メタデータ・レジストリの機能要件

メタデータ情報基盤構築事業
2010 年 11 月 18 日

目 次

1　スキーマ登録に関する要件

レジストリに既存スキーマを登録し、維持更新するための要件を定義する。ここで用語は以下の意味で用いる。

- 語彙定義とその記述規則（アプリケーション・プロファイルを含む）を総称して「**スキーマ**」と呼ぶ
- メタデータ記述に用いるクラス、プロパティを「**ターム**」と呼び、タームを一つの名前空間としてまとめたものを「**語彙**」と呼ぶ。ターム、語彙はそれぞれ URI によって参照される
- 語彙は RDFS（RDF Schema Language）および OWL（Web Ontology Language）で表現する。記述規則（アプリケーション・プロファイル）は、別途定める **DSP**（Description Set Profile）を用いて表現する（これらの基底モデルは RDF）
- スキーマを上記 RDF モデルで表現するとき、そのスキーマ表現に含まれる RDF トリプル全体の集合を「**グラフ**」と呼ぶ。グラフも URI で名前付けし、参照する

1.1　RDF グラフとして表現されるスキーマを格納、管理できる

レジストリに収録するスキーマは RDF で表現し、処理する。

1.1.1　スキーマの RDF トリプルをデータベースに格納する

適当な RDF 処理ライブラリとデータベース結合 API を用いて、スキーマを表現した RDF のトリプルをデータベースに格納し、照会、更新などの操作を行なう

1.1.2　スキーマごとに RDF グラフを区別し、指定グラフのみを処理できる

各スキーマのグラフを URI で名前付けする（**グラフ URI**）。各トリプルについて、それぞれが属するグラフ URI も保持（トリプル＋グラフ URI ＝四つ組みなど）し、指定するグラフのみを対象に操作できるようにする。データセットなどの概念でもよい。

1.1.3　スキーマ RDF グラフのメタデータを管理できる

各グラフ（スキーマ）について、名称、出所、登録日時などのメタデータを保持する。このメタデータも RDF で表現する。

1.2　RDF 形式の語彙定義を確認しながら登録できる

一つの名前空間で構成される語彙をレジストリに登録する。登録する語彙の定義はあらかじめ RDF で記述するものとする（複数の語彙を組み合わせる「記述規則」の登録要件は 1.3 で示す。表形式のものは記述規則として扱う）。

1.2.1 RDFS、OWL オントロジー、TopicMaps 語彙定義ファイルを読み込むことができる

- RDF/XML および Turtle 形式を読み込んで RDF トリプルを取得できる
- TopicMaps のファイルを読み込み、RDF に変換して処理できる
- 入力はローカルファイル選択のほか、URI 指定によるウェブからの読み込み、テキストエリアへの貼り付けに対応する

1.2.2 読み込んだ RDF グラフを、操作者が確認できるように画面表示する

- ターム一覧：タームの継承関係（サブクラス、サブプロパティ）を階層構造として表示する。この一覧を目次とし、次のターム概要にリンクする
- ターム概要：共通の主語（ターム URI）の元にトリプルをまとめてプロパティ＝値ペアのリストとして表現し、目的語が URI 参照である場合はハイパーリンクを設定する。リンク先は、同一語彙の場合はページ内リンク、レジストリに登録のある語彙（以下登録語彙）はそのターム詳細表示（2.1.6）、それ以外は目的語 URI とする
- プロパティに DC との関係が設定されていないとき、可能であればサブプロパティ関係を追加するよう促す（3.2.3 の dumb down で使用）
- 入力ファイルのパース中にエラーが生じた場合は、どの部分でどんなエラーが生じたかを操作者に分かりやすく示し、入力ファイルの該当部分前後を表示する

1.2.3 語彙に関するメタデータを登録するための画面を提供する

- グラフ URI は、語彙定義の名前空間が得られる場合はその URI をデフォルトとする
- 語彙定義によって語彙のメタデータが異なる場合（たとえばラベルプロパティが `rdfs:label`, `dc:title` など）、検索利用に不便をきたすので、一定の規則に基づいて記述プロパティを統一して登録する。
- スキーマにもともと記述されているメタデータは、そのままグラフ URI を主語として登録する。
- これに加え、追加分を含めた管理用メタデータは、別途メタデータ URI を主語として記述し、`foaf:primaryTopic` でグラフ URI と結びつける。

1.3 表などの半形式的定義をもつ記述規則 DSP を対話的に登録できる

記述規則は DSP（Description Set Profile）として表現する。DSP において、一つのレコードの記述規則を **DescriptionTemplate** と呼ぶ。また記述項目（プロパティ）ごとの記述制約規則を **StatementTemplate** と呼ぶ。DescriptionTemplate は 1 つ以上の StatementTemplate を「持つ」ことで構成される。DSP の詳細仕様については別途定める（概要は 4 参照）。

