RDFとメタデータの相互運用

第32回ディジタル図書館ワークショップ
2007-03-09　神崎正英

(メタ)データ相互運用の課題

- **データモデルと名前の相互運用**
  - どんなモデル(スキーマ)を採用するか
  - プロパティ(関係記述語彙)はどの程度詳細に設計すべきか、独自語彙か汎用語彙か
  - 記述対象をどのように識別するか(主語リソースの同一性)
  - 人物などの典拠、主題などの語彙をどうやって共有するか
    (※本稿では、最初の2つ、モデルとプロパティについて検討します)
- **相互運用のスコープ**
  - データの共有:統合レポジトリの構築や横断検索の実現
  - データの交換:AのデータをBが取り込みたいとき

データ相互運用とRDF/OWL

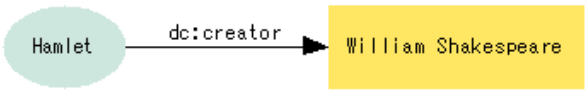
2

- **なぜRDF/OWLか**
  - **柔軟性**:シンプルで柔軟なグラフ構造でさまざまなモデルを表現できる
  - **意味論**:グラフの意味論[1]が定義されており、予備知識のないデータも共通の解釈ができる
  - **語彙の連携**:語彙間の関係を記述することができ、その関係を利用した基本的な推論ができる
  - **識別と統合**:URIによるリソースのグローバルな識別と、グラフの併合ができる
  - **標準クエリ**:グラフに対する問い合わせ言語の標準(SPARQL[2])が整いつつある
- **RDFバス**
  - 異なるデータソース間で直接変換を行うよりも、いったん共通項としてのRDF(RDF Bus)を経由するほうが有利
  - RDFとして識別、統合、推論を行うことで、さまざまなつながりを見出すことができる
  - 個々のデータはRDFでも、一般的なRDBMSでもよい
    - RDBMSなどは相互運用の際に共通項としてのRDFに変換すればよい
    - RDFの構文はXMLに限らない=RDFとして保持/交換する場合でもさまざまな方法がある

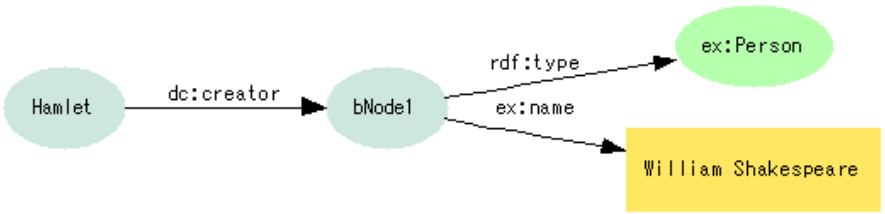
汎用語彙によるシンプルなRDFの記述

3

- **汎用語彙による記述のメリット、デメリット**
  - 長所:広い分野で利用でき、相互運用性が高い;シンプルで分かりやすい;(比較的)安定している
  - 短所:表現力が弱い;用法が不明瞭、不正確な場合がある(→相互運用の支障になる可能性)
- **Dublin Core記述モデルのばらつき**
  - 目的語(著者)を単純にリテラル(名前文字列)で表現する



  - 著者をエンティティ(人物)とし、そのプロパティとして名前を与える(より抽象モデルに忠実)

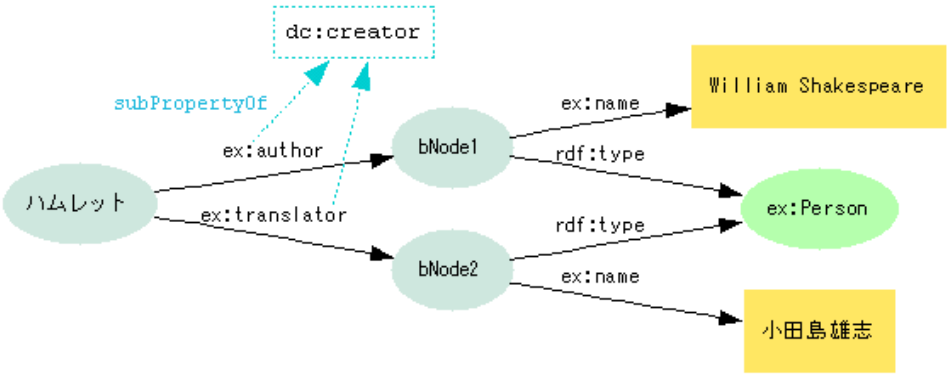


  - 単純なクエリでは両者を同時に検索できない
    - 過去の経緯で使い方が曖昧になっている
    - DCMI (Dublin Core Metadata Initiative) は2005年に抽象モデル[3]を定義、2007年には新たな抽象モデルと定義域、値域を定める案[4]が出されている

