セマンティックWebにおけるメタデータの問題点に関する研究

2000MT037 亀澤 真一　2000MT049 桑原 基彰

指導教員　青山 幹雄

1. はじめに

ここ数年でインターネットは爆発的に普及し，Web 上には膨大な量の情報が蓄積され続けているため．必要な情報を取り出すにはメタデータの活用が必要となる．

現在の Web では，メタデータは記述や管理，運用などの問題から普及しておらず, 上手く活用されていない．

本研究では，メタデータの記述と運用に着目し，セマンティック Web の第一歩として RDF を利用した図書システムの試作を行い，システムの評価と考察をする．

2. セマンティック Web と RDF

セマンティック Web[1]では，ユーザの要求をコンピュータによる自動処理で解決するために, Web 上の資源すべてに対してメタデータを RDF(Resource Description Framework)[2]で記述し，語彙間の詳細な関係をオントロジーを用いて定義する. コンピュータは, 記述されたメタデータを処理することにより，データの意味を理解することが可能となる．

RDF モデルは図 1 に示すようにリソース，プロパティ，リテラルの 3 つの値を持つステートメントが 1 つ以上存在することにより構成される．リテラルは，他のステートメントのリソースとなることができ，リソースは必ず URI で指定しなければならない．

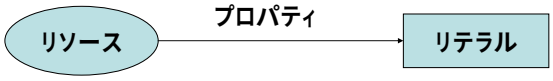

図 1:1 つのステートメントから構成される RDF モデル

RDF ではデータ間の関係を宣言的に表現でき, 曖昧さがなくなり単純になるため，メタデータをコンピュータで扱いやすくなる．

セマンティック Web において基盤技術となる RDF を有効に活用するためには，以下の問題点を解決する必要がある．

(1)　メタデータの信頼性
(2)　メタデータの収集と管理
(3)　メタデータの更新と運用
(4)　メタデータの用途に合わせた記述

本研究では，（4）についてデータモデルの設計を通して検討し，用途に合わせた記述方法を提案する．さらに，実際に RDF を用いたシステムを試作することにより（3）の例を示す．

3. RDF 図書システムの設計と実装

3.1. RDF 図書システムの概要

RDF 図書システムは，書籍のメタデータと書籍のグループ分けに関するメタデータを RDF モデルによって管理し，ユーザインタフェースとして Web ブラウザを使用するシステムである．

書籍に関するメタデータの種類には，ISBN コード，タイトル，著者名，目次，発行機関，発行日付，記述言語，関連書籍が含まれる．

本システムの機能としては，書籍の検索と，RDF ファイルの更新による書籍の追加である．

3.2. データモデルの設計

書籍のメタデータを定義するファイルと, 書籍のカタログ化とジャンルの階層構造を定義するファイルを用意する．システム内では，その 2 つのファイルから作成される RDF モデルを図 2 に示すように結合することで, 1 つの RDF モデルとして処理する．

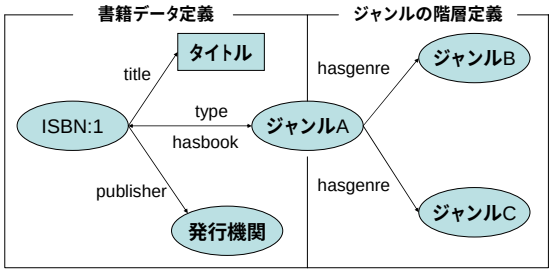

図 2:RDF モデルの結合

2 つの RDF モデルを結び付ける鍵となる書籍とジャンル間の関係を定義する方法として，以下の 3 つが考えられる．図 2 は，（3）の方法により作成される RDF モデルである．

(1)　ジャンル定義の中で書籍への関連を示す

ジャンルをリソースとし，書籍をリテラルとして関係を定義する. 図 2 では, 左向きの矢印が作成されるため, コンピュータは「ジャンル A が持っている書籍は ISBN:1 である」という関連は理解できるが，書籍から見た時のジャンルとの関係は理解できない．また，書籍 isbn:1 とジャンル B，ジャンル C は，意味的に全く異なる別物なので，混乱を避けるためにも異なるプロパティのリテラルとして定義する必要がある.