記述規則のほとんどは表の形で提供されていると考えられるので、ここでは表からの対話的登録の要件を定める。DSP として RDF で表現されているものは、1.3.3 の登録を直接行なう。

1.3.1 表から基本的な StatementTemplate を生成する

記述項目一覧の形で表がある場合、項目ごとに StatementTemplate を生成する。

- 記述項目名を StatementTemplate の URI 参照とし、さらに辞書などを用いて適当なプロパティ名にマップし:property とする。マップができない場合は、項目名をそのままプロパティとする
- 記述内容説明に相当する項目を、:usage とする
- 出現頻度に相当する項目を、:minOccur、:maxOccur にマップする（「推奨」など数値表現できないものの扱いは別途検討する）
- 値制約は、仮に:literalValue :PlainLiteral. とする。別テーブルの実体を ID などで参照することが分かる場合は、:nonLiteralValue <#table.field>. のような形とする。
- それ以外の欄がある場合は、いったん:comment に取り込む
- 記述項目名が ID に相当する場合は、StatementTemplate ではなく、DescriptionTemplate でレコード URI を与えるための制約として扱う（1.3.3）

表 1: 入力表の例

項目	記述内容	頻度	補足
タイトル	作品のタイトル	1	サブタイトルも含む
作者	作品の作者	必須	作者テーブルの ID を用いる
テーマ	作品のテーマ	推奨	国会図書館主題件名を用いる

たとえば、表 1 が与えられた場合、ここから概ね次の StatementTemplate を自動生成する。

```
<#タイトル> a StatementTemplate;
    rdfs:label "タイトル";
    :property dc:title;
    :minOccur 1;
    :maxOccur 1;
    :literalValue :PlainLiteral;
    :usage "作品のタイトル";
    :comment "サブタイトルも含む".

<#作者> a StatementTemplate;
    rdfs:label "作者";
    :property dc:creator;
    :minOccur 1;
    :literalValue :PlainLiteral;
    :usage "作品の作者";
    :comment "作者テーブルの ID を用いる".
```

```
<#テーマ> a StatementTemplate;
    rdfs:label "テーマ";
    :property undefined:テーマ;
    :minOccur 0;
    :literalValue :PlainLiteral;
    :usage "作品のテーマ";
    :comment "国会図書館主題件名を用いる".
```

1.3.2 自動生成した StatementTemplate を対話的に修正できる

生成した StatementTemplate を項目名ごとに画面表示し、表から読み取った定義をユーザが修正できるようにする。

- プロパティ名（:property の値）は、登録済み語彙から置き換え候補を選択できるようにする（上位・下位プロパティ関係も利用する）。適当なプロパティが登録されていない場合、「独自語彙定義」マークをつけて、後工程（1.3.4）で語彙登録を行なう。
- 値制約は、リテラル、非リテラルの選択と、値の制約（値域）の選択を行なえるようにする。
- リテラルの値域は、XML データ型のほか、NDC、NDLSH、BSH などの統制語彙をあらかじめ用意して（複数）選択可能にする。また、可能な値を列挙して限定する場合は、カンマ区切りなどのデータ入力で限定クラスをその場で定義する（:usage、:comment の欄からコピーして貼り付けられる）
- 非リテラルの値域は、foaf:Person などの登録済みクラスとともに、登録済み DescriptionTemplate を候補として示し、わかりやすく選択できるようにする。

こうした対話操作の結果、前項の StatementTemplate<#作者>、<#テーマ>を次のように修正できる。

```
<#作者> a StatementTemplate;
    rdfs:label "作者";
    :property dc:creator;
    :minOccur 1;
    :nonLiteralValue foaf:Person;   #非リテラルに変更
    :usage "作品の作者";
    :comment "作者テーブルの ID を用いる".

<#テーマ> a StatementTemplate;
    rdfs:label "テーマ";
    :property dc:subject;           #登録済みプロパティに変更
    :minOccur 0;
    :literalValue dcndl:NDLSH;      #データ型を指定
    :usage "作品のテーマ";
```

```
    :comment "国会図書館主題件名を用いる".
```

1.3.3 DescriptionTemplate の生成と登録

StatementTemplate の修正が完了したら、名前空間を対話的に決定し、DescriptionTemplate としてデータベースに登録する（表ではなく RDF 形式の DSP として提供される記述規則があれば、ファイルを読み込んでこのステップから登録を開始する）。

スキーマ名称、登録者、登録日時などのメタデータも合わせて記述・登録する。スキーマの分類（分野）の登録についても検討する。

```
<http://example.org/dsp/organization/tablename> a :DescriptionTemplate;
    rdfs:label "○○文書館スキーマ";
    rdfs:comment "文書の記述に用いる";
    :hasStatement
        <#タイトル>,
        <#作者>,
        <#テーマ>,
        ...;
    :statementOrder "#タイトル,#作者,#テーマ,...";
    :dcterms:created "2011-01-15";
    :dcterms:creator ex:someonesId;
    :owl:versionInfo "第 1.1 版".
```