関係記述の詳細化

4

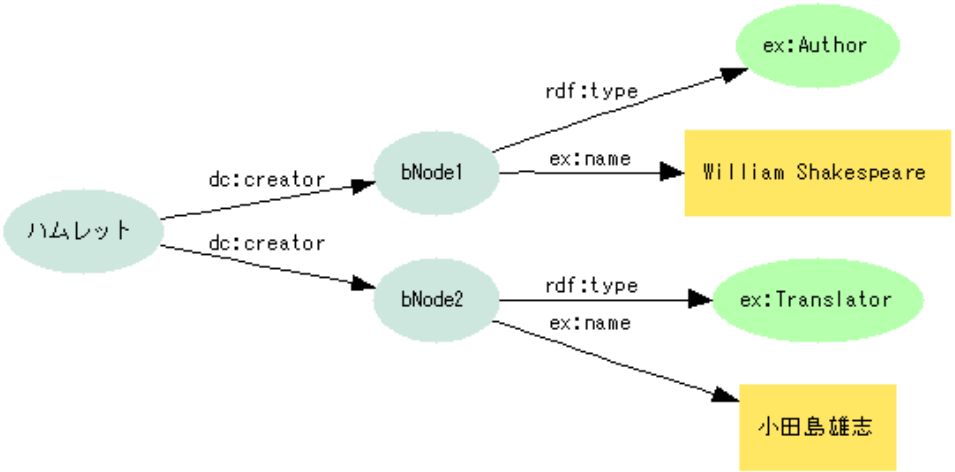
- **作者から著者と翻訳者へ**
  - William Shakespeare著, 小田島雄志訳『ハムレット』をどう記述するか
  - Dublin Coreの基本要素だけでは、著者、訳者ともにdc:creatorとなって区別ができない
- **詳細化のためのプロパティ定義とサブプロパティ**
  - 著者を表すex:author、翻訳者を表すex:translatorを定義
  - ex:authorとex:translatorをdc:creatorのサブプロパティとして関連付ける



クラスを利用した役割の定義

5

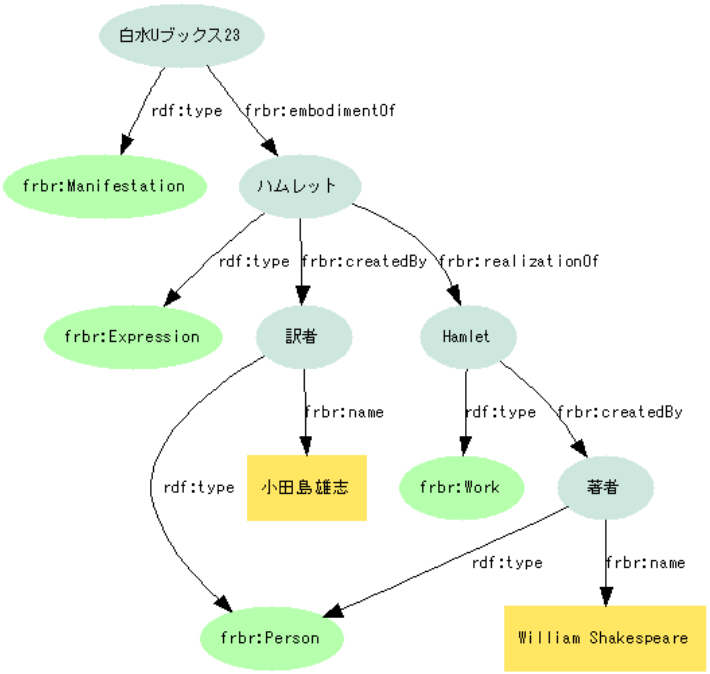
- **プロパティを増やさず役割クラスで明確化する方法**
  - ドメインのニーズを全て満たす詳細化プロパティの定義は複雑
  - 汎用プロパティの目的語に型(ロールを表すクラス)を与えることで役割を識別できる
  - 多くの役割クラスは汎用オントロジー(例えばWordNet/RDF[5]など)で定義されている可能性が高い
    - 独自定義の必要性が低い＝相互運用性が高い
- **著者、訳者をAuthor、Translatorクラスで表現**



