OOXML の管理手法
ハイブリッド型 XML データベースへの格納

小西　啓介（ソラン株式会社）

概要　Office Open XML File Formats (OOXML) [1]は，単一の XML ファイルではなく，複数の XML ファイルなどを圧縮した形式である．筆者は，OOXML 文書をファイルサーバではなくハイブリッド型 XML データベースに格納することによって，効率よく管理（閲覧・復元・更新）を行えるシステムを設計・実装した．OOXML 文書は，異なるコンテンツタイプを持つ多数のパーツを柔軟に組み上げることを可能にしたフォーマットであり，そのために各パーツへのアクセスに多くの処理を要する．複数の異なる処理パターンに対して，どのような DB スキーマを用いるべきかについても議論する．

1. はじめに

Office Open XML File Formats (OOXML) [1]が，2008 年に ISO/IEC 29500 として標準化されてから 2 年が経過した．オフィススイーツの対応の広がりと共に，少しずつビジネスシーンでの OOXML の利用が始まっている．現状では，OOXML 文書は従来のオフィスドキュメントと同様にファイルとしてファイルサーバなどで管理する方法が一般的である．しかし，OOXML 文書が主に XML ファイルで構成されている特徴を活かして，効率的に管理（閲覧・復元・更新）するためには，OOXML 文書の各要素がデータベースに格納されていることが望ましい．例えば，多数の画像を含む複数の OOXML 文書について，テキストの一部を閲覧・更新する際に，1 つ 1 つ展開し文書全体をメモリ上に読み込むのではなく，必要な部分のみにアクセスできる方が効率的である．

筆者は，ハイブリッド型 XML データベース[2]を利用することにより，OOXML 文書を効率よく管理する手法を考案し，実装した．本論文では，この実装から得られた知見に基づき，OOXML をどのようにハイブリッド型 XML データベースで処理するかを議論する．第 2 章では，OOXML の仕様と，OOXML 文書内のパーツへのアクセスの複雑さについて述べる．第 3 章では，ハイブリッド型 XML データベースを用いた OOXML 文書の格納法について議論する．第 4 章では，復元性や妥当性検証など，その他の知見について議論し，第 5 章でまとめと今後の展望について述べる．

2. OOXML の仕様

2.1 仕様の構成

OOXML には，用途ごとに WordprocessingML (WML)，SpreadsheetML (SML)，PresentationML (PML) の 3 種類のオフィス文書定義がある．オフィス文書内で共通に使用される情報については，DrawingML など共通の定義となっている．OOXML を格納するファイルコンテナとして OPC (Open Packaging Conventions) [1]と呼ばれる方式が用いられ ZIP 形式で圧縮した状態で管理を行っている．OOXML の仕様構成は図 1 の通りである．

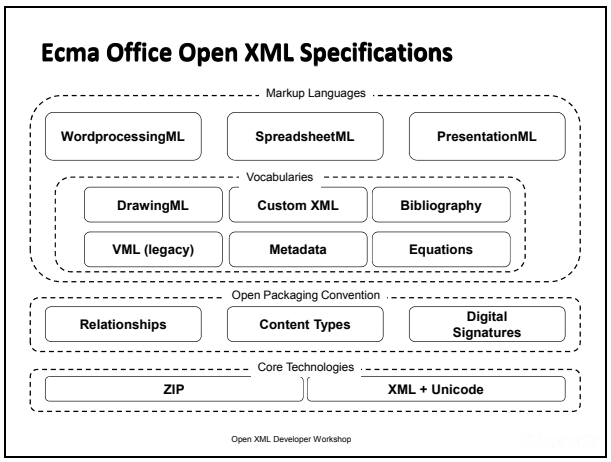

図 1. OOXML の仕様構成[3]

上記は OOXML が，ISO/IEC に提出される前の ECMA-376 [4]の仕様構成を表したものであるが，仕様構成に大きな変更はない．

2.2 論理的な構成

OPC で管理されるファイルコンテナをパッケージと呼ぶ．パッケージには，パーツおよびパーツのメタ情報としてリレーションシップやコンテンツタイプストリームが存在する（図 2）．

パーツは，文章や書式情報などの 80 種類以上に細分化されたアイテムで WML，SML，PML などで定義された

XML 形式のものと, 画像などのバイナリ形式のものがある．特に OOXML 文書として最初に参照される基点のパーツをドキュメントパーツと呼ぶ.

リレーションシップは，パッケージまたはパーツからパーツなどへの参照関係を定義したアイテムで，パッケージ単位のパッケージリレーションシップとパーツ単位のパーツリレーションシップが存在する．参照先 (Target) としては，パッケージ内 (Internal) に保有するパーツだけでなく，パッケージ外 (External) のパーツやハイパーリンクなども対象となる.