※ RDF として登録するとトリプルの順序が失われてしまうので、たとえば:statementOrder のような形で並び順を保持しておく。

なお、StatementTemplate 修正時に「独自語彙定義」マークが付けられている場合は、次項 1.3.4 の独自語彙登録を行なってプロパティ名を決定してからデータベースに登録する。

1.3.4 独自語彙の登録

1.3.2 の StatementTemplate 修正時に「独自語彙定義」マークをつけたプロパティを、対話形式に登録する。登録する項目は

- プロパティ名（URI、元表の項目名をデフォルト）
- ラベル（rdfs:label、元表の項目名をデフォルト）
- 上位プロパティ（rdfs:subPropertyOf、オプション）
- 定義域、値域（rdfs:domain、rdfs:range、オプション）
- 説明（rdfs:comment、推奨）

とする。
上位プロパティは、可能なら項目名から辞書を参照して候補を示す。定義域、値域は、登録済みクラスから選択するか、必要に応じて独自クラス登録サブ画面を提供する。
クラスを登録する場合は、

- クラス名（URI）
- ラベル（`rdfs:label`）
- 上位クラス（`rdfs:subClassOf`、オプション）
- 説明（`rdfs:comment`、推奨）

を入力項目とする。上位クラスは、可能であれば登録済みクラスから選択可能にする。
入力を終えたら、作者、組織、作成日などのメタデータとともに RDFS を生成し、データベースに登録する。
また、StatementTemplate で参照しているプロパティ名を、登録名に置き換える。

1.3.5 シンプル DC へのマッピングの登録

可能であれば、3.2.3 のシンプル DC へのマッピングを、記述規則登録者が作成する。これにより、シンプル DC への「公式マッピング」が提供されることになる。

1.4 スキーマファイルを解析して変換の前処理を行なう

表形式スキーマからの DSP 生成（1.3）の前処理として、各種フォーマットのスキーマファイルを DSP 化可能な表データに変換する。

1.4.1 エクセルのワークシートの読み取り

エクセルで作成された表を直接読み取ることができる。

- 左、上セルに余白がある場合は削除する。
- いくつかのフォーマットを想定して、可能であれば自動判定、でなければフォーマットを選択して読み込みを行なう。
- 作成日、版などの注記がある場合は可能な範囲でメタデータとして取り込む。

1.4.2 PDF ファイルの読み取り

表が PDF ファイルになっている場合、ページ区切りにおけるノンブルや脚注、表のセル内改行の扱いなど、抽出テキストをそのまま表として扱えないケースが多いので、フィルタリングを行なう。

1.4.3 一覧表形式ではないスキーマ定義

一覧表ではなく、項目ごとに表を作成しているスキーマも想定される。たとえばDC-NDL のアプリケーション・プロファイル定義は、次のようにプロパティごとに“ 表 ”を用意した PDF となっている。

```
プロパティURI   http://purl.org/dc/terms/title
QName           dcterms:title
定義の発生源    DCMI Metadata Terms
                http://purl.org/dc/terms/
語彙のタイプ    プロパティ
表示名          Title
使用法          当該情報資源のタイトルをここに収める。
補足説明        タイトル関連情報（サブタイトル等）がある場合は、...
語彙符号化スキームの使用    指定しない
値の記述形式    URI による   任意の文字列  構文符号化スキーム  入れ子
                記述         による記述    による記述          による記述
                不可         可            不可                可
表現例（RDF/XML）    <dcterms:title> ...
入力レベル      必須
```

この場合、各表の 1 列目を項目名、2 列目をその値として取り込んで、DSP 化可能な一覧データに変換しておく必要がある。自動変換が難しい場合も想定されるので、何らかの対話的な変換方法を用意する。

1.5 登録スキーマの更新ができる

登録済みスキーマの更新、バージョン管理、部分修正ができるようにする。

1.5.1 スキーマ全体の更新

スキーマ全体を更新する場合は、新規登録と同じ手順でデータを取り込み、新しいバージョンとして登録する。登録に際しては、メタデータは旧バージョンのものをデフォルトで示し、必要部分を変更登録できるようにする。

1.5.2 新旧バージョンの保存

旧バージョンは削除せず、遡って参照できるようにする（2.7.1 参照）。

- 最新スキーマと保存スキーマの 2 種のテーブルを用意し、スキーマ登録時に両方のテーブルに登録する（テーブルを分けなくても可）
- 最新スキーマテーブルには、スキーマのグラフ URI を用いて登録し、通常の検索などは全てこのテーブル、グラフ URI に対して実施する

- 保存テーブルでは、スキーマのグラフ URI に登録年月日を加えたグラフバージョン URI を用いて登録する
- スキーマ更新時には、(1) 最新テーブルのグラフは新しいものに置き換え、(2) 保存テーブルでは旧グラフはそのまま残して、新しいグラフをグラフバージョン URI を用いて追加登録する

