriptlink.xsd
39. office-drawing-2022-imageformula.xsd
40. office-drawing-comments-2020-reactions.xsd
41. office-word-2010-wordml.xsd
42. office-word-2010-wordml-2.xsd
43. office-word-2010-wordprocessingCanvas.xsd
44. office-word-2010-wordprocessingDrawing.xsd
45. office-word-2010-wordprocessingGroup.xsd
46. office-word-2010-wordprocessingShape.xsd
47. office-word-2012-wordml.xsd
48. office-word-2012-wordprocessingDrawing.xsd
49. office-word-2015-wordml-symex.xsd
50. office-word-2016-wordml-cid.xsd
51. office-word-2018-cex.xsd
52. office-word-2018-wordml.xsd
53. office-word-2020-oembed.xsd
54. office-word-2020-sdtdatahash.xsd
55. office-word-2023-word16du.xsd
56. pml.xsd
57. shared-additionalCharacteristics.xsd
58. shared-bibliography.xsd
59. shared-commonSimpleTypes.xsd
60. shared-customXmlDataProperties.xsd
61. shared-customXmlSchemaProperties.xsd
62. shared-documentPropertiesCustom.xsd
63. shared-documentPropertiesExtended.xsd
64. shared-documentPropertiesVariantTypes.xsd
65. shared-math.xsd
66. shared-relationshipReference.xsd
67. sml.xsd
68. vml-main.xsd
69. vml-officeDrawing.xsd
70. vml-presentationDrawing.xsd
71. vml-spreadsheetDrawing.xsd
72. vml-wordprocessingDrawing.xsd
73. wml.xsd
74. xml.xsd

Wszystkie prefiksy przestrzeni nazw ujednolicono, uzupełniono i zapisano w pliku Prefixes.csv. Prefiksy mają znaczenie jedynie przy przeglądaniu zawartości bazy danych. Chociaż starano się zapewnić zgodność prefiksów z deklaracjami stosowanymi w plikach XSD oraz dokumentach Office, to w wielu przypadkach ustalono je arbitralnie.

Wyniki

Liczby obiektów rozpoznanych i zapisanych w bazie danych przedstawiono w poniższej tabeli.

Tab. 1. Podsumowanie wyników

|  |  |
| --- | --- |
| Obiekty | Liczba |
| Pliki | 74 |
| Przestrzenie nazw | 76 |
| Używane przestrzenie nazw | 204 |
| Typy proste | 625 |
| Typy złożone | 1690 |
| Definicje atrybutów | 3411 |
| Odwołania do atrybutów | 140 |
| Grupy atrybutów | 31 |
| Odwołania do grup atrybutów | 107 |
| Wartości wyliczane | 3360 |
| Wzorce | 22 |
| Elementy list | 11 |
| Składowe unii | 74 |
| Grupy elementów | 89 |
| Odwołania do grup elementów | 206 |
| Elementy | 3975 |
| Partykuły inne | 1190 |

Uwaga dotycząca wydajności programu

Dzięki zastosowaniu słowników encji zależnych, przy wyszukiwaniu encji po nazwie jest przeszukiwany lokalny słownik bez odwoływania się do bazy danych. Dało to skrócenie czasu całkowitego przetwarzania z prawie 7 min (przy pustej bazie danych) do 7 sekund (przy braku konieczności zmian w bazie danych).

Inne uwagi

1. W sieci Web istnieje przynajmniej kilka różnych źródeł (katalogów) z plikami schematu OpenXml. Źródła różnią się liczbą i organizacją plików. W aktualnej wersji aplikacji zebrano i wyselekcjonowano źródła z dwóch katalogów:

* <https://msdn.microsoft.com/en-us/library/dd979921(v=office.12).aspx>
* <https://schemas.liquid-technologies.com/OfficeOpenXML/2006/>

Może się okazać, że nie wszystkie elementy rzeczywiście występujące w plikach aplikacji Office zostały w nich ujęte.

1. Po zebraniu plików XSD z różnych źródeł konieczne było skorygowanie przestrzeni nazw i deklaracji xsd:Include tak, aby uniknąć klasyfikowania tych samych typów do różnych przestrzeni nazw.
2. W czasie kompilacji schematów XSD pojawiły się liczne błędy związane ze sprawdzaniem poprawności właściwości defaultValue dla typów procentowych. Zastosowana metoda Compile z klasy XmlSchemaSet nie akceptowała wartości „0”, gdy typ procentowy wymagał na końcu znaku „%”. Aby uniknąć wielokrotnego poprawiania wartości wprowadzono dodatkowy typ prosty ST\_Zero akceptujący tylko zero (bez procenta) i zmieniono definicje typów procentowych na unie z typem ST\_Zero.
3. W tabeli wzorców zaobserwowano kilka identycznych wzorców opisujących typ ST\_Guid. Okazało się, że typ ten był deklarowany w kilku schematach. Aby uniknąć sztucznego powielania typów przeniesiono typ ST\_Guid do pliku shared-CommonSimpleTypes.xsd, a odwołania do typów lokalnych ST\_Guid zamieniono na odwołania do tego właśnie typu.
4. Nie analizowano i nie zapisywano adnotacji do definicji. Występowały one sporadycznie i nie zostały uznane za cenne informacje.

Uwaga podsumowująca

Po zestawieniu wyników parsowania plików XSD zapisanych w bazie danych XmlSchema z wynikami parsowania innych źródeł informacji o standardzie OpenXml mogą się pojawić różnice w postaci brakujących lub nadmiarowych typów, elementów i atrybutów. Dlatego do wyników niniejszej analizy należy podchodzić ostrożnie.