コンテンツタイプストリームは，パーツとリレーションシップのコンテンツタイプ（メディアタイプ）を定義するアイテムで, パッケージに一つ保有する．コンテンツタイプの定義には，拡張子毎 (Default) とパーツ毎 (Override) のものが存在する．パーツ毎の定義は，同一の拡張子（例えば，*.xml）のパーツであっても，コンテンツタイプを区別する場合に使用される．なお，拡張子毎の定義を行わず，全てパーツ毎の定義とすることも可能である.

OOXML は，単一の XML ファイルでは無い！

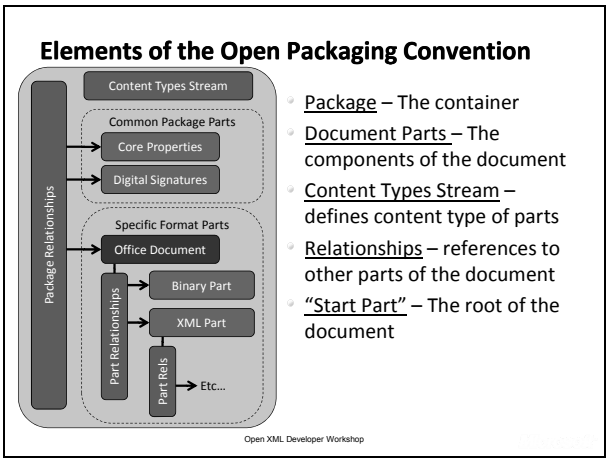

図 2. パッケージの論理的な構成[3]

各アイテムのパッケージ内での URI について，URI が固定されているのは，コンテンツタイプストリーム (/[Content_Types].xml) とパッケージリレーションシップ (/_rels/.rels) のみで，パーツは，任意の URI を持つことが可能である.

これは，常にリレーションシップを経由してパーツの特定を行う OPC の特徴である. パッケージで最初にドキュメントパーツを特定する場合も，(固定の URI となる) パッケージリレーションシップを経由して行われる．また，OPC では，パーツの URI をリレーションシップとコンテンツタイプストリームのみで管理することで，パーツの差し替えや参照先のパーツ変更を容易にしている.

なお，パーツリレーションシップについては，パーツの URI に対応する URI となっている．例えば，/document.xml がパーツの URI であれば, パーツリレーションシップの URI は/_rels/document.xml.rels となる.

2.3 パーツへのアクセス

OPC の柔軟な構造を採用した OOXML では, パーツへのアクセスのたびにリレーションシップとコンテンツタイプストリームへのアクセスが必要となるため，OOXML を管理する上で，これらへの複雑なアクセス[5]をいかに効率よく行うかが重要である．まず，ファイルベースで WML のドキュメントパーツにアクセスする場合の具体例を図 3 に示す．必要な処理は，順にパッケージリレーションシップよりドキュメントパーツの URI を取得, コンテンツタイプストリームからドキュメントパーツのコンテンツタイプを取得，ドキュメントパーツの取得である.

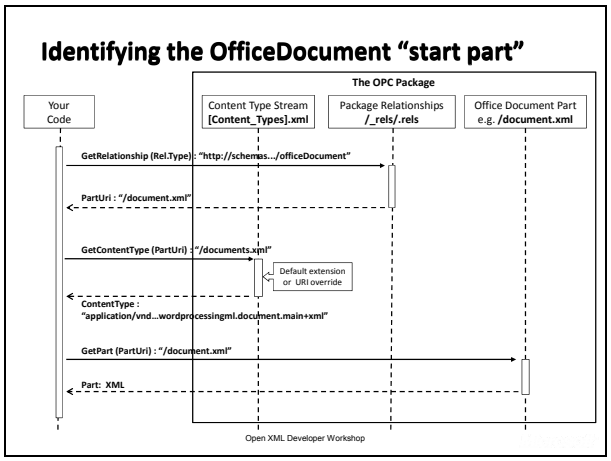

図 3. パーツへのアクセス例[3]

OOXML のアクセスのパターンは，パッケージ内の全てのパーツの閲覧や更新を行う場合と，必要なパーツのみの閲覧や更新を行う場合に大別される.