たとえば国会図書館 DC-NDL スキーマのグラフ URI が `http://ndl.go.jp/dcndl/terms/`であるとき、2011-01-15 に登録したスキーマのグラフバージョン URI は `http://ndl.go.jp/dcndl/terms/?20110115` のような形にする。

（バージョン URI の日付をクエリ（?で区切る情報）とするかどうかは要検討）

1.5.3 スキーマの部分修正

スキーマの最新グラフは、誤りの訂正などのために修正を施すことができるようにする。

- 指定されたスキーマのグラフをデータベースから取り出し、語彙登録確認（1.2.2）もしくは記述規則の確認編集（1.3.2、1.3.3）において修正を行なう
- グラフメタデータの修正日を自動更新する
- 最新テーブルおよび保存テーブルの当該スキーマのグラフを削除し、修正したものに置き換える。このとき、保存テーブルのグラフバージョン URI は**変更しない**（グラフバージョン URI は登録時の日付で固定）

この修正は、誤字訂正、登録ミスなど、基本的にスキーマの意味論を変えない範囲に限る。値制約の変更など意味論が変わる変更を伴う場合は、ひとつのターム、StatementTemplate の修正であっても、スキーマ全体の更新として扱い、バージョンを変える。

※登録直後に誤りに気付いて値制約を変更するなどのケースも考えられるので、この機能で意味論修正に関するチェック、禁止は行なわない。変更してしまうと、同じバージョンで意味論の異なるスキーマが存在することになってしまうので、注意を喚起する。

2 コンテンツ・メタデータ提供者のレジストリ利用に関する要件

基盤を利用してコンテンツ提供者がメタデータを付与したり、新たなスキーマを構築するための要件を定義する。

2.1 登録されている記述規則、語彙定義の一覧、詳細を表示できる

画面のメニュー／リンクをたどることで、登録されている記述規則、語彙定義をブラウズできるようにする。

2.1.1 記述規則の一覧表示

登録されている DescriptionTemplate を、登録組織別、または分類別に一覧して表示できる。一覧の表示にはラベルと説明を用いる。DescriptionTemplate を選択（クリック）すると、その詳細説明を表示する。

2.1.2 DescriptionTemplate の詳細表示

レコード記述規則（DescriptionTemplate）のメタデータ（タイトル、作成者ほか）と、そこで記述する StatementTemplate を表示する。

- メタデータは囲みあるいは背景色などで、概要であることが分かるように表示する。
- StatementTemplate は（出現順に）リストもしくは表形式で、ラベル、用法、頻度制約、値制約を一覧表示し、クリックで StatementTemplate 詳細画面を表示する。
- 一覧には、2.4.1 の記述規則コピー編集、2.4.2 の記述規則組み合わせで用いるために、StatementTemplate を選択するチェックボックスを用意する
- レコード記述規則を 3.3.1 の機械可読形式で取得するためのボタンを用意する

2.1.3 StatementTemplate の詳細表示

- 項目記述規則（StatementTemplate）の制約規則を、表もしくは定義型リストとして表示する。
- 2.4.2 の記述規則組み合わせで用いるために、利用選択チェックボックス（「この項目記述規を新規レコード記述規則に加える」など）を用意する。
- 2.4.3 の独自 StatementTemplate 作成で用いるために、編集ボタン（「この項目記述規をコピーして新規項目記述規を作成・編集する」など）を用意する。

2.1.4 語彙定義の一覧表示

登録されている語彙定義の RDFS を一覧して表示できる。一覧の表示にはラベルとコメント（の先頭部分）を用いる。語彙名を選択（クリック）すると、その詳細説明を表示する。

2.1.5 語彙定義の詳細表示

メタデータ（タイトル、作成者ほか）と、語彙に含まれるターム（クラス、プロパティ）を表示する。

- メタデータは囲みあるいは背景色などで、概要であることが分かるように表示する。
- タームは、クラス、プロパティごとにまとめて、リスト形式で示す。上位プロパティ、上位クラスがある場合は階層リストとして表示する。
- クリックで各タームの詳細画面を表示する。
- 語彙定義を 3.3.1 の機械可読形式で取得するためのボタンを用意する

2.1.6 ターム定義の詳細表示

- 各タームの URI、ラベル、および定義 RDF トリプルのプロパティ＝値ペアを、表もしくは定義型リストとして表示する。
- 上位プロパティ、上位クラス、定義域、値域の値となるタームは、登録されていればクリックすることでそれぞれの詳細を表示する。
- プロパティの場合、このターム（プロパティ）を用いている StatementTemplate をリスト表示する。これもクリックで詳細表示する。