  - ここでのクラスは、MODS[6]のrole属性の働きに近い

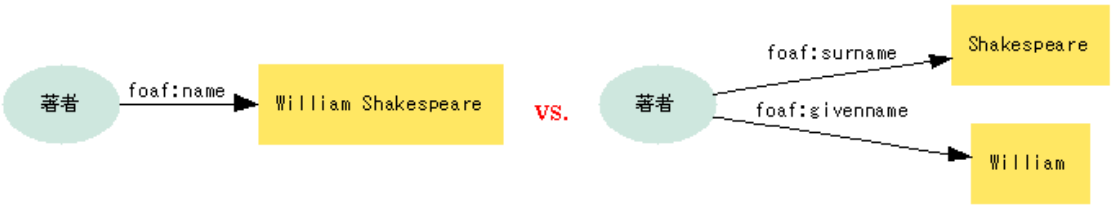
より掘り下げたモデル 6

- **FRBR[7]のエンティティモデルによるRDFグラフ**
  - Hamletという作品(`Work`)の日本語訳(`Expression`)の白水社版(`Manifestation`)という関係
  - 作品の作者はShakespeare、日本語訳の作者は小田島雄志として表現できる



部分を表現するプロパティ 7

- **1つのプロパティの関係が複数のプロパティに分割されることがある**
  - 「名前」を姓と名に分ける、「住所」を国、都道府県、市町村、番地に分ける、など



  - モデルは、`(著者)--name-->(名前)--surname-->"Shakespeare".`という間接的部分関係と考えられるが、直接関係として表現されることも多い。

- **共有・交換のボトルネックになりやすい**
  - 部分→全体の操作はできるが、全体→部分の操作は難しい
    - 姓と名から氏名を作ることはできるが、氏名を姓と名に分割できるとは限らない
  - RDFノードの結合(合成)は一般的な操作ではない
    - 姓ノードと名ノードから氏名ノードを作るには、基本的なグラフ操作ではなく、処理系依存の文字列連結関数などが必要

データの共有と交換

8

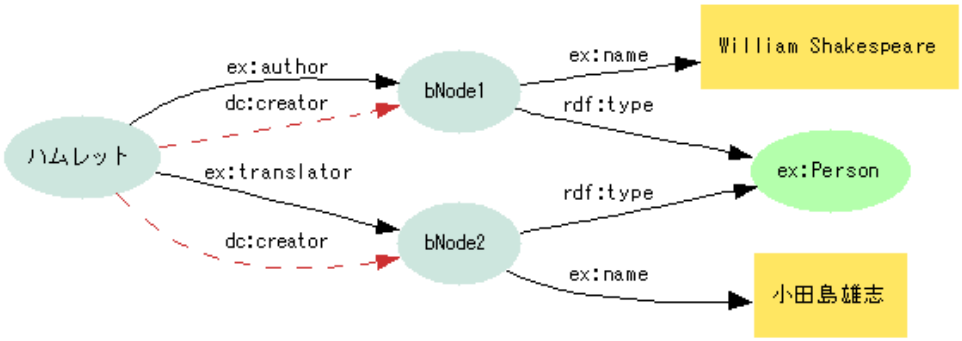
- **データの共有（共通利用）**
  - 異なるソース、異なるモデルのデータを共通の方法で利用する
    - 同じ手続きでデータにアクセスできる＝（仮想的に）共通のモデル、語彙を利用できる
    - あらかじめモデルと語彙を共通化する vs. 異なるモデル、語彙を推論や変換で橋渡しする
  - 共有方法は唯一無二ではなく、複数が並存し得る
    - Dublin Coreの基本要素のレベルで、できるだけ広い範囲を対象に共有するサービス
    - FRBRモデルに準拠したソースを対象に、精度の高い探索を行うサービス