OOXML のアクセスを簡略化する方法として，FlatOPC[6]と呼ばれる方式が存在する．FlatOPC は，全てのパーツを 1 つの XML とするもので，バイナリ形式のパーツについても BASE64 方式でエンコードし，XML 化する．単一の XML とすることで, XSLT などでの取り扱いが容易になり，パッケージ内の全てのパーツにアクセスする場合には有用と考える．一方，必要なパーツのみアクセスする場合でも常に大きな XML をメモリ上で

扱わなければならないので，パフォーマンス上のデメリットとなる．

OOXML を効率よく閲覧するために，ハイブリッド型 XML データベースの RDB としての機能を併用して，パーツを管理し必要なパーツのみにアクセスする方法を提案する．

3. ハイブリッド型 XML データベースへの格納

3.1 ハイブリッド型 XML データベース

ハイブリッド型 XML データベース[2]は，従来の RDBMS の機能に加えて XML データベースの機能としての XML の階層構造を管理する専用の機構を持っている．XML を専用に扱う XML 型（LOB 型でない）を使用すれば，テーブルの一項目として XML データの登録が可能となる（図 4）．また，RDBMS としての BLOB 型があり，バイナリデータの格納も可能である．

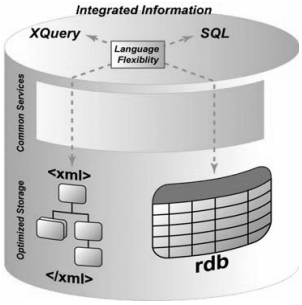

図 4. ハイブリット型 XML データベースのイメージ[2]

このため，OOXML に含まれる全てのパーツをハイブリット型 XML データベースによって管理することができる．

3.2 OOXML ファイル格納法の実装

OOXML のハイブリット型 XML データベースへの格納において，Web browser 上で OOXML ファイルの管理を行うプログラムが必要とする機能は以下の 4 つである．

- OOXML の登録（アップロード）機能．
- OOXML の復元（ダウンロード）機能．（図 5）
- 各パーツ，アイテムの表示・取得機能．（図 6，図 7）
- パーツの更新機能（共有文字列のインライン化のみ）．（図 6）

具体的には，Packages テーブル（表 1）と Parts テーブル（表 2）の 2 つのテーブルを使用する．Packages テーブルには，パッケージ単位の情報を格納する． Parts テーブルには，パーツ単位で保有する情報を格納する．これらのテーブルを関連付けるために，DOC_ID 列を設定する．

表 1. Packages テーブル

カラム名	型	KEY	NULL	説明
DOC_ID	INT	PK	不可	パッケージ単位の連番
DOC_NAME	VCHAR		不可	パッケージのファイル名
CONTENT_TYPES	XML		不可	パーツのデータタイプの定義
RELS	XML		不可	パッケージリレーションシップ
APP	XML		可	拡張ファイルのプロパティ
CORE	XML		可	作成者名や作成日など
CUSTOM	XML		可	ファイル共有情報など

表 1. Parts テーブル

カラム名	型	KEY	NULL	説明
DOC_ID	INT	PK FK	不可	Packagesテーブルの DOC_ID
PARTNAME	VCHAR	PK	不可	パーツ名（絶対パスのURI表記）
DEF_CONTENT TYPE	VCHAR		不可	コンテンツタイプスに定義されている,拡張子毎のコンテンツタイプ
OVR_CONTENT TYPE	VCHAR		可	コンテンツタイプスに定義されている,パーツ名毎のコンテンツタイプ
XML_PART	XML		可	BINARY_PARTと排他で利用
BINARY_PART	BLOB		可	XML_PARTと排他で利用
RELS	XML		可	パーツリレーションシップ

