ebXML　**解説書**

第 4 部

ebXML レジストリ・リポジトリ

平成 14 年 3 月

(財)日本情報処理開発協会 電子商取引推進センター

この解説書は、経済産業省委託「平成 13 年度 精算･調達・運用支援統合情報システムに関する調査研究」事業の成果です。

はじめに

1999 年 11 月に、国際標準 EDI（UN／EDIFACT）の利用グループの支援を受けた国連 CEFACT（Center for Trade Facilitation and eBusiness）と、先進的 IT ベンダーのコンソーシアムである OASIS (Organisation for Advanced Structured Information Standards)の協業で始められた ebXML イニシャチブは、2001 年 5 月、今後の電子ビジネスコラボレーション実現のフレームワークとなる一連の仕様（ebXML 仕様）の第 1 版を完成し公表した。

ebXML 仕様は、従来のレガシーEDI や WEB－EDI を XML 化するに留まらず、取引企業同士のそれぞれのアプリケーションが、情報交換により合意されたビジネスプロセスを遂行してビジネス目標を達成する、すなわち電子ビジネスコラボレーションを実現させるために必要な標準仕様を定めている。

今後、当該標準仕様は、IT ベンダーの戦略的製品やサービスに取り入れられるとともに、ユーザー業界においてはビジネスプロセス改善の仕組みに採用されて行くことが期待されている。

(財)日本情報処理開発協会では経済産業省の委託を受けて、2001 年 5 月に公表された ebXML 仕様を中心に、電子商取引推進協議会の平成 13 年度 XML／EDI 標準化専門委員会の討議結果を反映し、次の 6 部からなる解説書を作成した。

第1部　ebXML 概要
第2部　ebXML ビジネスプロセス
第3部　ebXML 情報構成要素
第4部　ebXML レジストリ・リポジトリ
第5部　ebXML 交換協定
第6部　ebXML 通信仕様

なお、ebXML 仕様は、2001 年 5 月以降、第 2 章－第 3 章関連は UN／CEFACT、第 4 章－第 6 章関連は OASIS が仕様の改訂・保守を継続しており、ebXML 仕様の実装においては該当組織より発表されている最新版の仕様を参照されることを推奨する。

平成 14 年 3 月
財団法人日本情報処理開発協会
電子商取引推進センター

第 4 部 ebXML レジストリ・リポジトリ　序文

「ebXML レジストリ・リポジトリ」は、ebXML イニシャチブにより作成された ebXML レジストリ情報モデル、および ebXML レジストリサービス仕様を掲載する。

<参照 ebXML 仕様書>

ebXML Registry Information Model V1.0	2001 年 5 月 8 日
ebXML Registry Service Specification V1.0	2001 年 5 月 10 日

ebXML レジストリ情報モデル v1.0

ebXML レジストリプロジェクトチーム

2001 年 5 月 8 日

技術検証　ECOM XML/EDI 標準化専門委員会

1 本書の位置付け

本書は、電子ビジネスコミュニティ向けの ebXML ドラフト標準を指定している。

本書は自由に配布できる。

本書の書式は、インターネット・ソサエティの標準 RFC に基づいている。

本書のバージョン：
http://www.ebxml.org/specs/ebRIM.pdf

最新バージョン：
http://www.ebxml.org/specs/ebRIM.pdf

2 ebXML の参加者

本書の開発・拡張に多大な貢献して頂いた以下の人々に感謝する。

Lisa Carnahan, NIST
Joe Dalman, Tie
Philippe DeSmedt, Viquity
Sally Fuger, AIAG
Len Gallagher, NIST
Steve Hanna, Sun Microsystems
Scott Hinkelman, IBM
Michael Kass, NIST
Jong.L Kim, Innodigital
Kyu-Chul Lee, Chungnam National University
Sangwon Lim, Korea Institute for Electronic Commerce
Bob Miller, GXS
Kunio Mizoguchi, Electronic Commerce Promotion Council of Japan
Dale Moberg, Sterling Commerce
Ron Monzillo, Sun Microsystems
JP Morgenthal, eThink Systems, Inc.
Joel Munter, Intel
Farrukh Najmi, Sun Microsystems
Scott Nieman, Norstan Consulting
Frank Olken, Lawrence Berkeley National Laboratory
Michael Park, eSum Technologies
Bruce Peat, eProcess Solutions
Mike Rowley, Excelon Corporation
Waqar Sadiq, Vitria
Krishna Sankar, Cisco Systems Inc.
Kim Tae Soo, Government of Korea
Nikola Stojanovic, Encoda Systems, Inc.
David Webber, XML Global
Yutaka Yoshida, Sun Microsystems
Prasad Yendluri, webmethods
Peter Z. Zhoo, Knowledge For the new Millennium

目 次

図目次

表目次

3 はじめに

3.1 本書の概要

本書は、ebXML レジストリの情報モデルを説明している。

別の文書「ebXML レジストリサービス仕様[ebRS]」は、ebXML レジストリの情報コンテンツへのアクセスを提供するレジストリサービスの構築方法を説明している。

3.2 一般規則

o 概念を簡潔に説明する手段として UML 図を使用する。UML 図は、特定の実装や手法の要件を特定するためのものではない。

o UML 図ではインタフェースが多用される。インタフェースは、属性のあるクラスの代わりに使用し、抽象的な定義を提供するもので、特定の実装を意図するものではない。特に、レジストリ内のオブジェクトにインタフェースを経由して直接アクセスすることを意味するものではない。レジストリ内のオブジェクトは、ebXML レジストリサービス仕様で説明しているインタフェースを経由してアクセスする。インタフェースの各 get メソッドは、マッピング先となる属性名を明確に示している。たとえば、getName メソッドは、name と名付けられた属性にマッピングする。

o 用語「リポジトリ項目」は、保管および保護のためにレジストリに登録されているオブジェクトを特定するために使用する（たとえば、XML 文書や DTD）。リポジトリ項目はすべて、RegistryEntry インスタンスによって表記する。

o 用語「RegistryEntry」は、コンテンツのインスタンス（リポジトリ項目）に関するメタデータを提供するオブジェクトを特定するために使用する。

o 用語「RegistryObject」は、一般的な用語「オブジェクト」との混同を避けるために、情報モデルの基本インタフェースを指すために使用する。ただし、用語「オブジェクト」を、情報モデルのクラスやインタフェースを指すために使用するときは、ほぼすべてのクラスは RegistryObject のインスタンスであるため、用語「オブジェクト」は RegistryObject を意味している。

情報モデルは、レジストリの実際のコンテンツ（リポジトリ）を扱わない。情報モデルのすべての要素は、コンテンツに関するメタデータを表し、コンテンツ自身を表すものではない。

ソフトウェアの実装者は、ebXML 準拠のソフトウェアを作成するときに、本書と他の ebXML 仕様文書を組み合わせて使用することができる。

本書で使用するキーワード『MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, OPTIONAL』は、RFC 2119 [Bra97]の説明通りに解釈されるものとする。

3.2.1 命名規則

本書における大文字の使用と命名規則を確実にするために、「Upper Camel Case (UCC)」と「Lower Camel Case (LCC)」の大文字スタイルは、以下の規則で使用する。

・要素名は UCC 規則に従う。
（例：<UpperCamelCaseElement/>）
・属性名は LCC 規則に従う。
（例：<UpperCamelCaseElement
lowerCamelCaseAttribute="Whatever"/）
・クラス、インタフェース名は UCC 規則を使用する。
（例：ClassificationNode, Versionable）
・メソッド名は LCC 規則を使用する。
（例：getName(), setName()）
また、大文字のイタリック体の単語は、ebXML 用語解説[ebGLOSS]で定義されている。

3.3 対象者

この仕様の対象者は、以下のソフトウェア開発者から構成されるコミュニティである。

o ebXML レジストリサービスの実装者
o ebXML レジストリクライアントの実装者

3.4 関連文書

本書の背景と関連情報のいくつかは、以下の仕様に基づいている。

a) ebXML レジストリサービス仕様- 情報モデルに基づいた実際のレジストリサービスを定義

b)コラボレーションプロトコルプロファイルおよびコラボレーションプロトコル合意書仕様– 取引当事者のプロファイルの定義方法と、2 取引当事者が当事者契約を定義するためのプロファイルの使用方法を定義

c) ebXML ビジネス・プロセス仕様スキーマ

d) ebXML テクニカルアーキテクチャ仕様

4 設計目的

4.1 目標

仕様のこのバージョンの目的は、以下の通りである。

o レジストリ内にある情報と、その情報の構成方法を知らせる。

o OASIS [OAS]およびISO 11179 [ISO]レジストリモデルで行われた作業をできるだけ多く利用する。

o その他の ebXML ワーキンググループの関連作業との調和を図る。

o 将来の ebXML レジストリ要件をサポートする。

o 他の ebXML 仕様と互換性を保つ。

5 システムの概要

5.1 ebXML レジストリの役割

レジストリは、登録組織によって提出される情報の永続的で安定した保管を提供する。こうした情報は、ebXML をベースとした企業対企業（B2B）パートナーシップとトランザクションを容易にするために使用される。提出されたコンテンツは、XML スキーマおよび文書、プロセス説明、コア構成要素、コンテキスト説明、UML モデル、取引当事者情報、さらにはソフトウェアコンポーネントにすることもできる。

5.2 レジストリサービス

レジストリのクライアントにレジストリコンテンツへのアクセスを提供するレジストリサービスのセットは、ebXML レジストリサービス仕様に定義されている。本書は、こうしたサービスの詳細を提供していないが、参照として使用することができる。

5.3 レジストリ情報モデルとは何か

レジストリ情報モデルは、ebXML レジストリの青写真または高レベルのスキーマを提供している。レジストリ情報モデルの第一の重要性は、ebXML レジストリを実装するためのものである。従って、レジストリ情報モデルは、このような実装に対し、レジストリに格納されているメタデータの型と、メタデータとクラス間の関連の情報を提供する。

レジストリ情報モデルとは、以下の通りである。

o どの型のオブジェクトがレジストリに格納されるのかを定義する。
o 格納されたオブジェクトがレジストリでどのように構成されるのかを定義する。
o いくつかのワーキンググループが作成した ebXML メタモデルに基づく。

5.4 レジストリ情報モデルの働き

ebXML レジストリの実装者は、レジストリの実装に含めるクラスと、こうしたクラスが持つ属性とメソッドを決定するために、情報モデルを使用できる。また、レジストリの実装に必要になる可能性があるデータベーススキーマの種類を決定するためにも、情報モデルを使用することができる。

[注意]　情報モデルは例証となるべきもので、実装についてどのような特定の選択も指定していない。

5.5 レジストリ情報モデルを実装できる場所

レジストリ情報モデルは、リレーショナルデータベーススキーマ、オブジェクトデータベーススキーマ、その他の物理的スキーマの形で、ebXML レジストリに実装することができる。また、レジストリ実装内のインタフェースおよびクラスとして実装することもできる。

5.6 ebXML レジストリへの適合性

実装がこの仕様への適合性を要求する場合、その実装は、ebXML レジストリサービスを通して見ることができる必要なすべての情報モデル・クラスおよびインタフェース、その属性および意味的定義をサポートする。

6 レジストリ情報モデル: 高レベルのパブリックビュー

本節では、レジストリ内で最も一般的なオブジェクトの高レベルのパブリックビューを説明する。

図 1 は、レジストリ内オブジェクトの高レベルのパブリックビューと、UML クラス図でその関連を示している。これは、継承、クラス属性またはクラスメソッドを示すものではない。
また、この情報モデルは、実際のリポジトリ項目のモデルではないことに注意して欲しい。

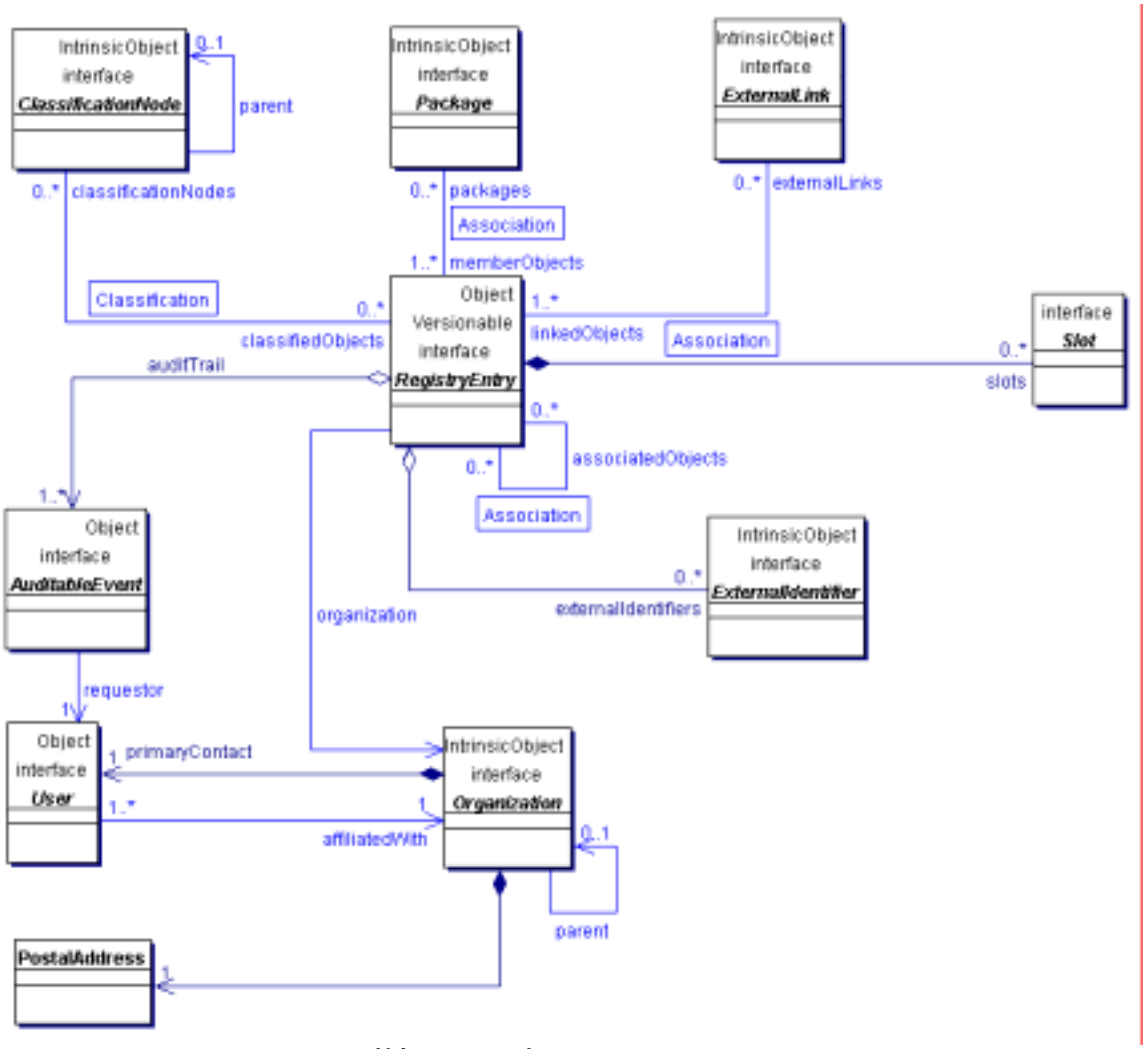


図 1: 情報モデルの高レベルパブリックビュー

6.1 RegistryEntry（レジストリエントリ）

情報モデルの中央のオブジェクトは RegistryEntry である。RegistryEntry のインスタンスは、レジストリに提出されたそれぞれのコンテンツインスタンスに対して存在する。
RegistryEntry クラスのインスタンスは、リポジトリ項目に関するメタデータを提供する。実際のリポジトリ項目（たとえば、DTD）は、RegistryEntry クラスのインスタンスには含まれ

ていない。情報モデルのほとんどのクラスは、RegistryEntry の特化されたサブクラスであることに注意して欲しい。各 RegistryEntry は、正確に 1 つのリポジトリ項目に関連付けられている。

6.2 Slot（スロット）

Slot インスタンスは、RegistryEntry インスタンスに任意の属性を追加する動的な方法を提供する。RegistryEntry インスタンスに動的に属性を追加するこの機能により、レジストリ情報モデル内の拡張性が可能になる。

6.3 Association（関連）

Association インスタンスは、情報モデル内のオブジェクト間で、多対多の関連を定義するために使用される RegistryEntry である。関連については、10 節で詳しく説明する。

6.4 ExternalIdentifier（外部識別子）

ExternalIdentifier インスタンスは、DUNS 番号、社会保障番号、または組織の別名といった補足的な識別子情報を RegistryEntry に提供する。

6.5 ExternalLink（外部リンク）

ExternalLink インスタンスは、レジストリによって管理されないコンテンツに対し、指定された URI のモデルを作成する RegistryEntry である。管理されたコンテンツとは異なり、こうした外部コンテンツは、レジストリの知らない間にいつでも変更や削除が実行される可能性がある。RegistryEntry は、任意の数の ExternalLink と関連付けることができる。登録組織がリポジトリ項目（たとえば、DTD）を提出し、特定の外部コンテンツをそのオブジェクト（たとえば、登録組織のホームページ）に関連付けることを希望する場合に、ExternalLink はこの機能を実現する。ExternalLink 機能は、ExternalLink を RegistryEntry に対して表示する GUI ツールの中にある。ユーザはこのようなリンクをクリックし、リンクが参照している外部の web ページに移動することができる。

6.6 ClassificationNode（分類ノード）

ClassificationNode インスタンスは、各ノードが ClassificationNode となるツリー構造を定義するために使用される RegistryEntry である。ClassificationNode と共に構成される分類ツリーは、分類スキームまたは存在論（オントロジー）を定義するために使用される。ClassificationNode については、11 節で詳しく説明する。

6.7 Classification（分類）

Classification インスタンスは、RegistryEntry インスタンスを分類スキーム内の ClassificationNode に関連付けることにより、リポジトリ項目を分類するために使用される RegistryEntry である。Classification については、第 11 章で詳しく説明する。

6.8 Package（パッケージ）

Package インスタンスは、論理的に関連する RegistryEntry をグループ化する RegistryEntry である。Package の使用目的は、オブジェクトのパッケージ全体で操作を実行できるようにすることにある。たとえば、1 つの Package に属するすべてのオブジェクトを単独の依頼で削

除することができる。

6.9 AuditableEvent（監査可能イベント）

AuditableEvent インスタンスは、RegistryEntry を監査追跡するために使用する Object である。AuditableEvent については、8 節で詳しく説明する。

6.10 User（ユーザ）

User インスタンスは、レジストリに登録しているユーザの情報を提供するために使用する Object である。User オブジェクトは、RegistryEntry に対する監査追跡で使用される。User については、8 節で詳しく説明する。

6.11 PostalAddress（郵便アドレス）

PostalAddress は、郵便アドレスの属性を定義する、再利用可能な単一のエンティティクラスである。

6.12 Organization（組織）

Organization インスタンスは、登録組織のような組織の情報を提供する RegistryEntry である。それぞれの Organization インスタンスは、親 Organization への参照を持つことができる。

7 レジストリ情報モデル: 詳細ビュー

本節では、情報モデルクラスについて、パブリックビューよりも詳しく説明する。詳細ビューでは、情報モデルのパブリックビューで説明しなかったモデルに補足的ないくつかのクラスを導入する。

図 2 は、継承、すなわち、情報モデル内のクラス間の「is a」関連を示しており、これは、「has a」関連を示していない。「has a」関連は図 1 ですでに解説している。図 2 では、クラス属性とクラスメソッドも示されていない。ほとんどのインタフェースとクラスのメソッドおよび属性の詳細な説明は、各クラスの説明後に表の形で説明している。

インタフェース Association については、10 節で別に詳しく説明する。インタフェース Classification および ClassificationNode は、11 節で詳しく説明する。

サンプルの情報モデルは、実際のリポジトリ項目のモデルを作成していない。

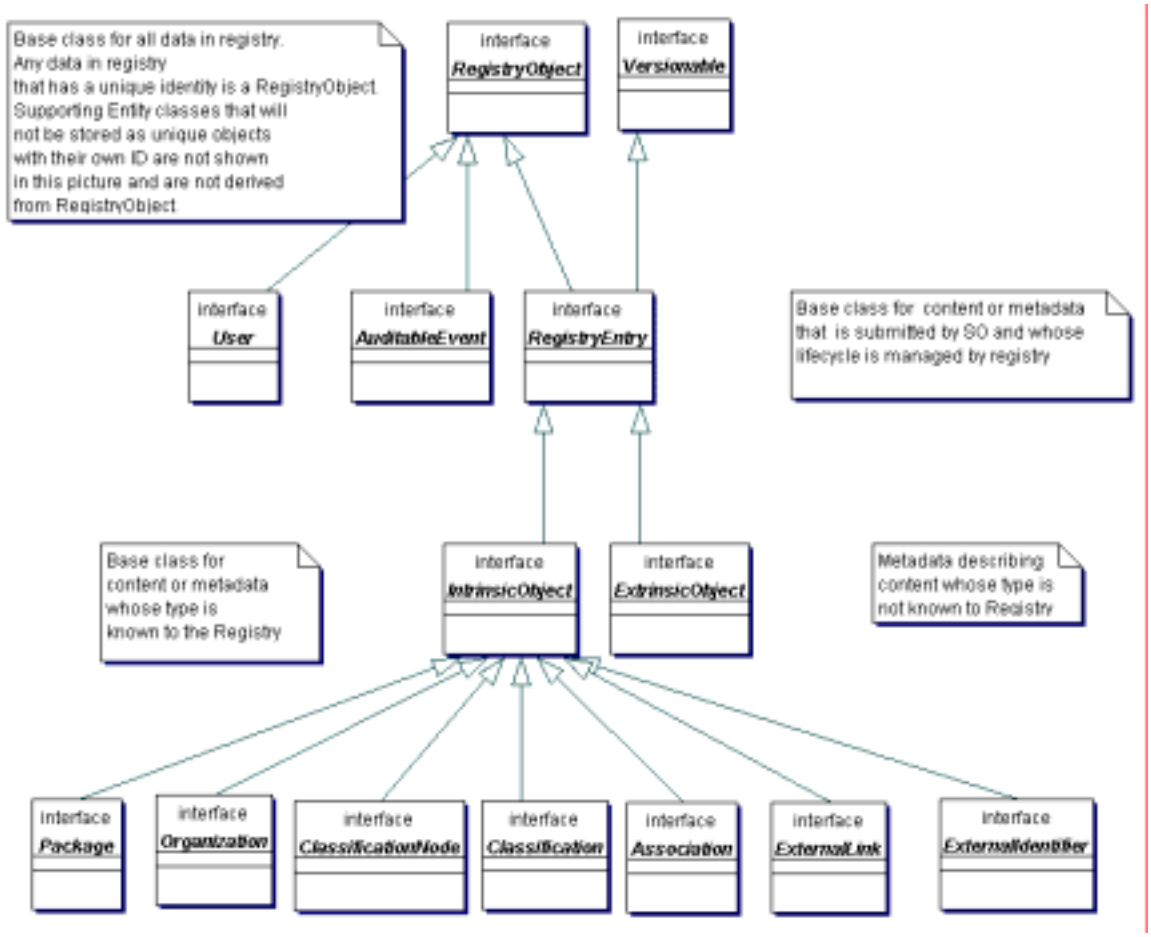


図 2: 情報モデル継承ビュー

7.1 インタフェース RegistryObject

既知のすべての下位インタフェース:
Association, Classification, ClassificationNode, ExternalLink, ExtrinsicObject, IntrinsicObject, RegistryEntry, Organization, Package, User, AuditableEvent, ExternalIdentifier

RegistryObject は、情報モデル内のほとんどすべてのオブジェクトに対して共通の基本インタフェースを提供する。インスタンスが一意のアイデンティティと独立したライフサイクルを持っている情報モデルクラスは、RegistryObject クラスのインスタンスである。

Slot および PostalAddress は、RegistryObject クラスのインスタンスではない。その理由は、それらのインスタンスは、独立した存在と一意のアイデンティティを持っていないからである。これらは、常にその他のクラスのインスタンスである（たとえば、Organization は PostalAddress を持っている）。

RegistryObject メソッドの概要

AccessControlPolicy	getAccessControlPolicy () Object と関連付けられた AccessControlPolicy オブジェクトを取得する。AccessControlPolicy は、その RegistryObject といっしょに “ 誰が何をすることを許可されているか ” に関して、RegistryObject に関連付けられたセキュリティモデルを定義する。accessControlPOlicy と名付けられた属性にマッピングする。
String	getDescription () RegistryObject に対して、コンテキスト独立のテキスト説明を取得する。description と名付けられた属性にマッピングする。
String	getName () リポジトリ内の RegistryObject のユーザフレンドリーなコンテキスト独立名称を取得する。name と名付けられた属性にマッピングする。
String	getID () RegistryObject に対して普遍的に一意の ID ([UUID]と定義される) を取得する。id と名付けられた属性にマッピングする。
void	setDescription (String description) RegistryObject に対してコンテキスト独立のテキスト説明を設定する。
void	setName (String name) RegistryObject のユーザフレンドリーなコンテキスト独立の名称を設定する。
void	setID (String id) RegistryObject に対して普遍的に一意の ID ([UUID]と定義される) を設定する。

7.2 インタフェース Versionable

既知のすべての下位インタフェース:
Association, Classification, ClassificationNode, ExternalLink, ExtrinsicObject, IntrinsicObject, RegistryEntry, Organization, Package, ExternalIdentifier

Versionable インタフェースは、インスタンスのバージョンを作成することができるクラスに共通の動作を定義する。現在のところ、すべての RegistryEntry クラスは、Versionable インタフェースを実装するために必要である。

Versionable メソッドの概要	
int	getMajorVersion () Versionable オブジェクトのこのバージョンに対し、メジャー改訂番号を取得する。majorVersion と名付けられた属性にマッピングする。
int	getMinorVersion () Versionable オブジェクトのこのバージョンに対し、マイナー改訂番号を取得する。minorVersion と名付けられた属性にマッピングする。
void	setMajorVersion (int majorVersion) Versionable オブジェクトのこのバージョンに対し、メジャー改訂番号を設定する。
void	setMinorVersion (int minorVersion)

	Versionable オブジェクトのこのバージョンに対し、マイナー改訂番号を設定する。

7.3 インタフェース RegistryEntry

すべての上位インタフェース:
RegistryObject, Versionable

既知のすべての下位インタフェース:
Association, Classification, ClassificationNode, ExternalLink, ExtrinsicObject, IntrinsicObject, Organization, Package, ExternalIdentifier

RegistryEntry は、ライフサイクルがレジストリによって管理される提出されたコンテンツを説明するすべてのメタデータに対する共通の基本クラスである。レジストリに提出されたコンテンツを説明するすべてのメタデータは、RegistryEntry の ExtrinsicObject および IntrinsicObject サブクラスによってさらに特化される。

RegistryEntry メソッドの概要	
Collection	getAssociatedObjects () RegistryObject に関連付けられた RegistryObject の一覧を返す。associatedObjects と名付けられた属性にマッピングする。
Collection	getAuditTrail () AuditableEvent オブジェクトの命令された Collection として、RegistryObject の状態変化に影響を与えたすべての結果の完全な監査追跡を返す。auditTrail と名付けられた属性にマッピングする。
Collection	getClassificationNodes () RegistryObject に関連付けられた ClassificationNodes の一覧を返す。classificationNodes と名付けられた属性にマッピングする。
Collection	getExternalLinks () RegistryObject に関連付けられた ExternalLinks の一覧を返す。externalLinks と名付けられた属性にマッピングする。
Collection	getExternalIdentifiers () RegistryObject に関連付けられた ExternalIdentifiers の一覧を返す。externalIdentifiers と名付けられた属性にマッピングする。
String	getObjectType () RegistryEntry と関連付けられた事前定義オブジェクト・タイプを取得する。これは、7.3.2 に説明されているように、オブジェクト・タイプの名前にしなければならない。objectType と名付けられた属性にマッピングする。
Collection	getOrganizations () RegistryObject に関連付けられた Organizations の一覧を返す。organizations と名付けられた属性にマッピングする。
Collection	getPackages () RegistryObject に関連付けられた Packages の一覧を返す。packages と名付けられた属性にマッピングする。
String	getStatus () レジストリ内にある RegistryEntry のライフサイクル・ステ

	ータスを取得する。これは、7.3.1 に説明されているように、RegistryEntry ステータス型の名前にしなければならない。status と名付けられた属性にマッピングする。
String	getUserVersion () レジストリ内にある RegistryEntry の userVersion 属性を取得する。userVersion は、ユーザによって割り当てられたような RegistryEntry に対するバージョンである。
void	setUserVersion (String UserVersion) レジストリにある RegistryEntry の userVersion 属性を設定する。
String	getStability () レジストリにある RegistryEntry に対してスタビリティインジケータを取得する。スタビリティインジケータは、コンテンツに対する安定性の保証として、提出者によって提供される。これは、7.3.3 に説明されているように、Stability 型の名前にしなければならない。stability と名付けられた属性にマッピングする。
Date	getExpirationDate () レジストリ内にある RegistryEntry の expirationDate 属性を取得する。この属性は、Stability によって提供された安定性保証の有効期限を定義する。expirationDate に達すると、Stability 属性は、コンテンツをいつでもどの方法でも変更できることを意味する STABILITY_DYNAMIC になる。ヌル値は、Stability 属性について期限がないことを意味する。 expirationDate と名付けられた属性にマッピングする。
void	setExpirationDate (Date ExpirationDate) レジストリ内にある RegistryEntry の expirationDate 属性を設定する。
Collection	getSlots () この RegistryObject に動的に追加されたスロットの一覧を取得する。slots と名付けられた属性にマッピングする。
void	addSlots (Collection newSlots) 1 つまたは複数のスロットをこの RegistryObject に追加する。スロット名は、この RegistryObject 内でローカルに一意の名称にしなければならない。どのような既存のスロットも影響を受けない。
void	removeSlots (Collection slotNames) この RegistryObject から 1 つまたは複数のスロットを削除する。削除されるスロットはその名前によって識別される。

インタフェース RegistryObject から継承されるメソッド
getAccessControlPolicy, getDescription, getName, getID, setDescription, setName, setID

インタフェース Versionable から継承されるメソッド
getMajorVersion, getMinorVersion, setMajorVersion, setMinorVersion