2.2 登録されている記述規則、語彙定義を検索し、該当するものを表示できる

利用法に応じてレジストリを検索し、必要な記述規則、語彙定義を発見できるようにする。

2.2.1 自由キーワードによる記述規則、語彙定義検索

- 自由キーワードを用いて、記述規則、語彙定義のラベル、説明などのメタデータ項目をまたいだ検索を行なう
- 検索結果をレコード記述規則、項目記述規則、語彙定義、ターム定義に分けて一覧表示する。一覧の表示方法は、2.1 の一覧表示に準じる

2.2.2 語彙による検索

語彙を指定して、その語彙を利用するレコード記述規則を一覧表示する。検索には DescriptionTemplate の生成と登録時（1.3.3）に用意した、レコード記述規則の使用語彙リストを用いる。

2.2.3 プロパティによる項目記述規則検索

プロパティを指定し、そのプロパティを用いる項目記述規則（StatementTemplate）を検索、一覧表示する。

- プロパティは、登録語彙ごとにメニューなどで選択できるようにする
- 語彙が分からずプロパティ名のみが念頭にある場合（`creator` など）は、自由入力として、ラベルを対象にまずプロパティを検索し、そのプロパティを用いて StatementTemplate を検索する
- プロパティが下位プロパティを持つ場合（たとえば `dc:creator` の下位プロパティとして `ex:composer` が独自に登録されている場合）、これらも含めて検索対象にするオプションを用意する

たとえば、`dc:creator` およびその下位プロパティを用いて記述する項目記述規則の検索は、SPARQL では次のようになる。

```
SELECT ?template WHERE {
    {?template a :StatementTemplate;
        :property dc:creator.}
    UNION
    {?template a :StatementTemplate;
        :property [rdfs:subPropertyOf dc:creator].}
}
```

2.3 記述規則、語彙定義を複数フォーマットで取得できる

ウェブブラウザ画面での閲覧に加え、記述規則、語彙定義の RDF グラフそのものを取得、利用できるようにする。提供フォーマットは複数用意する。

2.3.1 複数形式の RDF 記述

- デフォルトで RDF/XML、ほかに Turtle を選択できるようにする。
- ダウンロードに用いる URI は、3.3.1 のコンピュータ問い合わせ用 REST の URI と同じになるようにする。

2.3.2 定義 RDF を TopicMaps に変換して取得できる

2.4 独自記述規則の作成を支援できる

登録済み記述規則では不十分あるいはオーバースペックである場合、独自の記述規則を作成できる。登録済み記述規則をベースに、対話的に操作ができる。

2.4.1 独自の記述規則を対話的に定義・登録できる

ウイザード的に手順を追いながら、独自の記述規則を定義・登録する機能を提供する。

- 2.1.2 の記述規則詳細画面からレコード記述規則をコピーし、一部を編集する形で独自記述規則を作成できる
- 不要な StatementTemplate を削除したり、登録済み StatementTemplate から選んだものを追加したり、新たな StatementTemplate を作成して追加することができる（次項以降参照）
- 白紙の新規記述規則状態からはじめることもできる（StatementTemplate を持たないダミー記述規則をコピーして編集する）。
- 作成した記述規則で 2.6.3 の入力 HTML を表示してみて、意図どおりの規則になっているかどうか確認することができる。
- 確認の結果、前のステップに戻って修正することができる。

- 作成を終えた記述規則に、「DescriptionTemplate の生成と登録」（1.3.3）と同様の画面でメタデータを与えて登録する。

2.4.2 登録済項目記述規則を選んで、独自レコード記述規則に組み込むことができる

新しいレコード記述規則を作るときに、登録されている項目記述規則を部品として組み合わせることができる。

- ブラウズあるいは検索結果の規則一覧で、必要な StatementTemplate を複数選択できる。
- StatementTemplate の詳細を確認してからの選択もできるよう、詳細画面でも同様に選択できる。

選択が終了したら、「DescriptionTemplate の生成と登録」（1.3.3）と同様の画面でメタデータを入力する。このとき、StatementTemplate の削除や追加などの変更を可能にする

2.4.3 登録済項目記述規則を編集して新しい規則を作成し、独自レコード記述規則に組み込むことができる

検索しても適切な項目記述規則が見当たらない場合、既存の StatementTemplate をベースにして制約記述を編集し、新たな StatementTemplate を作成できるようにする。

- 編集の画面は、「自動生成した StatementTemplate を対話的に修正」（1.3.2）のステップに準じる。
- コピー元となった StatementTemplate との関係をメタデータに加える。
- 作成した StatementTemplate を登録し、独自レコード記述規則に組み込むことができる

2.4.4 新規の項目規則を定義・登録できる

- 空のテンプレートとなるダミーの StatementTemplate を用意しておき、このテンプレートを 2.4.3 の手順で編集することで、新規の項目規則を作成できるようにする。
- 未登録プロパティを用いたい場合は、RDFS/OWL の形で記述されている語彙定義であれば、「RDF 形式の語彙定義を確認しながら登録」（1.2）の手順に従いレジストリに追加登録できる。まずこの登録を行なって、登録済みプロパティとして利用可能にした上で、StatementTemplate を編集する。