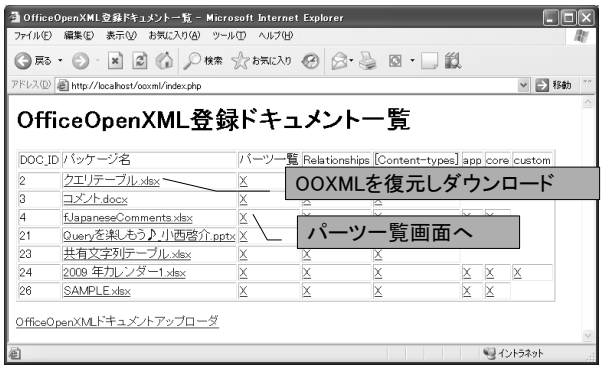

図 4. 登録された OOXML の一覧画面

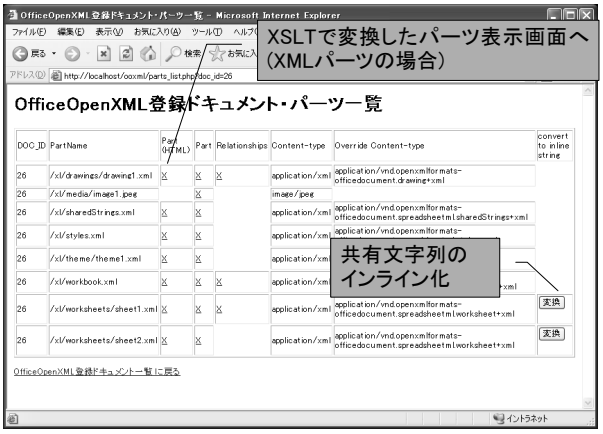

図 6. パーツ一覧画面

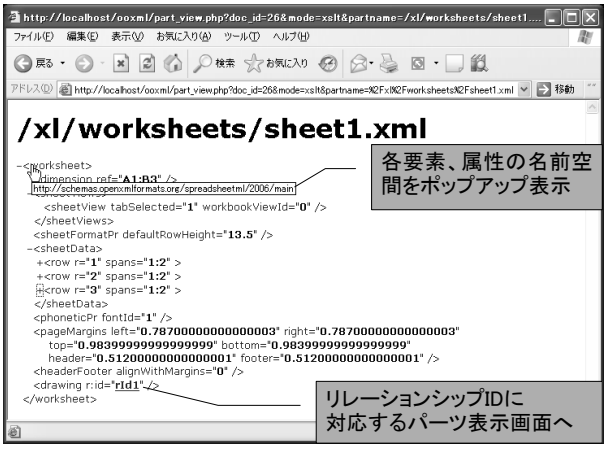

図 5. XSLT で変換した XML パーツ表示画面

コンテンツタイプストリームは，復元のし易さとパーツの取り扱い易さを両立させるために，XML の状態で Packages テーブルに保管するとともに，Parts テーブルにもパーツ毎にシュレッディングした状態で保管することとした．また，ファイルプロパティパーツについては，パッケージ単位の検索を容易にするために，Packages テーブルに保管することとした．その他のパーツについては XML 形式の場合には，XML 型の XML_PART 列に，それ以外の場合には，BLOB 型の BINARY_PART 列に格納する．各パーツに対応するリレーションシップが存在する場合には，XML 型の RELS 列に格納する．

実行環境として Windows XP + Apatch2.2.11 + PHP5.2.6 + DB2 9.5.2 Express-C を使用した.このプログラムは，筆者が作成し 2009 年に IDUG (International DB2 Users Group) 主催で，インフォテリア（株）と日本 IBM（株）が開催した「XML プログラミングコンテスト」への応募作品で，最優秀賞を受賞したものである．

上述した OOXML を格納する適切な DB スキーマに対して実装した格納法には，2 つの問題点がある．すなわち，パッケージに紐付くパーツの格納方式，および，OOXML のメタ情報の格納方法である．

【パッケージに紐付くパーツの格納方式】

パッケージに紐付くパーツのうち 3 つのファイルプロパティパーツのみ Packages テーブルに格納されるが，その他のパッケージに紐付くパーツ（サムネイルパーツ，デジタル署名パーツなど)は Parts テーブルへ格納される．Packages テーブルにファイルプロパティパーツを格納することでの作成者や更新時間などの情報にリレーションシップを経由せずに直接参照が可能となる．しかし，他のパーツと区別して格納するため，アクセス処理を共通化することが出来ず，3 つのファイルプロパティパーツ毎にアクセス処理の実装が必要となる．

【メタ情報の格納方法】

OOXML のメタデータであるコンテンツタイプストリームが Packages テーブルに XML として格納されているだけでなく，さらに，表示および処理を容易にするために，Parts テーブルにもシュレッディングした状態で格納されている．この結果，コンテンツタイプストリームに対して正規化できていない．

1 番目の問題に対しては，パッケージに紐付くパーツを含め，全てのパーツについて Parts テーブルに正規化して管理する方法を提案する．2 番目の問題に対して，OOXML のメタデータである，コンテンツタイプストリームとリレーションシップについて，管理方法を XML または，シュレッディングした状態のいずれかに統一する必要があることを指摘する．

DB スキーマとして OOXML のメタ情報を，XML として保有する場合とシュレッディングして保有する場合の比較を 3.3 節および 3.4 節で行う．なお，比較にあたり，メタ情報の取得はパーツを取得する毎に行うこと，および，パーツの XML 処理はアプリケーション側で行うことを前提とする．

デジプラコラム：「OOXML 内の文字列情報」

実務者としてオフィスドキュメントが XML 化された大きなメリットは，ドキュメント内の文字列情報を容易に閲覧できるようになったことである．WML や PML については，ドキュメントパーツやスライドパーツの text 要素 (t) で主な文字列情報が抽出可能である．SML については，重複した文字列を効率よく管理するため，通常ワークブックパーツ単位で共有文字列テーブルパーツを保有し，ワークシートパーツには，共有文字列テーブルパーツの index のみを保有する．このため，SML の文字列操作の際は，常に index の処理が必要である．

ただし，SML の規格上ワークシートパーツに文字列情報を保有（インライン化）する形式も存在する．事前に文字列情報のインライン化を行えば，人間に見やすいだけでなく，セルの更新処理を簡素化することも可能である．