7.3.1 事前定義 RegistryEntry ステータス型

以下の表は、RegistryEntry ステータス属性のために事前定義された選択の一覧である。これらの事前定義ステータス・タイプは、Classification スキームとして定義されている。スキー

ムを簡単に拡張することができるが、レジストリは以下のリストにあるステータスをサポートしなければならない。

名前	説明
Submitted	レジストリに提出されたコンテンツをカタログ化する RegistryEntry のステータス
Approved	レジストリに提出されてから承認を受けたコンテンツをカタログ化する RegistryEntry のステータス
Deprecated	レジストリに提出されてから却下されたコンテンツをカタログ化する RegistryEntry のステータス
Withdrawn	レジストリから回収されたコンテンツをカタログ化する RegistryEntry のステータス

7.3.2 事前定義オブジェクト・タイプ

以下の表は、事前定義オブジェクト型のリストである。ExtrinsicObject に対して、ExtrinsicObject がカタログ化するリポジトリ項目の型に基づいて定義される多くの型があることに注意が必要である。さらに、具体的なインスタンスを持つこともできる IntrinsicObject サブクラスに対して定義されたオブジェクト型もある。

これらの事前定義ステータス型は、Classification スキームとして定義されている。スキームを簡単に拡張することができるが、レジストリは以下のリストにあるステータスをサポートしなければならない。

名前	説明
Unknown	タイプが指定されていないか未知のコンテンツをカタログ化する ExtrinsicObject
CPA	XML 文書コラボレーションプロトコル合意書 (CPA) をカタログ化する型の ExtrinsicObject。本書は 2 つの当事者間の技術的合意で、特定のプロトコルを使って互いに通信する方法を提示する。
CPP	コラボレーションプロトコルプロファイル (CPP) と呼ばれる文書をカタログ化する型の ExtrinsicObject。本書は、取引トランザクションに参加する当事者に関する情報を提供する。
Process	プロセスを説明する文書をカタログ化する型の ExtrinsicObject
Role	コラボレーションプロトコルプロファイル (CPP) の役割について XML 説明をカタログ化する型の ExtrinsicObject
ServiceInterface	コラボレーションプロトコルプロファイル仕様書 で定義されている通り、サービスインタフェースの XML 説明をカタログ化する型の ExtrinsicObject
SoftwareComponent	ソフトウェアコンポーネントをカタログ化する型の ExtrinsicObject（たとえば、EJB またはクラスライブラリ）
Transport	コラボレーションプロトコルプロファイル仕様書で定義されている通り、トランスポート設定（transport configuration）の XML 説明をカタログ化する型の ExtrinsicObject
UMLModel	UML モデルをカタログ化する型の ExtrinsicObject
XMLSchema	XML スキーマ（DTD, XML Schema, RELAX 文法など）をカタログ化する型の ExtrinsicObject

Package	Package オブジェクト
ExternalLink	ExternalLink オブジェクト
ExternalIdentifier	ExternalIdentifier オブジェクト
Association	Association オブジェクト
Classification	Classification オブジェクト
ClassificationNode	ClassificationNode オブジェクト
AuditableEvent	AuditableEvent オブジェクト
User	User オブジェクト
Organization	Organization オブジェクト

7.3.3 事前定義 RegistryEntry Stability Enumerations（スタビリティ列挙）

以下の表は、RegistryEntry スタビリティ属性に対する事前定義選択の一覧である。これらの事前定義スタビリティ型は、Classification スキームとして定義されている。スキームを簡単に拡張することができるが、レジストリは以下の一覧にあるステータスをサポートできる。

名前	説明
Dynamic	コンテンツが動的で、提出者によっていつでも自由に変更される可能性があることを示す RegistryEntry のスタビリティ
DynamicCompatible	コンテンツが動的で、提出者によっていつでも下位互換の方法で変更される可能性があることを示す RegistryEntry のスタビリティ
Static	コンテンツが静的で、提出者によって変更されないことを示す RegistryEntry のスタビリティ

7.4 インタフェース Slot

Slot インスタンスは、RegistryEntry インスタンスに自由に属性を追加するための動的な方法を提供する。RegistryEntry インスタンスに属性を動的に追加するこの機能は、レジストリ情報モデルの拡張を可能にする。

このモデルでは、RegistryEntry はゼロかそれ以上の Slot を持つことができる。スロットは、名称、slotType、値のコレクションで構成されている。スロットの名称は、RegistryEntry インスタンス内でローカルに一意である。同様に、Slot の値は、スロットインスタンス内でローカルに一意である。Slot はその値がコレクションとなることもできる拡張可能な属性を示すため、Slot は単一の値ではなく値のコレクションを持つことができる。slotType 属性は、スロットに対して自由に型またはカテゴリを指定することもできる。

Slot メソッドの概要	
String	getName () この RegistryObject の名前を取得する。name と名付けられた属性にマッピングする。
void	setName (String name) RegistryObject の名前を設定する。Slot 名は、RegistryEntry インスタンス内でローカルに一意である。
String	getSlotType () スロットの slotType またはカテゴリを取得する。slotType と名付けられた属性にマッピングする。
void	setSlotType (String slotType)

	このスロットに対して slotType またはカテゴリを設定する。
Collection	getValues () オブジェクトに対して値のコレクションを取得する。各値のタイプは String である。values と名付けられた属性にマッピングする。
void	setValues (Collection values) RegistryObject に対して値のコレクションを設定する。

7.5 インタフェース ExtrinsicObject

すべての上位インタフェース:
RegistryEntry, RegistryObject, Versionable

ExtrinsicObjects は、レジストリに型が知らされていないために補足的な属性によって説明しなければならない型（たとえば、mime 型）を持つ、提出されたコンテンツを説明するメタデータを提供する。

ExtrinsicObject によって説明されるコンテンツの例には、コラボレーションプロトコルプロファイル (CPP)、ビジネスプロセスの説明、スキーマが含まれている。

ExtrinsicObject メソッドの概要	
String	getContentURI () ExtrinsicObject によってカタログ化されるコンテンツを指す URI を取得する。レジストリは、この URI が解決可能であることを保証しなければならない。contentURI と名付けられた属性にマッピングする。
String	getMimeType () ExtrinsicObject によってカタログ化されるコンテンツに関連付けられた mime 型を取得する。mimeType と名付けられた属性にマッピングする。
boolean	isOpaque () ExtrinsicObject によってカタログ化されたコンテンツがレジストリに曖昧（レジストリによって解読できない）かどうかを決定する。場合により、登録組織は、暗号化されてレジストリによって解読さえできないコンテンツを提出することができる。opaque と名付けられた属性にマッピングする。
void	setContentURI (String uri) ExtrinsicObject によってカタログ化されたコンテンツに URI を設定する。
void	setMimeType (String mimeType) ExtrinsicObject によってカタログ化されたコンテンツに関連付けられた mime 型を設定する。
void	setOpaque (boolean isOpaque) ExtrinsicObject によってカタログ化されたコンテンツがレジストリに曖昧（レジストリによって解読できない）かどうかを設定する。

このインタフェースの基本インタフェースから継承されるメソッドは、表示されていないことに注意が必要である。

7.6 インタフェース IntrinsicObject

すべての上位インタフェース:
RegistryEntry, RegistryObject, Versionable

すべての既知の下位インタフェース:
Association, Classification, ClassificationNode, ExternalLink, Organization, Package, ExternalIdentifier

IntrinsicObject は、型がレジストリに知られていて、ebXML レジストリ仕様によって定義されている、提出されたコンテンツをカタログ化する派生クラスに対する、共通の基本クラスとなる。

現在のところ、このインタフェースはどのような属性もメソッドも定義していない。このインタフェースの基本インタフェースから継承されたメソッドは表示されていない。

7.7 インタフェース Package

すべての上位インタフェース:
IntrinsicObject, RegistryEntry, RegistryObject, Versionable

論理的に関連付けられた RegistryEntry は Package にグループ化することができる。これは、Registry Services は、将来オブジェクトのパッケージ全体に対する操作を許可することを想定している。

Package メソッドの概要	
Collection	getMemberObject s() Package のメンバーである RegistryEntries のコレクションを取得する。memberObjects と名付けられた属性にマッピングする。

7.8 インタフェース ExternalIdentifier

すべての上位インタフェース:
IntrinsicObject, RegistryEntry, RegistryObject, Versionable

ExternalIdentifier インスタンスは、DUNS 番号、社会保障番号、または組織の別名といった補足的な識別子情報を RegistryEntry に提供する。RegistryObject から継承された属性 name は、識別スキーム（社会保障番号など）を含むために使用され、属性 value は、実際の情報を含む。各 RegistryEntry は、ゼロまたはそれ以上の関連を ExternalIdentifier と持つことができる。

以下も参照:

ExternalIdentifier メソッドの概要	
String	getValue () ExternalIdentifier の値を取得する。value と名付けられた属性にマッピングする。
Void	setValue (String value) ExternalIdentifier の値を設定する。

このインタフェースの基本インタフェースから継承されたメソッドは、表示されていないことに注意が必要である。

7.9 インタフェース ExternalLink

すべての上位インタフェース:
IntrinsicObject, RegistryEntry, RegistryObject, Versionable

ExternalLinks は、レジストリ内のコンテンツと、レジストリの外側にあるコンテンツを関連付けるために、URI を使用する。たとえば、DTD を提出する組織は、その DTD と組織のホーム・ページを関連付けるために ExternalLink を使用することができる。

ExternalLink メソッドの概要	
Collection	getLinkedObjects () 外部リンクを使用する RegistryObject のコレクションを取得する。linkedObjects と名付けられた属性にマッピングする。
URI	getExternalURI () 外部のコンテンツに URI を取得する。externalURI と名付けられた属性にマッピングする。
void	setExternalURI (URI uri) 外部のコンテンツに URI を設定する。

このインタフェースの基本インタフェースから継承されたメソッドは、表示されていない。

8 レジストリ監査追跡

本節では、レジストリの監査追跡機能をサポートする情報モデル要素について説明する。この章のいくつかのクラスは、関連する属性のモデルを作成するためにラッパーとして使用されるエンティティクラスである。こうしたエンティティクラスは、関連付けられた動作を持っていない。これらは、C プログラミング言語の"struct"構造に似ている。

RegistryEntry の getAuditTrail()メソッドは、AuditableEvents の命じられた一覧を返す。これらの AuditableEvents は、RegistryEntry に対する監査追跡を構成する。AuditableEvents には、イベントに対するタイムスタンプが含まれている。各 AuditableEvent は、結果として特定のユーザを識別する User インターフェースを参照している。各 User インターフェースは、Organization インスタンスと関連付けられており、その組織は一般的に 登録組織である。

8.1 インタフェース AuditableEvent

すべての上位インタフェース:
RegistryObject

AuditableEvent インスタンスは、RegistryEntry の状態の変化に影響を与えるイベントの長期的な記録を提供する。RegistryEntry は、その RegistryObject に対して完全な監査追跡を提供する AuditableEvent インスタンスの一覧と関係している。

AuditableEvents は、一般的に、クライアントが開始した依頼の結果である。AuditableEvent インスタンスは、こうしたイベントを記録するためにレジストリサービスによって生成される。

こうしたイベントは、RegistryEntry のライフサイクルの変化に度々影響を受ける。たとえば、クライアントの依頼は、RegistryEntry を Create（作成）、Update（更新）、Deprecate（却下）、Delete（削除）することができる。どのような AuditableEvent も、RegistryEntry の状態が変化する依頼に対して作成される。特に、読み取り専用依頼は、AuditableEvent を生成しない。どのような AuditableEvent も、分類されたり、Package に割り当てられたり、他の RegistryObject と関連付けられる場合は、RegistryEntry に対して生成されない。

8.1.1 事前定義された監査可能イベントの型

以下の表は、事前定義された監査可能イベント型の一覧である。これらの事前定義されたイベント型は、Classification スキームとして定義されている。スキームを簡単に拡張することができるが、レジストリは以下の一覧にあるイベント型をサポートしなければならない。

名前	説明
Created	RegistryEntry を生成したイベント
Deleted	RegistryEntry を削除したイベント
Deprecated	RegistryEntry を却下したイベント
Updated	RegistryEntry を更新したイベント
Versioned	RegistryEntry にバージョン番号を付けたイベント

AuditableEvent メソッドの概要	
User	getUser () イベントを生成した依頼を送った User を取得する。user と名付けられた属性にマッピングする。
String	getEventType () 8.1.1 節で定義しているように、イベント型の名前属性によって定義されたようなこのイベント型。eventType と名付けられた属性にマッピングする。
RegistryEntry	getRegistryEntry () AuditableEvent と関連付けられた RegistryEntry を取得する。registryEntry と名付けられた属性にマッピングする。
Timestamp	getTimestamp () イベントが発生したときに Timestamp を取得する。timestamp と名付けられた属性にマッピングする。

このインタフェースの基本インタフェースから継承されたメソッドは、表示されない。

8.2 インタフェース User

すべての上位インタフェース:
RegistryObject

User インスタンスは、AuditableEvent を生成した依頼を送信した依頼者のアイデンティティの経路を維持するために、AuditableEvent を使用する。

User メソッドの概要	
Organization	getOrganization () 登録組織の情報を取得する。organization と名付けられた属性にマッピングする。
PostalAddress	getAddress ()

	このユーザに対して郵便アドレスを取得する。address と名付けられた属性にマッピングする。
String	getEmail () ユーザの電子メールアドレスを取得する。email と名付けられた属性にマッピングする。
TelephoneNumber	getFax () ユーザの FAX 番号を取得する。fax と名付けられた属性にマッピングする。
TelephoneNumber	getMobilePhone () ユーザの携帯電話番号を取得する。mobilePhone と名付けられた属性にマッピングする。
PersonName	getPersonName () 担当者の名前を取得する。personName と名付けられた属性にマッピングする。
TelephoneNumber	getPager () ユーザのポケットベル電話番号を取得する。pager と名付けられた属性にマッピングする。
TelephoneNumber	getTelephone () ユーザのデフォルトの固定電話番号を取得する。telephone と名付けられた属性にマッピングする。
URL	getUrl () コンテンツの Web ページへの URL を取得する。url と名付けられた属性にマッピングする。

8.3 インタフェース Organization

すべての上位インタフェース:
IntrinsicObject, RegistryEntry, RegistryObject, Versionable

Organization インスタンスは、登録組織などの組織に関する情報を提供する。各 Organization インスタンスは、親 Organization を参照できる。さらに、組織内の第 1 連絡先を定義するコンタクト属性を持つことができる。Organization は、アドレス属性も持つことができる。

Organization メソッドの概要	
PostalAddress	getAddress () 組織の PostalAddress を取得する。address と名付けられた属性にマッピングする。
User	getPrimaryContact () その組織の第 1 連絡先を取得する。第 1 連絡先は、User オブジェクトへの参照である。primaryContact と名付けられた属性にマッピングする。
TelephoneNumber	getFax () 組織の FAX 番号を取得する。fax と名付けられた属性にマッピングする。
Organization	getParent () 組織の親 Organization を取得する。parent と名付けられた属性にマッピングする。
TelephoneNumber	getTelephone () 組織の代表電話番号を取得する。telephone と名付けられた

	属性にマッピングする。

このインタフェースの基本インタフェースから継承されたメソッドは、表示されていない。

8.4 クラス PostalAddress

PostalAddress は、再利用可能な単一のエンティティで、郵便アドレスの属性を定義する。

項目の概要	
String	city 都市
String	country 国
String	postalCode 郵便番号
String	state 州、地方
String	street 番地

8.5 クラス TelephoneNumber

電話番号の属性を定義する、再利用可能な単一のエンティティクラス

項目の概要	
String	areaCode 地域番号
String	countryCode 国番号
String	extension 内線番号（内線がある場合）
String	number 国または地域番号に含まれていない電話番号接尾部
String	url この番号に電子的にダイアルできる URL

8.6 クラス PersonName

従業員に対する単一のエンティティクラス

項目の概要	
String	firstName この従業員の名前
String	lastName この従業員の苗字 (姓)
String	middleName この従業員のミドルネーム

9 RegistryEntry の名前付け

RegistryEntry は、Registry 内で一意にできる、または一意にしてはならない名称を持つ。

さらに、RegistryEntry は、特定の分類スキームのコンテキスト内だけで有効な、コンテキストに繊細な代替名を任意の数だけ持つこともできる。コンテキストに関係する代替名は、レジストリ情報モデルの今後のバージョンで扱われる。

10 RegistryEntry の関連

RegistryEntry は、ゼロまたはそれ以上の RegistryObject と関連付けることができる。情報モデルは、Association クラスを定義する。Association クラスのインスタンスは、RegistryEntry と別の RegistryObject 間の関連を示す。こうした関連の例は、新しいコラボレーションプロトコルプロファイル (CPP) と古いコラボレーションプロトコルプロファイルをカタログ化する ExtrinsicObjects の間である。ここでは、図 3 に示したように、新しい CPP はより古い CPP に取り替えられる。

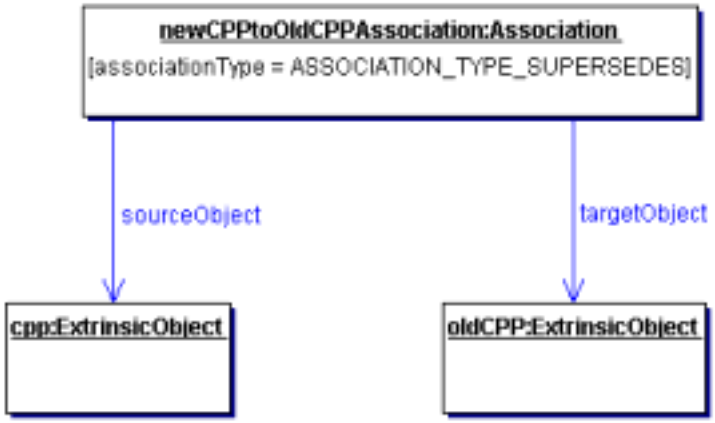


図 3: RegistryEntry 関連の例

10.1 インタフェース Association

すべての上位インタフェース:
IntrinsicObject, RegistryEntry, RegistryObject, Versionable

Association インスタンスは、情報モデル内の RegistryObject 間で多対多関連を定義するために使用される。

Association クラスのインスタンスは、2 つの RegistryObject 間の関連を示す。

Association メソッドの概要	
String	getAssociationType () Association の関連型を取得する。これは、10.1.1 で定義するように、関連・タイプの名前属性でなければならない。associationType と名付けられた属性にマッピングする。
Object	getSourceObject () Association のソースである RegistryObject にマッピングする。
String	getSourceRole () Association のソース RegistryObject によって実行される役割

	の名前を取得する。sourceRole と名付けられた属性にマッピングする。
Object	getTargetObject () Association のターゲットである RegistryObject を取得する。targetObject と名付けられた属性にマッピングする。
String	getTargetRole () Association のターゲット RegistryObject によって実行される役割の名前を取得する。targetRole と名付けられた属性にマッピングする。
boolean	isBidirectional () Association が双方向であるかどうかを決定する。bidirectional と名付けられた属性にマッピングする。
void	setBidirectional (boolean bidirectional) Association が双方向であるかどうかを設定する。
void	setSourceRole (String sourceRole) Association のソース RegistryObject によって実行される役割の名前を設定する。
void	setTargetRole (String targetRole) Association のターゲット RegistryObject によって実行される役割の名前を設定する。

10.1.1 事前定義された関連型

以下の表は、事前定義された関連型の一覧である。これらの事前定義された関連型は、Classification スキームとして提示されている。このスキームは簡単に拡張できるが、レジストリは以下の表の関連型をサポートしなければならない。

名前	説明
RelatedTo	ソース RegistryObject がターゲット RegistryObject と関連することを定義する。
HasMember	ソース Package オブジェクトがターゲット RegistryEntry オブジェクトをメンバーとして持つことを定義する。 RegistryEntries パッケージングするときに使用するために予約済み
ExternallyLinks	ソース ExternalLink オブジェクトがターゲット RegistryEntry オブジェクトに外部でリンクすることを定義する。 ExternalLinks を RegistryEntries に関連付けるときに使用するために予約済み
ExternallyIdentifies	ソース ExternalIdentifier オブジェクトがターゲット RegistryEntry オブジェクトを識別することを定義する。 ExternalIdentifiers を RegistryEntries に関連付けるときに使用するために予約済み
ContainedBy	ソース RegistryObject がターゲット RegistryObject によって含まれることを定義する。
Contains	ソース RegistryObject がターゲット RegistryObject を含むことを定義する。
Extends	ソース RegistryObject がターゲット RegistryObject から継承するか、ターゲット・オブジェクトを特化することを定義する。

Implements	ソース RegistryObject が、ターゲット RegistryObject によって定義された機能を実行することを定義する。
InstanceOf	ソース RegistryObject がターゲット RegistryObject のインスタンスであることを定義する。
SupercededBy	ソース RegistryObject がターゲット RegistryObject によって代替されることを定義する。
Supercedes	ソース RegistryObject がターゲット RegistryObject を代替することを定義する。
UsedBy	ソース RegistryObject がある方法でターゲット RegistryObject によって使用されることを定義する。
Uses	ソース RegistryObject がある方法でターゲット RegistryObject を使用することを定義する
ReplacedBy	ソース RegistryObject がある方法によってターゲット RegistryObject よって置き換えられることを定義する。
Replaces	ソース RegistryObject がある方法によってターゲット RegistryObject を置き換えることを定義する。

[注意] Extends や Implements といった関連型については、その関連が RegistyObject 間にあっても、その型により特定される実際の関連は registryObject により指定されるリポジトリ項目間にある。

11 RegistryEntry の分類

本節では、情報モデルがサポートする RegistryEntry の分類方法について説明する。これは、OASIS 分類モデル[OAS]の簡易バージョンである。

RegistryEntry は、いくつかの方法で分類することができる。たとえば、同じコラボレーションプロトコルプロファイル (CPP) に対する RegistryEntry は、産業、販売製品、地理的場所によって分類することができる。

一般的な分類スキーマは、分類ツリーとして見ることができる。図 4 に示した例では、コラボレーションプロトコルプロファイルを示す RegistryEntry は、影付きのボックスとして表示されている。各コラボレーションプロトコルプロファイルは、自動車メーカーを示している。各コラボレーションプロトコルプロファイルは、Industry と名付けられたルート ClassificationNode の下の、Automotive と名付けられた ClassificationNode によって分類されている。さらに、US Automobile メーカーは、Geography ClassificationNode の下の US ClassificationNode によって分類されている。同様に、欧州自動車メーカーは、Geography ClassificationNode の下の Europe ClassificationNode によって分類されている。

この例は、RegistryEntry が複数の分類スキームによって分類できる方法を示している。分類スキームは、分類ツリーのルートである ClassificationNode によって定義される（たとえば、Industry や Geography）。

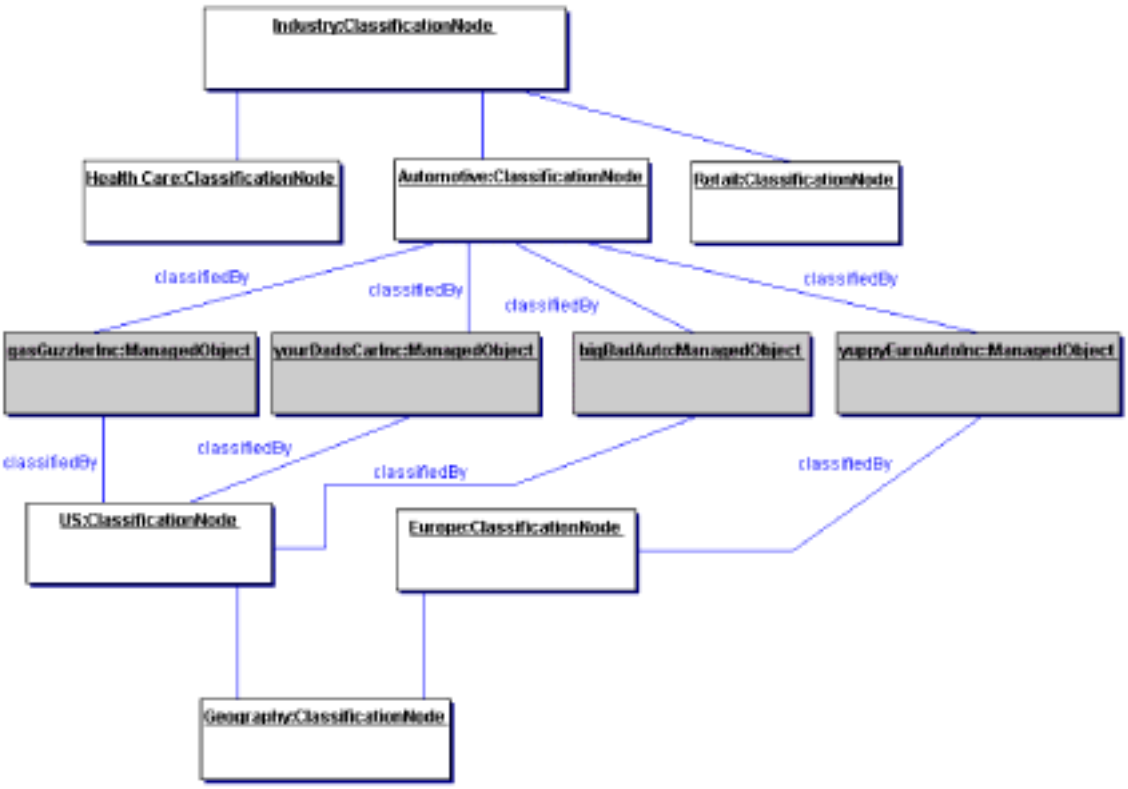


図 4: 分類ツリーを示した例

[注意] 影の付いたノード (gasGuzzlerInc、yourDadsCarInc など) は、分類ツリーの一部でないことを指摘することは重要である。分類ツリーのリーフノードは、Health Care、Automotive、Retail、US、Europe である。影の付いたノードは、この図に表示されていない Classification インスタンスを経由して分類ツリーと関連付けられている。

シングルレベルと同様にマルチレベルの分類がサポートできる一般分類スキームをサポートするために、情報モデルは図 5 で示したクラスと関連を定義している。

図 5: 情報モデル分類ビュー

Classification は、Association の特化された形である。図 6 は、自らが属する Industory を表す ClassificationNode によって分類されるコラボレーションプロトコルプロファイル (CPP) オブジェクトに対する ExtrinsicObject インスタンスの例である。

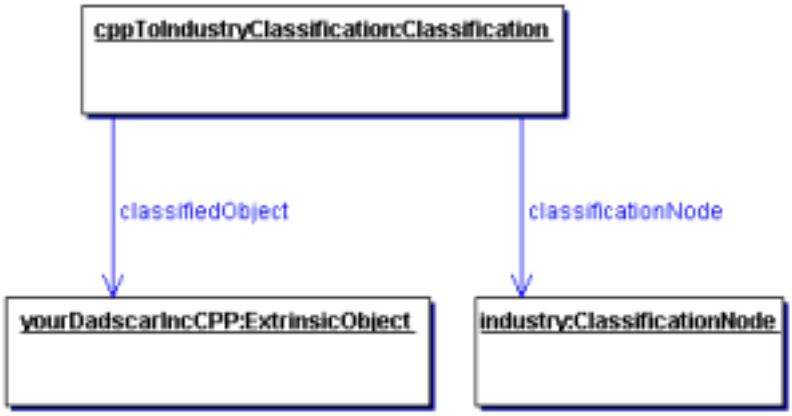


図 6: 分類インスタンス図

11.1 インタフェース ClassificationNode

すべての上位インタフェース:
IntrinsicObject, RegistryEntry, RegistryObject, Versionable

ClassificationNode インスタンスは、ツリー内の各ノードが ClassificationNode であるツリー構造を定義するために使用される。ClassificationNode といっしょに構成されたこうした分類ツリーは、分類スキーマまたはオントロジーを定義するために使用される。

以下も参照:
Classification

ClassificationNode メソッドの概要	
Collection	getClassifiedObjects () ClassificationNode によって分類された RegistryObjects のコレクションを取得する。classifiedObjects と名付けられた属性にマッピングする。
ClassificationNode	getParent () ClassificationNode に対する親 ClassificationNode を取得する。parent と名付けられた属性にマッピングする。
String	getPath () ClassificationNode のルート祖先からパスを取得する。パスは[XPATH]表記構文に従う（たとえば、 “ Geography/Asia/Japan ”)。path と名付けられた属性にマッピングする。
void	setParent (ClassificationNode parent) ClassificationNode に対して親 ClassificationNode を設定する。
String	getCode () ClassificationNode に対してコードを取得する。詳細は 11.4 節を参照。code と名付けられた属性にマッピングする。
void	setCode (String code) ClassificationNode に対してコードを設定する。詳細は 11.4 節を参照。

このインタフェースの基本インタフェースから継承されたメソッドは、表示されていないことに注意が必要である。

図 4 では、ClassificationNode の複数のインスタンスが定義されている（明るい色のすべてのボックス）。ClassificationNode は、その親に対してゼロまたは 1 つの ClassificationNode を、そのすぐ下の子に対してはゼロまたはそれ以上の ClassificationNode を持っている。ClassificationNode に親がない場合は、その ClassificationNode が分類ツリーのルートである。全体的な分類ツリーは、単一の情報モデル要素 ClassificationNode によって再帰的に定義されることに注意が必要である。

11.2 インタフェース Classification

すべての上位インタフェース:
IntrinsicObject, RegistryEntry, RegistryObject, Versionable

Classification インスタンスは、その RegistryEntry インスタンスを分類スキーム内で ClassificationNode インスタンスと関連付けることにより、リポジトリ項目を分類するために使用される。

図 4 では、Classification インスタンスは明示的に表示されていないが、RegistryEntries (影の付いたリーフノード) と関連付けられた ClassificationNode の間の関連として示されている。

Classification メソッドの概要	
RegistryObject	getClassifiedObject () Classification によって分類される RegistryObject を取得する。classifiedObject と名付けられた属性にマッピングする。
RegistryObject	getClassificationNode () Classification で RegistryObject を分類する ClassificationNode を取得する。classificationNode と名付けられた属性にマッピングする。