2.5 複数記述規則間の関連を調べ、分かりやすく示すことができる

記述規則をブラウズ、検索しただけではどれが適切なものか分からない場合などに、複数の記述規則の類似関係を調べ、選択判断の補助とすることができる。

2.5.1 複数のレコード記述規則を表形式で列挙し、共通プロパティ項目規則をハイライトできる

複数のレコード記述規則の間で、同じプロパティを用いる項目記述規則（共通プロパティ項目規則）がどれであるかを示す。

- 基準となるレコード記述規則を一つ選び、比較記述規則を選択すると、各規則に含まれる StatementTemplate を縦に並べた表として表示できる
- 基準規則の StatementTemplate に対して、比較記述規則の StatementTemplate の中で同じプロパティを用いるものを調べ、ハイライト表示する（可能であれば線で結ぶなどする。あるいは同じ行に並ぶようにする）
- 同様に、上位・下位関係にあるプロパティを用いるものをハイライト表示する

2.5.2 共通プロパティ項目規則を表形式で複数列挙し、共通する制約規則をハイライトできる

記述に用いるプロパティに注目し、同じプロパティを用いる項目記述規則（共通プロパティ項目規則）の間で、制約規則がどの程度共通か、あるいは異なるかを示す。

「プロパティによる項目記述規則検索」（2.2.3）の結果表示のオプションとして、StatementTemplate を横に、制約規則を縦に並べた表形式の表示を行い、共通する制約規則があればハイライト表示する。

※この機能は複雑すぎる場合は省いてよい

2.5.3 プロパティ類似度による記述規則検索ができる

選択したレコード記述規則に対し、登録されている他レコード記述規則それぞれにおいて《共通プロパティを用いる項目記述規則の数》を調べ、その数が多い（一致度が高い）順に一覧表示する。オプションで、結果から選択したレコード記述規則間の関係を 2.5.1 の表を用いて表示する。

※実用的な時間内に検索が困難であれば、この機能は省いてもよい

2.6 選択した記述規則に基づくメタデータ記述の支援できる

記述規則を利用したメタデータ活用のため、メタデータの検証、規則に基づくメタデータの作成・変換を支援する。

2.6.1 メタデータが記述規則どおりに書かれているかどうか検証できる

- 入力フィールド、もしくはファイルから読み込んだメタデータを、選択した記述規則に対して検証する
- 規則に反する部分があれば、入力メタデータのどの箇所が、どの規則に合致しないかを分かりやすく示す
- 構文レベルではなく、RDF グラフに対する検証を行なう（一般に同じグラフを異なる構文で記述できる）

※構文による検証は入力データに対してはできないが、(1) いったん RDF グラフとして取り込んだうえ、(2) で内部的に RDF ライブラリの RDF/XML 出力機能を使って定型の XML とし、(3) 構文規則から生成した XML スキーマを用いて、(4) 既存の XML 妥当性検証ツールで検証する、という方法を採ることはできる（ライブラリの生成する RDF/XML は一定の構文になるよう調整できる）。ただしこの場合、エラーの報告を XML 構文エラーとして示すのではなく、元の入力規則に対するエラーに翻訳する必要がある。

2.6.2 記述規則に基づいて、表形式のデータを RDF に変換できる

- 表データの 1 行目が項目名となっており、
- 記述規則の StatementTemplate 名（URI 参照のローカル名）が表の項目名と対応するとき、
- DescriptionTemplate で ID 欄となっている項の値目から各レコードの URI を生成し、
- StatementTemplate のプロパティ、制約規則に基づいて、各項目のデータを RDF トリプルに変換する

2.6.3 記述規則に基づいて、データ入力の HTML フォームテンプレートを用意できる

- 記述規則の StatementTemplate 名をラベルとし、対応する `input` 要素を持つ HTML の `form` 要素を生成する
- `input` 要素の `name` 属性値にも記述規則の StatementTemplate 名を用いる
- 制約規則に基づき、必須入力などをスクリプトで制御する。複数入力可の場合、スクリプトで入力フィールドを追加できるようにする
- 制約が列挙型の選択である場合、`select` 要素を用いて選択できるようにする

利用者は、このフォームの HTML ソースをコピーして、データベース用の入力フォームを容易に作成できる。レジストリでこのフォームにデータを入力して送信した場合は、前項 2.6.2 の手順で RDF に変換した結果を表示する。