- **データの交換**
  - 異なるソースのデータを（相互に）移植する
    - 完全な交換：AとBが双方向に情報を失わずにデータを交換できる
    - 一方向的な交換：A→Bで失われる情報はないが、B→Aでは一部の情報が失われる
    - 部分的な交換：A→B、B→Aともに、一部の情報が失われる
  - 直接のデータ交換を行う場合もあれば、中間形式を経る間接的交換もある

共有(1)：同じ深さのモデルで独自語彙を用いる場合

9

- **サブプロパティ定義があれば推論可能**
  - ex:author、ex:translatorのサブクラス定義からdc:creatorを推論できる（dc:creatorでもex:authorでも検索できる）



- **個別のデータ変換は不要**
  - RDFスキーマの基本推論（RDFS推論）で汎用プロパティのトリプルを追加できる
  - 元のプロパティも失われない（推論結果で置き換えるのではなく追加される）
  - データ提供者は、スキーマを公開するだけでよく、自分でデータ変換する必要がない

共有(2):役割クラスを用いる場合

10

- **単純検索ならそのまま可能**
  - プロパティが共通なので、集約したデータをそのまま検索できる(変換も推論も不要)

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ex: <http://example.org/terms/>
SELECT ?book ?title
WHERE {
 ?book dc:creator ?who . ?who ex:name "William Shakespeare" .
 ?book dc:title ?title .}
```

- **絞込検索には役割クラスを指定する**
  - 翻訳者名のみから検索する場合は、クラスを表すトリプルを検索パターンに追加

```
WHERE {
 ?book dc:creator ?who . ?who ex:name "小田島雄志" .
 ?book dc:title ?title . ?who a ex:Translator .}
```

  - 汎用オントロジーの階層化されたクラスを利用すれば、役割クラスを横断した検索もできる

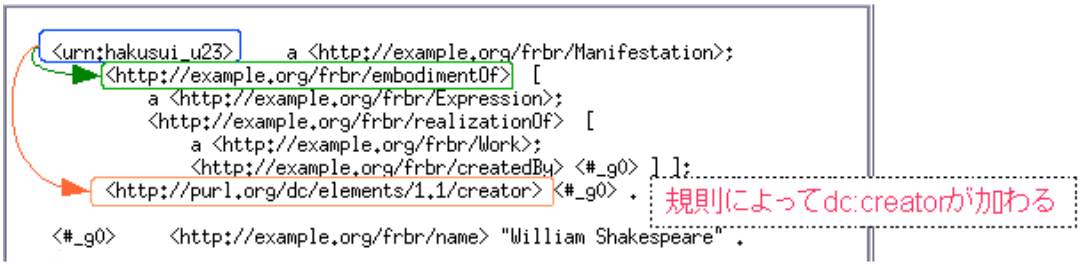
共有(3):モデルの深さが異なる場合

11

- **独自の推論規則で汎用的なトリプルを追加**
  - FRBRモデルからDCプロパティを生成する推論規則を定める

```
{?book frbr:embodimentOf ?expr .
 ?expr frbr:realizationOf ?work .
 ?work frbr:createdBy ?author .}
 => {?book dc:creator ?author .}
```

  - 推論エンジンを使ってdc:creatorのトリプルを加えることができる(例はcwm[8]による推論)



  - 生成されたトリプルを利用して、前節と同様のSPARQLで検索ができる
  - ただし、RDFS推論は主な処理ライブラリで提供されるが、こちらは独自の規則定義と推論エンジンが必要

共有(4):部分関係に分割されたプロパティ

12

- **姓と名に分かれたプロパティでの著者検索**
  - Surname(?author, "Shakespeare") ∧ Givenname(?author, "William") ⇒ Name(?author, "William Shakespeare") とするには?
  - 単純なグラフの推論ではカバーできない(姓と名のノードを合わせてひとつのノードにする手続きがない)
  - 文字列連結関数のある推論システムで「名前」ノードを生成した上で検索
- **推論と文字列操作**
  - cwmによる推論の例:特殊なプロパティ(string:conatenation)を組み込み関数として扱い、文字列を操作する

```
@prefix string: <http://www.w3.org/2000/10/swap/string#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
{?x foaf:surname ?sn;
 foaf:givenname ?gn.
 (?gn " " ?sn) string:concatenation ?fn . }
=>
{?x foaf:name ?fn.}.
```

共有(5):語彙とモデルの変換

13

- **XSLTによる変換**
  - データがXML形式にならば、XSLTテンプレートでdc:などの共通プロパティに変換することも可能
  - 例:FRBRモデルからシンプルなDCモデルに変換する