(2)　書籍定義の中でジャンルへの関連を示す

書籍をリソースとし, ジャンルをリテラルとして関係を定義する方法である．図 2 では，右向きの矢印が作成されるため，コンピュータは「書籍 ISBN:1 のタイプはジャンル A である」という関連は理解できるが，（1）で示した具体的な関係は理解できない.

(3)　（1）と（2）をともに行う

ジャンルと書籍は，それぞれリソースにもリテラルにもなる．図 2 に示した RDF モデルが作成されるため，（1）と（2）で表現した関係をコンピュータは理解することができる．この場合の注意点として，書籍から見たジャンルと，ジャンルから見た書籍が持つ意味は異なるため，異なるプロパティでそれぞれを定義しないと混乱を招く原因となる.

3.3. メタデータの記述

(1)　ジャンルに関するメタデータ定義

ジャンルの階層構造を示す RDF ファイルは, リスト 2 に示す文書構造をとることにする．”my”は，ファイルの先頭で定義されている URI を表す名前空間の接頭辞とし，”hasbook”，”hasgenre”プロパティはその URI を用いて参照される独自に定義したプロパティである.

リスト 1:ジャンルのメタデータ定義

```
<rdf:Description rdf:ID=”ジャンル名”>
  <my:hasbook>
    <rdf:Bag>
      <rdf:li rdf:resource=”ISBN コード” />
    </rdf:Bag>
  </my:hasbook>
  <my:hasgenre>
     <rdf:Bag>
       <rdf:li rdf:resource=”ジャンルを示す URI” />
     </rdf:Bag>
  </my:hasgenre>
</rdf:Description>
```

ひとつのジャンルには，複数の書籍とジャンルが含まれることもあるため，RDF コンテナの Bag コンテナを利用してそれぞれ記述する.

(2)　書籍に関するメタデータ定義

3 つの記述方法は，データ間の関連におけるコンピュータの解釈に違いがあるため，用途に合わせて使い分ければよい．本システムは，特定のジャンルに含まれる書籍を検索するので，（1）の記述方法で十分である．書籍のメタデータを定義する RDF ファイルは，書誌情報に関するプロパティを定義している Dublin Core[3]を利用し，リスト 1 に示す文書構造をとることにする.

リスト 2:書籍のメタデータ定義

```
<rdf:Description rdf:about=”ISBN コード”>
  <dc:titile>書籍のタイトル</dc:title>
  <dc:creator>
    <rdf:Bag>
      <rdf:li rdf:resource=”著者名を示す URI” />
    </rdf:Bag>
  </rdf:creator>
  <dc:description>
     <rdf:Seq><rdf:li>目次</rdf:li></rdf:Seq>
  </dc:description>
  <dc:publisher rdf:resource=”発行機関を示す URI” />
  <dc:date>発行日付</dc:date>
  <dc:language>記述言語</dc:language>
  <dc:relation>
     <rdf:Bag><rdf:li rdf:resource=”関連書籍” /></rdf:Bag>
  </dc:relation>
</rdf:Description>
```

リスト 1 は 1 冊の書籍を定義する RDF である. 著者, 関連書籍は 1 冊の書籍に対して複数存在し，表記の順番に意味はないため RDF の Bag コンテナを利用する．目次も複数存在するが, 表記の順番に意味を持つため RDF の Seq コンテナを利用する．発行機関，人間など現実世界で 1 つしか存在しないものが，RDF モデル中で複数存在すると困るので，それらを表すプロパティのリテラルをリソースとして定義することにした.

3.4. RDF 図書システムの実装

本システムは，Web ブラウザ上で動作する Web アプリケーションとし, Servlet と JSP を利用して実装した.

書籍のメタデータとカタログに関するデータを保持する RDF ファイルは，”BookData.rdf”，”BookType.rdf” とした．メタデータは，各ファイル内でリスト 1 とリスト 2 に示すように RDF の構文規則に従った XML で記述されている．プログラムからこれらの RDF ファイルにアクセスするためには，あらかじめ RDF パーサにより解析しておく必要があるため，本システムでは Jena SemanticWeb Toolkit version2.0 に含まれる RDF パーサを使用した.