3.3 正規化手法 A

OOXML のメタ情報を XML として保有する格納法として 3.2 節の格納法と同様に，2 つのテーブルを使用する．3.2 節の格納法からの変更点は，以下の通りである．

Packages テーブル（表 3）には，コンテンツタイプス

表 3. Packages テーブル

カラム名	型	KEY	NULL	説明
DOC_ID	INT	PK	不可	パッケージ単位の連番
DOC_NAME	VCHAR		不可	パッケージのファイル名
CONTENT_TYPES	XML		不可	パーツのデータタイプの定義
RELS	XML		不可	パッケージリレーションシッ

表 2. Parts テーブル

カラム名	型	KEY	NULL	説明
DOC_ID	INT	PK FK	不可	Packagesテーブルの DOC_ID
PARTNAME	VCHAR	PK	不可	パーツ名（絶対パスのURI表記）
XML_PART	XML		可	BINARY_PARTと排他で利用
BINARY_PART	BLOB		可	XML_PARTと排他で利用
RELS	XML		可	パーツリレーションシップ

トリームとパッケージリレーションシップのみ格納する．Parts テーブル（表 4）には，ファイルプロパティパーツを含め全てのパーツを格納し，パーツ毎のコンテンツタイプは格納しない．

本格納法を用いて SML の最初のワークシートパーツにアクセスする場合の具体例を図 8 に示す．

必要な処理は，順に Packages テーブルから指定したファイル名で DOC_ID およびドキュメントパーツの URI を取得 (#1)，アプリケーションにてドキュメントパーツの絶対パスの URI および拡張子を取得， Packages テーブルからドキュメントパーツのコンテンツタイプを取得 (#2)，Parts テーブルからドキュメントパーツの取得 (#3)，アプリケーションにて最初のシートのリレーションシップの ID を取得， Parts テーブルからワークシートパーツの URI を取得 (#4)，アプリケーションにてワークシートパーツの絶対パスの URI および拡張子を取得，Packages テーブルからワークシートパーツのコンテンツタイプを取得 (#5)，Parts テーブルからワークシートパーツの取得 (#6) である．このため，DB アクセスは計 6 回となる．

本格納法のメリットは，メタデータを含めパッケージ内の XML を全て XML 型に格納しているので，パッケージへの復元が容易なことである．デメリットは，リレーションシップを辿る毎に，Xquery でのアクセスまたは XML のパースを行い，参照先の URI を絶対パスに変換してから Parts テーブルへのアクセスを行う必要があることである．また，現状のハイブリット型 XML データベースでは，通常の RDB 項目と XML 型の内部のノード値との制約を設定できないので，リレーションシップによる参照関係の整合性をアプリケーション側で担保する必要があることである．

```
--#1:DOC_ID およびドキュメントパーツの URI 取得
SELECT DOC_ID , XMLCAST(XMLQUERY('$RELS//*:Relationship[@*:Type="http://…/relationships/officeDocument"]/data(@*:Target)') AS VARCHAR(1000)) FROM PACKAGES  WHERE DOC_NAME = 'SAMPLE.xlsx';
--return 26,'xl/workbook.xml'

/* 絶対パスへの変換および拡張子の算出 */

--#2: ドキュメントパーツのコンテンツタイプ取得
SELECT XMLCAST(XMLQUERY('$CONTENT_TYPES//(*:Override[@*:PartName="/xl/workbook.xml"],*:Default[@*:Extension="xml"])[1]/data(@*:ContentType)') AS VARCHAR(1000)) FROM PACKAGES
 WHERE DOC_ID = 26;
-- return 'application/....sheet.main+xml'

--#3: ドキュメントパーツ（ワークブックパーツ）取得
SELECT XML_PART FROM PARTS WHERE DOC_ID=26 AND PARTSNAME='/xl/workbook.xml';
--return XML

/* XML より最初のシートの ID(rId1) を取得 */

--#4: ワークシートパーツの URI 取得
SELECT XMLCAST (XMLQUERY('$RELS//*:Relationship[@*:Id="rId1"]/data(@*:Target)') AS VARCHAR(1000))
 FROM PARTS WHERE DOC_ID=26 AND PARTSNAME ='/xl/workbook.xml';
--return 'worksheets/sheet1.xml'

/* 絶対パスへの変換および拡張子の取得 */

--#5: ワークシートパーツのコンテンツタイプ取得
SELECT XMLCAST(XMLQUERY('$CONTENT_TYPES//(*:Override[@*:PartName="/xl/worksheets/sheet1.xml"],*:Default[@*:Extension="xml"])[1]/data(@*:ContentType)') AS VARCHAR(1000)) FROM
 PACKAGES WHERE DOC_ID = 26;
--return 'application/....worksheet+xml'

--#6: ワークシートパーツ取得
SELECT XML_PART FROM PARTS WHERE DOC_ID=26 AND PARTSNAME='/xl/worksheets/sheet1.xml';
--return XML
```