このインタフェースの基本インタフェースから継承されたメソッドは、表示されていないことに注意が必要である。

11.2.1 コンテキストに繊細な分類

図 7 に示したケースを例に取ると、ACME Inc.に対するコラボレーションプロトコルプロファイルは、Geography 分類スキームの下の Japan ClassificationNode によって分類されている。この分類に対するコンテキストがないと、その意味は曖昧になる。コンテキストは ACME が日本にあることを意味にしているのか、ACME が日本に製品を出荷することを意味しているのか明確でなく、別の意味がある場合もある。この曖昧性を解決するには、Classification に対して不明なコンテキストを提供する別の ClassificationNode（この例では、isLocatedIn）と Classification を自由に関連付けることができる。MyParcelService に対する別のコラボレーションプロトコルプロファイルは、Japan ClassificationNode によって分類することができる。ここでは、ACME Inc.によって使用されたコンテキストとは異なるコンテキストを指定するために、この Classification を異なる ClassificationNode（たとえば、shipsTo）と関連付けることができる。

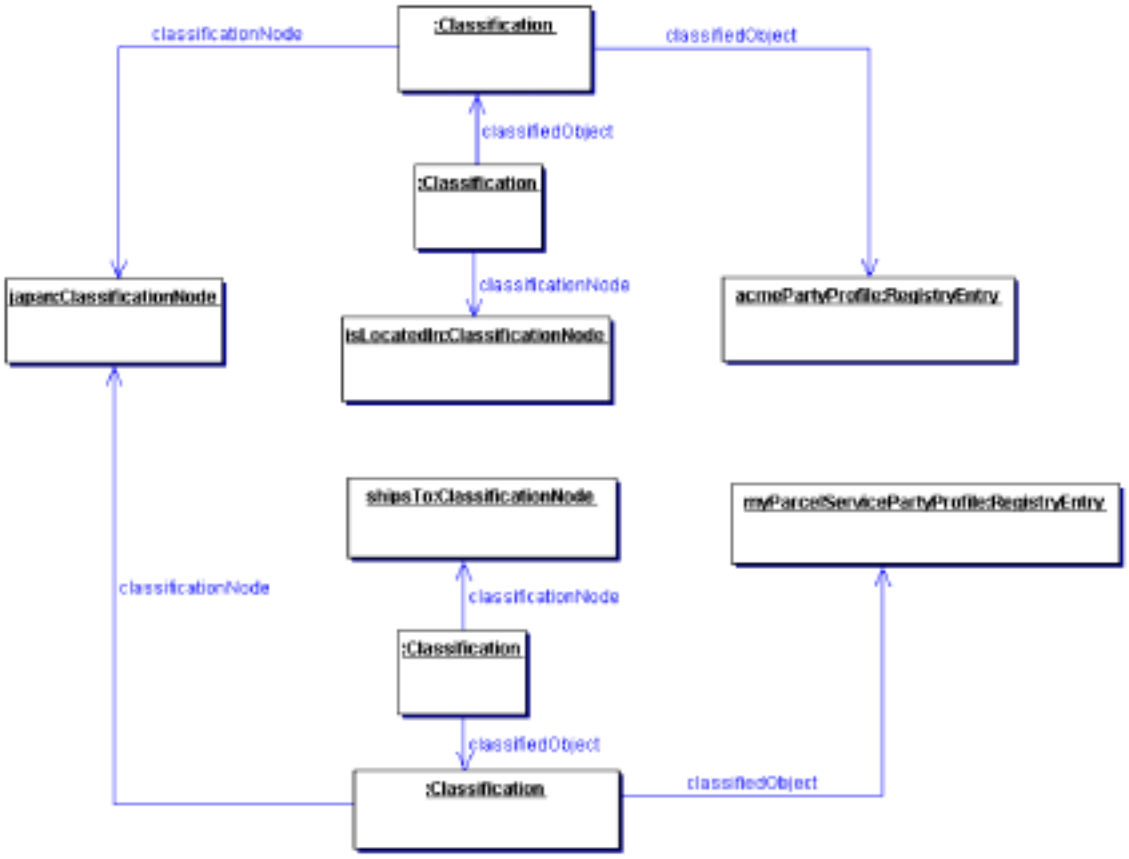


図 7: コンテキストに影響する分類

したがって、複数のコンテキスト内の Classification の可能性をサポートするために、Classification は、不足しているコンテキストを提供する ClassificationNodes に最初の Classification を結びつける任意の数の Classifications によって自分自身が分類される。

要約すると、情報モデル内の分類スキームに対して一般化されたサポートにより、以下が可能になる。

o 分類ツリーで ClassificationNode と関連付ける Classification を定義することにより、RegistryEntry を分類することができる。

o 複数の ClassificationNodes と関連付ける複数の分類を持つことによって、RegistryEntry を複数のファセットに応じて分類することができる。

o RegistryEntry に対して定義された分類を、RegistryEntry が分類されているコンテキストによって限定することができる。

11.3 分類スキームの例

以下の表は、情報モデルによって可能になった分類スキームの例をいくつか示している。これらのスキームは、ebXML ビジネスプロセスプロジェクトチームとコア構成要素プロジェクトチームによって識別されたコンテキスト関連概念のサブセットに基づいている。この一覧は、例証のためのもので、規定を示すものではない。

分類スキーム（コンテキスト）	使用例
Industry	自動車産業のすべての取引当事者を検索
Process	Process を実行する ServiceInterface を検索
Product	製品を販売するビジネスを検索
Locale	日本にある Supplier を検索
Temporal	24 時間以内に出荷できる Supplier を検索
Role	“ Seller ”の役割を持つすべての Suppliers を検索

表 1: サンプル分類スキーム

11.4 標準分類法のサポート

オントロジーまたはコーディングスキームとも呼ばれる標準分類法はさまざまな産業に存在し、構造的にコード化された語彙を提供している。ebXML レジストリは、特定の分類法に対するサポートを定義していない。その代わり、さまざまな分類法によって定義されたコードに RegistryEntries をリンクするための一般的な機能を提供している。

情報モデルは、RegistryEntries の分類に対して標準分類法を使用するために、2 つの方法を提供している。

11.4.1 完全機能の分類法に基づく分類

情報モデルは、Classification および ClassificationNode インスタンスに基づき、完全機能の分類法に基づく分類方法を提供する。この方法は、標準分類法が ClassificationNode インスタンスから構成される分類ツリーとして Registry にインポートされることを必要としている。この仕様は、標準分類法を ebXML Registry 分類ツリーに変換するために必要な変換ツールとして規定しているのではない。しかし、変換は以下を保証する必要がある。

1. ルート ClassificationNode の名前属性は、標準分類法の name である（たとえば、NAICS, ICD-9, SNOMED）

2. 標準分類法のすべてのコードは、ClassificationNode の code 属性に保存される。

3. 標準分類法が意図する構造は、ClassificationNode ツリーに保存される。したがって、多形ブラウズ（polymorphic browse）とドリルダウン探索が可能になる。 つまり、Asia によって分類されたエントリを検索すると、Asia のインスタンス（たとえば、Japan と Korea）によって分類されたエントリが見つかる。

11.4.2 簡易分類法に基づいた分類

情報モデルは、標準分類法によって定義されたコードごとに RegistryEntry インスタンスを分類するための簡易分類方法も備えている。この方法では、提出者は本来の分類スキームとして完全な分類法をインポートすることを希望しない。

この分類方法では、提出者は、提出されたリポジトリ項目に対する RegistryEntry に、Slot に関連した 1 つまたは複数の分類法を追加する。各 Slot の名称は、標準化された分類法を識別し、Slot の値は、指定された分類法のコードとなる。こうしたスロット関連分類法は、Classification の slotType と一緒に定義されなければならない。

たとえば、RegistryEntry が名前 「NAICS」という Slot と、「Classification」の slotType、値「51113」を持っていると、RegistryEntry は、NAICS 分類法の「Book Publishers」に対するコードによって分類される。この例では、NAICS 分類法全体をインポートする必要もないし、ClassificationNode または Classification のインスタンスを生成する必要もない。

以下のポイントは、この簡易分類方法で重要である。

・「Classification」の名前と値の評価は、SO の責任であり ebXM レジストリ自身の責任ではない。

・探索はスロット名とスロット値の完全一致に基づき、完全機能の分類方法で利用できる柔軟な「閲覧およびドリルダウン探索」に基づくのではない。

12 情報モデル: セキュリティビュー

本節では、レジストリのセキュリティ機能に関する情報モデルの側面を説明する。

図 8 は、セキュリティの観点から Registry のオブジェクトを描いている。ここでは、オブジェクトの関連は UML クラス図として示されている。後節で説明するクラス属性またはクラスメソッドは示されていない。これは例証のためのもので、規定を示すものではない。

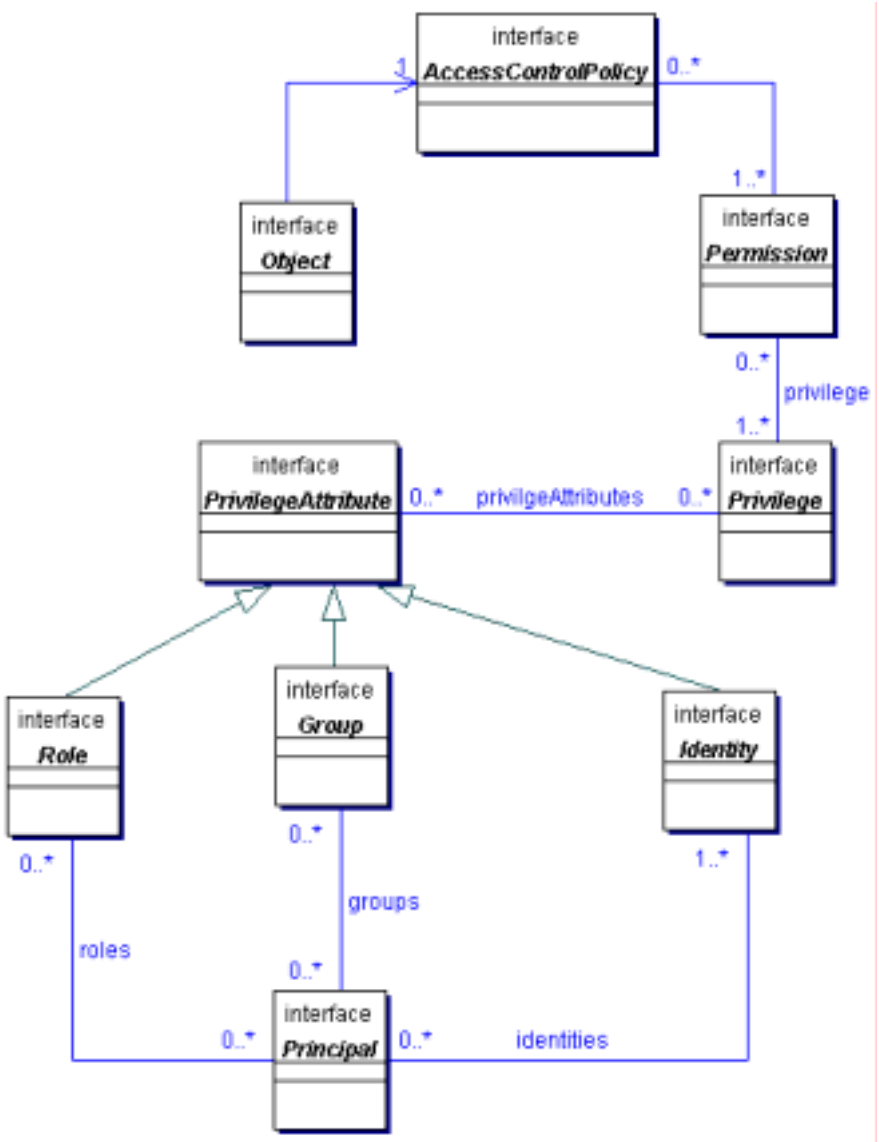


図 8: 情報モデル: セキュリティ・ビュー

12.1 インタフェース AccessControlPolicy

それぞれの RegistryObject は、確実に 1 つの AccessControlPolicy と関連付けられている。AccessControlPolicy は、その RegistryObject に関して実行されるアクセスまたはメソッドを管理するポリシールールを定義する。

AccessControlPolicy メソッドの概要	
Collection	getPermissions () AccessControlPolicy に対して定義された Permissions を取得する。permissions と名付けられた属性にマッピングする。

12.2 インタフェース Permission

Permission オブジェクトは、レジストリの RegistryObject への権限およびアクセス制御に対して使用される。RegistryObject に対する Permissions は、AccessControlPolicy オブジェクト

で定義されている。

Permission オブジェクトは、依頼している Principal に Permission で定義された任意の Privileges があれば、RegistryObject のメソッドにアクセスを許可する。

以下も参照:
Privilege, AccessControlPolicy

Permission メソッドの概要	
String	getMethodName () Permission によって指定された Privilege のある Principal にアクセスできるメソッド名を取得する。methodName と名付けられた属性にマッピングする。
Collection	getPrivileges () Permission と関連付けられた Privileges を取得する。privileges と名付けられた属性にマッピングする。

12.3 インタフェース Privilege

Privilege オブジェクトは、ゼロまたはそれ以上の PrivilegeAttributes を含む。PrivilegeAttribute は、Group（グループ）、Role（役割）、または Identity（アイデンティティ）になることができる。

依頼する Principal は、保護された RegistryObject のメソッドにアクセスするために、Privilege の中で指定された PrivilegeAttributes のすべてを持たなければならない。RegistryObject の AccessControlPolicy の中で定義された Permissions は、特定のメソッドにアクセスを許可できる Privileges を定義する。

このメカニズムは、Roles（役割）、Identities（アイデンティティ）、Groups（グループ）の任意の組み合わせに基づくオブジェクトアクセス制御ポリシーを柔軟に設定することができる。

以下も参照:
PrivilegeAttribute, Permission

Privilege メソッドの概要	
Collection	getPrivilegeAttributes () Privilege に関連付けられた PrivilegeAttributes を取得する。privilegeAttributes と名付けられた属性にマッピングする。

12.4 インタフェース PrivilegeAttribute

すべての既知の上位インタフェース:
Group, Identity, Role

PrivilegeAttribute は、Principal に特定のアクセス制御特権を与えるために使用されるセキュリティ属性のすべての型に対する共通の基本クラスである。Principal は、PrivilegeAttributes の複数の異なる型を持つこともできる。PrivilegeAttributes の特定の組み合わせは、Privilege オブジェクトとして定義することができる。

以下も参照:

Principal, Privilege

12.5 インタフェース Role

すべての上位インタフェース:
PrivilegeAttribute

セキュリティに関する Role PrivilegeAttribute である。たとえば、病院では Nurse（看護婦)、Doctor（医師)、Administrator（管理者）といった Role を使用することができる。Roles は Principals に Privileges を与えるために使用される。たとえば、Doctor の役割は 処方箋を書くことができるが、Nurse の役割はそれができない。

12.6 インタフェース Group

すべての上位インタフェース:
PrivilegeAttribute

セキュリティに関する Group PrivilegeAttribute である。Group は、異なる役割を持つことができるユーザの集合である。たとえば、病院は、特定の臨床試験に参加する Nurses と Doctors に対して定義された Group を持つことができる（たとえば、AspirinTrial グループ)。Groups は、Principals に Privileges を与えるために使用される。たとえば、AspirinTrial グループのメンバーは、Aspirin に対して処方箋を書くことができる（一般的には、Nurse 役割は処方箋を書くことができない)。

12.7 インタフェース Identity

すべての上位インタフェース:
PrivilegeAttribute

セキュリティに関する Identity PrivilegeAttribute である。これは、人、組織、ソフトウェアサービスを識別するために一般的に使用される。Identity 属性はデジタル証明書の形にすることができる。

12.8 インタフェース Principal

Principal は、人とソフトウェア・システムの両方を含めるために、セキュリティコミュニティーによって使用される、完全に汎用的な用語である。Principal オブジェクトは、PrivilegeAttributes のセットを持ったエンティティである。これらの PrivilegeAttributes は、少なくとも 1 つのアイデンティティと、随意の役割メンバーシップ、グループメンバーシップ、またはセキュリティ許可のセットを含んでいる。プリンシパルは、依頼者を認証し、Principal と関連付けられた PrivilegeAttributes に基づいて、依頼されたアクションを認可するために使用される。

以下も参照:
PrivilegeAttributes, Privilege, Permission

Principal メソッドの概要	
Collection	getGroups () Principal と関連付けられた Groups を取得する。groups と名付けられた属性にマッピングする。
Collection	getIdentities ()

	Principal と関連付けられた Identities を取得する。identities と名付けられた属性にマッピングする。
Collection	getRoles () Principal と関連付けられた Roles を取得する。roles と名付けられた属性にマッピングする。

13 関連文献

[ebGLOSS] ebXML 用語集
http://www.ebxml.org/documents/199909/terms_of_reference.htm
[ebTA] ebXML テクニカルアーキテクチャ仕様
http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.4.pdf
[OAS] OASIS 情報モデル
http://xsun.sdct.itl.nist.gov/regrep/OasisRegrepSpec.pdf
[ISO] ISO 11179 情報モデル
http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument
[BRA97] IETF (Internet Engineering Task Force). RFC 2119: Key words for use in RFCs to Indicate Requirement Levels
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2119.html
[ebRS] ebXML レジストリサービス仕様
http://www.ebxml.org/specdrafts/ebXML_RS_v1.0.pdf
[ebBPSS] ebXML Business Process Specification Schema
http://www.ebxml.org/specdrafts/Busv2-0.pdf
[ebCPP] ebXML コラボレーションプロトコルプロファイルおよびコラボレーションプロトコル合意書仕様
http://www.ebxml.org/specfrafts/
[UUID] DCE 128 bit Universal Unique Identifier
http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
http://www.opengroup.org/publications/catalog/c706.htmttp://www.w3.org/TR/REC-xml
[XPATH] XML Path Language (XPath) Version 1.0
http://www.w3.org/TR/xpath

14 免責

本書で表現したビューおよび仕様は、著者のビューおよび仕様であり、必ずしも雇用者のビューおよび仕様を示すものではない。著者およびその雇用者は、この設計の正確なまたは不正確な実装または使用から生じるどのような問題に対しても責任を負わないことを明言する。

15 連絡先

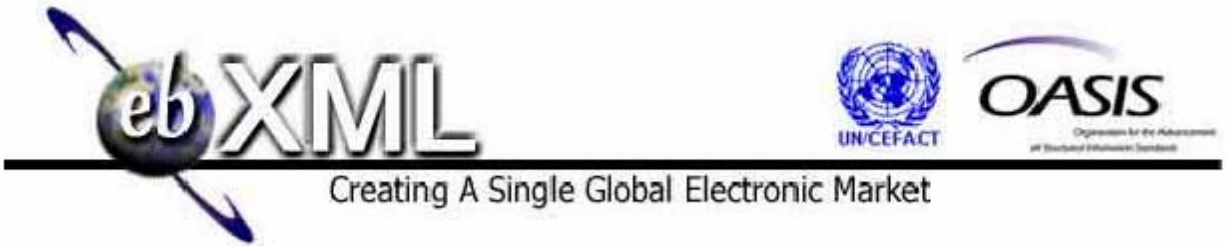
チーム・リーダー
名前: スコット・ニーマン
会社: ノースタン・コンサルティング
住所: 5101 Shady Oak Road
住所: Minnetonka, MN 55343
国: USA
電話番号: 952.352.5889
電子メール: Scott.Nieman@Norstan

副チーム・リーダー
名前: ユタカ・ヨシダ
会社: サン・マイクロシステムズ
住所: 901 San Antonio Road, MS UMPK17-102
住所: Palo Alto, CA 94303
国: USA
電話番号: 650.786.5488
電子メール: Yutaka.Yoshida@eng.sun.com

編集者
名前: Farrukh S. Najmi
会社: サン・マイクロシステムズ
住所: 1 Network Dr., MS BUR02-302
住所: Burlington, MA, 01803-0902
国: USA
電話番号: 781.442.0703
電子メール: najmi@east.sun.com

著作権について

Copyright Statement



ebXML レジストリサービス仕様 v1.0

ebXML レジストリプロジェクトチーム

2001 年 5 月 10 日

技術検証　ECOM XML/EDI 標準化専門委員会

1 本書の位置付け

本書は電子ビジネスコミュニティ向けの ebXML ドラフト標準を指定している。

本書は自由に配布できる。

文書形式はインターネット・ソサイアティーの標準 RFC に基づいている。

本バージョン:
http://www.ebxml.org/specs/ebRS.pdf

最新バージョン:
http://www.ebxml.org/specs/ebRS.pdf

2 ebXML の参加者

ebXML Registry Services v1.0 は、ebXML レジストリプロジェクトチームによって開発された。この仕様が承認されたときの ebXML レジストリプロジェクトチームのメンバは以下の通りである。

Lisa Carnahan, NIST
Joe Dalman, Tie
Philippe DeSmedt, Viquity
Sally Fuger, AIAG
Len Gallagher, NIST
Steve Hanna, Sun Microsystems
Scott Hinkelman, IBM
Michael Kass, NIST
Jong.L Kim, Innodigital
Kyu-Chul Lee, Chungnam National University
Sangwon Lim, Korea Institute for Electronic Commerce
Bob Miller, GXS
Kunio Mizoguchi, Electronic Commerce Promotion Council of Japan
Dale Moberg, Sterling Commerce
Ron Monzillo, Sun Microsystems
JP Morgenthal, eThink Systems, Inc.
Joel Munter, Intel
Farrukh Najmi, Sun Microsystems
Scott Nieman, Norstan Consulting
Frank Olken, Lawrence Berkeley National Laboratory
Michael Park, eSum Technologies
Bruce Peat, eProcess Solutions
Mike Rowley, Excelon Corporation
Waqar Sadiq, Vitria
Krishna Sankar, Cisco Systems Inc.
Kim Tae Soo, Government of Korea
Nikola Stojanovic, Encoda Systems, Inc.
David Webber, XML Global
Yutaka Yoshida, Sun Microsystems
Prasad Yendluri, webmethods
Peter Z. Zhoo, Knowledge For the new Millennium

目 次

図目次

表目次

3 はじめに

3.1 本書の総括

本書では、ebXML レジストリサービスのインタフェースのほか、相互作用プロトコル、メッセージ定義、および XML スキーマの定義を行う。

別の文書「ebXML レジストリ情報モデル [ebRIM]」では、レジストリに格納されるメタデータ型のほか、各種メタデータクラス間の関係の情報を提供する。

3.2 一般規約

本書では以下の規約を使用する。

o 概念を簡潔に説明する手段として UML 図を使用する。特定の実装や方法の要件を伝えるために UML 図を使用するのではない。

o 「リポジトリ項目」という用語は、保管および保護の目的でレジストリに登録されているオブジェクトを参照するために使用する（たとえば、XML 文書や DTD）。あらゆるリポジトリ項目は、RegistryEntry インスタンスで表記する。

o 「RegistryEntry」という用語は、リポジトリ項目のメタデータを提供するオブジェクトを参照するために使用する。

o 大文字のイタリック体の単語は、ebXML 用語解説[ebGLOSS]で定義されている。

キーワード MUST(〜しなければならない)、MUST NOT(〜してはならない)、REQUIRED(要求される)、SHALL(〜するものとする)、SHALL NOT(〜しないものとする)、SHOULD(〜すべきである)、SHOULD NOT(すべきでない)、RECOMMENDED(推奨される)、MAY(〜してよい)、および OPTIONAL(オプション)が本書で使用される場合、RFC 2119[Bra97]の規定に従って解釈するものとする。

3.3 対象読者

この仕様書の対象読者は、次に該当するソフトウェア開発者のコミュニティである。

o ebXML レジストリサービスの実装担当者

o ebXML レジストリクライアントの実装担当者

3.4 関連文書

以下の仕様書が背景情報と関連情報を読者に提供する。

a) ebXML レジストリ情報モデル [ebRIM]

b) ebXML メッセージ取扱サービス仕様[ebMS]

c) ebXML ビジネスプロセス仕様スキーマ[ebBPM]

d) ebXML コラボレーションプロトコルプロファイルおよびコラボレーションプロトコル合

意書[ebCPP]

4 設計目的

4.1 目標

仕様書の本バージョンの目標は以下の通りである。

o レジストリサービスの機能をソフトウェア開発者に伝える
o レジストリクライアントとレジストリのインタフェースを明示する
o より複雑な ebXML レジストリ要件に対する将来のサポートの基礎を提供する
o 他の ebXML 仕様との整合性を確保する

4.2 警告および前提条件

レジストリサービス仕様書は段階的に配布が可能な文書の最初のものである。本書の今後のバージョンには、将来開発が計画されている追加機能を含める予定である。

以下の事項を前提とする。

1. 相互運用性の要件により、ebXML メッセージ取扱サービス仕様は、ebXML レジストリと ebXML レジストリクライアントの間で使用しなければならない。その他のコミュニケーション手段を使用できないのではない。しかし、その他のコミュニケーション手段を使用する場合は相互運用性を想定できない。その他のコミュニケーション手段は、本仕様書では扱わない。

2. レジストリのコンテンツへのアクセスは、すべてレジストリサービスのために定義されるインタフェースを介して公開する。

3. レジストリは、リポジトリを利用してレジストリサービスが必要とする永続化情報を格納し、検索する。これは実装の細部であり、本仕様では詳しく規定しない。

5 システムの概要

5.1 ebXML レジストリの機能

ebXML レジストリは、ebXML 仕様書に基づいて取引当事者間のビジネスプロセスを統合するために、当事者間の情報の共有を可能とするサービスを提供する。共有される情報は、本書で定義する ebXML レジストリサービスがオブジェクトとしてリポジトリに保持・管理する。

5.2 ebXML レジストリの仕組み

本節では、レジストリクライアントがレジストリサービスを利用して企業対企業（B2B）取引を実行する仕組みを例示するユースケースの概要を説明する。これは例示的なものであり、規範ではない。

以下のシナリオは、レジストリクライアントとレジストリとの相互作用の観点からユース

ケースの概要をテキストで示している。これは想定されるユースケースをすべて網羅するものではない。このシナリオでは、例示を目的として、ロゼッタネット PIP3A4 Purchase Order ビジネスプロトコルを用いて B2B 取引を実行することを望む買い手と売り手を前提とする。買い手と売り手の両者ともサードパーティが提供する同一のレジストリサービスを用いているものと仮定する。アーキテクチャが他の可能性(たとえば、当事者がそれぞれ非公開レジストリを使用する場合)をサポートすることに注意が必要である。

5.2.1 スキーマ文書の申請

業界のコンソーシアムや標準団体などのサードパーティは、7.3 節で説明するレジストリのオブジェクトマネージャサービスを用いて、ロゼッタネット PIP3A4 Purchase Order ビジネスプロトコルが要求する必要なスキーマ文書をレジストリに申請する。

5.2.2 ビジネスプロセス文書の申請

業界のコンソーシアムや標準団体などのサードパーティは、7.3 節で説明するレジストリのオブジェクトマネージャサービスを用いて、ロゼッタネット PIP3A4 Purchase Order ビジネスプロトコルが要求する必要なビジネスプロセス文書をレジストリに申請する。

5.2.3 売り手のコラボレーションプロトコルプロファイルの申請

売り手は[ebCPP]が定義するコラボレーションプロトコルプロファイル、すなわち CPP をレジストリに公開する。CPP は売り手、売り手の役割、売り手が提供するサービス、アクセス方法に関する技術の詳細を記述する。 売り手はレジストリの柔軟な分類機能により、コラボレーションプロトコルプロファイルを分類する。

5.2.4 買い手が売り手を見つける

買い手はレジストリブラウザ GUI ツールを用いてレジストリ内で定義された分類スキームを使用するレジストリを閲覧し、適切な売り手を見つける。たとえば、買い手は自動車産業に属し、売り手の役割を果たし、ロゼッタネット PIP3A4 プロセスをサポートし、カーステレオを販売するすべての当事者を検索できる。

買い手は売り手の CPP を見つけ、売り手との間で取引を行うことを決定する。

5.2.5 CPA の確立

買い手は売り手の CPP と自分の CPP を入力とし、[ebCPP]が定義するコラボレーションプロトコル合意書、すなわち CPA を売り手に対して一方的に作成する。買い手は一方的な CPA を使って取引を提案する。提案された CPA を売り手が受け入れると、取引が確立する。

売り手が CPA を受け入れると、両当事者が[ebMS]の定義に従って B2B 取引の実行を開始する。

5.3 レジストリサービスが実装できる場所

レジストリサービスはいくつかの方法で実装できる。公開 Web サイトとしての実装、非公開 Web サイトとしての実装、ASP プロバイダまたは VPN プロバイダのホスティングによる実装を含む。

5.4 実装の適合性

実装が 5.4.1 節の条件を満たす場合、その実装は ebXML に適合するレジストリである。実装が 5.4.2 節の条件を満たす場合は、その実装は ebXML に適合するレジストリクライアントである。実装が 5.4.1 節および 5.4.2 節の条件を満たす場合は、その実装は ebXML に適合するレジストリであり、かつ ebXML に適合するレジストリクライアントである。実装は、ebXML に適合するレジストリか、ebXML に適合するレジストリクライアント、または、ebXML に適合するレジストリであってかつ ebXML に適合するレジストリクライアントでなければならない。

5.4.1 ebXML レジストリとしての適合性

実装が ebXML として本仕様書に適合するのは、以下の条件を満たす場合である。

1. ebXML レジストリ情報モデル[ebRIM]に適合する。

2. レジストリインタフェースとセキュリティモデルの構文と意味情報をサポートする。

3. 定義された ebXML レジストリ DTD（付録 A）をサポートする。

4. オプションで 8.3 節の SQL クエリサポートの構文と意味情報をサポートする。

5.4.2 ebXML レジストリクライアントとしての適合性

実装が本仕様書に適合するのは、ebXML レジストリクライアントとして以下の条件を満たす場合である。

1. ebXML CPA およびブートストラッププロセスをサポートする。

2. レジストリクライアントインタフェースの構文と意味情報をサポートする。

3. 定義された ebXML エラーメッセージ DTD をサポートする。

4. 定義された ebXML レジストリ DTD をサポートする。

6 レジストリのアーキテクチャ

ebXML レジストリのアーキテクチャは、ebXML レジストリおよび ebXML レジストリクライアントから構成される。レジストリクライアントのインタフェースは、レジストリにとってローカル、またはユーザにとってローカルとすることができる。図 1 ではレジストリのアーキテクチャがサポートする、レジストリおよびレジストリクライアントに関して可能なトポロジを 2 つ示す。

左側の図は、レジストリが共通 Web ブラウザによりユーザが使用できるレジストリにアクセスするための Web ベースの「シンクライアント」アプリケーションを提供するシナリオを示す。このシナリオでは、レジストリクライアントのインタフェースはインターネット側にあり、ユーザから見ればレジストリ側のローカルである。