4. 評価と考察

本システムでは次の 5 項目について評価し, それぞれ

考察を行う．4.1 節と 4.2 節は，設計したデータモデルに関する評価であり，4.3 節～4.5 節は，本システムの機能に対する評価である．

4.1. RDF によるカタログ化の有用性

XML を利用してカタログ化を行った時と比べることにより，RDF の有用性を評価する．本システムで RDF を用いて作成したカタログは，同じジャンルに属する書籍をまとめ，ジャンルの階層構造を示すものである．

(1) RDF によるカタログ化

RDF では，書籍のメタデータの中にジャンル情報をリソースとして持たせるだけで，図 3 に示すように書籍を表すノードからその書籍が属するジャンルのノードへアークが引かれるため，自然に書籍のグループ化を行える．ジャンルを表すノードへアクセスすれば，そのアークを逆に辿ることにより，そのジャンルに関する書籍を全て知ることができる．

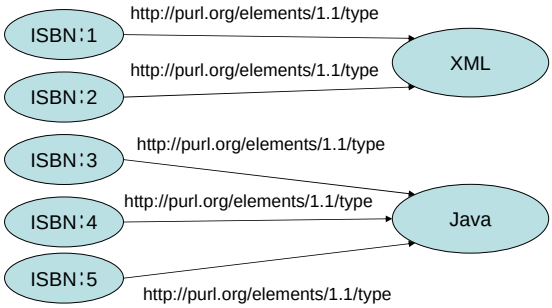

図 3: 書籍のジャンル分けを示す RDF モデル

(2) XML によるカタログ化

図 3 を出力した RDF をただの XML として考え，XML パーサによって解析した結果を図 4 に示す．

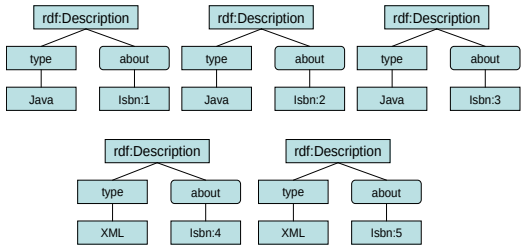

図 4: XML パーサの解析結果

図 4 では，それぞれの書籍のタイプが何かを理解することはできても，そのジャンルには他にどのような書籍が含まれるのかを理解することはできない．仮に，図 4 に加えて，ジャンルと書籍の階層構造を定義したとしても，同じ書籍，同じジャンルに対してもそれぞれが別々のノードを作成するため，コンピュータはその 2 つのノードが同じ書籍，もしくは同じジャンルを表しているとは理解できないだろう. XML で図 3 に示した RDF モデルが表す関係を完全に表現するのは不可能だと考える．

4.2. データの意味の理解度

RDF を用いてメタデータを定義する最大のメリットは，XML 以上にデータの意味を精密化でき，その意味をコンピュータにも理解させることにある．

本システムでは，Dublin Core をメタデータの意味定義に利用したため, RDF を処理するソフトウェア・エージェントは Dublin Core を知ることにより，メタデータの持つ意味が書籍のタイトルなのか，それとも著者名なのかなどを理解することが可能となる．

しかし，Dublin Core のみで書籍に関するメタデータの意味を全て網羅することは不可能である．本システムでは，Dublin Core のみでメタデータを定義しようとしたために，データの意味が一部曖昧になってしまった．

その例として, ある書籍に対する関連書籍を定義する際に，リソースである書籍と関連書籍の関係がすべて，リスト 1 に示すようにプロパティ”dc:relation”により定義されている. Dublin Core で relation プロパティは「関連するリソースへの参照」としか意味が定義されていないため，このままでは書籍と関連書籍の間に，具体的にどんな関係があるのかをコンピュータは理解できない．書籍と関連書籍の関係を「内容が似ている」，「著者が同じ」，「発行機関が同じ」など具体的に表現するためには，その関係を表すプロパティを新たに定義する必要がある．その際には，新たに定義するプロパティも書籍間の関係を表現するプロパティであることを示すために，”dc:relation”プロパティのサブプロパティとして定義すればよいと考える．定義された語彙は，オントロジーで語彙間の関係を定義することで，コンピュータはより深く意味を理解できるだろう．また，独自に定義した語彙は, Web 上に存在するコンピュータに共有させる必要があり，その方法を確立することが課題である．

4.3. 問題追跡機能

本システムでは, あるジャンルからそのジャンルがリテラルとして持つ他ジャンルへ処理対象を移行することと，ある書籍の関連書籍を辿ることを繰り返すことにより 2 つの問題追跡機能を実現している．