2.6.4 記述規則に基づいて、データ入力のエクセル入力ウイザードを用意できる

※ニーズがありそうならば、前項 2.6.3 の HTML フォームに準じて作成する。オプション

2.7 コンテンツの長期保存・利用のためのメタデータ記述標準

コンテンツの長期保存・利用のために、メタデータも長期にわたって適切に利用できるよう、記述規則を提供する。

2.7.1 記述規則のバージョンを管理し、指定時期の規則を参照できる

バージョン情報と登録日によって、過去の記述規則を参照することができる。

3　サービス提供者の高度なメタデータ利用に関する要件

ウェブサービスなどの提供者が、コンテンツ提供者のメタデータを利用して、マッシュアップなど高度なサービスを展開するための要件を定義する。

3.1　レジストリを検索し、語彙、記述規則を調べることができる

コンテンツ提供者のメタデータをサービス提供者（メタデータ／コンテンツ利用者）が理解できるよう、メタデータに用いられている語彙や記述規則を検索して調べることができる。

3.1.1　キーワードで記述規則、語彙の一覧を検索できる

コンテンツ・メタデータ提供者のキーワード検索（2.2.1）を参照。

3.1.2　URI を指定して語彙、タームを検索できる

語彙定義の RDF をグラフ URI を指定して取得する（3.3.1）のとは別に、URI から 2.1.5 の定義詳細を表示できるようにする。

3.1.3　メタデータから、使用語彙一覧と類似記述規則一覧を得ることができる

メタデータインスタンス（RDF）を入力して、使用されている語彙の一覧と、条件の近い記述規則の一覧を得ることができる。

- 名前空間宣言から使用している語彙を調べて、登録されているものを一覧表示。未登録語彙の場合は、名前空間 URI へのリンクを表示
- 使用語彙で記述規則を絞り込み、用いられているプロパティの種類が一致もしくは近いものを表示する

3.2　メタデータ変換の支援

コンテンツ提供者のメタデータをサービス提供者（メタデータ／コンテンツ利用者）がより利用しやすい形で扱えるよう、変換を支援する。

3.2.1　メタデータを、JSON フォーマットに変換する

登録記述規則によるメタデータ（インスタンス RDF）を JSON 形式に変換し、ウェブアプリケーションから利用しやすくする。

- レコードを JSON オブジェクトとして表現する
- 主語 URI を `id` の値とする
- メタデータのプロパティはローカル名のみを取り出して JSON のプロパティとする

- 値はすべてリテラルとしてあつかう。入れ子の RDF は、JSON オブジェクトの入れ子構造で表現する。同じプロパティの繰り返しは、JSON 配列とする
- 変換結果のサンプルをユーザに示し、必要に応じて JSON を修正できるようにする
- 得られた変換アルゴリズムを、JavaScript、PHP などポピュラーな言語で提供し、利用者がプログラムの部品として使えるようにする

3.2.2 メタデータを、TopicMaps に変換する

3.2.3 メタデータを、シンプル DC による記述に変換する（dumb down）

マッシュアップなどの利用を容易にするために、異なるスキーマによるメタデータの共通化・標準化として、シンプル DC へのマッピングを提供する。

1.3.5 の「公式マッピング」もしくは第三者によるマッピングが登録されていれば、そのマッピング ID を、メタデータ（作成者、日付など）とともに利用者に知らせる。マッピングがなければ、利用者は次の手順でマッピングを作成して登録できる。

- 登録されている記述規則を用いた任意のメタデータ RDF を、プロパティの上位プロパティ関係（および記述規則）を利用して、すべてリテラル値のシンプル DC に変換する。
- シンプル DC との関係が定義されていないプロパティはいったんすべて `dc:description` とする
- メタデータのサンプルレコードをフォームに入力するか、ファイルを読み込んで、マッピング案を自動作成して利用者に示す
- マッピング案は利用者が修正できる
- シンプル DC は、JSON フォーマットでの出力も用意し、利用者が選択できる
- 得られたマッピングアルゴリズムを、JavaScript、PHP などポピュラーな言語で提供し、利用者がプログラムの部品として使えるようにする
- マッピングは ID を与え、メタデータとともにレジストリに登録し、同様のメタデータ変換で再利用できるようにする（3.3.3 も参照）

※記述規則は、たとえば `title` プロパティ値が構造化されて読みも含む場合、`dc:title` へは本来のタイトル（`rdf:value` 値など）のみを取り込み、読みの値は別プロパティに振り分けるなどの最適化に利用する。

3.3 コンピュータによる問い合わせインターフェイス

サービス提供者は、人手によるレジストリ利用の他に、API 経由でプログラムからレジストリを利用し、自動化サービスに結びつけることができる。

3.3.1 記述規則、語彙定義の RDF を、グラフ URI を指定して取得できる

- `{レジストリ URI}?graph={グラフ URI}`の形で問い合わせ、取得できる
- パラメータとしてフォーマットを指定できる（`format=xml,turtle,json`）。デフォルトは XML