右側の図は、ユーザが「ファットクライアント」レジストリブラウザを使ってレジストリにアクセスするシナリオを示す。このシナリオでは、レジストリクライアントのインタフ

ェースはレジストリブラウザツールにあり、ユーザから見ればレジストリ側のローカルである。このシナリオでは、レジストリクライアントのインタフェースがインターネット上でレジストリと通信する。

レジストリのアーキテクチャで可能となる第 3 のトポロジは、レジストリクライアントのインタフェースが購買ビジネスコンポーネントなどのサーバ側ビジネスコンポーネントにあるというトポロジである。このトポロジでは、直接のユーザインタフェースやユーザの介入は発生しない。その代わりに、購買ビジネスコンポーネントが、自動的にレジストリにアクセスし、その時のビジネスニーズに基づいて売り手やサービス提供者の候補を選択できる。

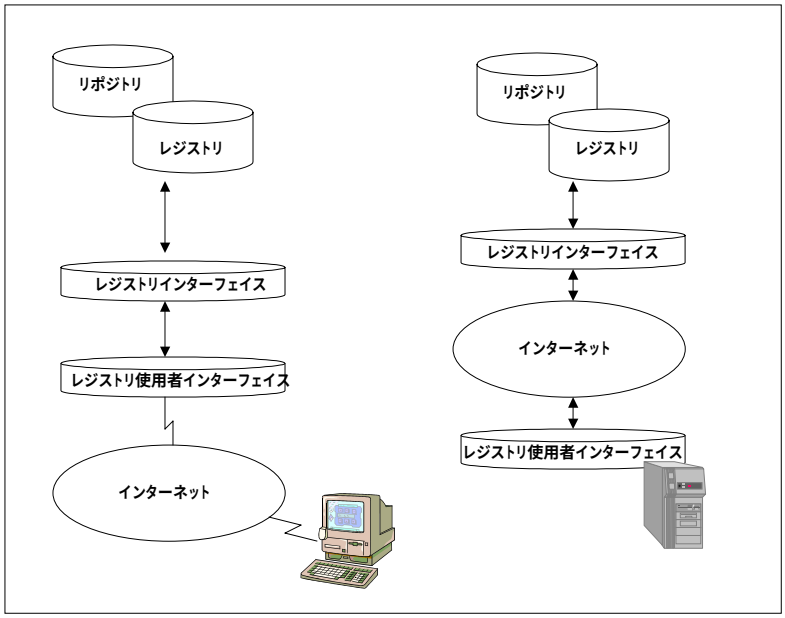


図 1: レジストリアーキテクチャは柔軟なトポロジをサポートする

クライアントは、ebXML アプリケーション同士が通信する場合と同じ方法で、ebXML メッセージ取扱サービスを使ってレジストリと通信する。

本仕様書の将来のバージョンでは、分散レジストリをサポートするために、レジストリアーキテクチャを明示的に拡張するための追加サービスを提供するかもしれない。しかし、仕様書の現在のバージョンでは、情報を共有するために互いに協力することから ebXML レジストリを除外しておらず、ebXML レジストリのオーナがその ebXML レジストリを他のレジストリシステム、カタログまたはディレクトリといっしょに登録することを妨げるものでもない。

例には以下のものがある。
・中央登録ポイントとして機能するレジストリの ebXML レジストリ
・連合（federation）でレジストリが互いに登録し合う、協調（cooperative）ebXML レジストリ
・ホワイトページやイエローページとして機能する他のレジストリシステムを伴う ebXML レジストリの登録。文書[ebXML-UDDI]に、UDDI として知られるホワイト／イエローページの検索システムで見つかる ebXML レジストリの一例を掲載している。

6.1 ebXML レジストリのプロファイルおよび合意書

ebXML CPP 仕様[ebCPP]は、2 当事者がそれぞれのビジネスプロセスに関する情報を共有する機構として、コラボレーションプロトコルプロファイル（CPP）とコラボレーションプロトコル合意書（CPA）を定義している。その仕様は、B2B 取引を行うために両当事者が CPA に合意済みであることを想定している。

本仕様書では、レジストリとレジストリクライアントの間で CPA を使用しなければいけないわけではない。しかし、レジストリが CPP を使用しない場合、CPP が提供するサービスと他の情報をレジストリクライアントが見つけられるように、レジストリは代替機構をレジストリクライアントに提供しなければならない。この代替機構は、単なる URL とすることができる。

クライアントとレジストリ間の CPA は、レジストリに固有の相互作用に対してレジストリとクライアントが互いに公開するインタフェースを記述しなければならない。これらのインタフェースは図 2 と以後の節で説明する。レジストリ CPP テンプレートとレジストリクライアント CPP テンプレートの定義は、本仕様書では扱わない。

6.2 クライアントからレジストリへの通信ブートストラップ

レジストリとレジストリクライアントとの間にはまだ CPA が確立されていないため、クライアントはレジストリに対する転送用通信アドレスを少なくとも 1 つ知っておかなければならない。この通信アドレスは一般的にはレジストリに対する URL である。ただし、通信アドレスが電子メールアドレスなど、他の型のアドレスの場合もある。

たとえば、レジストリが使用する通信が HTTP である場合、通信アドレスは URL である。この例では、クライアントはレジストリの公開 URL を使ってレジストリとの暗黙の CPA を作成する。クライアントがレジストリに要求を送信すると、クライアントは自ら URL を指定する。レジストリはクライアントの URL を用いて、レジストリ側でもクライアントとの間の暗黙の CPA を作成する。この時点で、レジストリ内でセッションが確立される。

クライアントのレジストリとのセッション中に、本仕様書で定義する相互作用プロトコルの要求に従ってメッセージを双方向に交換できる。

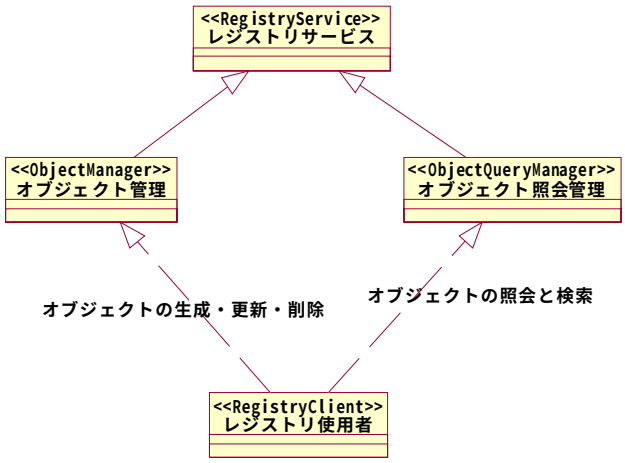


図 2:　ebXML レジストリインタフェース

6.3 インタフェース

本仕様書は、レジストリによって公開されるインタフェース（6.4 節）と、レジストリクライアントによって公開されるインタフェース（6.5 節）を定義している。図 2 は、インタフェース間の関係と、特定のレジストリインタフェースと特定のレジストリクライアントインタフェースのマッピングを示している。

6.4 レジストリによって公開されるインタフェース

ebXML メッセージ取扱サービス仕様を用いる際、ebXML レジストリサービス要素は、以下のようにメッセージ取扱サービス要素に対応する。
・MessageHeader のサービス要素の値は、ebXML レジストリサービスインタフェース名となる（たとえば、「ObjectManager」）。サービス要素の型属性は、「ebXMLRegistry」の値を持たなければならない。
・MessageHeader のアクション要素の値は、ebXML レジストリサービスメソッド名となる（たとえば、「submitObjects」）。

上記のことは、メッセージごとに 1 組のインタフェース/メソッドのみをレジストリクライアントに許可することに注意すること。これは、レジストリに対する要求に対して、レジストリクライアントは指定されたインタフェース上で 1 つのメソッドしか呼び出すことができないことを意味している。

6.4.1 同期および非同期応答

レジストリが公開するインタフェース上のすべてのメソッドは、応答メッセージを返す。

・非同期応答
o MessageHeader のみ
o レジストリ応答要素なし（たとえば、AdHocQueryResponse および GetContentResponse）
・同期応答
o MessageHeader
o 以下を含むレジストリ応答要素
・ステータス属性（成功または失敗）
・オプションの ebXML エラー

ebXML レジストリは、以下のインタフェースをそのサービス（レジストリサービス）として実行する。

6.4.2 RegistryService インタフェース

これはレジストリにより実装される主要なインタフェースである。このインタフェースは、クライアントがレジストリによって実装されるサービス固有のインタフェースを発見するために使用するメソッドを提供する。

RegistryService メソッドの概要	
ObjectManager	getObjectManager () レジストリサービスによって実装される ObjectManager インタフェースを返す。
ObjectQueryManager	getObjectQueryManager () レジストリサービスによって実装される ObjectQueryManager イ

	ンタフェースを返す。

6.4.3 ObjectManager インタフェース

これはレジストリのライフサイクル管理機能を実装するレジストリサービスによって公開されるインタフェースである。そのメソッドはレジストリクライアントによって呼び出される。たとえば、クライアントはこのインタフェースを用いて、オブジェクトの申請、オブジェクトの分類と関連付け、そしてオブジェクトの廃止および削除を実行できる。本仕様書で使用している申請、分類、関連付け、廃止、削除の意味は、[ebRIM]で解説している。

ObjectManager メソッドの概要	
RegistryResponse	approveObjects (ApproveObjectsRequest req) すでに申請済みの 1 つまたは複数のオブジェクトを承認する。
RegistryResponse	deprecateObjects (DeprecateObjectsRequest req) すでに申請済みの 1 つまたは複数のオブジェクトを廃止する。
RegistryResponse	removeObjects (RemoveObjectsRequest req) すでに申請済みの 1 つまたは複数のオブジェクトをレジストリから削除する。
RegistryResponse	submitObjects (SubmitObjectsRequest req) 1 つまたは複数のオブジェクト、そして場合によっては関係および分類など、関連するメタデータを申請する
RegistryResponse	addSlots (AddSlotsRequest req) スロットを 1 つまたは複数のレジストリエントリに追加する。
RegistryResponse	removeSlots (RemoveSlotsRequest req) 指定されたスロットを 1 つまたは複数のレジストリエントリから削除する。

6.4.4 ObjectQueryManager インタフェース

これはレジストリのオブジェクトクエリ管理サービスを実装するレジストリが公開するインタフェースである。そのメソッドはレジストリクライアントが呼び出す。たとえば、クライアントは、このインタフェースによりレジストリのコンテンツに対してクエリまたはアドホッククエリの実行、ブラウズ、およびドリルダウンできる。

ObjectQueryManager メソッドの概要	
RegistryResponse	getClassificationTree (GetClassificationTreeRequest req) GetClassificationTreeRequest で指定された ClassificationNode にある ClassificationNode ツリーを返す。
RegistryResponse	getClassifiedObjects (GetClassifiedObjectsRequest req) 指定された ClassificationItem に分類される RegistryEntries への参照のコレクションを返す。
RegistryResponse	getContent () 指定されたコンテンツを返す。応答には応答メッセージ内の追加搬送内容として要求で指定されたすべてのコンテンツが含まれる。

RegistryResponse	getRootClassificationNodes (GetRootClassificationNodesRequest req) GetRootClassificationNodesRequest 要求の namePattern 属性と一致するルート ClassificationNodes をすべて返す。
RegistryResponse	submitAdhocQuery (AdhocQueryRequest req) アドホッククエリ要求を申請する。

6.5 レジストリクライアントによって公開されるインタフェース

ebXML レジストリクライアントは以下のインタフェースを実装する。

6.5.1 RegistryClient インタフェース

これはレジストリクライアントが実装する主要なインタフェースである。クライアントはレジストリと接続するときにこのインタフェースを提供する。このインタフェースは、クライアントに非同期応答を提供するためにレジストリが使用するメソッドを提供する。クライアントとレジストリ間の[CPA]が非同期応答をサポートしていない場合は、クライアントは RegistryClient インタフェースを提供する必要がないことに注意すること。

レジストリは、オペレーションに対するすべての非同期応答を onResponse メソッドに送信する。

RegistryClient メソッドの概要	
void	onResponse (RegistryResponse resp) 前回申請された要求にレジストリが送信した応答をクライアントに通知する。

6.6 レジストリ応答クラス階層

レジストリからの多くの応答は共通する属性を持っているため、以下のクラス階層に整理される。この階層は、レジストリ DTD に反映される。

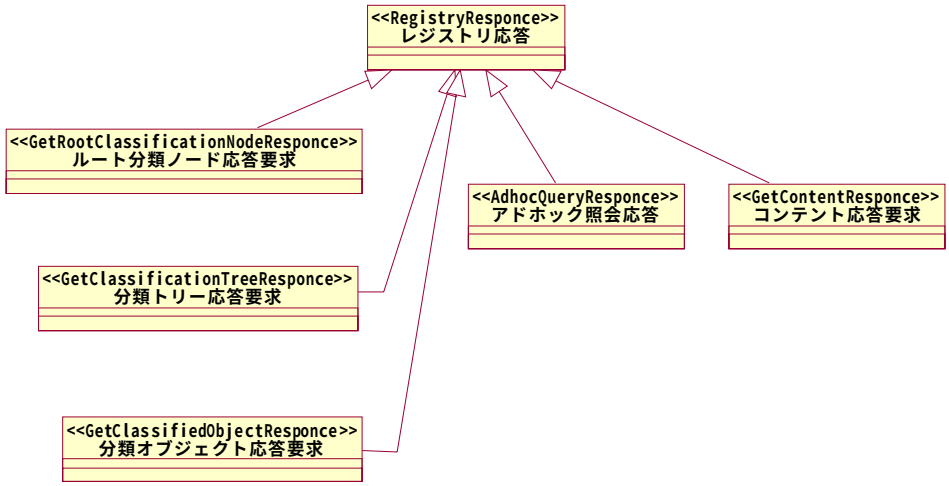


図 3: レジストリ応答クラス階層

7 オブジェクト管理サービス

本節では、レジストリのオブジェクト管理サービスを定義する。オブジェクト管理サービスは、レジストリサービスのサブサービスである。オブジェクト管理サービスは、RegistryClients が要求するリポジトリ項目（たとえば、ebXML ビジネスプロセスで要求される XML 文書）のライフサイクルを管理する機能を提供する。オブジェクト管理サービスは、すべてのタイプのリポジトリ項目のほかにも、分類と関係など、[ebRIM]で指定されるメタデータオブジェクトすべてに対して使用できる。

コンテンツに ebXML レジストリによって認められる認証機関が発行した証明書によるデジタル署名がある場合、ebXML レジストリの最小限のセキュリティポリシは、どのクライアントからもコンテンツを受けるとする。申請組織はコンテンツを申請する前に登録する必要がない。

7.1 リポジトリ項目のライフサイクル

オブジェクト管理サービスの主な目的は、リポジトリ項目のライフサイクルを管理することにある。

図 4 はリポジトリ項目の一般的なライフサイクルを示す。本仕様書の現在のバージョンではオブジェクトのバージョン管理をサポートしないことに注意が必要である。オブジェクトのバージョン管理は本仕様書の将来のバージョンで追加される予定である。

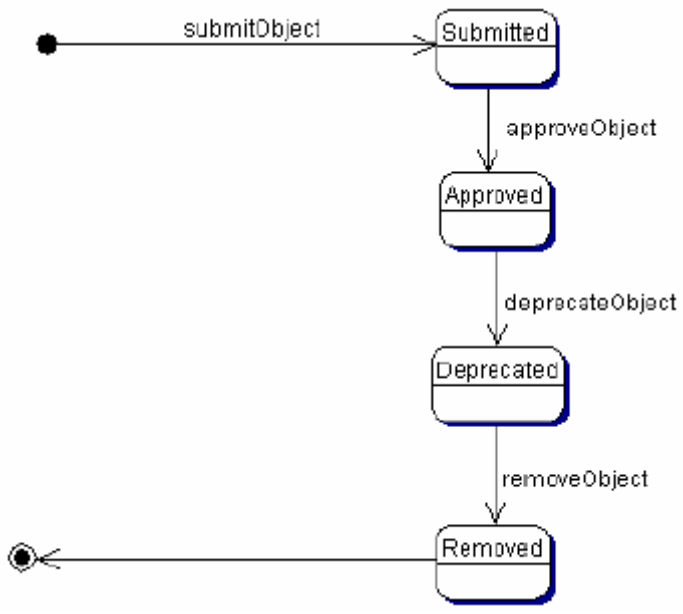


図 4: リポジトリ項目のライフサイクル

7.2 RegistryObject の属性

リポジトリ項目は[ebRIM]で規定する RegistryObject クラスとそのサブクラスの属性として定義される 1 組の標準メタデータと関係付けられる。これらの属性は、実際のリポジトリ項目およびリポジトリ項目に関するカタログ記述情報の外部に存在する。ExtrinsicObject および IntrinsicObject と呼ばれる XML 要素（詳細については、付録 A を参照のこと）に[ebRIM]で XML 属性として定義されるすべてのオブジェクトのメタデータ属性が含まれる。

7.3 オブジェクト申請プロトコル

本節では、RegistryClient が申請組織を代表して ObjectManager を用いて 1 つまたは複数のリポジトリ項目をリポジトリに申請するために使用するレジストリサービスのプロトコルを規定する。これは付録 B の説明のように UML の表記法で表現する。

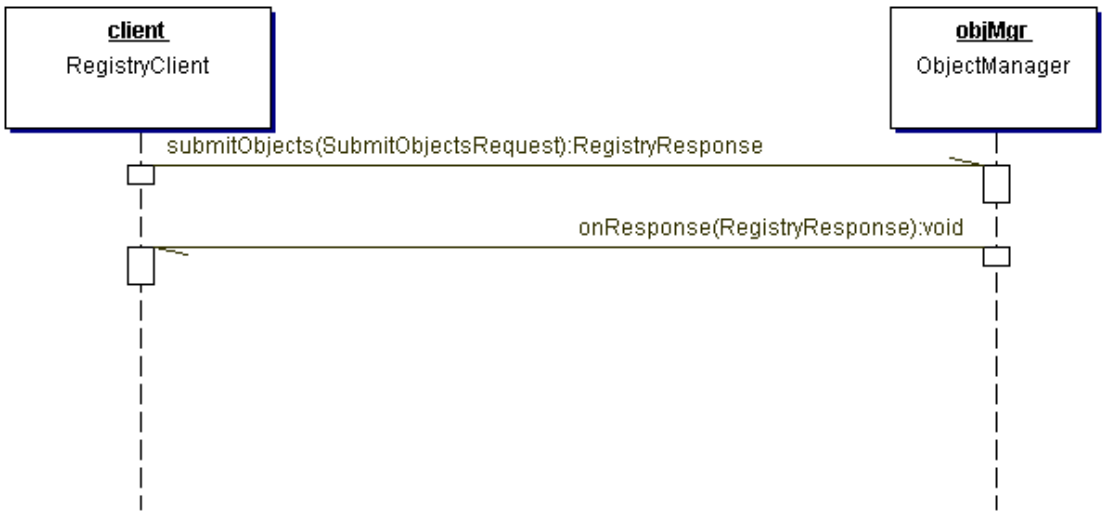


図 5: オブジェクト申請シーケンス図

このプロセスで示される取引文書のスキーマに関する詳細は、付録 A を参照のこと。

SubmitObjectRequest メッセージには RegistryEntryList 要素が含まれる。

RegistryEntryList 要素は、1 つ以上の ExtrinsicObject、あるいは分類、関係、ExternalLinks、または Packages といった他の RegistryEntry を指定する。

ExtrinsicObject 要素は、[ebRIM]が定義する通りレジストリが申請するコンテンツに関して必要なメタデータを提供する。これらの標準 ExtrinsicObject 属性はリポジトリ項目自体とは別個のものであるため、ebXML レジストリはどのオブジェクト型のオブジェクトをもカタログに登録できることに注意が必要である。

成功した場合は、レジストリは「success」のステータスと共に RegistryResponse をクライアントに返す。失敗した場合は、レジストリは「failure」のステータスと共に RegistryResponse をクライアントに返す。

7.3.1 絶対一意識別子（UUID）の生成

[ebRIM]で規定している通り、レジストリ内のオブジェクトにはすべて一意の id がある。この id は、絶対一意識別子 (UUID)でなければならず、また、[UUID]で定義している DCE 128 ビット UUID を指定する URN の書式に適合しなければならない。
(たとえば、urn:uuid:a2345678-1234-1234-123456789012)

この id は通常レジストリが生成する。申請されたオブジェクトの id 属性は、クライアントが随意に供給できる。クライアントが id を供給し、それが DCE 128 ビット UUID を指定す

る URN の書式に適合する場合は、レジストリは、クライアントがオブジェクトに id を指定しようとしていると判断される。この場合、レジストリはクライアントが指定した id を優先し、それをオブジェクトの id 属性としてレジストリで使用しなければならない。id が絶対一意でないことがレジストリによって判明した場合、レジストリはエラー条件 InvalidIdError を発生させなければならない。

クライアントが申請オブジェクトに id を指定しないときは、レジストリが絶対一意の id を生成しなければならない。id がクライアントによって生成される場合でも、レジストリによって生成される場合でも、id は[UUID]の仕様に従って DCE 128 ビット UUID 生成アルゴリズムを用いて生成しなければならない。

7.3.2 ID 属性とオブジェクト参照

オブジェクトの id 属性は、他のオブジェクトが最初のオブジェクトを参照するときに使用できる。そのような参照は SubmitObjectsRequest 内でも、レジストリ内でも共通である。SubmitObjectsRequest 内では、id 属性は SubmitObjectsRequest 内のオブジェクトを参照する場合でも、レジストリ内のオブジェクトを参照する場合にも使用できる。要求文書内で参照が必要な SubmitObjectsRequest 内のオブジェクトに対して、申請者は要求の中で参照できるように id を割り当てることができる。申請者はオブジェクト固有の uuid URN を指定でき、その場合、id はレジストリ内のオブジェクトに永続的に割り当てられる。別の方法として、申請者はその id が要求文書内で一意であるかぎり、任意の id(固有の uuid URN ではない)を割り当てることができる。この場合、id は要求文書内で連結機構の役割を果たすが、レジストリ側ではこれを無視して、申請され次第レジストリによって生成される id に置換しなければならない。

SubmitObjectsRequest 内のオブジェクトがすでにレジストリにあるオブジェクトを参照する必要がある場合、要求には id 属性がレジストリ内のオブジェクトの id である ObjectRef 要素が含まれることが必要である。この id は当然、固有の uuid URN である。ObjectRef はレジストリにあるオブジェクトに対する要求の内部のプロキシと考えることができる。

7.3.3 SubmitObjectsRequest のサンプル

次の例は単一の SubmitObjectRequest における数種類のユースケースを示す。これはリポジトリ項目に関するメッセージにメッセージヘッダと追加搬送内容を含む完全な ebXML メッセージを示すものではない。

SubmitObjectsRequest には、任意の数の申請オブジェクトを含む RegistryEntryList が含まれる。また、申請オブジェクトをすでにレジストリにあるオブジェクトとリンクする任意の数の ObjectRefs が含まれる場合がある。

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE SubmitObjectsRequest SYSTEM "file:////home/najmi/Registry.dtd">

<SubmitObjectsRequest>
  <RegistryEntryList>

    <!—
    The following 3 objects package specified ExtrinsicObject in specified
    Package, where both the Package and the ExtrinsicObject are
    being  submitted
    -->
    <Package id = "acmePackage1" name = "Package #1" description = "ACME's package #1"/>
    <ExtrinsicObject id = "acmeCPP1" contentURI = "CPP1"
      objectType = "CPP" name = "Widget Profile"
      description = "ACME's profile for selling widgets"/>
    <Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages"
      sourceObject = "acmePackage1" targetObject = "acmeCPP1"/>

    <!—
    The following 3 objects package specified ExtrinsicObject in specified Package,
    Where the Package is being submitted and the ExtrinsicObject is
    already in registry
    -->
    <Package id = "acmePackage2" name = "Package #2" description = "ACME's package #2"/>
    <ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
    <Association id = "acmePackage2-alreadySubmittedCPP-Assoc"
      associationType = "Packages" sourceObject = "acmePackage2"
      targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>

    <!—
    The following 3 objects package specified ExtrinsicObject in specified Package,
    where the Package and the ExtrinsicObject are already in registry
    -->
    <ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
    <ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
    <!-- id is unspecified implying that registry must create a uuid for this object -->
    <Association associationType = "Packages"
      sourceObject = "urn:uuid:b2345678-1234-1234-123456789012"
      targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>

    <!—
    The following 3 objects externally link specified ExtrinsicObject using
    specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
    are being submitted
    -->
    <ExternalLink id = "acmeLink1" name = "Link #1" description = "ACME's Link #1"/>
    <ExtrinsicObject id = "acmeCPP2" contentURI = "CPP2" objectType = "CPP"
      name = "Sprockets Profile" description = "ACME's profile for selling sprockets"/>
    <Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
      sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>

    <!--
    The following 2 objects externally link specified ExtrinsicObject using specified
    ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
    is already in registry. Note that the targetObject points to an ObjectRef in a
    previous line
    -->
    <ExternalLink id = "acmeLink2" name = "Link #2" description = "ACME's Link #2"/>
    <Association id = "acmeLink2-alreadySubmittedCPP-Assoc"
      associationType = "ExternallyLinks" sourceObject = "acmeLink2"
      targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>

    <!--
    The following 2 objects externally identify specified ExtrinsicObject using specified
    ExternalIdentifier, where the ExternalIdentifier is being submitted and the
    ExtrinsicObject is already in registry. Note that the targetObject points to an
    ObjectRef in a previous line
    -->
    <ExternalIdentifier id = "acmeDUNSId" name = "DUNS" description = "DUNS ID for ACME"
      value = "13456789012"/>
    <Association id = "acmeDUNSId-alreadySubmittedCPP-Assoc"
      associationType = "ExternallyIdentifies" sourceObject = "acmeDUNSId"
```

```
      targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>

    <!--
    The following show submission of a brand new classification scheme in its entirety
    -->
    <ClassificationNode id = "geographyNode" name = "Geography"
      description = "The Geography scheme example from Registry Services Spec" />
    <ClassificationNode id = "asiaNode" name = "Asia"
      description = "The Asia node under the Geography node" parent="geographyNode" />
    <ClassificationNode id = "japanNode" name = "Japan"
      description ="The Japan node under the Asia node" parent="asiaNode" />
    <ClassificationNode id = "koreaNode" name = "Korea"
      description ="The Korea node under the Asia node" parent="asiaNode" />
    <ClassificationNode id = "europeNode" name = "Europe"
      description = "The Europe node under the Geography node" parent="geographyNode" />
    <ClassificationNode id = "germanyNode" name = "Germany"
      description ="The Germany node under the Asia node" parent="europeNode" />
    <ClassificationNode id = "northAmericaNode" name = "North America"
      description = "The North America node under the Geography node"
      parent="geographyNode" />
    <ClassificationNode id = "usNode" name = "US"
      description ="The US node under the Asia node" parent="northAmericaNode" />

    <!--
    The following show submission of a Automotive sub-tree of ClassificationNodes that
    gets added to an existing classification scheme named 'Industry'
    that is already in the registry
    -->
    <ObjectRef id="urn:uuid:d2345678-1234-1234-123456789012" />
    <ClassificationNode  id = "automotiveNode" name = "Automotive"
      description = "The Automotive sub-tree under Industry scheme"
      parent = "urn:uuid:d2345678-1234-1234-123456789012"/>
    <ClassificationNode id = "partSuppliersNode" name = "Parts Supplier"
      description = "The Parts Supplier node under the Automotive node"
      parent="automotiveNode" />
    <ClassificationNode id = "engineSuppliersNode" name = "Engine Supplier"
      description = "The Engine Supplier node under the Automotive node"
      parent="automotiveNode" />

    <!--
    The following show submission of 2 Classifications of an object  that is already in
    the registry using 2 ClassificationNodes. One ClassificationNode
    is being submitted in this request (Japan) while the other is already in the registry.
    -->
    <Classification id = "japanClassification"
      description = "Classifies object by /Geography/Asia/Japan node"
      classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
      classificationNode="japanNode" />
    <Classification id = "classificationUsingExistingNode"
      description = "Classifies object using a node in the registry"
      classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
      classificationNode="urn:uuid:e2345678-1234-1234-123456789012" />
    <ObjectRef id="urn:uuid:e2345678-1234-1234-123456789012" />