```
<xsl:template match="frbr:embodimentOf">
 <dc:creator>
  <ex:Person>
   <ex:name>
    <xsl:value-of
         select="*/frbr:realizationOf/*/frbr:createdBy/*/frbr:name"/>
   </ex:name>
  </ex:Person>
 </dc:creator>
</xsl:template>
```

- **XSLT変換のメリット、デメリット**
  - 長所:元データがRDFでなく一般のXMLであっても適用できる(MARCXML[9]やMODS形式の書誌データの利用など);文字列操作が容易
  - 短所:RDFS推論が利用できない(全ての場合にテンプレートが必要);特定のXML構文を前提とする;モデルや語彙の変換を伴う場合、元のプロパティは保持されない

    ※ここでは規則の論理的帰結によってグラフを追加していく操作を**推論**、マッピングによってもとのデータを別の構文、モデル、語彙に置き換える操作を**変換**と呼んでいます

データの交換

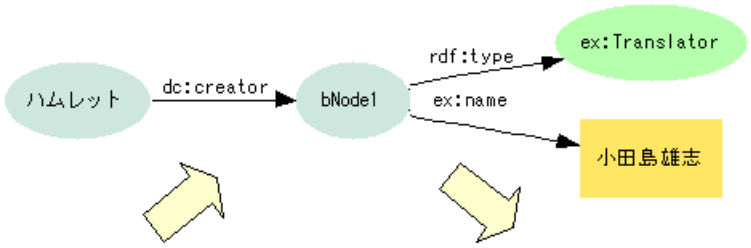
14

- **データベース間での交換の前提**
  - 相手のデータベースの対応する項目をできるだけ直接取り込み、対応しない場合はできるだけ情報を保持して変換したい
  - 自分のデータベースで用いる属性は固定(プロパティの追加は考えない)
- **RDFを利用した交換**
  - 役割クラスもしくは共通の詳細化プロパティによるRDFグラフを中間形式として用いる → モデルが一致すれば完全な交換も可能
  - 汎用プロパティとのサブプロパティ関係を利用して変換 → 一方的、もしくは部分的な交換は可能
    - a:翻訳、b:訳者がともにdc:creatorのサブプロパティであっても、a:翻訳 → b:訳者の変換はできない
- **直接データ変換を利用した交換**
  - 相互のデータを交換するための変換規則を用意する
  - 完全～一方向の交換ができるが、スケーラビリティが低い
  - 頻繁にデータ交換を行う重要な関係で用いる

RDFを利用したデータ交換

15

- **役割クラスを介する交換**
  - データを汎用属性＋役割クラスとしてエクスポート(公開)する



	翻訳	
	小田島雄志	

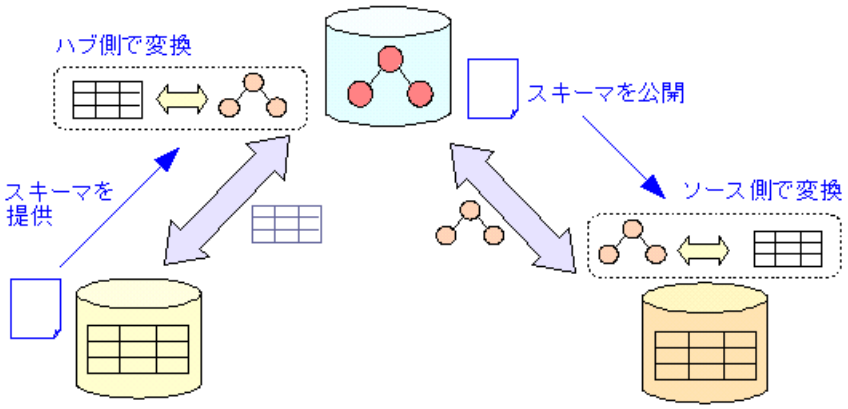
	訳者	
	小田島雄志	

  - 役割クラスに対応するフィールドがあれば、データを直接取り込むことができる
  - 直接対応するフィールドがなければ、汎用プロパティ(dc:creator)に対応するフィールド(作者など)に取り込む
- **共通の詳細化プロパティを用いる交換**
  - 役割クラスよりも直接的に交換が可能
  - 対応するフィールドがない場合はサブクラス定義を利用する
  - 共通詳細化プロパティをどの程度定義し、普及させるかが課題

データ交換とハブ

16

- **ハブとなるサービスを介した交換**
  - 直接交換は対象が増えると効率が悪い → 共通のレポジトリ(ハブ)を介しての交換
  - ハブ段階で情報が失われる(簡略化される)と回復できない → ハブのモデル設計が重要
- **ハブ側変換とソース側変換**
  - データハブが推論・変換を請け負う
    - ソースの負担が軽く、多くの参加を得やすい
  - データ公開(ソース)側がハブ向けに変換を行う
    - データ共有のステップとしては確実だが、ソース側に変換技術が必要
    - ソース側のモデルに応じたきめ細かな変換が必要な場合



相互運用可能な語彙の考え方

17

- **安心して利用できる語彙設計**
  - 名前、URIを変更しなくてよいようにする
    - 名前空間URIにバージョンを入れない(バージョン管理はOWL語彙やドキュメンテーションで)
  - 定義域、値域は慎重に
    - 異なるコミュニティで利用したりグラフを併合したときに不整合を生じる恐れがある
    - OWLクラスのプロパティ制約でローカルな定義を行う