3.3.2 キーワード検索、プロパティ指定検索を、REST API で実行できる

- 結果は記述規則、語彙定義のグループに分けた上で、スキーマのラベルと前項 3.3.1 のグラフ取得 URI をセットにする
- オプションで結果フォーマットを指定できる（XML もしくは JSON、デフォルトは XML）

XML 結果フォーマットの例：

```
<search>
  <request>
    <type>property</type>
    <value>dc:contributor</value>
  </request>
  <results_set>
    <vocabulary_results>
      <vocaburary label="Dublin Core"
         uri="http://meta-proj.jp/?graph=http://purl.org/dc/elements/1.1/"/>
      ...
    </vocabulary_results>
    <template_results>
      <template label="国会図書館 DC-NDL"
         uri="http://meta-proj.jp/?graph=http://ndl.go.jp/dcndl/terms/"/>
      ...
    </template_results>
  </results_set>
</search>
```

※この XML は、(1)search 要素を rdf:RDF に置き換えて名前空間宣言；(2)vocabulary_results、template_results 要素に属性 rdf:parseType="Resource"を追加；(3)label、uri 属性をそれぞれ rdfs:label、rdf:resource に置き換え；という手順で RDF/XML として扱うこともできる。

3.3.3 登録マッピングを用いて、URI で指定したメタデータを変換できる

ウェブアプリケーションは、変換アルゴリズムを自身のプログラムに組み込む代わりに、レジストリのサービスをゲートウェイとしてメタデータを変換・利用することも可能にする。

- dumb down マッピング（3.2.3）としてあらかじめ登録したシンプル DC へのマッピングと、変換対象メタデータの URI を、{レジストリ URI}?map={マッピング ID}&source={対象メタデータ URI} の形で指定する
- レジストリは、対象メタデータ URI から取得したデータを登録マッピングのアルゴリズムで変換し、シンプル DC としてのメタデータを返す
- オプションで結果フォーマットを指定できる（XML もしくは JSON、デフォルトは XML）

4　DSP（Description Set Profile）の概要

機能要件で求められる DSP の役割を理解するために、概要を説明する。

4.1　DescriptionTemplate と StatementTemplate

DSP において、一つのレコードの記述規則を **DescriptionTemplate** と呼ぶ。また記述項目（プロパティ）ごとの記述制約規則を **StatementTemplate** と呼ぶ。DescriptionTemplate は 1 つ以上の StatementTemplate を「持つ」ことで構成される。

4.1.1　DSP の RDF 表現

DSP は RDF を用いて表現する（以下の RDF 記述は暫定案）。

```
<#sample_template> a :DescriptionTemplate;
    :hasStatement
        [a :StatementTemplate;
            #制約規則
        ...],
        [a :StatementTemplate;
            #制約規則
        ...],
```

4.1.2　再利用可能な StatementTemplate

StatementTemplate は独自の URI 参照を持つものとして記述し、再利用することもできる。

```
<#title_template> a :StatementTemplate;
    #制約規則
    ...

<#sample_template> a :DescriptionTemplate;
    :hasStatement
        <#title_template>,
        <#name_template>,
        ...
```

メタデータ・レジストリは、スキーマ（記述規則）の共有と再利用を目的とするので、原則として StatementTemplate の記述はこの形を用いる。レジストリは、これら URI 参照を持つ DescriptionTemplate、StatementTemplate のトリプルを一つのまとまりとして扱う（登録、取り出し、更新）ことができなければならない。

4.2 StatementTemplate の基本構造

StatementTemplate は、

- メタデータでの項目記述に用いるプロパティ名
- （レジストリで）StatementTemplate 自身の表示に用いるラベル
- 当該記述項目の出現回数制約（必須、繰り返し可など）
- 値の制約（文字列、日時などのデータ型つき文字列、人物などのエンティティなど）
- 記述に関する文章による説明

で構成する。値の制約は、リテラル値と非リテラル値（エンティティ）を区別して扱う。

```
<#title_template> a  :StatementTemplate;
    rdfs:label "タイトル";
    :property dcterms:title;
    :minOccur 1;
    :maxOccur 1;
    :literalValue :PlainLiteral;
    :usage "作品のタイトルを記述する…".
```

値がリテラルでなくエンティティである場合、再利用可能なものはそれ自体を DescriptionTemplate として定義し、URI で参照する。

```
<#creator_template> a  :StatementTemplate;
    rdfs:label "作者";
    :property dcterms:creator;
    :minOccur 1;
    :nonLiteralValue <#Agent_description>;
    :usage "作品の作者を記述する…".
```

4.3 DescriptionTemplate によるエンティティの定義

エンティティはそれ自身が独立したレコード記述になり得るので、DescriptionTemplate として記述する。

```
<#Agent_description> a :DescriptionTemplate;
    :hasURI "optional";
    :resourceClass foaf:Agent;
    :hasStatement
        [:property foaf:name; minOccur 1; maxOccur 1],
        [:property foaf:homepage; minOccur 0]
```