  </RegistryEntryList>
</SubmitObjectsRequest>
```

7.4 スロット追加プロトコル

本節では、クライアントが ObjectManager を用いてスロットをすでに申請済みのレジストリエントリに追加できるようにするためのレジストリサービスのプロトコルを説明する。スロットは[ebRIM]が定義するレジストリエントリを拡張するための動的機構を提供する。

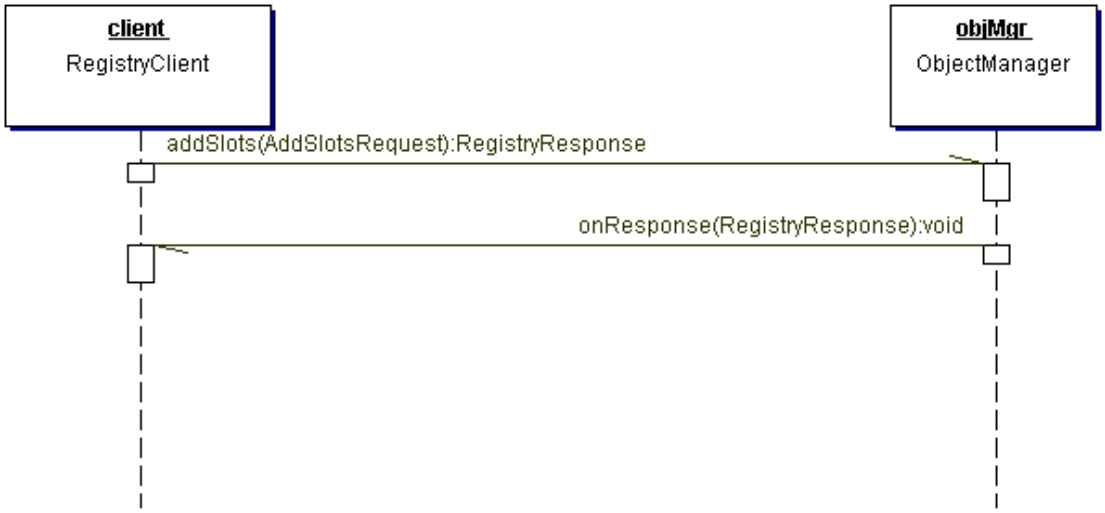


図 7: スロット追加シーケンス図

成功した場合は、レジストリは「success」のステータスと共に RegistryResponse をクライアントに返す。失敗した場合は、レジストリは「failure」のステータスと共に RegistryResponse をクライアントに返す。

7.5 スロット削除プロトコル

本節では、クライアントが ObjectManager を用いてスロットをすでに申請済みのレジストリエントリに追加できるようにするためのレジストリサービスのプロトコルを説明する。

図 8: スロット削除シーケンス図

成功した場合は、レジストリは「success」のステータスと共に RegistryResponse をクライアントに返す。失敗した場合は、レジストリは「failure」のステータスと共に RegistryResponse をクライアントに返す。

7.6 オブジェクト承認プロトコル

本節では、クライアントが ObjectManager を用いて 1 つまたは複数の申請済みリポジトリ項目を承認できるようにするためのレジストリサービスのプロトコルについて説明する。リポジトリ項目が承認されると、その項目は取引当事者が使用できるようになる（たとえば、新規の CPA およびコラボレーションプロトコルプロファイルの作成時）。

図 9: オブジェクト承認シーケンス図

成功した場合は、レジストリは「success」のステータスと共に RegistryResponse をクライアントに返す。失敗した場合は、レジストリは「failure」のステータスと共に RegistryResponse をクライアントに返す。

このプロセスで示される取引文書のスキーマに関する詳細については、付録 A を参照のこと。

7.7 オブジェクト廃止プロトコル

本節では、クライアントが ObjectManager を用いて 1 つまたは複数の申請済みのリポジトリ項目を廃止できるようにするためのリポジトリサービスのプロトコルを説明する。オブジェクトが廃止されると、そのオブジェクトに対する新規の参照（たとえば、新規の関係、分類、および ExternalLinks）は申請できなくなる。ただし、廃止されたオブジェクトの既存の参照は引き続き正常に機能する。

図 10: オブジェクト廃止シーケンス図

成功した場合は、レジストリは「success」のステータスと共に RegistryResponse をクライアントに返す。失敗した場合は、レジストリは「failure」のステータスと共に RegistryResponse をクライアントに返す。

このプロセスで示される取引文書のスキーマに関する詳細については、付録 A を参照のこと。

7.8 オブジェクト削除プロトコル

本節では、クライアントがオブジェクトマネージャを用いて 1 つまたは複数のレジストリエントリ、またはリポジトリ項目、またはこれらの両方を削除できるようにするためのレジストリサービスのプロトコルを説明する。

RemoveObjectsRequest メッセージは、RegistoryEntry のインスタンス、またはリポジトリ項目を削除するためにクライアントに送信される。RemoveObjectsRequest 要素には、以下の節で定義する値を持ちうる列挙である、deletionScope と呼ばれる XML 属性が含まれる。

7.8.1 削除範囲 DeleteRepositoryItemOnly

この deletionScope は、その要求によって指定されたレジストリエントリに対するリポジトリ項目は削除するが、指定されたレジストリエントリは削除してはならないことを指定する。これはレジストリエントリへの参照を有効のまま維持する上で有効である。

7.8.2 削除範囲 DeleteAll

この deletionScope は、その要求によって RegistryEntry、および指定されたレジストリエントリに対するリポジトリ項目の両方を削除することを指定する。RegistryEntry に対する参照(たとえば、関係、分類、ExternalLinks)がすべて削除された場合に限り、その RegistryEntry は RemoveObjectsRequest を使って deletionScope DeleteAll により削除できる。まだ RegistryEntry に参照があるときに RegistryEntry を削除すると、エラー条件 InvalidRequestError が発生する。

オブジェクト削除プロトコルは付録 B の規定に従って UML 表記法で表現する。

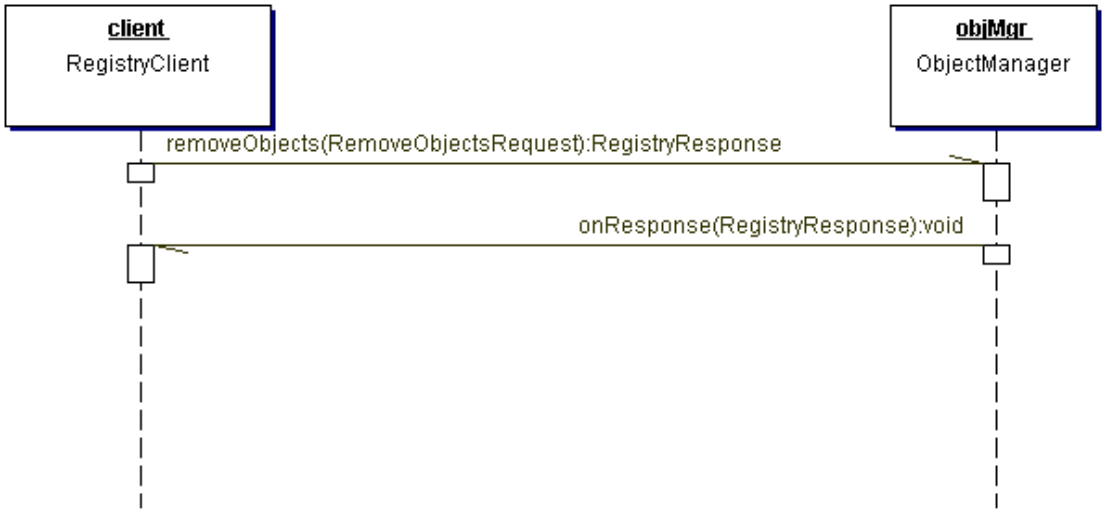


図 11: オブジェクト削除シーケンス図

成功した場合は、レジストリは「success」のステータスと共に RegistryResponse をクライアントに返す。失敗した場合は、レジストリは「failure」のステータスと共に RegistryResponse をクライアントに返す。

このプロセスで示される取引文書のスキーマに関する詳細については、付録 A を参照のこと。

8 オブジェクトクエリ管理サービス

本節では、クライアント(ObjectQueryManagerClient)がレジストリの ObjectQueryManager インタフェースを用いて ebXML レジストリ内の RegistryEntries を検索、または照会できるレジストリサービスの機能を説明する。

レジストリはいくつかのクエリ機能をサポートする。それには次の機能が含まれる。

1. ブラウズドリルダウンクエリ

2. フィルタクエリ

3. SQL クエリ

8.1 節のブラウズドリルダウンクエリと 8.2 節のフィルタクエリ機構は、いずれのレジストリ実装によってもサポートされる。SQL クエリ機構はオプションであり、レジストリ実装によって提供してもよい。ただし、ベンダが SQL クエリ機能を ebXML レジストリに提供するときは、本書に適合するものとする。したがって、この機能は規範的ではあるが、任意の機能である。

本仕様書の将来のバージョンでは、W3C XQuery 構文がもう 1 つのクエリ構文として検討されるかもしれない。

クエリ要求メッセージにエラーがあると、それが該当するクエリ応答メッセージで指示される。

8.1 ブラウズドリルダウンクエリのサポート

ブラウズドリルダウンクエリスタイルは、ObjectQueryManagerClient と ObjectQueryManager との間の相互作用プロトコルのセットによりサポートされる。8.1.1 節、8.1.2 節、8.1.3 節でこれらのプロトコルを説明する。

8.1.1 ルート分類ノード取得要求

ObjectQueryManagerClient はこの要求を送信し、リポジトリで定義されているルート ClassificationNodes の一覧を取得する。ルート分類ノードは親を持たないノードであると定義される。ルート ClassificationNodes の名前属性にフィルタをかけることができる namePattern 属性を指定できることに注意が必要である。namePattern は[SQL]の定義に従って SQL-92 LIKE 文節により定義されるワイルドカードパタンを用いて指定しなければならない。

図 12: ルート分類ノード取得シーケンス図

成功した場合は、レジストリは「success」のステータスと共に GetRootClassificationNodeResponse をクライアントに返す。失敗した場合は、レジストリは「failure」のステータスと共に GetRootClassificationNodeResponse をクライアントに返す。

このプロセスで示される取引文書のスキーマに関する詳細については、付録 A を参照のこと。

8.1.2 分類ツリー取得要求

ObjectQueryManagerClient はこの要求を送信し、リポジトリ内のこの要求により指定される ClassificationNodes で定義された ClassificationNode サブツリーを取得する。GetClassificationTreeRequest が深さと呼ばれる整数属性を指定して指定された深さまでのサブツリーを取得できることに注意が必要である。深さがデフォルト値の 1 である場合、指定された ClassificationNodeList の直下の子だけが返される。深さが 0 または負の数の場合は、サブツリー全体が検索される。

図 14: 分類ツリー取得シーケンス図

成功した場合は、レジストリは「success」のステータスと共に GetRootClassificationNodeResponse をクライアントに返す。失敗した場合は、レジストリは「failure」のステータスと共に GetRootClassificationNodeResponse をクライアントに返す。

このプロセスで示される取引文書のスキーマに関する詳細については、付録 A を参照のこと。

8.1.3 分類済みオブジェクト取得要求

ObjectQueryManagerClient はこの要求を送信し、要求内の ObjectRefList により指定された通り、指定された ClassificationNodes(または、その派生のいずれか)すべてによって分類された RegistryEntries の一覧を取得する。

RegistryEntries は複数の分類との一致を基に取得することが可能である。ClassificationNode を指定することは、指定された ClassificationNode のすべての派生の論理和を暗黙に指定することである。

GetClassifiedObjectsRequest が ObjectQueryManager に送られるとき、次の条件に該当するオブジェクトを返すべきである。

1. 指定された ClassificationNode によって直接分類されるオブジェクト、もしくは

2. 指定された ClassificationNode の派生によって直接分類されるオブジェクト

8.1.3.1 分類済みオブジェクト取得要求のサンプル

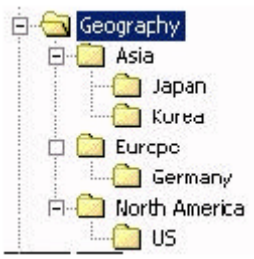


図 16: ジオグラフィ分類のサンプル

分類ツリーが図 16 で示す構造であるとする。

・地理ノードが GetClassifiedObjectsRequest で指定される場合、GetClassifiedObjectsResponse には地理、すなわち北米、米国、アジア、日本、韓国、欧州、ドイツによって直接分類される RegistryEntries をすべて含まなければならない。

・アジアノードが GetClassifiedObjectsRequest で指定される場合、GetClassifiedObjectsResponse には、アジア、日本、または韓国によって直接分類される RegistryEntries をすべて含まなければならない。

・日本および韓国のノードが GetClassifiedObjectsRequest で指定される場合、GetClassifiedObjectsResponse には、日本および韓国の両方によって直接分類される RegistryEntries をすべて含まなければならない。

・北米およびアジアのノードが GetClassifiedObjectsRequest で指定される場合、GetClassifiedObjectsResponse には（北米または米国）および(アジア、日本、または韓国)によって直接分類される RegistryEntries をがすべて含まなければならない。

図 17: 分類されたオブジェクト取得シーケンス図

成功した場合は、レジストリは「success」のステータスと共に GetRootClassificationNodeResponse をクライアントに返す。失敗した場合は、レジストリは「failure」のステータスと共に GetRootClassificationNodeResponse をクライアントに返す。

8.2 フィルタクエリのサポート

FilterQuery は、ebXML に合致するレジストリ実装に対して、簡単なクエリ機能を提供する XML 構文である。それぞれの代替クエリは、ebXML レジストリ情報モデル（ebRIM）によって定義された単一クラスに対して指示される。こうしたクエリの結果は、そのクラスのインスタンスに対する識別子のセットとなる。FilterQuery は、スタンドアローン型クエリとなることも、ReturnRegistryEntry クエリまたは ReturnRepositoryItem クエリの初期アクションとなることもできる。

クライアントは、FilterQuery、ReturnRegistryEntry クエリ、または ReturnRepositoryItem クエ

リを、AdhocQueryRequest の一部として ObjectQueryManager に申請する。ObjectQueryManager は、ここで指定された適切な FilterQueryResponse、ReturnRegistryEntryResponse、または ReturnRepositoryItemRespose を同封し、AdhocQueryRespose をクライアントに返す。AdhocQueryRequest および AdhocQueryResponse に関するシーケンス図については、節 8.4 で指定されている。

それぞれの FilterQuery 代替は、ebRIM バインディングと関連している。この ebRIM バインディングは、ebRIM が定義する通り、単一クラスと、他のクラスとのその関係から派生したクラスの階層を識別する。クラスが選択されるたびに、ツリーとして照会できる仮想 XML 文書が事前定義される。たとえば、C をクラスとし、C に直接関連するクラスを Y と Z とし、Z と関連するクラスを V とする。C に対する ebRIM バインディングは図 19 のようになる。

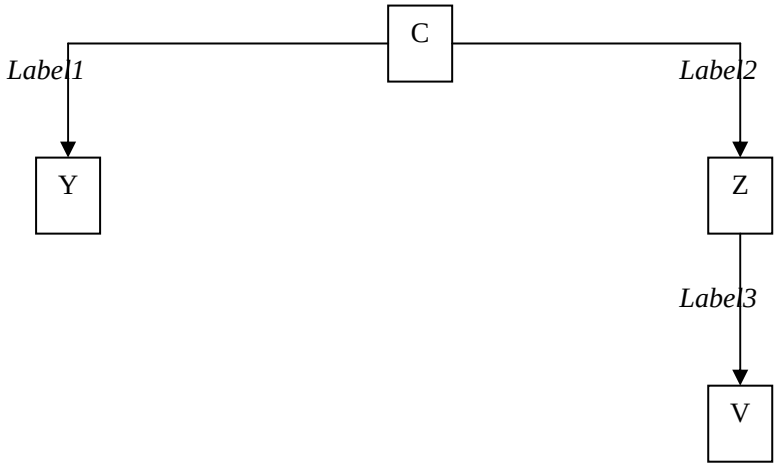


図 19:　ebRIM バインディングの例

ラベル 1 は、C から Y への関連を識別する。ラベル 2 は C から Z への関連を、ラベル 3 は Z から V への関連をそれぞれ識別する。どの ebRIM 関連が指定されるかについて曖昧さが全くない場合は、ラベルを省略できる。クエリの名前はルートクラスによって決定される。すなわち CQuery に対する ebRIM バインディングである。ツリー内の Y ノードはラベル 1 で識別される関連によって C に結びつけられる Y インスタンスのセットに限定される。同様に、Z および V ノードは、識別される関連によってその親ノードに結びつけられるインスタンスに限定される。

それぞれの FilterQuery 代替は、1 つまたは複数のクラスフィルタに依存する。ここでは、クラスフィルタは単一クラスの属性に関する限定的述語文節である。サポートされるクラスフィルタについては 8.2.9 節で、サポートされる述語文節については 8.2.10 節で指定する。FilterQuery は、指定されたクラスフィルタをどの分岐が満たしているかを決定するためにツリーを探索する要素から構成され、クエリの結果はこうした分岐をサポートするルートノードインスタンスのセットとなる。

上記の例では、CQuery 要素は 3 種類の下位要素を持つ。最初の下位要素は C クラス上の CFilter で、CFilter の述語を満たさない C インスタンスを排除する。2 番目の下位要素は Y クラス上の YFilter で、関連のターゲットが YFilter を満たさない C から Y への分岐を排除する。最後の下位要素は、C から Z を通って V に至るパスに沿って分岐を排除する。この 3 番目の要素は、それぞれのクラス上のクラスフィルタに対して X から V へのパスを許可するため、分岐要素と呼ばれる。一般的に、分岐要素は、それ自身がクラスフィルタであ

る下位要素と、他の分岐要素、または、パス内のターミナルクラス上の完全なクエリを持つ。

クラス C からクラス Y への関連が 1 対 0 または 1 対 1 の場合、Y 上の多くとも 1 つの分岐かフィルタ要素が許可される。しかし、関連が 1 対多の場合、複数のフィルタまたは分岐要素が許可される。これにより、C のインスタンスが分岐要素を満たすと報告される前に、C のこのインスタンスが Y の複数のインタフェースと関連を持たなければならないことを指定できる。

FilterQuery 構文は、ebRIM で定義する構造に従う。ebRIM は安定性指向なので、FilterQuery 構文は安定している。しかし、新しい構造が ebRIM に追加される場合は、FilterQuery 構文と意味を同時に拡張できる。

FilterQuery のサポートは、ebXML に合致するレジストリ実装に対して要求されているが、その他のクエリオプションも可能である。レジストリは、サポートされたすべての AdhocQuery オプションを識別する自己記述型 CPP を保持する。このプロファイルは、6.1 節で説明している。

8.2.2 節から 8.2.6 節の RIM バインディングの段落は、それぞれの FilterQuery 代替に対する仮想階層を示す。各クエリ代替の意味情報規則は、そのバインディングがクエリの意味情報に及ぼす結果を指定する。

以下に定義する ReturnRegistryEntry および ReturnRepositoryItem サービスは、RegistryEntryQuery の結果の拡張として、XML 文書を構造化する方法を提供する。8.2.7 節で指定する ReturnRegistryEntry 要素により、RegistryEntryQuery の結果によって識別される各レジストリエントリとともに返すメタデータを指定できる。8.2.8 節で指定する ReturnRepositoryItem により、RegistryEntryQuery の結果によって識別されるレジストリエントリとの関係に基づいて返すポジトリ項目を指定できる。

8.2.1 FilterQuery

目的

特定のレジストリクラスからのレジストリインスタンスのセットを識別する。それぞれのクエリ型は ebRIM に対する特定のバインディングを前提とする。各クエリ型に対するクエリ結果は、バインディングにより指定されるルートクラスのインスタンスへの参照のセットである。ステータスは処理の成功を示すものであるか、警告や例外の集まりである。

定義

```
<!ELEMENT FilterQuery
 (    RegistryEntryQuery
   |  AuditableEventQuery
   |  ClassificationNodeQuery
   |  RegistryPackageQuery
   |  OrganizationQuery        )>

<!ELEMENT FilterQueryResult
  (   RegistryEntryQueryResult
   |  AuditableEventQueryResult
   |  ClassificationNodeQueryResult
   |  RegistryPackageQueryResult
```

```
   |  OrganizationQueryResult   )>

<!ELEMENT RegistryEntryQueryResult ( RegistryEntryView* )>

<!ELEMENT RegistryEntryView EMPTY >
<!ATTLIST RegistryEntryView
   objectURN     CDATA     #REQUIRED
   contentURI    CDATA     #IMPLIED
   objectID      CDATA     #IMPLIED  >

<!ELEMENT AuditableEventQueryResult ( AuditableEventView* )>

<!ELEMENT AuditableEventView EMPTY >
<!ATTLIST AuditableEventView
   objectID      CDATA     #REQUIRED
   timestamp     CDATA     #REQUIRED  >

<!ELEMENT ClassificationNodeQueryResult
               (ClassificationNodeView*)>

<!ELEMENT ClassificationNodeView EMPTY >
<!ATTLIST ClassificationNodeView
   objectURN     CDATA     #REQUIRED
   contentURI    CDATA     #IMPLIED
   objectID      CDATA     #IMPLIED  >

<!ELEMENT RegistryPackageQueryResult ( RegistryPackageView* )>

<!ELEMENT RegistryPackageView EMPTY >
<!ATTLIST RegistryPackageView
   objectURN     CDATA     #REQUIRED
   contentURI    CDATA     #IMPLIED
   objectID      CDATA     #IMPLIED  >

<!ELEMENT OrganizationQueryResult ( OrganizationView* )>

<!ELEMENT OrganizationView EMPTY >
<!ATTLIST OrganizationView
   orgURN        CDATA     #REQUIRED
   objectID      CDATA     #IMPLIED  >
```

意味情報規則

1. 各 FilterQuery 型の意味情報規則は以後のサブ節で指定する。

2. 各 FilterQueryResult は、結果セットの各インスタンスを識別するための XML 参照要素のセットである。各 XML 属性は以下の通りレジストリ情報モデルで指定する属性の値から導かれる値を持つ。

a) objectID は RegistryObject クラスの ID 属性値であり、

b) objectURN および orgURN はオブジェクト ID から求められる URN 値であり、

c) contentURL は RegistryEntry クラスの contentURI 属性から求められる URL 値であり、

d) タイムスタンプは AuditableEvent クラスのタイムスタンプ属性の値を表すリテラル値で

ある。

3. FilterQuery の実行の任意の部分でエラー条件が発生すると、XML RegistryResult のステータス属性は「failure」に設定され、クエリ結果のどのような要素も返されない。その代わり、RegistryErrorList 要素は、「error」に設定されたその highestSeverity 要素と共に返されなければならない。RegistryErrorList の少なくとも 1 つの RegistryError 要素は、「error」に設定された重要度属性を持つ。

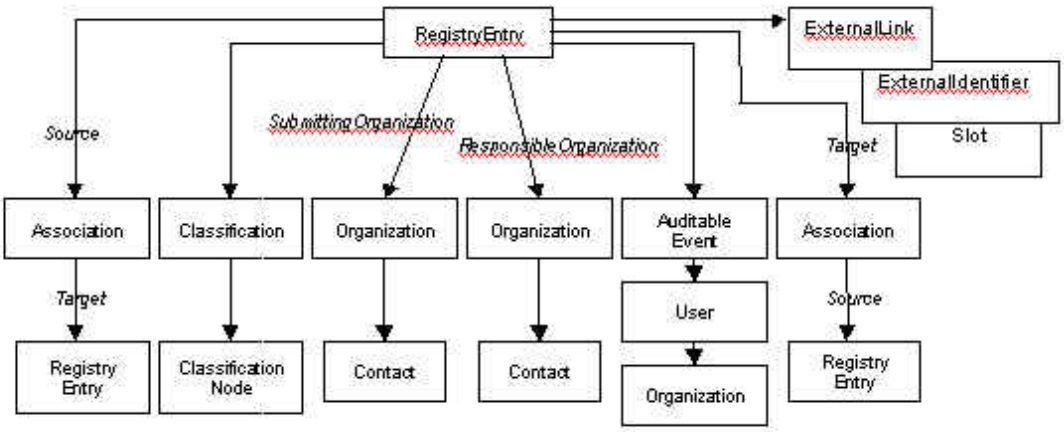
4. FilterQuery の実行でどのようなエラー条件も発生しない場合は、XML RegistryResult のステータス属性は「success」に設定され、適切なクエリ結果要素を含まなければならない。RegistryErrorList も返される場合は、RegistryErrorList の highestSeverity 属性は「warnig」に設定され、それぞれの RegistryError の重要度属性は「warnig」に設定される。

8.2.2 RegistryEntryQuery

目的

特定のレジストリメタデータに対するクエリの結果としてレジストリエントリのインスタンスのセットを識別すること。

ebRIM バインディング



定義

```
<!ELEMENT RegistryEntryQuery
   (  RegistryEntryFilter?,
      SourceAssociationBranch*,
      TargetAssociationBranch*,
      HasClassificationBranch*,
      SubmittingOrganizationBranch?,
      ResponsibleOrganizationBranch?,
      ExternalIdentifierFilter*,
      ExternalLinkFilter*,
      SlotFilter*,
      HasAuditableEventBranch*         )>

<!ELEMENT SourceAssociationBranch
  (   AssociationFilter?,
```

```
        RegistryEntryFilter?             )>

   <!ELEMENT TargetAssociationBranch
     (    AssociationFilter?,
        RegistryEntryFilter?             )>

   <!ELEMENT HasClassificationBranch
     (    ClassificationFilter?,
        ClassificationNodeFilter?        )>

   <!ELEMENT SubmittingOrganizationBranch
     (    OrganizationFilter?,
        ContactFilter?                   )>

   <!ELEMENT ResponsibleOrganizationBranch
     (    OrganizationFilter?,
        ContactFilter?                   )>

   <!ELEMENT HasAuditableEventBranch
     (    AuditableEventFilter?,
        UserFilter?,
        OrganizationFilter?              )>
```

意味情報規則

1. RE はレジストリ内の永続的 RegistryEntry インスタンスすべてを示すものとする。以下のステップにより、指定されたフィルタの条件を満たさない RE におけるインスタンスが取り除かれる。

a) RegistryEntryFilter が指定されない場合、または RE が空である場合は、次に続ける。そうでない場合、x を RE におけるレジストリエントリであるとする。もし x が 8.2.9 節で定義するように RegistryEntryFilter を満たさなければ、x を RE から削除する。

b) SourceAssociationBranch 要素が指定されない場合、または RE が空である場合は、次に続ける。そうでない場合、x が RE における残りのレジストリエントリであるものとする。x が何らかの Association インスタンスのソースオブジェクトでない場合は、x を RE から削除する。そうでない場合、SourceAssociationBranch の各要素を次のように別々に取り扱う。

SourceAssociationBranch の内部で AssociationFilter が指定されない場合は、AF が x をソースオブジェクトとして持つすべての Association インスタンスのセットであるものとする。そうでない場合、AF は AssociationFilter を満たし、かつ x をソースオブジェクトとする Association インスタンスのセットであるものとする。AF が空である場合は、x を RE から削除する。RegistryEntryFilter が SourceAssociationBranch の内部で指定されない場合、RET が AF のどれかの要素の対象オブジェクトであるすべての RegistryEntry インスタンスのセットであるものとする。そうでない場合、RET が RegistryEntryFilter を満たし、かつ AF の何らかの要素の対象オブジェクトである RegistryEntry インスタンスのセットであるものとする。RET が空である場合、x を RE から削除する。

c) TargetAssociationBranch 要素が指定されない場合、または RE が空である場合は、次に続ける。そうでない場合、x は RE における残りのレジストリエントリであるものとする。x が何らかの Association インスタンスの対象オブジェクトでないときは、x を RE から削除する。そうでない場合、各 TargetAssociationBranch 要素を次の通り別個に取り扱う。

AssociationFilter が TargetAssociationBranch 内で指定されない場合、AF が x を対象オブジェクトとするすべての Association インスタンスのセットであるものとする。そうでない場合、AF が AssociationFilter を満たし、x を対象オブジェクトとする Association インスタンスのセットであるものとする。AF が空である場合は、x を RE から削除する。RegistryEntryFilter が TargetAssociationBranch 内で指定されない場合は、RES が AF のいずれかの要素のソースオブジェクトであるすべての RegistryEntry のセットであるものとする。そうでない場合、RES が RegistryEntryFilter を満たし、AF のいずれかの要素のソースオブジェクトである RegistryEntry インスタンスのセットであるものとする。RES が空である場合は、x を RE から削除する。

d) HasClassificationBranch 要素が指定されない場合、または RE が空である場合、次に続ける。そうでない場合、x が RE における残りのレジストリエントリであるものとする。x がいずれかの Classification インスタンスのソースオブジェクトでない場合は、x を RE から削除する。そうでない場合、各 HasClassificationBranch 要素を次のように別個に取り扱う。

ClassificationFilter が HasClassificationBranch で指定されない場合、CL が x をソースオブジェクトとするすべての Classification インスタンスのセットであるものとする。そうでない場合、CL が ClassificationFilter を満たし、x をソースオブジェクトとする Classification インスタンスのセットであるものとする。CL が空である場合、x を RE から削除する。
ClassificationNodeFilter が HasClassificationBranch で指定されない場合、CN が CL のどれかの要素の対象オブジェクトであるすべての ClassificationNode インスタンスのセットであるものとする。そうでない場合、CN が ClassificationNodeFilter を満たし、CL のどれかの要素の対象オブジェクトである RegistryEntry インスタンスのセットであるものとする。CN が空である場合、x を RE から削除する。

e) SubmittingOrganizationBranch 要素が指定されない場合、または RE が空である場合、次に続ける。 そうでない場合、x が RE における残りのレジストリエントリであるものとする。x に申請組織がない場合は、x を RE から削除する。SubmittingOrganizationBranch 内で OrganizationFilter が指定されない場合、SO は x の申請組織であるすべての Organization インスタンスのセットであるものとする。そうでない場合、SO は OrganizationFilter を満たし、x の申請組織である Organization インスタンスのセットであるものとする。SO が空であるときは、x を RE から削除する。SubmittingOrganizationBranch 内で ContactFilter が指定されない場合は、CT が SO のどれかの要素の連絡先であるすべての Contact インスタンスのセットであるものとする。そうでない場合、CT が ContactFilter を満たし、SO のどれかの要素の連絡先である Contact インスタンスのセットであるものとする。CT が空である場合、x を RE から削除する。

f) ResponsibleOrganizationBranch 要素が指定されない場合、または RE が空である場合、次に続ける。そうでない場合、x が RE における残りのレジストリエントリであるものとする。x に責任のある組織がない場合、x を RE から削除する。ResponsibleOrganizationBranch 内で OrganizationFilter が指定されない場合は、RO が x に対して責任のある組織であるすべての Organization インスタンスのセットであるものとする。そうでない場合、RO が OrganizationFilter を満たし、x に対して責任のある組織である Organization インスタンスのセットであるものとする。RO が空である場合は、s を RE から削除する。SubmittingOrgFilter 内で ContactFilter が指定されない場合は、CT は RO のどれかの要素の連絡先であるすべての Contact インスタンスのセットであるものとする。そうでない場合、CT は ContactFilter を満たし、RO のどれかの要素の連絡先である Contact インスタンスのセットであるものとする。CT が空である場合、x を RE から削除する。

g) ExternalLinkFilter 要素が指定されない場合、または RE が空である場合は、次に続ける。そうでない場合、x が RE における残りのレジストリエントリであるものとする。x がどれかの ExternalLink インスタンスとリンクされない時は、x を RE から削除する。そうでない場合、次のように各 ExternalLinkFilter 要素を別個に取り扱う。

EL が ExternalLinkFilter を満たし、x とリンクされる ExternalLink インスタンスのセットであるものとする。EL が空である時は、x を RE から削除する。

h) ExternalIdentifierFilter 要素が指定されない場合、または RE が空である場合、次に続ける。そうでない場合、x が RE における残りのレジストリエントリであるものとする。x がどれかの ExternalIdentifierFilter インスタンスとリンクされない場合は、x を RE から削除する。そうでない場合、次のように各 ExternalIdentifierFilter 要素を別個に取り扱う。

EI が ExternalIdentifierFilter を満たし、x とリンクされる ExternalIdentifier インスタンスのセットであるものとする。EI が空である時は、x を RE から削除する。

i) SoftFilter 要素が指定されない場合、または RE が空である場合は、以下を続行する。そうでない場合は、RE 内の残りのレジストリエントリに x がなるようにする。x が Slot インスタンスに関連付けられていない場合は RE から x を削除し、関連付けられている場合はそれぞれの SlotFilter 要素を以下のように個別に処理する。

SoftFilter を満たさせて x に関連付けられる Slot インスタンスのセットに SL がなるようにする。SL が空の場合は、RE から x を削除する。

j) HasAuditableEventBranch 要素が指定されない場合、または RE が空である場合は、以下を続行する。そうでない場合は、RE 内の残りのレジストリエントリに x がなるようにする。x が AuditableEvent インスタンスに関連付けられていない場合は、RE から x を削除し、関連付けられている場合はそれぞれの HasAuditableEventBranch 要素を以下のように個別に処理する。

AuditableEventsFilter が HasAuditableEventBranch 内で指定されない場合は、x に対するすべての AuditableEvent インスタンスのセットに AE がなるようにする。そうでない場合は、AuditableEventFilter を満たし、x に対する監査イベントである AuditableEvent インスタンスのセットに AE がなるようにする。AE が空である場合は、x を RE から削除する。UserFilter が HasAuditableEventBranch 内で指定されない場合は、AE の要素と関連付けられるすべての User インスタンスのセットに AI がなるようにする。そうでない場合は、UserFilter を満たして AE の要素と関連付けられる User インスタンスのセットに AI がなるようにする。AI が空である場合は、x を RE から削除する。OrganizationFilter が HasAuditableEventBranch 内で指定されない場合は、AI の要素と関連付けられるすべての Organization インスタンスのセットに OG がなるようにする。そうでない場合は、OrganizationFilter を満たして AI の要素と関連付けられる Organization インスタンスのセットに OG がなるようにする。OG が空である場合は、x を RE から削除する。

2. RE が空である時は、registry entry query result is empty（レジストリエントリクエリ結果は空です）という警告を発行する。

3. RegistryEntryQuery の結果として RE を返す。

例

あるクライアントは XYZ Corporation との間で取引関係を結ぶことを望み、XYZ Corporation がレジストリに取引文書のいずれかを登録しているか知りたいと考えているとする。次のクエリにより、名前に文字列「XYZ」が含まれる組織から申請された現在登録されている項目に対するレジストリエントリ識別子のセットが返される。このクエリでは、取替、置換、廃止、または撤回された項目のレジストリエントリ識別子は返されない。