図 6. ワークシートパーツへのアクセス例

XML データは，全て XML 型に格納すべきか

3.4 正規化手法 B

OOXML のメタ情報をシュレッディングした状態で保有する格納法として 3.2 節の格納法に 2 テーブルを加えた 4 つのテーブルを使用する．3.2 節の格納法からの主な変更点としては，以下の通りである．

Packages テーブル（表 5）には，パッケージのファイル名のみ格納する．Parts テーブル（表 6）にはファイルプロパティパーツを含め全てのパーツを格納し，パーツ

表 5. Packages テーブル

カラム名	型	KEY	NULL	説明
DOC_ID	INT	PK	不可	パッケージ単位の連番
DOC_NAME	VCHAR		不可	パッケージのファイル名

表 6. Parts テーブル

カラム名	型	KEY	NULL	説明
DOC_ID	INT	PK UK FK	不可	Packagesテーブルの DOC_ID
PART_ID	INT	PK		パーツ単位の連番 オフィスドキュメントパーツの場合1とする
PARTNAME	VCHAR	UK	不可	パーツ名（絶対パスのURI表記）
EXTENSION	VCHAR		可	パーツ名の拡張子
CONTENTTYPE	VCHAR		不可	パーツのメディアタイプ
XML_PART	XML		可	BINARY_PARTと排他で利用
BINARY_PART	BLOB		可	XML_PARTと排他で利用

毎の PART_ID を設定する．コンテンツタイプは，CONTENTTYPE に統一する． Relationships テーブル（表 7）を新たに追加し，リレーションシップをシュレッディングして格納する．パッケージリレーションシップの場合は，参照元のパーツは存在しないため，PART_ID を 0 として登録する．また，頻繁にアクセスが行われる，ドキュメントパーツについては，PART_ID を常に 1 として登録する．Extensions テーブル（表 8）には，コンテンツタイプストリームの拡張子毎のコンテンツタイプをシュレッディングして保管する．なお，コンテンツタイプストリームを復元の際に全てのパーツ，リレーションシップについてパーツ毎の定義 (Override) とする場合には，本テーブルを設置しないことも可能である．

本格納法を用いて SML の最初のワークシートパーツにアクセスする場合の具体例を図 9 に示す．必要な処理は，順に Packages テーブルから指定したファイル名で DOC_ID を取得 (#1)，Relationships テーブルからドキュメントパーツの PART_ID を取得 (#2)，Parts テーブルからドキュメントパーツのパーツ自体およびコンテンツタイプを取得 (#3)，アプリケーションにて最初のシートのリレーションシップの ID を取得，Relationships テーブルからワークシートパーツの PART_ID を取得 (#4)，Parts テーブルからワークシートパーツのパーツ自体およびコンテンツタイプを取得 (#5)である．このため，DB アクセスは計 5 回となる．また，ドキュメントパーツの PART_ID 取得 (#2) では結果が常に 1 となる（よう格納している）ため省略可能であり，省略した場合は，計 4 回の DB アクセスとなる．

表 7. Relationships テーブル

カラム名	型	KEY	NULL	説明
DOC_ID	INT	PK FK	不可	Packagesテーブルの DOC_ID
PART_ID	INT	PK	不可	PartsテーブルのPART_ID パッケージリレーションシップの場合は0
RELS_ID	VCHAR	PK	不可	リレーションシップのID
TYPE	VCHAR		不可	リレーションシップタイプの URI
TARGETMODE	CHAR		不可	INTERNAL,または,EXTERNAL
INTERNALTARGET	INT	FK	可	パッケージ内パーツの PART_ID
EXTERNALTARGET	VCHAR		可	パッケージ外パーツ等の URI

表 8. Extensions テーブル

カラム名	型	KEY	NULL	説明
DOC_ID	INT	PK FK	不可	Packages テーブルの DOC_ID
EXTENSION	VCHAR	PK	不可	拡張子
CONTENTTYPE	VCHAR		不可	拡張子のコンテンツタイプ