前者の例として，ジャンル”プログラミング言語”で検索した場合, 順番にこのジャンルが持つ”C”や”Java”といったジャンルに移動し，さらにそれらが持つジャンルにまで移動して検索を行う．これは，移動先のジャンルが他ジャンルを保持している限り続けられる. 結果として,

ジャンル”プログラミング言語”で検索を行った場合でも，関連のある他ジャンルまで検索対象にすることが可能となった．後者に関しても同様であり，書籍が関連書籍を持ち，処理要求がある限りノード間を追跡し続ける．このため, RDF モデルの中で無限ループに陥らないために，既に検索を行ったノードに目印を付けておく必要がある．

本来意味検索における問題追跡機能では, ソフトウェア・エージェントがユーザからの要求に対してデータの意味を基に最適な追跡先と結果を判断するため，コンピュータがデータの意味の理解度を深めることにより，この機能は充実する．本システムでは，オントロジーを使用しなかったため，コンピュータに対してデータ間の関連における意味定義が不完全となり，それを行うことが不可能であった．また，書籍のメタデータとしてジャンル情報を持たせていないため，もし書籍からジャンルを検索する場合は，コンピュータはそれぞれが持つ意味を理解できないため意味検索は不可能となる．常に意味検索が要求されるセマンティック Web では，データ間の関連を明確にする必要があるため，リソース間の関連は常に双方向とも定義しておく方が良い．

4.4. メタデータ入力インタフェース

コンピュータによる RDF の自動生成を行う場合でも，メタデータを人間が入力する必要がある．コンピュータは入力されたメタデータを基にして RDF モデルを作成する．その際に，メタデータ入力のインタフェースは．記述形式が単純な N-Triples のように，人間にとって解りやすい構成になっていなければならない．本システムでは，このインタフェースを図 5 に示すように，N-Triples に似せることで，メタデータを入力しながら作成する RDF モデルをイメージし易くすることでメタデータ入力インタフェースが持つ課題を解決した．

図 5:メタデータ入力のインタフェース

図 5 に示したインタフェースが N-Triples に似ている点は，各行のデータの並びと，1 行が 1 ステートメントを表現している点である．

4.5. RDF の自動生成

本システムの RDF 自動生成機能では．ユーザから入力されたメタデータを基にして, 設計段階で考えた RDF モデルと同一なそれをシステム内で作成する．ファイルに出力する際に，それを XML 形式に変換するが，出力された XML の文書構造は，リスト 1 で示す設計段階で考えた XML と異なる．これは，RDF の自動生成に使用した Java の API が持つ仕様のためだと考える. しかし，RDF では RDF モデルの構造が同じならば XML の文書構造が異なっても問題ではない. 実際に，RDF パーサで解析[4]したところ，どちらの XML からも同一な RDF モデルが出力された．これは，コンピュータにとって 2 つの XML が持つ意味は同一であることを表している．

ただし，XML パーサによる解析結果では，2 つの XML は異なる文書構造を持つため，異なるツリーが作成されてしまう．

5. おわりに

本研究では，セマンティック Web 発展段階の第 1 段階である「メタデータの活用」に属し，書籍のカタログ化と書籍に関するメタデータの定義を RDF を用いて行い，それらを利用した書籍検索機能と RDF の自動生成機能を持った図書システムを Java 言語により試作した．

RDF を用いて作成したカタログの解析結果を XML で作成したカタログの解析結果と比べることで，RDF メタデータが XML メタデータよりも効果的に処理が可能であることを示した．

また, 図書システムを評価, 考察することによりセマンティック Web 発展段階の第 2 段階へ向けた課題を以下のように整理することができた．

(1) オントロジーによる詳細な語彙の定義と方法
(2) オントロジーのアプリケーションへの適用
(3) 独自に定義した語彙の共有方法の確立

参考文献

[1] 荻野達也，特集セマンティック Web，情報処理 Vol.43 No.7，p.707-750 （2002）
[2] 神崎正英，RDF-リソース表現のフレームワーク，http://www.kanzaki.com/docs/sw/rdf-model.html （2002）
[3] Dublin Core Metadata Initiative, http://dublincore.org/（2003）
[4] W3C，RDF Validation Service, http://www.w3.org/RDF/Validator/ （2003）