```
<RegistryEntryQuery>
   <RegistryEntryFilter>
         status EQUAL "Approved"          -- code by Clause, Section 8.2.10
   </RegistryEntryFilter>
   <SubmittingOrganizationBranch>
      <OrganizationFilter>
            name CONTAINS "XYZ"           -- code by Clause, Section 8.2.10
      </OrganizationFilter>
   </SubmittingOrganizationBranch>
</RegistryEntryquery>
```

クライアントは United Nations Standard Product and Services Classification (UNSPSC) 分類スキームを用いており、「集積回路部品」、すなわち UNSPSC コード「321118」として分類される製品を取り扱う企業を全部識別したいと考える。クライアントは企業がその当事者プロファイル文書をレジストリに登録済みであること、また、プロファイルがそれぞれ企業の取り扱う製品別に分類されていることを知っている。次のクエリにより、集積回路部品を取り扱う企業のプロファイルに対するレジストリエントリ識別子のセットが返される。

```
<RegistryEntryQuery>
   <RegistryEntryFilter>
      objectType EQUAL "CPP" AND          -- code by Clause, Section 8.2.10
      status EQUAL "Approved"
   </RegistryEntryFilter>
   <HasClassificationBranch>
      <ClassificationNodeFilter>
         id STARTSWITH "urn:un:spsc:321118"   -- code by Clause, Section
                                                 8.2.10
      </ClassificationNodeFilter>
   <HasClassificationBranch>
</RegistryEntryQuery>
```

クライアントアプリケーションは 2 種類の分類スキームによって分類されるすべての項目を必要とする。一つは「Industry」に基づくスキーム、もう一つは「Geography」に基づくスキームである。どちらのスキームも ebXLM によって定義され、登録されている。各スキームのルートノードは、それぞれ「urn:ebxml:cs:industry」および「urn:ebxml:cs:geography」と指定される。次のクエリでは、「Industry/Automotive」および「Geography/Asia/Japan」によって分類される登録済みの全項目に対するレジストリエントリが識別される。

```
<RegistryEntryQuery>
   <HasClassificationBranch>
      <ClassificationNodeFilter>
         id STARTSWITH "urn:ebxml:cs:industry" AND
         path EQUAL "Industry/Automotive"       -- code by Clause, Section
                                                 8.2.10
      </ClassificationNodeFilter>
      <ClassificationNodeFilter>
         id STARTSWITH "urn:ebxml:cs:geography" AND
```

```
            path EQUAL "Geography/Asia/Japan"    -- code by Clause, Section
                                                 8.2.10
         </ClassificationNodeFilter>
      </HasClassificationBranch>
   </RegistryEntryQuery>
```

クライアントアプリケーションは、パッケージのメンバとして特定のレジストリエントリを持つレジストリ Package インスタンスをすべて識別しようとする。次のクエリでは、URN「urn:path:myitem」によって識別されるレジストリエントリをメンバとして含むレジストリパッケージがすべて識別される。

```
   <RegistryEntryQuery>
      <RegistryEntryFilter>
         objectType EQUAL "RegistryPackage"      -- code by Clause, Section
                                                 8.2.10
      </RegistryEntryFilter>
      <SourceAssociationBranch>
         <AssociationFilter>                     -- code by Clause, Section
                                                 8.2.10
            associationType EQUAL "HasMember" AND
            targetObject EQUAL "urn:path:myitem"
         </AssociationFilter>
      </SourceAssociationBranch>
   </RegistryEntryQuery>
```

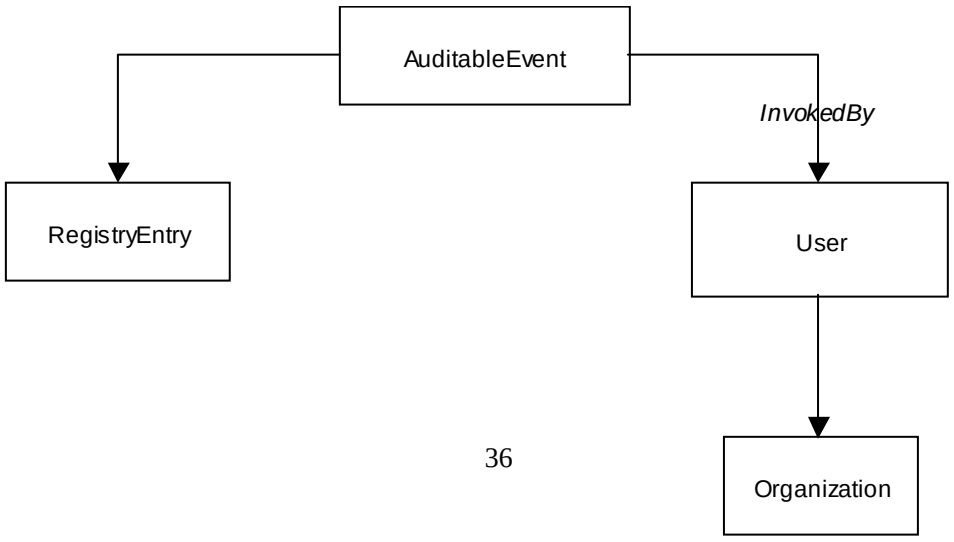
クライアントアプリケーションは、名前または記述の一部に特定のキーワードを含むすべての ClassificationNode インスタンスを識別しようとする。次のクエリでは、キーワード「transistor」を名前の一部もしくは記述の一部として含むすべてのレジストリ分類ノードが識別される。

```
   <RegistryEntryQuery>
      <RegistryEntryFilter>
         ObjectType="ClassificationNode" AND
         (name CONTAINS "transistor" OR      -- code by Clause, Section 8.2.10
         description CONTAINS "transistor")
      </RegistryEntryFilter>
   </RegistryEntryQuery>
```

8.2.3 AuditableEventQuery

目的

特定のレジストリメタデータに対するクエリの結果として監査対象イベントのセットを識



別すること。

ebRIM バインディング

定義

```
<!ELEMENT AuditableEventQuery
  (   AuditableEventFilter?,
      RegistryEntryQuery*,
      InvokedByBranch?     )>

<!ELEMENT InvokedByBranch
  (   UserFilter?,
      OrganizationQuery?   )>
```

意味情報規則

1. AE はレジストリ内の永続的 AuditableEvent イベントすべてのセットを示すものとする。次のステップにより、指定されたフィルタの条件を満たさない AE におけるインスタンスが取り除かれる。

a) AuditableEventFilter が指定されない場合、または AE が空である場合は、次に続ける。 そうでない場合、x が AE における監査対象イベントであるものとする。x が 8.2.9 節で定義する AuditableEventFilter を満たさない時は、x を AE から削除する。

b) RegistryEntryQuery 要素が指定されない場合、または AE が空である場合、次に続ける。そうでない場合、x が AE における残りの監査対象イベントであるものとする。それぞれの RegistryEntryQuery 要素は以下の通り別個に取り扱う。

RE は 8.2.2 節で定義する通り RegistryEntryQuery の結果セットであるものとする。x が RE 内のどれかのレジストリエントリに監査対象イベントでない場合は、x を AE から削除する。

c) InvokedByBranch 要素が指定されない場合、または AE が空である場合は、以下のステップを続行する。そうでない場合は、x が AE 内の残りの監査可能イベントになるようにする。

x を呼び出すユーザインスタンスが u であるとする。UserFilter 要素が InvokedByBranch で指定されていない場合、そして u がそのフィルタを満たさない場合は、AE から x を削除する。そうでない場合は以下を続行する。

OrganizationQuery が InvokedByBranch 内で指定されない場合は、以下を続行する。そうでない場合は、u の組織属性によって識別され、OrganizationQuery の結果のセットにある Organization インスタンスのセットに OG がなるようにする。OG が空である場合は、x を AE から削除する。

2. AE が空であるときは、auditable event query result is empty(監査対象イベントクエリ結果が空です)の警告を発行する。

3. AE を AuditableEventQuery の結果として返す。

例

レジストリクライアントが項目を登録し、その項目に URN 識別子"urn:path:myitem"が割り当てられた。クライアントはその項目に影響を及ぼした年初からのすべてのイベントを調べようとする。次のクエリにより、そのようなイベントすべてに対する AuditableEvent 識別子のセットが返される。

```
<AuditableEventquery>
   <AuditableEventFilter>
      timestamp GE "2001-01-01" AND         -- code by Clause, Section 8.2.10
      registryEntry EQUAL "urn:path:myitem"
   </AuditableEventFilter>
</AuditableEventQuery>
```

クライアント企業は多くのオブジェクトをレジストリに登録している。レジストリにより、他の組織から申請されたイベントは、自社で登録した項目、たとえば、新規の分類および新規の関係に影響を及ぼすことができる。次のクエリでは、他の当事者に呼び出され、責任ある組織“ myorg ”により申請された項目に影響するすべての検査可能イベントの識別子のセットが返される。

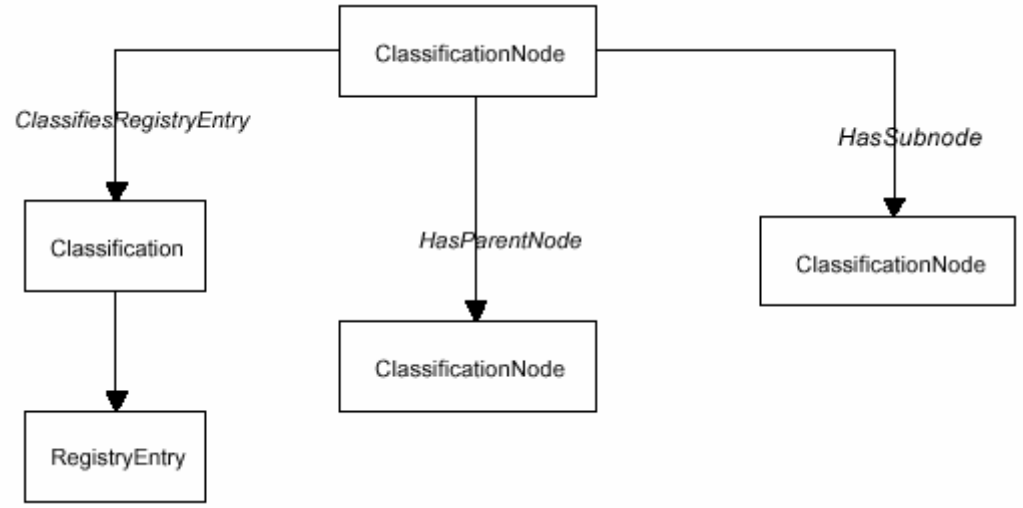
```
<AuditableEventQuery>
   <RegistryEntryQuery>
      <SubmittingOrganizationBranch>
         <OrganizationFilter>
            id EQUAL "urn:somepath:myorg"       -- code by Clause, Section
                                                8.2.10
         </OrganizationFilter>
      </SubmittingOrganizationBranch>
      <ResponsibleOrganizationBranch>
         <OrganizationFilter>
            id EQUAL "urn:somepath:myorg"       -- code by Clause, Section
                                                8.2.10
         </OrganizationFilter>
      </ResponsibleOrganizationBranch>
   </RegistryEntryQuery>
   <InvokedByBranch>
      <OrganizationQuery>
         <OrganizationFilter>
            id -EQUAL "urn:somepath:myorg"      -- code by Clause, Section
                                                8.2.10
         </OrganizationFilter>
      </OrganizationQuery>
   </InvokedByBranch>
</AuditableEventQuery>
```

8.2.4 ClassificationNodeQuery

目的

特定のレジストリメタデータに対するクエリの結果として、分類ノードインスタンスのセットを識別する。

ebRIM バインディング



定義

```
<!ELEMENT ClassificationNodeQuery
   (  ClassificationNodeFilter?,
      PermitsClassificationBranch*,
      HasParentNode?,
      HasSubnode*               )>

<!ELEMENT PermitsClassificationBranch
   (  ClassificationFilter?,
      RegistryEntryQuery?       )>

<!ELEMENT HasParentNode
   (  ClassificationNodeFilter?,
      HasParentNode?            )>

<!ELEMENT HasSubnode
   (  ClassificationNodeFilter?,
      HasSubnode*               )>
```

意味情報規則

1. CN はレジストリ内のすべての永続的 ClassificationNode インスタンスのセットを示すものとする。次のステップにより、CN 内にある、指定されたフィルタの条件を満たさないインスタンスが取り除かれる。

a) ClassificationNodeFilter が指定されない場合、または CN が空である場合は、次に続ける。そうでない場合、x が CN における分類ノードであるものとする。x が 8.2 節.9 で定義された ClassificationNodeFilter を満たさない場合、x を AE から削除する。

b) ClassifiesRegistryEntry 要素が指定されない場合、または CN が空である場合は、次に続ける。そうでない場合、x が CN における残りの分類ノードであるものとする。x がどれかの Classification インスタンスの対象オブジェクトでない場合、x を CN から削除する。そうでない場合、各 PermitsClassificationBranch インスタンスを次のように別個に取り扱う。

ClassificationFilter が PermitsClassificationBranch 要素で指定されない場合、CL が x を対象オ

ブジェクトとするすべての Classification インスタンスのセットであるものとする。そうでない場合、CL が ClassificationFilter を満たし、x を対象オブジェクトとする Classification インスタンスのセットであるものとする。CL が空である場合、x を CN から削除する。RegistryEntryQuery が PermitsClassificationBranch 要素で指定されない場合、RES が CL におけるいずれかの分類インスタンスのソースオブジェクトであるすべての RegistryEntry インスタンスのセットであるものとする。そうでない場合、RE は 8.2.2 節で定義される RegistryEntryQuery の結果セットであるものとし、RES は RE 内にあって CL におけるいずれかの分類のソースオブジェクトであるすべてのインスタンスのセットであるものとする。RES が空である場合は、x を CN から削除する。

c) HasParentNode 要素が指定されない場合、または CN が空である場合は、次に続ける。そうでない場合、x が cn における残りの分類ノードであるものとし、次の段落を n=x として実行する。

n は分類ノードインスタンスであるものとする。n に親ノードがない場合（すなわち、n がルートノードである場合）、x を CN から削除する。p は n の親ノードであるものとする。ClassificationNodeFilter 要素が HasParentNode に直接含まれ、p が ClassificationNodeFilter を満たさないときは、x を CN から削除する。

別の HasParentNode 要素がこの HasParentNode ノードに直接含まれる場合は、前の段落を n=p で繰り返す。

d) HasSubnode 要素が指定されない場合、または CN が空である場合は、次に続ける。そうでない場合、x が CN における残りの分類ノードであるものとする。x がどれかの ClassificationNode インスタンスの親ノードではないときは、x を CN から削除する。そうでない場合、それぞれの HasSubnode 要素を別個に取り扱い、次の段落を n=x で実行する。

n は分類ノードインスタンスであるものとする。ClassificationNodeFilter が HasSubnode 要素で指定されない場合は、CNC が n を親ノードとするすべての分類ノードのセットであるものとする。そうでない場合、CNC は ClassificationNodeFilter を満たし、n をその親ノードとするすべての分類ノードのセットであるものとする。CNC が空である場合は、x を CN から削除する。そうでない場合、y が CNC の要素であるものとし、次の段落を続ける。

HasSubnode 要素が終端である場合、すなわち、HasSubnode 要素に別の HasSubnode 要素が直接含まれない場合は、次に続ける。そうでない場合、前の段落を新規の HasSubnode 要素と n=y で繰り返す。

2. CN が空である場合は、classification node query result is empty（分類ノードクエリ結果は空です）の警告を発行する。

3. CN を ClassificationNodeQuery の結果として返す。

例

クライアントアプリケーションは、レジストリで定義された分類ノードのうち、ルートノードであって名前に語句"product code"または語句"product type"が含まれるすべての分類ノードを識別しようとする。注: 規約では、分類ノードに親がない（すなわち、ルートノードである）場合、そのインスタンスの親属性は null に設定され、0 長文字列によって直定数として表現される。

```
<ClassificationNodeQuery>
   <ClassificationNodeFilter>
      (name CONTAINS “product code” OR     -- code by Clause, Section 8.2.10
       name CONTAINS “product type”) AND
       parent EQUAL “”
   </ClassificationNodeFilter>
</ClassificationNodeQuery>
```

クライアントアプリケーションは分類スキーム階層の第 3 レベルにある分類ノードすべてを識別しようとする。クライアントには、ルートノードの URN 識別子が urn:ebxml:cs:myroot であることがわかっている。次のクエリでは、第 2 レベルの「myroot」にあるすべてのノード（すなわち、第 3 レベル全体）のノードがすべて識別される。

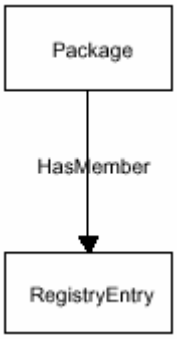
```
<ClassificationNodeQuery>
   <HasParentNode>
      <HasParentNode>
         <ClassificationNodeFilter>
            id EQ “urn:ebxml:cs:myroot”   -- code by Clause, Section 8.2.10
         </ClassificationNodeFilter>
      </HasParentNode>
   </HasParentNode>
</ClassificationNodeQuery>
```

8.2.5 RegistryPackageQuery

目的

特定のレジストリメタデータに対するクエリの結果としてレジストリパッケージインスタンスのセットを識別する。

ebRIM バインディング



定義

```
<!ELEMENT RegistryPackageQuery
  (   PackageFilter?,
      HasMemberBranch*     )>

<!ELEMENT HasMemberBranch
  (   RegistryEntryQuery?   )>
```

意味情報規則

1. RP はレジストリ内の永続的な Packages インスタンスすべてのセットを表すものとする。以下のステップにより、指定されたフィルタの条件を満たさない RP 内のインスタンスが取り除かれる。

a) PackageFilter が指定されない場合、または RP が空である場合、次に続ける。そうでない場合は、x が RP 内のパッケージインスタンスであるものとする。x が 8.2.9 節の定義に従って PackageFilter を満たさない場合は、x を RP から削除する。

b) HasMembeBranch 要素が RegistryPackageQuery に直接含まれない場合、または RP が空である場合は、次に続ける。そうでない場合は、x が RP 内の残りのパッケージインスタンスであるものとする。x が空のパッケージであるときは、x を RP から削除する。そうでないときは、次のように HasMembeBranch の各要素を別個に取り扱う。

RegistryEntryQuery 要素が PHasMembeBranch 要素に直接含まれない場合、PM がパッケージ x のメンバである RegistryEntry のすべてのインスタンスのセットであるものとする。そうでない場合は、RE が 8.2.2 節で定義されているように RegistryEntryQuery が返す RegistryEntry インスタンスのセットであるものとし、PM はパッケージ x のメンバである RE のサブセットであるものとする。PM が空の場合、x を RP から削除する。

2. RP が空である場合、registry package query result is empty（レジストリパッケージクエリ結果は空）の警告を発行する。

3. RP を RegistryPackageQuery の結果として返す。

例

クライアントアプリケーションは、レジストリ内の、外部オブジェクト Invoice をパッケージのメンバとして含むパッケージインスタンスをすべて識別しようとしている。

```
<RegistryPackageQuery>
   <HasMemberBranch>
      <RegistryEntryQuery>
         <RegistryEntryFilter>
            objectType EQ “Invoice”      -- code by Clause, Section 8.2.10
         </RegistryEntryFilter>
      </RegistryEntryQuery>
   </HasMemberBranch>
</RegistryPackageQuery>
```

クライアントアプリケーションは、レジストリ内の空でないパッケージインスタンスをすべて識別しようとしている。

```
<RegistryEntryQuery>
   <HasMemberBranch/>
</RegistryEntryQuery>
```

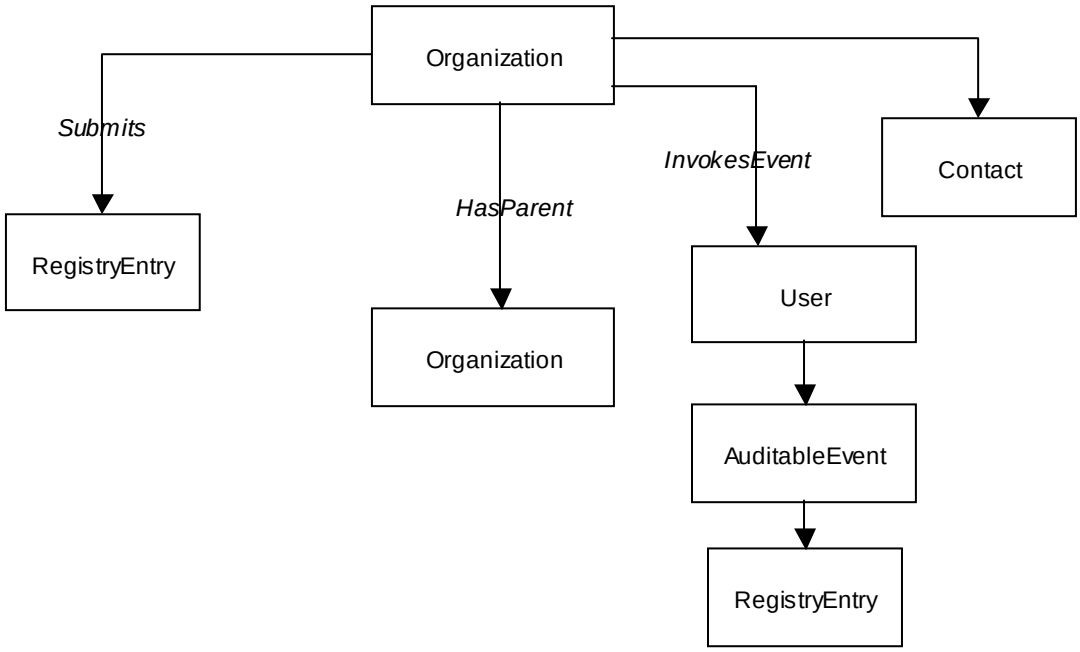
クライアントアプリケーションは、レジストリ内の空のパッケージインスタンスをすべて識別しようとしている。RegistryPackageQuery は否定を実行するように設定されていないため、クライアントは 2 種類の RegistryPackageQuery 要求を実行しなければならない。一つはすべてのパッケージを検索するもの、もう一つは空でないパッケージをすべて検索したうえで、その差のセットを作るものである。別の方法として、もっと複雑な RegistryEntryQuery

を実行して、パッケージとそのメンバのパッケージング関係が存在しないことをチェックできる。

注: レジストリパッケージはそのメンバとの関係によって完全に決定される固有 RegistryEntry インスタンスである。したがって、RegistryPackageQuery は常に適切な「Source」と「Target」の関係を使って同等の RegistryEntryQuery として再指定できる。ただし、同等の RegistryEntryQuery は記述が複雑になることが多い。

8.2.6 OrganizationQuery

目的



特定のレジストリメタデータに対するクエリの結果として組織インスタンスのセットを識別すること。

ebRIM バインディング

定義

```
<!ELEMENT OrganizationQuery
  (   OrganizationFilter?,
      SubmitsRegistryEntry*,
      HasParentOrganization?,
      InvokesEventBranch*,
      ContactFilter            )>

<!ELEMENT SubmitsRegistryEntry ( RegistryEntryQuery? )>

<!ELEMENT HasParentOrganization
  (   OrganizationFilter?,
      HasParentOrganization?   )>
```

```
<!ELEMENT InvokesEventBranch
  (   UserFilter?,
      AuditableEventFilter?,
      RegistryEntryQuery?      )>
```

意味情報規則

1. ORG はレジストリ内のすべての永続的 Organization インスタンスのセットを表すものとする。以下のステップにより、ORG 内で指定されたフィルタの条件を満たさないインスタンスが取り除かれる。

a) OrganizationFilter 要素が OrganizationQuery 要素に直接含まれない場合、または ORG が空である場合、次に続ける。そうでない場合、x が ORG 内の組織インスタンスであるものとする。x が 8.2.9 節の定義に従って OrganizationFilter を満たさない場合は、x を RP から削除する。

b) SubmitsRegistryEntry 要素が OrganizationQuery で指定されない場合、または ORG が空である場合は、次に続ける。そうでない場合、各 SubmitsRegistryEntry 要素を次のように別個に取り扱う。

RegistryEntryQuery が SubmitsRegistryEntry 要素で指定されない場合は、RES が組織 x によってレジストリに申請された RegistryEntry のすべてのインスタンスのセットであるものとする。そうでない場合、RE が 8.2.2 節で定義されている RegistryEntryQuery の結果であるものとし、RES が組織 x によりレジストリに申請された RE 内のすべてのインスタンスのセットであるものとする。RES が空であるときは、x を ORG から削除する。

c) HasParentOrganization 要素が OrganizationQuery で指定されない場合、または ORG が空である場合、次に続ける。そうでない場合、次の段落を o=x で実行する。

o は組織インスタンスとする。OrganizationFilter が HasParentOrganization で指定されず、o に親がない(すなわち、o が Organization 階層のルート組織である)場合、x を ORG から削除する。そうでない場合、p が o の親組織であるとする。p が OrganizationFilter を満たさないときは、x を ORG から削除する。

別の HasParentOrganization 要素がこの HasParentOrganization 要素に直接含まれる場合は、前の段落を o=p で繰り返す。

d) InvokesEvent 要素が OrganizationQuery で指定されない場合、または ORG が空である場合は、次に続ける。そうでない場合、各 InvokesEvent を次のように別個に取り扱う。

UserFilter が指定されず、x がどれかの AuditableEvent インスタンスの申請組織でない場合は、x を ORG から削除する。AuditableEventFilter が指定されない場合、AE が x を申請組織とするすべての AuditableEvent インスタンスのセットであるものとする。そうでない場合、AE は AuditableEventFilter を満たし、x を申請組織とする AuditableEvent インスタンスのセットであるものとする。AE が空であるときは、x を ORG から削除する。RegistryEntryQuery が InvokesEventBranch 要素で指定されない場合は、RES が AE 内のイベントと関係するすべての RegistryEntry インスタンスのセットであるものとする。そうでない場合、RE は 8.2.2 節で指定される RegistryEntryQuery の結果セットであるものとし、RES は RE のうち x により申請されたエントリのサブセットであるものとする。RES が空の場合は、x を ORG から削除する。

e) ContactFilter が OrganizationQuery 内で指定されない場合、または ORG が空である場合は、次に続ける。そうでない場合、各 ContactFilter を次のように別個に取り扱う。

CT が ContactFilter を満たし、組織 x の連絡先である Contact インスタンスのセットであるものとする。CT が空の場合、x を ORG から削除する。

2. ORG が空である場合は、organization query result is empt（組織クエリ結果が空）の警告を発行する。

3. ORG を OrganizationQuery の結果として返す。

4. 蓄積された警告や例外を OrganizationQuery に対する StatusResult として返す。

例

クライアントアプリケーションがフランスに本拠を置き、外部オブジェクト PartyProfile を今年申請した組織のセットを識別しようとしている。

```
<OrganizationQuery>
   <OrganizationFilter>
      country EQUAL “France”          -- code by Clause, Section 8.2.10
   </OrganizationFilter>
   <SubmitsRegistryEntry>
      <RegistryEntryQuery>
         <RegistryEntryFilter>
            objectType EQUAL “CPP”   -- code by Clause, Section 8.2.10
         </RegistryEntryFilter>
         <HasAuditableEventBranch>
            <AuditableEventFilter>
               timestamp GE “2001-01-01”   -- code by Clause, Section 8.2.10
            </AuditableEventFilter>
         </HasAuditableEventBranch>
      </RegistryEntryQuery>
   </SubmitsRegistryEntry>
</OrganizationQuery>
```

クライアントアプリケーションが XYZ, Corporation を親とするすべての組織のセットを識別しようとしている。クライアントでは、XYZ, Corp.の URN が urn:ebxml:org:xyz であることがわかっているが、XYZ の子会社が同じ書式の URL を用いている保証がないため、全体に対するクエリが要求される。