メタデータは，シュレッディングして格納すべきか

本格納法のメリットは，パーツへのアクセスがパーツ名ではなく PART_ID のみで可能なため，アプリケーション側での URI の絶対パスへの変換などが不要となり，パーツの取り扱いが容易に行えることである．

```
--#1: ドキュメント名から DOC_ID 取得
SELECT DOC_ID FROM PACKAGES WHERE DOC_NAME =
 'SAMPLE.xlsx';
-- return 26

--#2: (省略可) ドキュメントパーツの PART_ID 取得
--(パッケージリレーションシップの PART_ID は，0 固定)
SELECT INTERNALTARGET FROM RELATIONSHIPS WHE
RE DOC_ID=26 AND PART_ID=0 AND TYPE='http://
…/relationships/officeDocument';
-- return 1

--#3: ドキュメントパーツ取得
--(ドキュメントパーツの PART_ID は，1 固定)
SELECT XMLPART,CONTENTTYPE FROM PARTS WHERE
DOC_ID=26 AND PARTID=1;  --1=ドキュメントパーツ
-- return XML, 'application/….sheet.main+xml'

/* XML より最初のシートの ID(rId1) を取得 */

--#4: ワークシートパーツの PART_ID 取得
SELECT INTERNALTARGET FROM RELATIONSHIPS WHE
RE DOC_ID=26 AND PART_ID=1 AND RID='rId1';
--return 7

--#5: ワークシートパーツ取得
SELECT XMLPART,CONTENTTYPE FROM PARTS WHERE
DOC_ID=26 AND PART_ID=7;
--return XML, 'application/….worksheet+xml'
```

図 7. ワークシートパーツへのアクセス例

デメリットは，パッケージの復元時にシュレッディングしたメタデータに対して, XML への変換が必要なことである．

3.2 節に記載した前提を条件とすると，正規化手法 A の格納法に比べて，DB アクセス数の少なくパーツへのアクセスが容易な，正規化手法 B の格納法が優れていると考えられる．

なお，今回は，パーツの XML の処理をアプリケーション側で行う前提としたが，アプリケーション側で行うかデータベース側で行うかは，システムの性能に影響する重要な要因であるため, 注意して決定する必要がある．

4. その他の知見

4.1 OOXML の復元性

パーツの復元性について，パーツの形式により違いが生じる．バイナリ形式のパーツは，BLOB 型で保管しているので, バイナリレベルで復元でき, 問題はない. XML 形式のパーツは，XML 型として保管しているが，XML 型は XDM (XQuery 1.0 and XPath 2.0 Data Model) [7]での管理でありバイナリレベルで復元することはできない．ただし，XML 形式のパーツについては，そもそも XML レベルでの処理を前提としているので，XDM で管理を行えば，通常問題は発生しない．

しかし，OOXML でサポートされているデジタル署名[8]では，必ずしも復元性が保証されないという問題が発生する．例えば, Office 2010 でデジタル署名を設定し 3.2 節の実装で格納, 復元を行い再び Office 2010 で開いた場合に, 元のデジタル署名は無効と判断されることがある．署名時と復元後の差異は，署名時のパーツに存在する XML 宣言の standalone 属性と XML 宣言直後の改行である．XDM では，XML 宣言の情報は保存していないが，デジタル署名時には，Canonical XML[9]で正規化を行っており, XML 宣言や改行の差異は吸収されるべきである．ただし，Canonical XML では，バージョン番号と文字エンコーディングを含む XML 宣言を削除すると規定されているものの，standalone 属性が含まれる場合の動作は明確には規定されていない．改行の有無に伴う挙動の違いを含め原因については，規格上の問題か実装上の問題かなど今後さらに調査が必要である．

パッケージの復元性について, ZIP 形式は, 圧縮形式, 圧縮率，圧縮順序，環境などにより，異なるバイナリファイルとなるので，バイナリレベルで復元することは難しいが，（デジタル署名の問題を除き）パッケージとしては論理的に復元されると判断できる．

4.2 妥当性検証

ハイブリット型 XML データベースで OOXML の管理を行う場合，格納や更新の際に，妥当性検証が必要となることが考えられる．OOXML の妥当性検証の標準的な方法は確立されていないが，大きく分けてパッケージ (OPC) の妥当性，パーツ単位（XML-Schema など）の妥当性，意味的制約定義（パーツを跨る参照関係など）の妥当性の３種類が必要だと考えられる．

XML-Schema を使用した妥当性検証以外は汎用的な仕組みを使用出来ないため、オープンソースでも Office-o-tron[10]や ISO/IEC 29500 (OOXML) Validator[11]などの妥当性検証ツールが開発されている．しかし現状では, 意味的制約定義を含めて妥当性検証を行えるのは, Microsoft 社の Open XML SDK Ver2.0[12]のみである．

本論文の格納法では，ハイブリット型 XML データベースに装備されている XML-Schema の妥当性検証機能によるパーツ単位の妥当性検証が可能である．また，3.4 節の正規化手法 B の格納法では，格納の際にメタ情報をシュレッディングする事でパッケージレベルの妥当性検証が可能であり，格納中は，RDBMS の外部制約機能により，パッケージレベルの妥当性を維持することが可能である．