- **必要十分なモデルと語彙**
  - 必要な表現が可能なモデルと語彙を定義する
    - 必要事項が表現できなければ使えない
    - 語彙を無理に単純化する(汎用語彙に絞る)必要はない → 推論、変換で共有する
  - オーバースペックでないほうが使いやすい
    - 利用目的に適したモデルと語彙を用意する(複雑すぎては使われない)
    - 全て自前の語彙でカバーしようとしない(日付にdc:dateが使えるなら使う)
  - プロパティとクラスのバランスを考える
    - 恐らく、プロパティの詳細化よりも役割クラスを利用するほうが共有、交換には有利

詳細プロパティと役割クラスについて

- **汎用性、柔軟性なら役割クラス**
  - 一般にクラス(名詞)のほうがプロパティ(動詞)よりも汎用的なものが提供されている
  - スキーマを参照せずにデータから一段階の絞込みができる(ex:Translatorクラス→dc:creatorプロパティ)
- **記述の簡便さなら詳細プロパティ**
  - 型を省略すれば、グラフ、構文が多少シンプルになる
- **両方同時に使う必要性は低い**
  - 役割クラスTranslatorを使うならプロパティはdc:creatorでよく、詳細化したhasTranslatorを用いなくても同等の情報が伝わる。逆も然り。

参照先

- **このスライド**
  - <http://www.kanzaki.com/works/2007/pub/0309dlw.html>
- **参照したページ**

1. RDF Semantics, by Patrick Hayes and Brian McBride (ed.), 2004-02-10, W3C Recommendation <http://www.w3.org/TR/rdf-mt/>
2. SPARQL Query Language for RDF, by Eric Prud'hommeaux and Andy Seaborne (ed.) <http://www.w3.org/TR/rdf-sparql-query/>
3. DCMI Abstract Model, by A. Powell, M. Nilsson, A. Naeve, P. Johnston, 2005-03-07, DCMI Recommendation <http://dublincore.org/documents/abstract-model/>
4. Domains and Ranges for DCMI Properties, by Andy Powell, 2007-02-05 <http://dublincore.org/documents/2007/02/05/domain-range/>
5. RDF/OWL Representation of WordNet, by M. Assem, A. Gangemi and G. Schreiber (ed.), 2006-06-19, W3C Working Draft <http://www.w3.org/TR/wordnet-rdf/>
6. Metadata Object Description Schema (MODS), The Library of Congress <http://www.loc.gov/standards/mods/>
7. Functional Requirements for Bibliographic Records – Final Report, by IFLA, 1998 <http://www.ifla.org/VII/s13/frbr/frbr.htm>
8. cwm – a general purpose data processor for the semantic web, by Tim Berners-Lee et al. <http://www.w3.org/2000/10/swap/doc/cwm.html>
9. MARC 21 XML Schema, The Library of Congress <http://www.loc.gov/standards/marcxml/>