```
<OrganizationQuery>
   <HasParentOrganization>
      <OrganizationFilter>
         id EQUAL “urn:ebxml:org:xyz”      -- code by Clause, Section 8.2.10
      </OrganizationFilter>
   </HasParentOrganization>
</OrganizationQuery>
```

8.2.7 ReturnRegistryEntry

目的

RegistryEntryQuery によって識別されるレジストリエントリに関係する特定のレジストリメタデータを含む XML 文書を作成する。注:最初は RegistryEntryQuery を単一のレジストリ項目の URN 識別子とすることができる。

定義

```
<!ELEMENT ReturnRegistryEntry
  (   RegistryEntryQuery,
      WithClassifications?,
      WithSourceAssociations?,
      WithTargetAssociations?,
      WithAuditableEvents?,
      WithExternalLinks?              )>

<!ELEMENT WithClassifications  ( ClassificationFilter? )>
<!ELEMENT WithSourceAssociations  ( AssociationFilter? )>
<!ELEMENT WithTargetAssociations  ( AssociationFilter? )>
<!ELEMENT WithAuditableEvents  ( AuditableEventFilter? )>
<!ELEMENT WithExternalLinks  ( ExternalLinkFilter? )>

<!ELEMENT ReturnRegistryEntryResult
   (  RegistryEntryMetadata*)>

<!ELEMENT RegistryEntryMetadata
    ( RegistryEntry,
      Classification*,
      SourceAssociations?,
      TargetAssociations?,
      AuditableEvent*,
      ExternalLink*                        )>

<!ELEMENT SourceAssociations ( Association* )>
<!ELEMENT TargetAssociations ( Association* )>
```

意味情報規則

1. ReturnRegistryEntryResult に含まれる RegistryEntry、Classification、Association、AuditableEvent、および ExternalLink の要素は 付録 A で指定される ebXML Registry DTD で定義される。

2. RegistryEntryQuery を 8.2.2 節で指定の意味情報規則に従って実行し、R をレジストリエントリインスタンスに対する識別子の結果セットであるものとする。S は返される警告およびエラーのセットであるものとする。S にあるいずれかの要素がエラー条件である場合は、実行を中止し、ReturnRegistryEntryResult と共に警告とエラーの同じセットを返す。

3. R のセットが空である場合は、サブ要素 RegistryEntryMetadata を ReturnRegistryEntryResult で返さない。その代わりに、no resulting registry entry（レジストリエントリの結果なし）の警告を発行する。この警告を RegistryEntryQuery によって返されるエラーリストに追加し、この拡張エラーリストを ReturnRegistryEntryResult とともに返す。

4. R の要素が参照するレジストリエントリ E それぞれについて、E の属性を使って付録 A で定義されるように新規の RegistryEntry 要素を作成する。次に、前記の定義に従ってその RegistryEntry 要素の親要素とする新規の RegistryEntryMetadata 要素を作成する。

5. With オプションが指定されない場合、作成される RegistryEntryMetadata 要素には、サブ要素 Classification、SourceAssociations、TargetAssociations、AuditableEvent、または ExternalData が含まれない。RegistryEntryQuery からのエラーリストを含む RegistryEntryMetadata 要素のセットは ReturnRegistryEntryResult として返される。

6. WithClassifications が指定される場合は、R 内の E それぞれに対して次の処理を実行する。ClassificationFilter が存在しない場合、C は E とリンクされるいずれかの分類インスタンスであるものとする。そうでない場合、C は E とリンクされ、ClassificationFilter を満たす分類インスタンスであるものとする（8.2 節.9）。そのような C それぞれに対し、付録 A の定義に従って新規の Classification 要素を作成する。これらの Classification 要素をその親の RegistryEntryMetadata 要素に追加する。

7. WithSourceAssociations が指定される場合は、R 内の E それぞれについて、次の処理を実行する。AssociationFilter が存在しない場合は、A が E をソースオブジェクトとする関係インスタンスであるものとする。そうでない場合、A が AssociationFilter（8.2.9 節）を満たし、E をソースオブジェクトとする関係インスタンスであるものとする。そのような A それぞれに対して、付録 A の定義に従って新規の Association 要素を作成する。これらの Association 要素を WithSourceAssociations のサブ要素として追加し、その要素をその親の RegistryEntryMetadata 要素に追加する。

8. WithTargetAssociations が指定される場合は、R 内の E それぞれに対して、次の処理を実行する。AssociationFilter が存在しない場合は、A が E を対象オブジェクトとする関係インスタンスであるものとする。そうでない場合、A が AssociationFilter（8.2.9 節）を満たし、E を対象オブジェクトとする関係インスタンスであるものとする。そのような A それぞれに対して、付録 A の定義に従って新規の Association 要素を作成する。これらの Association 要素を WithTargetAssociations のサブ要素として追加し、その要素をその親の RegistryEntryMetadata 要素に追加する。

9. WithAuditableEvents が指定される場合は、R 内の E それぞれに対して、次の処理を実行する。AuditableEventFilter が存在しない場合は、A が E とリンクされるいずれかの監査対象イベントインスタンスであるものとする。そうでない場合、A は E とリンクされ、AuditableEventFilter（8.2 節.9）を満たすいずれかの監査対象イベントであるものとする。そのような A それぞれに対して、付録 A の定義に従って新規の AuditableEvent 要素を作成する。これらの AuditableEvent 要素をその親の RegistryEntryMetadata 要素に追加する。

10. WithExternalLinks が指定されない場合、R 内の E それぞれに対して、次の処理を実行する。ExternalLinkFilter が存在しない場合は、L が E とリンクされる外部リンクインスタンスであるものとする。そうでない場合、L は E とリンクされ、ExternalLinkFilter（8.2 節.9）を満たす外部リンクインスタンスであるものとする。そのような D それぞれに対し、付録 A で定義されるように新規の ExternalLink 要素を作成する。これらの ExternalLink 要素をその親の RegistryEntryMetadata 要素に追加する。

11. 警告またはエラー条件が発生するときは、コードとメッセージを RegistryEntryQueryResult として生成された RegistryResponse 要素に追加する。

12. RegistryEntryMetadata 要素のセットを ReturnRegistryEntryResult のコンテンツとして返す。

例

XYZ Corporation の顧客は XYZ が以前に登録した Purchase Order DTD を用いてきた。その URN 識別子は「urn:com:xyz:po:325」である。顧客は特にその DTD が廃止または置換される場合、その DTD の最新ステータスをチェックし、最新の分類をすべて取得したいと考える。次のクエリ要求により、既存の DTD に対するレジストリエントリをルートとして含み、その分類のすべてを含み、その DTD に優先する、またはそれに取って代わった項目に対するレジストリエントリとの関係を含む XML 文書が返される。

```
<ReturnRegistryEntry>
   <RegistryEntryQuery>
      <RegistryEntryFilter>
         id EQUAL "urn:com:xyz:po:325"       -- code by Clause, Section
                                             8.2.10
      </RegistryEntryFilter>
   </RegistryEntryQuery>
   <WithClassifications/>
   <WithSourceAssociations>
      <AssociationFilter>                    -- code by Clause, Section
                                             8.2.10
         associationType EQUAL "SupercededBy" OR
         associationType EQUAL "ReplacedBy"
      </AssociationFilter>
   </WithSourceAssociations>
</ReturnRegistryEntry>
```

レジストリのクライアントは数年前に XML DTD を登録したが、現在その DTD を改訂版に置き換えることを検討している。既存の DTD に対する識別子は「urn:xyz:dtd:po97」である。改訂版は既存の DTD とは完全に上位互換となっていない。クライアントは互換性のない変更の影響を評価できるようにするため、既存の DTD を使用する登録済みの項目すべての一覧が欲しいと考える。次のクエリにより、特定の DTD を使用するか、それを含む、あるいはそれを拡張する登録済みの項目を表すすべての RegistryEntry 要素の一覧である XML 文書が返される。また、本書は一覧にある各 RegistryEntry 要素を識別された関係に対する要素とリンクする。

```
<ReturnRegistryEntry>
      <RegistryEntryQuery>
         <SourceAssociationBranch>
            <AssociationFilter>              -- code by Clause, Section 8.2.10
               associationType EQUAL "Contains" OR
               associationType EQUAL "Uses" OR
               associationType EQUAL "Extends"
            </AssociationFilter>
            <RegistryEntryFilter>            -- code by Clause, Section 8.2.10
               id EQUAL "urn:xyz:dtd:po97"
            </RegistryEntryFilter>
         </SourceAssociationBranch>
      </RegistryEntryQuery>
      <WithSourceAssociations>
         <AssociationFilter>                  -- code by Clause, Section 8.2.10
            associationType EQUAL "Contains" OR
            associationType EQUAL "Uses" OR
            associationType EQUAL "Extends"
         </AssociationFilter>
      </WithSourceAssociations>
   </ReturnRegistryEntry>
```

ユーザはレジストリをブラウズしていて、ユーザの問題を解決するはずのコア構成要素のパッケージを記述するレジストリエントリを発見した。パッケージの URN 識別子は "urn:com:cc:pkg:ccstuff"である。そのため、ユーザはパッケージに何が入っているか知りたいと考える。次のクエリにより、パッケージの各メンバに対するレジストリエントリをメンバの Uses および HasMemberBranch の関係とともに含む XML 文書を返す。

```
<ReturnRegistryEntry>
   <RegistryEntryQuery>
      <TargetAssociationBranch>
         <AssociationFilter>           -- code by Clause, Section 8.2.10
             associationType EQUAL "HasMember"
         </AssociationFilter>
         <RegistryEntryFilter>         -- code by Clause, Section 8.2.10
            id EQUAL " urn:com:cc:pkg:ccstuff "
         </RegistryEntryFilter>
      </TargetAssociationBranch>
   </RegistryEntryQuery>
   <WithSourceAssociations>
      <AssociationFilter>               -- code by Clause, Section 8.2.10
         associationType EQUAL "HasMember" OR
         associationType EQUAL "Uses"
      </AssociationFilter>
   </WithSourceAssociations>
</ReturnRegistryEntry>
```

8.2.8 ReturnRepositoryItem

目的

所要のオブジェクトを保持するレジストリ/リポジトリに RegistryEntryQuery を申請することにより、1 つまたは複数のリポジトリ項目、およびいくつかの関係メタデータを含む XML 文書を作成する。注:最初は、RegistryEntryQuery は単一のレジストリエントリに対する URN 識別子とすることができる。

定義

```
<!ELEMENT ReturnRepositoryItem
(  RegistryEntryQuery,
   RecursiveAssociationOption?,
   WithDescription?              )>

<!ELEMENT RecursiveAssociationOption ( AssociationType+ )>
<!ATTLIST RecursiveAssociationOption
      depthLimit   CDATA   #IMPLIED   >

<!ELEMENT AssociationType EMPTY >
<!ATTLIST AssociationType
      role   CDATA   #REQUIRED  >

<!ELEMENT WithDescription EMPTY >

<!ELEMENT ReturnRepositoryItemResult
   ( RepositoryItem*)>

<!ELEMENT RepositoryItem
   (    ClassificationScheme
```

```
      | RegistryPackage
      | ExtrinsicObject
      | WithdrawnObject
      | ExternalLinkItem      )>
<!ATTLIST RepositoryItem
   identifier      CDATA   #REQUIRED
   name            CDATA   #REQUIRED
   contentURI      CDATA   #REQUIRED
   objectType      CDATA   #REQUIRED
   status          CDATA   #REQUIRED
   stability       CDATA   #REQUIRED
   description     CDATA   #IMPLIED   >

<!ELEMENT ExtrinsicObject  (#PCDATA) >
<!ATTLIST ExtrinsicObject
   byteEncoding    CDATA    "Base64"  >

<!ELEMENT WithdrawnObject EMPTY >

<!ELEMENT ExternalLinkItem EMPTY >
```

意味情報規則

1. RecursiveOption 要素が存在しない場合は、Limit=0 に設定する。RecursiveOption 要素が存在する場合、その depthLimit 属性を整数の直定数と解釈する。depthLimit 属性が存在しない場合は、Limit=-1 に設定する。0 の Limit は再帰が発生しないことを意味する。-1 の Limit は再帰が無限に発生することを意味する。depthLimit 値が存在しても、それが正の整数であると解釈できないときは、実行を中止し、例外 invalid depth ,ilit(無効な深さ限度)を発行する。そうでない場合、Limit=N に設定する。ここで、N はその正の整数である。限度 N は、プロセスがその限度よりも前に終了する場合を除き、ちょうど N 回の再帰ステップが実行されることを意味する。

2. Depth=0 に設定する。Result は ReturnRepositoryItemResult の一部として返される RepositoryItem 要素のセットを表すものとする。最初は Result が空である。意味情報規則 4 から 10 までが Result の内容を決定する。

3. WithDescription 要素が存在するときは、WSD="yes"に設定する。そうでない場合、WSD="no"に設定する。

4. 8.2.2 節の指定に従って RegistryEntryQuery を実行し、R はレジストリエントリインスタンスに対する識別子の結果セットであるものとする。S は返される警告およびエラーのセットであるものとする。S における要素がエラー条件である場合は実行を中止し、同じ警告およびエラーのセットを ReturnRepositoryItemResult と共に返す。

5. x を R から求められるレジストリ参照のセットとして意味情報規則 6 および 7 を実行する。これらの規則を実行したあとで、Depth が Limit と等しければ、Result の内容を RepositoryItem 要素のセットとして ReturnRepositoryItemResult 要素に返す。そうでない場合、意味情報規則 8 を続ける。

6. X が RegistryEntry インスタンスのセットであるものとする。X 内のレジストリエントリ E それぞれに対し、次の処理を実行する。

a) E.contentURL がこのレジストリ/リポジトリ内のリポジトリ項目を参照する場合は、意味

情報規則 7 の指定に従って求められたその属性値を使って新規の RepositoryItem 要素を作成する。

1) E.objectType="ClassificationScheme"のときは、参照される ClassificationScheme DTD をこの RepositoryItem のサブ要素として表す。[注: DTD 仕様が必要!]

2) E.objectType="RegistryPackage"のときは、参照される RegistryPackage DTD をこの RepositoryItem のサブ要素として表す。[注: DTD 仕様が必要!]

3) そうでない場合、すなわち、E が参照するオブジェクトに不明な内部構造がある場合、リポジトリ項目のコンテンツをこの RepositoryItem の新規の ExtrinsicObject サブ要素の#PCDATA として表す。

b) E.objectURL が他のレジストリ/リポジトリにある登録済みオブジェクトを参照するときは、意味情報規則 7 で指定されるように、求められるその属性の値を使って新規の RepositoryItem 要素を作成し、新規の ExternalLink 要素をこの RepositoryItem のサブ要素として作成する。

c) E.objectURL が無効である場合、すなわち、E.objectURL が参照するオブジェクトが取り消されている場合、意味情報規則 7 で指定されるように、求められるその属性の値を使って新規の RepositoryItem 要素を作成し、新規の WithdrawnObject 要素をこの RepositoryItem のサブ要素として作成する。

7. E はレジストリエントリとし、RO は意味情報規則 6 で作成された RepositoryItem 要素であるものとする。RO の属性を E の該当する属性から求められる値に設定する。WSD="yes" のときは、記述属性の値を含める。そうでない場合、それを含めない。この新規の RepositoryItemyouso 要素を Result セットに挿入する。

8. R は意味情報規則 3 で定義される通りであるとする。Y を R によって参照される RegistryEntry インスタンスのセットとして意味情報規則 9 を実行する。次に意味情報規則 10 を続行する。

9. Y は RegistryEntry インスタンスへの参照のセットであるとする。NextLevel は RegistryEntry インスタンスの空のセットであるものとする。Y におけるレジストリエントリ E それぞれに対して、また RecursiveAssociationOption の AssociationType A それぞれに対して、次の処理を実行する。

a) Z は E をソースオブジェクト、E'を対象オブジェクト、そして A を AssociationType とする関係インスタンスにより E とリンクされる対象項目 E'のセットであるものとする。

b) Z の要素を NextLevel に追加する。

10. X は NextLevel にあるものの、まだ結果セットで表現されない新規レジストリエントリのセットであるものとする。

事例:

a) X が空のときは、Result の内容を RepositoryItem 要素のセットとして ReturnRepositoryItemResult 要素に返す。

b) X が空のときは､x を入力セットとして意味情報規則 6 および 7 を実行する｡終了したら、X の要素を Y に追加し、Depth=Depth+1 に設定する。ここで Depth が Limit と等しい場合、Result の内容を RepositoryItem 要素のセットとして ReturnRepositoryItemResult 要素に返す。そうでない場合、レジストリエントリの新規セット Y を使って意味情報規則 9 および 10 を繰り返す。

11. 上記の処理の実行時に例外、警告、またはその他のステータス条件が発生する場合は、RegistryResult の該当する RegistryError 要素を意味情報規則 5 または意味情報規則 10 で作成された ReturnRepositoryItemResult 要素と関連して返す。

例

レジストリクライアントがコア構成要素項目に対するレジストリエントリを発見した。その項目の URN 識別子は「urn:ebxml:cc:goodthing」である。しかし、「goodthing」は他の登録済み項目を多数使用する複合項目である。クライアントは「goodthing」の完全な実装に必要な全項目のコレクションを望む。次のクエリにより、必要な項目すべてのコレクションである XML 文書が返される。

```
<ReturnRepositoryItem>
   <RegistryEntryQuery>
      <RegistryEntryFilter>              -- code by Clause, Section 8.2.10
         id EQUAL "urn:ebxml:cc:goodthing"
      </RegistryEntryFilter>
   </RegistryEntryQuery>
   <RecursiveAssociationOption>
      <AssociationType  role="Uses"  />
      <AssociationType  role="ValidatesTo"  />
   </RecursiveAssociationOption>
</ReturnRepositoryItem>
```

レジストリクライアントが特定のビジネスプロセスを実装するコア構成要素ルーチン（「urn:ebxml:cc:rtn:nice87」）への参照を発見した。クライアントでは、すべてのルーチンにその定義元である UML 仕様との必要な関係があることがわかっている、以下のクエリにより、ルーチンとその UML 仕様が単一の XML 文書内の 2 つの項目のコレクションとして返される。

```
<ReturnRepositoryItem>
   <RegistryEntryQuery>
      <RegistryEntryFilter>              -- code by Clause, Section 8.2.10
         id EQUAL "urn:ebxml:cc:rtn:nice87"
      </RegistryEntryFilter>
   </RegistryEntryQuery>
   <RecursiveAssociationOption   depthLimit="1" >
      <AssociationType  role="ValidatesTo"  />
   </RecursiveAssociationOption>
</ReturnRepositoryItem>
```

あるユーザは北米産業分類システム(NAICS)の 1997 年版が URN 識別子「urn:nist:cs:naics-1997」でレジストリに格納されているという話を聞いた。次のクエリにより、1810 のノードをすべて含む完全な分類スキームが分類スキーム DTD に適合する XML 文書として検索される。

```
<ReturnRepositoryItem>
```

```
      <RegistryEntryQuery>
         <RegistryEntryFilter>              -- code by Clause, Section 8.2.10
            id EQUAL "urn:nist:cs:naics-1997"
         </RegistryEntryFilter>
      </RegistryEntryQuery>
   </ReturnRepositoryItem>
```

注: ReturnRepositoryItemResult には、以下の URL により決定されるコンテンツの ClassificationScheme 文書で構成される単一の RepositoryItem が含まれる。
ftp://xsun.sdct.itl.nist.gov/regrep/scheme/naics.txt

8.2.9 レジストリ・フィルタ

目的

特定のレジストリクラスのすべての永続的インスタンスのセットのサブセットを識別する。

定義

```
<!ELEMENT ObjectFilter ( Clause )>

<!ELEMENT RegistryEntryFilter ( Clause )>

<!ELEMENT IntrinsicObjectFilter ( Clause )>

<!ELEMENT ExtrinsicObjectFilter ( Clause )>

<!ELEMENT PackageFilter ( Clause )>

<!ELEMENT OrganizationFilter ( Clause )>

<!ELEMENT ContactFilter ( Clause )>

<!ELEMENT ClassificationNodeFilter ( Clause )>

<!ELEMENT AssociationFilter ( Clause )>

<!ELEMENT ClassificationFilter ( Clause )>

<!ELEMENT ExternalLinkFilter ( Clause )>

<!ELEMENT ExternalIdentifierFilter ( Clause )>

<!ELEMENT SlotFilter ( Clause )>

<!ELEMENT AuditableEventFilter ( Clause )>

<!ELEMENT UserFilter ( Clause )>
```

意味情報規則

1. Clause 要素は 8.2.10 節の Clause で定義される。

2. ObjectFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される RegistryObject UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 object attribute error(オブジェクト属性エラー)を発行する。ObjectFilter は属性

値が Clause の述語に対して真である RegistryObject インスタンスの識別子のセットを返す。

3. RegistryEntryFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される RegistryEntry UML クラスの public 属性を識別するものとする。

これに該当しない場合、例外 registry entry attribute error（レジストリエントリ属性エラー）を発行する。RegistryEntryFilter は、属性値が Clause の述語に対して真となる RegistryEntry インスタンスの識別子のセットを返す。

4. IntrinsicObjectFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される IntrinsicObject UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 intrinsic object attribute error（内部オブジェクト属性エラー）を発行する。IntrinsicObjectFilter は、属性値が Clause の述語に対して真となる IntrinsicObject インスタンスの識別子のセットを返す。

5. ExtrinsicObjectFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される ExtrinsicObject UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 extrinsic object attribute error（外部オブジェクト属性エラー）を発行する。ExtrinsicObjectFilter は、属性値が Clause の述語に対して真となる ExtrinsicObject インスタンスの識別子のセットを返す。

6. PackageFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される Package UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 package attribute error（パッケージ属性エラー）を発行する。PackageFilter は属性値が Clause の述語に対して真となる Package インスタンスの識別子のセットを返す。

7. OrganizationFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される Organization または PostalAddress UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 organization attribute error（組織属性エラー）を発行する。OrganizationFilter は、属性値が Clause の述語に対して真となる Organization インスタンスの識別子のセットを返す。

8. ContactFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される Contact または PostalAddress UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 contact attribute error（連絡先属性エラー）を発行する。ContactFilter は、属性値が Clause の述語に対して真となる Contact インスタンスの識別子のセットを返す。

9. ClassificationNodeFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される ClassificationNode UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 classification node attribute error（分類ノード属性エラー）を発行する。ClassificationNodeFilter は、属性値が Clause の述語に対して真となる ClassificationNode インスタンスの識別子のセットを返す。

10. AssociationFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される Association UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 association attribute error（関係属性エラー）を発行する。AssociationFilter は、属性値が Clause の述語に対して真となる Association インスタンスの識別子のセットを返す。

11. ClassificationFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は

[ebRIM]で定義される Classification UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 classification attribute error（分類属性エラー）を発行する。ClassificationFilter は、属性値が Clause の述語に対して真となる Classification インスタンスの識別子のセットを返す。

12. ExternalLinkFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される ExternalLink UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 external link attribute error（外部リンク属性エラー）を発行する。ExternalLinkFilter は、属性値が Clause の述語に対して真となる ExternalLink インスタンスの識別子のセットを返す。

13. ExternalIdentifierFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は、[ebRIM]で定義された ExternalIdentifier UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 external identifier attribute error（外部識別子属性エラー）を発行する。ExternalIdentifierFilter は、属性値が Clause の述語に対して真となる ExternalIdentifier インスタンスの識別子のセットを返す。

14. SlotFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は、[ebRIM]で定義された Slot UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 slot attribute　error（スロット属性エラー）を発行する。SlotFilter は、属性値が Clause の述語に対して真となる Slot インスタンスの識別子のセットを返す。

15. AuditableEventFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される AuditableEvent UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 auditable event attribute error（監査対象イベント属性エラー）を発行する。AuditableEventFilter は、属性値が Clause の述語に対して真となる AuditableEvent インスタンスの識別子のセットを返す。

16. UserFilter XML 要素ごとに、これを含む SimpleClause の leftArgument 属性は[ebRIM]で定義される User UML クラスの public 属性を識別するものとする。これに該当しない場合、例外 auditable identity attribute error（監査対象 ID 属性エラー）を発行する。UserFilter は、属性値が Clause の述語に対して真となる User インスタンスの識別子のセットを返す。

例

以下は、その objectType 属性が「CPP」であり、かつそのステータス属性が「Approved」である RegistryEntry のインスタンスのセットを返すための RegistryEntryFilter の Clause 拡張と結合された RegistryEntryQuery の完全な例である。

```
<RegistryEntryQuery>
   <RegistryEntryFilter>
       <Clause>
          <CompoundClause   connectivePredicate="And" >
             <Clause>
                <SimpleClause  leftArgument="objectType" >
           <StringClause  stringPredicate="equal" >CPP</StringClause>
                </SimpleClause>
             </Clause>
             <Clause>
                <SimpleClause  leftArgument="status" >
           <StringClause  stringPredicate="equal"
                                        >Approved</StringClause>
```

```
                </SimpleClause>
              </Clause>
            </CompoundClause>
          </Clause>
      </RegistryEntryFilter>
  </RegistryEntryQuery>
```

8.2.10 XML Clause の制約の表現

目的

単純な XML FilterQuery は述語節に基づく正式な XML 構造を利用する。述語節は制約機構を正式に定義するために利用され、本仕様書では単に Clauses と言う。

UML の概念図

Clause の基本構造を概説する概念図は以下の通りである。これは図式化のために UML で表現されている。

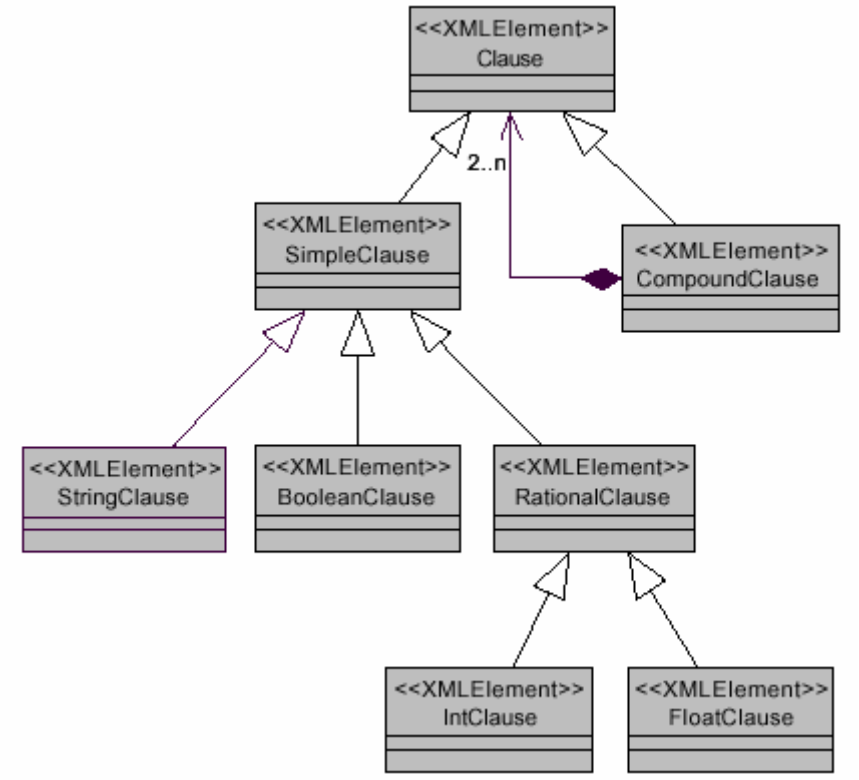


図 20: Clause の基本構造

意味情報規則

述語と引数は「LeftArgument - 述語 – RightArgument」の書式に結合され、Ckause を形成する。Clauses には SimpleClauses と CompoundClauses の次の 2 種類がある。

SimpleClauses

SimpleClause は常に leftArgument をテキスト文字列として定義する。この文字列は文節の主

語と呼ばれることもある。SimpleClause 自体は不完全（抽象的）なものであり、拡張しなければならない。SimpleClause は BooleanClause、StringClause、および RationalClause (abstract) をサポートするために拡張される。

BooleanClause は暗黙に右引数をブールとして述語を「equal to」として定義する。StringClause は、述語を該当する文字列と比較演算の列挙属性として、右引数を要素のテキストデータとして定義する。有理数のサポートは該当する有理数比較演算の列挙を提供する共通の RationalClause によって提供され、RationalClause はさらにそれぞれ右引数に適切な署名のある IntClause と FloatClause に拡張される。

CompoundClauses

CompoundClause には、2 つ以上の Clauses（Simple または Compound）および連結述語が含まれる。これにより任意の複雑な Clauses を構成できる。

定義

```
<!ELEMENT Clause ( SimpleClause | CompoundClause )>

<!ELEMENT SimpleClause
   ( BooleanClause | RationalClause | StringClause )>
<!ATTLIST SimpleClause
   leftArgument  CDATA #REQUIRED >

<!ELEMENT CompoundClause ( Clause, Clause+ )>
<!ATTLIST CompoundClause
   connectivePredicate ( And | Or ) #REQUIRED>

<!ELEMENT BooleanClause EMPTY >
<!ATTLIST BooleanClause
   booleanPredicate ( True | False ) #REQUIRED>

<!ELEMENT RationalClause ( IntClause | FloatClause )>
<!ATTLIST RationalClause
   logicalPredicate ( LE | LT | GE | GT | EQ | NE ) #REQUIRED >

<!ELEMENT IntClause ( #PCDATA )
<!ATTLIST IntClause
   e-dtype NMTOKEN #FIXED 'int' >

<!ELEMENT FloatClause ( #PCDATA )>
<!ATTLIST FloatClause
   e-dtype NMTOKEN #FIXED 'float' >

<!ELEMENT StringClause ( #PCDATA )>
<!ATTLIST StringClause
   stringPredicate
       ( contains | -contains |
         startswith | -startswith |
         equal | -equal
         endswith | -endswith ) #REQUIRED >
```

例

Simple BooleanClause:"Smoker"=True

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
  <SimpleClause leftArgument="Smoker">
    <BooleanClause booleanPredicate="True"/>
  </SimpleClause>
</Clause>
```

Simple StringClause:"Smoker" contains "mo"

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
  <SimpleClause leftArgument="Smoker">
    <StringClause stringcomparepredicate="contains">
      mo
    </StringClause>
  </SimpleClause>
</Clause>
```

Simple IntClause: "Age" >= 7

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
  <SimpleClause leftArgument="Age">
    <RationalClause logicalPredicate="GE">
      <IntClause e-dtype="int">7</IntClause>
    </RationalClause>
  </SimpleClause>
</Clause>
```