なお，OOXML の XML-Schema は，名前空間単位に提供されるため，厳密にはパーツ単位の妥当性検証として XML-Schema だけでなくルート要素とコンテンツタイプの関係もチェックすべきである．また，OOXML の XML-Schema には，最新の仕様に厳密な (Strict) ものと過去の仕様を許容する (Transitional) もの，Ecma 仕様のもの，製品独自の拡張など複数存在するので，どのレベルでチェックしどこまで許容すべきかについて注意が必要である．

4.3 パーツの共有

本論文では，パッケージ毎にパーツの管理を行う前提で格納法の設計し，実装を行った．現実の応用では，常に利用する巨大な画像データを使用するパーツをパッケージ間で共用するといったことが求められる. 3.4 節の格納法を拡張すれば，このようなパーツを共有することが可能と考えられる．

5. まとめと今後の展望

本論文では，ZIP で固めたファイル群として格納される OOXML 文書に対して，ハイブリッド型 XML データベースに格納することによって，効率よく管理（閲覧・

復元・更新）を行えるシステムについて提案を行った．OOXML は，異なるコンテンツタイプを持つ多数のパーツを柔軟に組み上げることを可能としている反面，各パーツへのアクセスに多くの処理を要する．複数の異なる処理パターンに対して，提案手法に基づき，どのようなDB スキーマを用いるべきかについても議論を行った．

XML データベースに格納した OOXML の編集方法には，本論文で対象とした直接格納されたパーツの更新だけで済む場合だけでなく，アプリケーションのオブジェクトモデルでの編集が必要な場合もある．後者の場合には，すでに提供されている API との連携が必要となる．特に，API 上でパッケージのオブジェクトモデルが提供されている場合には，パーツへのアクセスをハイブリットデータベースへのアクセスに変更することで API 上の機能を利用することも可能だと考えられる．

OPC は，OOXML 以外での利用も考慮した規格となっており，今後，複雑なデータを取り扱うアプリケーションで OPC が保存形式として普及する可能性がある．既に OOXML より先に OPC の構成で規格化された XPS (XML Paper Specification) [13]も存在する．OPC の構成であれば，本論文の格納法を利用可能だと考えられる．

謝辞 論文などには縁遠い私に，この様な発表の場への挑戦を強く勧めて下さり，多くの助言を頂いた丸山　宏氏，本論文の編集担当を務めて頂いた京都大学教授の奥乃博氏に深謝いたします．

参考文献

1) ISO/IEC29500 Information technology — Document description and processing languages —Office Open XML File Formats.
2) 日本 IBM Redbooks: DB2 9 - pureXML ガイド(翻訳), http://www-06.ibm.com/jp/domino01/mkt/dminfo.nsf/doc/000A6934 (2010) .
3) Microsoft Developer Network: Open XML Developer Workshop, http://msdn.microsoft.com/office/bb738430.aspx (2010).
4) Standard ECMA-376 Office Open XML File Formats, http://www.ecma-international.org/publications/standards/Ecma-376.htm (2010).
5) 鈴木俊哉: OOXML に求められるフォント情報交換規定, 画像電子学会第 23 回 VMA 研究会，VNA23-2，2008-12-01 (2008).
6) Eric White's Blog: The Flat OPC Format, http://blogs.msdn.com/b/ericwhite/archive/2008/09/29/the-flat-opc-format.aspx .
7) W3C: XQuery 1.0 and XPath 2.0 Data Model (XDM), http://www.w3.org/TR/xpath-datamodel/ .
8) 山地 一禎, 片岡 俊幸, 宮地 直人, 曽根原 登: Office Open XML に対する長期署名の付与, 情報知識学会誌, Vol.20, No.1, pp.1-14 (2010) .
9) W3C: Canonical XML Version 1.0, http://www.w3.org/TR/xml-c14n .
10) Probatron: Office-o-tron, http://www.probatron.org:8080/officeotron/officeotron.html (2010).
11) Jesper: ISO/IEC 29500 document validator, http://is29500validator.codeplex.com/ (2010).
12) Microsoft Developer Network: Open XML SDK 2.0 for Microsoft Office へようこそ, http://msdn.microsoft.com/ja-jp/library/bb448854.aspx (2010).
13) Microsoft Hardware Developer Center: XML Paper Specification: 概要, http://www.microsoft.com/japan/whdc/device/print/xps/default.mspx (2010).

小西　啓介（非会員）
E-mail: konishi.keisuke@sorun.co.jp
1980 年生まれ．2004 年ソラン（株）に入社．金融機関向けのシステム開発業務に従事．2008 年 XML コンソーシアム XMLDB 部会に参加．2009 年 XML プログラミングコンテストで最優秀賞を受賞．

投稿受付：2010 年 9 月 21 日
採録決定：2010 年 11 月 16 日
編集担当：奥乃　博 (京都大学)