Simple FloatClause: "Size" = 4.3

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
  <SimpleClause leftArgument="Size">
    <RationalClause logicalPredicate="E">
      <FloatClause e-dtype="float">4.3</FloatClause>
    </RationalClause>
  </SimpleClause>
</Clause>
```

2 つの SimpleClause を持つ CompoundClause(("Smoker" = False)AND("Age" =< 45))

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
  <CompoundClause connectivePredicate="And">
    <Clause>
      <SimpleClause leftArgument="Smoker">
        <BooleanClause booleanPredicate="False"/>
      </SimpleClause>
    </Clause>
    <Clause>
      <SimpleClause leftArgument="Age">
```

```
          <RationalClause logicalPredicate="EL">
            <IntClause e-dtype="int">45</IntClause>
          </RationalClause>
        </SimpleClause>
      </Clause>
    </CompoundClause>
  </Clause>
```

1 つの SimpleClause と 1 つの CoumpoundClause を持つ CompoundClause
(("Smoker" = False)And(("Age" =< 45)Or("American"=True)))

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
  <CompoundClause connectivePredicate="And">
    <Clause>
      <SimpleClause leftArgument="Smoker">
        <BooleanClause booleanPredicate="False"/>
      </SimpleClause>
    </Clause>
    <Clause>
      <CompoundClause connectivePredicate="Or">
        <Clause>
          <SimpleClause leftArgument="Age">
            <RationalClause logicalPredicate="EL">
              <IntClause e-dtype="int">45</IntClause>
            </RationalClause>
          </SimpleClause>
        </Clause>
        <Clause>
          <SimpleClause leftArgument="American">
            <BooleanClause booleanPredicate="True"/>
          </SimpleClause>
        </Clause>
      </CompoundClause>
    </Clause>
  </CompoundClause>
</Clause>
```

8.3 SQL クエリのサポート

レジストリは、より複雑なクエリ機能を要求するレジストリクライアントのために設計された SQL ベースのクエリ機能をオプションでサポートできる。AdhocQueryRequest 内のオプションの SQLQuery 要素により、クライアントは宣言型クエリ言語を使って複雑な SQL クエリを申請できる。

レジストリの SQLQuery で使用する構文は、ISO/IEC 9075:1992, Database Language SQL [SQL] によって定義され、ISO/IEC 9075-4 [SQL-PSM]で指定される<sql invoked routines>（stored procedures とも言う）、および付録 C.3 でテンプレート書式で定義される事前定義済みのルーチンを加えて拡張された Entry レベルの"SELECT"文の適切なサブセットの様式化された使用によって定義される。レジストリクエリ言語の正確な構文については、C.1 の BNF 文法によって定義される。

SQLQuery のための SQL 構文のサブセットを使用することは、レジストリの実装にリレーショナルデータベースを使用する要件を意味するものではないことに注意が必要である。

8.3.1 SQL クエリ構文の[ebRIM]とのバインディング

SQL クエリは、付録 C.1 のクエリ構文と、付録 C.3 で定義される固定関係スキーマに基づいて定義される。関係スキーマは以下の節で説明する[RIM]へのアルゴリズムバインディングである。

8.3.1.1 インタフェースとクラスのバインディング

[ebRIM]で定義されるインタフェースとクラス名のサブセットは、SQL クエリによって照会できるテーブル名にマッピングされる。付録 C.3 は SQL クエリによって照会できる RIM インタフェースおよびクラスの名前を定義する。

付録 C.3 のテーブル定義に対する[ebRIM]クラスのバインディングを定義するために使用されるアルゴリズムは以下の通りである。

・具体的なインスタンスのあるクラスとインタフェースのみが関係テーブルにマッピングされる。これにより、SQL テーブルにマッピングされない RegistryObject および IntrinsicObject など、継承階層に中間インタフェースが作られる。この規則の例外は次に定義される RegistryEntry である。

・SQL クエリを RegistryEntry インスタンスに対して実行できるようにするため、RegistryEntry と呼ばれる特別なビューが定義される。これは[ebRIM]で定義されているもののうち、具体的なインスタンスがないが SQL クエリによって照会可能な唯一のインタフェースである。

・関係テーブルの名前は該当する[ebRIM]のクラスまたはインタフェースの名前と同じである。ただし、名前のバインディングは大文字と小文字を区別しない。

・付録 C.3 のテーブルにマッピングする[ebRIM]のクラスまたはインタフェースには、それぞれ付録 C.3 のカラム定義が含まれる。この場合、カラム定義は[ebRIM]内のそのクラスまたはインタフェースに対して定義された属性のサブセットに基づく。カラムにマッピングする属性には、[ebRIM]クラスまたはインタフェースの継承属性が含まれる。付録 C.3 のコメントは、どの先祖クラスまたはインタフェースがどのカラム定義のもとになっているのかを示す。

付録 C.3 で定義されていない SQLQuery は、エラー条件 InvalidQueryException を発生させることができる。

SQL カラム定義に対する[ebRIM]のマッピング属性のアルゴリズムは、以下の節で説明する。

8.3.1.2 属性バインディングへのアクセス方法

[ebRIM]インタフェースメソッドのほとんどは、属性に直接マッピングする単純な get メソッドである。たとえば、RegistryObject に対する getName メソッドは String 型の名前属性にマッピングする。[ebRIM]による get メソッドはそれぞれ、それが[ebRIM]におけるインタフェース定義でマッピングする正確な属性名を定義する。

8.3.1.3 基本属性バインディング

[ebRIM]によって定義される基本型（たとえば、String）の属性は、SQL ではカラム名と同じように使用できる。この場合も、正確な属性名は[ebRIM]のインタフェース定義で定義さ

れる。名前は大文字と小文字が混在するが、SQL-92 は大文字と小文字を区別しないことに注意が必要である。したがって、クエリに[ebRIM]で定義される大文字/小文字の別と完全には一致しない属性名が含まれていても有効である。

8.3.1.4 参照属性バインディング

[ebRIM]インタフェースメソッドのいくつかは[ebRIM]によって定義されるインタフェースまたはクラスのインスタンスへの参照を返す。たとえば、RegistryObject クラスの getAccessControlPolicy メソッドは AccessControlPolicy オブジェクトのインスタンスへの参照を返す。

そのような場合、参照は参照されるオブジェクトの id 属性にマッピングする。作成されるカラムの名前は 8.3.1.3 節で定義した[ebRIM]内の属性名と同じである。カラムのデータ型は付録 C.3 で定義されるように UUID である。

参照属性が null 参照を保持する場合、その参照属性は SQL バインディングで null 値を返す。これは[SQL]によって定義されるように<null specification>を使って検査できる。

参照属性バインディングは基本属性マッピングの特別なケースである。

8.3.1.5 複合属性バインディング

[ebRIM]インタフェースのいくつかは基本型でない属性を定義する。逆に、これらの[ebRIM]インタフェースは[RIM]におけるエンティティクラスによって定義される複合型のものである。その例には、インタフェース Organization およびクラス Contact における型 TelephoneNumber、Contact、PersonName などの属性が含まれる。

SQL クエリスキーマは、親テーブル内の多重基本属性などの複合属性をアルゴリズム的にマッピングする。マッピングは親テーブル内のエンティティクラス属性を平坦化する。平坦化された属性の属性名は参照チェーン内の属性名の連結から成る。たとえば、Organization には Contact 型の連絡先属性がある。Contact には型 PostalAddress の住所属性がある。PostalAddress には city という名前の String 属性がある。この city 属性は contact_address_city と命名される。

8.3.1.6 コレクション属性バインディング

[ebRIM]インタフェースメソッドのいくつかは、[ebRIM]によって定義されるインタフェースまたはクラスのインスタンスに対する参照のコレクションを返す。たとえば、ManagedObject クラスの getPackages メソッドは、オブジェクトがメンバとして含まれる Packages のインスタンスへの参照のコレクションを返す。

[ebRIM]クラスにおけるそのようなコレクション属性は付録 C.3 にあるストアドプロシージャにマッピングされているため、これらのストアドプロシージャは id 属性値のコレクションを返す。これらのストアドプロシージャの返り値は、SQL におけるテーブルサブクエリの結果として取り扱うことができる。

これらのストアドプロシージャは、SQL IN 文節の右側として使用し、前記の参照のコレクションにオブジェクトがメンバとして含まれているかどうかを検査できる。

8.3.2 クエリ構文に対する意味条件

本節では、クエリ構文に対して BNF で表現できないクエリ構文に対する単純化制約を定義する。これらの制約はクエリの意味情報分析で適用しなければならない。

1. クラス名と属性名は大文字と小文字の区別に関係なく処理しなければならない。

2. ストアドプロシージャの呼び出しに使用する構文は、ISO/IEC 9075- 4 [SQL/PSM]によって指定されるように、SQL プロシージャの呼び出しの構文と整合性がなければならない。

3. 仕様の本バージョンでは、SQL select カラムリストは 1 つだけのカラムから構成され、必ず t.id としなければならない。この場合、t は FROM 文節にあるテーブル参照である。

8.3.3 SQL クエリの結果

SQL クエリの結果は、8.4 節の AdHocQueryResponse によって定義されるように、常に ObjectRefList である。すなわち、SQL クエリの結果は常に[ebRIM]における RegistryObject インタフェースのサブクラスのインスタンスへの参照のコレクションである。これは意味条件で反映されており、意味条件では、指定される SQLselect カラムが本バージョンの仕様では付録 C.3 のテーブルにある id カラムでなければならないことが要求される。

8.3.4 シンプルなメタデータ基準クエリ

SQL クエリの最も単純な形式は、[ebRIM]内の単一クラスに対して指定されたメタデータ属性を基本とする。本節では単純なメタデータを基本とするクエリの例をいくつか示す。

たとえば、名前にワード'Acme' が含まれ、バージョンが 1.3 より大きい ExtrinsicObjects のコレクションを取得するためには、以下のクエリ述語をサポートしなければならない。

```
SELECT id FROM ExtrinsicObject  WHERE name LIKE ‘%Acme%’ AND
       majorVersion >= 1 AND
       (majorVersion >= 2 OR  minorVersion > 3);
```

クエリ構文では、前記の例で示すように、より単純な述語を結合して複雑なクエリにできる。

8.3.5 RegistryEntry クエリ

RegistryEntry インタフェースは ebRIM で中心的な役割を果たすため、SQL クエリのスキーマは、実際の具体的な型やテーブル名とは関係なくすべての RegistryEntry インスタンスに対する多様型クエリの実行が可能な RegistryEntry と呼ばれる特別なビューを定義する。

次の例は 8.3.4 節と同じであるが、ExtrinsicObject インスタンスのみでなくすべての RegistryEntry インスタンスに対して適用される点が異なる。結果セットには、名前にワード 'Acme'が含まれ、バージョンが 1.3 よりも大きい、適合するすべての RegistryEntry インスタンスの id が含まれる。

```
SELECT id FROM RegistryEntry WHERE name LIKE ‘%Acme%’ AND
       objectType = ‘ExtrinsicObject’ AND
       majorVersion >= 1 AND
       (majorVersion >= 2 OR  minorVersion > 3);
```

8.3.6 分類クエリ

本節では、サポートしなければならない分類関連の各種クエリを説明する。

8.3.6.1 ClassificationNodes の識別

[ebRIM]内のすべてのオブジェクトのように、ClassificationNodes はそれぞれの ID によって識別される。しかし、ClassificationNodes は、その ClassificationNode を含む ClassificationNode ツリーを表現する XML 文書におけるルート分類ノードから指定された分類ノードへの XPATH 表現[XPT]を指定するパス属性によっても識別できる。

8.3.6.2 ルート分類ノードの取得

ルート ClassificationNodes のコレクションを取得するには、次のクエリ述語をサポートしなければならない。

```
SELECT cn.id FROM ClassificationNode cn WHERE parent IS NULL
```

前記のクエリは親属性が null に設定されている ClassificationNodes をすべて返す。特定のルート ClassificationNode が必要なときは、前記のクエリで名前に対する述語を指定することもできることに注意が必要である。

8.3.6.3 指定 ClassificationNode の子の取得

そのノードの ID を割り当てられた ClassificationNode の子を取得するには、次のようなスタイルのクエリをサポートしなければならない。

```
SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
```

前記のクエリは ID によって指定されたノードをその親属性とする ClassificationNodes をすべて返す。

8.3.6.4 ClassificationNode によって分類されたオブジェクトの取得

指定された ClassificationNodes によって分類される ExtrinsicObjects のコレクションを取得するには、次のようなスタイルのクエリをサポートしなければならない。

```
SELECT id FROM ExtrinsicObject
WHERE
   id IN (SELECT classifiedObject FROM Classification
          WHERE
               classificationNode IN (SELECT id FROM ClassificationNode
                                      WHERE path = '/Geography/Asia/Japan'))
  AND
   id IN (SELECT classifiedObject FROM Classification
          WHERE
               classificationNode IN (SELECT id FROM ClassificationNode
                                      WHERE path = '/Industry/Automotive'))
```

前記のクエリにより、Automotive Industry および Japan Geography によって分類される ExtrinsicObjects のコレクションが取得される。GetClassifiedObjectsRequest に対して定義される意味情報に従って、クエリには指定された ClassificationNodes の派生によって分類されるオブジェクトも含まれることに注意が必要である。

8.3.6.5 オブジェクトを分類する ClassificationNodes の取得

指定された Object を分類する ClassificationNodes のコレクションを取得するには、次のスタ

イルのクエリをサポートしなければならない。

```
SELECT id FROM ClassificationNode
       WHERE id IN (RegistryEntry_classificationNodes(<id>))
```

8.3.7 関係クエリ

本節では、サポートしなければならない Association 関連の各種クエリについて説明する。

8.3.7.1 ソースとしての指定オブジェクトとのすべての関係の取得

指定されたオブジェクトをソースとする Associations のコレクションを取得するには、次のクエリをサポートしなければならない。

```
SELECT id FROM Association WHERE sourceObject = <id>
```

8.3.7.2 ターゲットとしての指定オブジェクトとのすべての関係の取得

指定されたオブジェクトをそのターゲットとする Associations のコレクションを取得するには、次のクエリをサポートしなければならない。

```
SELECT id FROM Association WHERE targetObject = <id>
```

8.3.7.3 関係属性に基づく結合オブジェクトの取得

指定された Association 属性を持つ Associations のコレクションを取得するには、次のクエリをサポートしなければならない。

指定された名前を持つ Associations を選択する。

```
SELECT id FROM Association WHERE name = <name>
```

指定されたソース役割名を持つ Associations を選択する。

```
SELECT id FROM Association  WHERE sourceRole = <roleName>
```

指定されたターゲット役割名を持つ Associations を選択する。

```
SELECT id FROM Association WHERE targetRole = <roleName>
```

指定された関係型を持つ Associations を選択する。ここで、関係型は[ebRIM]で規定される該当するフィールド名を含む文字列である。

```
SELECT id FROM Association WHERE
       associationType = <associationType>
```

8.3.7.4 複合関係クエリ

さまざまな形式の Association クエリを複雑な述語にまとめることができる。次のクエリにより、指定された id を持つオブジェクトからの sourceRole "buysFrom"および targetRole "sellsTo"に該当する Associations が選択される。

```
SELECT id FROM Association WHERE
       sourceObject = <id> AND
       sourceRole = ‘buysFrom’ AND
       targetRole = ‘sellsTo’
```

8.3.8 パッケージクエリ

指定された ExtrinsicObject が属する Packages をすべて検索するときは、以下のクエリを指定する。

```
SELECT id FROM Package WHERE id IN (RegistryEntry_packages(<id>))
```

8.3.8.1 複合パッケージクエリ

次のクエリにより、指定されたオブジェクトが属する Packages のうち、廃止されておらず、名前に"RosettaNet"が含まれる Packages がすべて取得される。

```
SELECT id FROM Package WHERE
       id IN (RegistryEntry_packages(<id>)) AND
       name LIKE ‘%RosettaNet%’ AND
       status <> ‘Deprecated’
```

8.3.9 ExternalLink クエリ

指定された ExtrinsicObject のリンク先である ExternalLinks をすべて検索するときは、次のクエリを指定する。

```
SELECT id From ExternalLink WHERE id IN (RegistryEntry_externalLinks(<id>))
```

指定された ExternalLink によってリンクされた ExtrinsicObjects をすべて検索するときは、次のクエリを指定する。

```
SELECT id From ExtrinsicObject WHERE id IN (RegistryEntry_linkedObjects(<id>))
```

8.3.9.1 複合 ExternalLink クエリ

次のクエリにより、指定された ExtrinsicObject が属する ExternalLinks のうち、ワード'legal' を記述に含み、その externalURI に対する URL があるものすべてが取得される。

```
SELECT id FROM ExternalLink WHERE
       id IN (RegistryEntry_externalLinks(<id>)) AND
       description LIKE ‘%legal%’ AND
       externalURI LIKE ‘%http://%’
```

8.3.10 監査トレイルクエリ

指定された ManagedObject に対する AuditableEvent オブジェクトの完全なコレクションを取得するには、次のクエリを指定する。

```
SELECT id FROM AuditableEvent WHERE  registryEntry = <id>
```

8.4 アドホッククエリ要求/応答

クライアントは、AdhocQueryRequest を送信することにより、アドホッククエリを ObjectQueryManager に申請する。AdhocQueryRequest には、サポートされる Registry クエリ機構の 1 つでクエリを定義するサブ要素が含まれる。

ObjectQueryManager は AdhocQueryResponse を同期または非同期でクライアントに返す。AdhocQueryResponse は、要素型が[ebRIM]の RegistryEntry 階層のリーフノードによって表される要素型のセットに含まれるオブジェクトのコレクションを返す。

図 21: アドホッククエリ申請シーケンス図

このプロセスで示される取引文書のスキーマに関する詳細については、付録 A を参照のこと。

8.5 コンテンツの検索

クライアントは GetContentRequest を ObjectQueryManager に送信することにより、コンテンツをレジストリを介して検索する。GetContentRequest は検索が必要な Objects に対するオブジェクト参照の一覧を指定する。ObjectQueryManager は、GetContentResponse メッセージをクライアントの ObjectQueryManagerClient インタフェースに送信することにより、指定されたコンテンツを返す。エラーが発生しなかった場合、GetContentResponse メッセージに指定されたコンテンツがメッセージ内の追加搬送内容として含まれる。RegistryResponse の搬送内容のほかに、要求されたコンテンツごとに追加搬送内容が 1 つ含まれる。エラーが発生した場合は、GetContentResponse の搬送内容にエラーが含まれ、コンテンツ固有の追加搬送内容は含まれない。

8.5.1 コンテンツ・搬送内容の識別

GetContentResponse メッセージは複数のリポジトリ項目を追加搬送内容として含んでよいため、それぞれの搬送内容をメッセージ内で識別する手段を用意することが必要である。この識別を容易なものとするため、レジストリは次の処理を実行しなければならない。

・リポジトリ項目を ebXMLHeader の Manifest 要素でそのオブジェクトに対する DocumentReference の DocumentLabel 要素として記述する RegistryEntry の各インスタンスに対する ID を使用する。

8.5.2 GetContentResponse メッセージの構造

次のメッセージ断片は、要求されるオブジェクトの ID を指定した GetContentRequest の結果として CPP のコレクションを返す GetContentResponse メッセージの構造を示す。追加搬送内容としてメッセージで検索される各オブジェクトの ID は、ebXMLHeader の Manifest でその DocumentLabel として使用されることに注意が必要である。

```
…

--PartBoundary
…
<eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
…
  <eb:Service eb:type=”ebXMLRegistry”>ObjectManager</eb:Service>
  <eb:Action>submitObjects</eb:Action>
…
</eb:MessageHeader>
…
<eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
  <eb:Reference xlink:href=”cid:registryentries@example.com” …>
    <eb:Description xml:lang="en-us">XML instances that are parameters         for the
particular Registry Interface / Method. These are RIM structures that don’t include repository
items, just a reference – contentURI to them.</eb:Description>
  </eb:Reference>
  <eb:Reference xlink:href=”cid:cpp1@example.com” …>
    <eb:Description xml:lang="en-us">XML instance of CPP 1. This is a repository
item.</eb:Description>
  </eb:Reference>
  <eb:Reference xlink:href=”cid:cpp2@example.com” …>
    <eb:Description xml:lang="en-us">XML instance of CPP 2. This is a repository
item.</eb:Description>
  </eb:Reference>
</eb:Manifest>

--PartBoundary
Content-ID: registryentries@example.com
Content-Type: text/xml
…
<?xml version="1.0" encoding="UTF-8"?>
<RootElement>
<SubmitObjectsRequest>
  <RegistryEntryList>
    <ExtrinsicObject … contentURI=”cid:cpp1@example.com” …/>
    <ExtrinsicObject … contentURI=”cid:cpp2@example.com” …/>
  </RegistryEntryList>
</SubmitObjectsRequest>
</RootElement>
--PartBoundary
Content-ID: cpp1@example.com
Content-Type: text/xml
…
<CPP>
…
</CPP>

--PartBoundary
Content-ID: cpp2@example.com
Content-Type: text/xml
…
<CPP>
…
</CPP>

--PartBoundary--
```

8.6 クエリと検索: 一般的なシーケンス

次の図では、ブラウズ/ドリルダウンとアドホッククエリの両方を用いた後、クエリによって選択されたコンテンツの検索を行っている。

図 23: 代表的なクエリと検索のシーケンス

9 レジストリのセキュリティ

本章では、ebXML Registry のセキュリティ機能について説明する。ここでは読者が[ebRIM]で説明されるレジストリ情報モデルにおけるセキュリティ関連のクラスに詳しいことを前提とする。

本仕様の現行バージョンでは、レジストリのセキュリティに関して最低限必要なものだけを組み込むという手法をとっている。「既知のエンティティはいずれもコンテンツを公開でき、誰でも公開コンテンツを閲覧できる」という考え方である。レジストリ情報モデルは本仕様の将来のバージョンでより高度なセキュリティポリシを実装できるように設計されている。

9.1 レジストリコンテンツの完全性

ほとんどのビジネスレジストリには申請されるコンテンツの正しさを確認する手段がないと考えられる。レジストリが提供しなければならない最低限の完全性は、申請組織（SO）が申請するコンテンツが途中でも、レジストリ内でも不正に改竄されることなくレジストリ内で維持されるようにすることである。そのうえ、レジストリは任意のレジストリコンテンツの SO を明確に識別できるようにしなければならない。

9.1.1 メッセージ搬送内容の署名

レジストリコンテンツの完全性を確保するため、申請されるコンテンツはすべて[SEC]が定義する通りレジストリクライアントによる署名を受けなければならない。申請されるコンテンツの署名は以下のことを保証する。

・コンテンツが途中またはレジストリ内で不正に改竄されていない。
・コンテンツの正しさは、特定の申請組織との関係によって確実なものとできる。

9.2 認証

レジストリはクライアントの要求と関連付けられる本人の同一性を証明できなければならない。コンテンツの所有権を識別すると同時に、レジストリ内の特定のオブジェクトに関して本人に割り当てることができる「特権」を識別するため、認証が必要である。

レジストリは要求があるたびに認証を実行しなければならない。セキュリティの観点から見ると、すべてのメッセージは独立したもので、複数のメッセージまたは会話を含むセッションという概念は存在しない。セッションのサポートは、本仕様の将来のバージョンで最適化機能として追加されることがありうる。

レジストリはデジタル証明書と署名に基づく証明書を基本とする認証機構を実装しなければならない。レジストリは署名の証明書 DN を使ってユーザを認証する。

9.2.1 メッセージヘッダの署名

メッセージヘッダへの署名は、[SEC]で定義している通り ebXML メッセージ取扱サービスを送信することで行うことができる。この仕様はまだ最終決定されていないため、このバージョンではメッセージヘッダの署名は要求しない。メッセージヘッダの署名がないため、要求元クライアントの同一性を認証するために搬送内容署名を使用する。

9.3 機密性

9.3.1 オンザワイヤメッセージの機密性

クライアントとレジストリとの間で交換するメッセージ搬送内容を送信時に暗号化することが推奨されるが、これは必須ではない。搬送内容の暗号化にあたっては、[SEC]で定められた制約を遵守しなければならない。

9.3.2 レジストリコンテンツの機密性

本仕様の現行バージョンでは、レジストリのコンテンツの機密性を保護するための規定はない。レジストリに申請されるすべてのコンテンツはいずれのクライアントでも発見し、読むことができる。したがって、申請されたコンテンツを受信し、それをリポジトリに格納するまえに、レジストリはコンテンツを解読できなければならない。このことはレジストリとクライアントとの間に暗号化アルゴリズムに関する事前の取り決め、キー交換に関する取り決めなどがあることを意味する。このサービスは本仕様では規定しない。

9.4 権限

レジストリは[ebRIM]で定義された情報モデルに基づく権限付与機構を提供しなければな

らない。このバージョンの仕様では、権限付与機構はレジストリユーザに対する事前定義済みの役割セットに対して定義されたデフォルトのアクセス制御方針に基づく。この仕様の将来のバージョンでは、申請組織がカスタムアクセス制御方針を定義できるようになる予定である。

9.4.1 レジストリユーザの事前定義済みの役割

レジストリでは以下の役割を事前定義しなければならない。

役割	内容説明
ContentOwner	レジストリコンテンツの申請者または所有者。ISO 11179 における申請組織（SO）
RegistryAdministrator	レジストリの管理者である「スーパー」ユーザ。ISO 11179 における登録機関（RA）
RegistryGuest	レジストリの非認証ユーザ。レジストリをブラウズするクライアントは認証を受ける必要がない。

9.4.2 デフォルトのアクセス制御方針

レジストリは、レジストリユーザに割り当てられた役割に基づき、レジストリのユーザにデフォルト許可を与えるデフォルトの AccessControlPolicy オブジェクトを作成しなければならない。

次の表は、レジストリによってデフォルト AccessControlPolicy に基づいてレジストリユーザのさまざまな事前定義済みの役割に与えられる許可を定義するものである。

役割	許可
ContentOwner	ContentOwner によって所有される Registry Objects 上のすべてのメソッドへのアクセス
RegistryAdministrator	すべての Registry Objects 上のすべてのメソッドへのアクセス
RegistryGuest	すべての Registry Objects 上のすべての読み取り専用（getXXX）メソッドへのアクセス（すべてのコンテンツへの読み取り専用アクセス）

次の一覧はデフォルトの役割に基づく AccessControlPolicy をまとめたものである。

・レジストリはデフォルト AccessControlPolicy を実装し、それをレジストリ内のすべてのオブジェクトと関連付けなければならない。

・誰でもコンテンツを公開できるが、認証を受ける必要がある。

・誰でも認証を必要とせずにコンテンツにアクセスできる。

・ContentOwner は自らが所有するレジストリオブジェクトに対するすべてのメソッドへのアクセス権を持つ。

・RegistryAdministrator は、すべてのレジストリオブジェクトに対するすべてのメソッドへのアクセス権を持つ。

・認証を受けないユーザはすべての読み取り専用(getXXX)メソッドにアクセスできる。

・コンテンツの申請時点で、レジストリは申請メッセージ内の証明によって認証されるデフォルトの ContentOwner の役割を申請組織(SO)に割り当てなければならない。この仕様の現行バージョンでは、証明書によって指定される DN である。

・レジストリをブラウズするクライアントは証明書を使用する必要がない。レジストリはデフォルトの RegistryGuest の役割をそのようなクライアントに割り当てなければならない。

付録 A ebXML レジストリ DTD 定義

次に本書で説明している各種 ebXML Message 搬送内容に関する定義を記載する。

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Begin information model mapping. -->

<!--
ObjectAttributes are attributes from the RegistryObject interface in
                                                ebRIM.

id may be empty. If specified it may be in urn:uuid format or be in some
arbitrary format. If id is empty registry must generate globally unique
                                                id.

If id is provided and in proper UUID syntax (starts with urn:uuid:)
registry will honour it.

If id is provided and is not in proper UUID syntax then it is used for
linkage within document and is ignored by the registry. In this case the
registry generates a UUID for id attribute.

id must not be null when object is being retrieved from the registry.
-->
<!ENTITY % ObjectAttributes "
   id          ID     #IMPLIED
   name        CDATA  #IMPLIED
   description CDATA  #IMPLIED
">

<!--
Use as a proxy for an Object that is in the registry already.
Specifies the id attribute of the object in the registry as its id attribute.
id attribute in ObjectAttributes is exactly the same syntax and semantics
                                                as
id attribute in RegistryObject.
-->
<!ELEMENT ObjectRef EMPTY>
<!ATTLIST ObjectRef
    id ID #IMPLIED
>

<!ELEMENT ObjectRefList (ObjectRef)*>

<!--
RegistryEntryAttributes are attributes from the RegistryEntry interface
in ebRIM.
It inherits ObjectAttributes
-->
```

```
<!ENTITY % RegistryEntryAttributes " %ObjectAttributes;
   majorVersion     CDATA  '1'
   minorVersion     CDATA  '0'
   status           CDATA  #IMPLIED
   userVersion      CDATA  #IMPLIED
   stability        CDATA  'Dynamic'
   expirationDate   CDATA  #IMPLIED">

<!ELEMENT RegistryEntry (SlotList?)>
<!ATTLIST RegistryEntry
        %RegistryEntryAttributes; >
<!ELEMENT Value (#PCDATA)>
<!ELEMENT ValueList (Value*)>
<!ELEMENT Slot (ValueList?)>
<!ATTLIST Slot
        name CDATA #REQUIRED
        slotType CDATA #IMPLIED
>
<!ELEMENT SlotList (Slot*)>

<!--
ExtrinsicObject are attributes from the ExtrinsicObject interface in
                                                ebRIM.
It inherits RegistryEntryAttributes
-->


<!ELEMENT ExtrinsicObject EMPTY >
<!ATTLIST ExtrinsicObject
        %RegistryEntryAttributes;
        contentURI CDATA #REQUIRED
        mimeType CDATA #IMPLIED
        objectType CDATA #REQUIRED
        opaque (true | false) "false"
>


<!ENTITY % IntrinsicObjectAttributes " %RegistryEntryAttributes;">

<!-- Leaf classes that reflect the concrete classes in ebRIM -->
<!ELEMENT RegistryEntryList
(Association | Classification | ClassificationNode | Package |
 ExternalLink | ExternalIdentifier | Organization |
 ExtrinsicObject | ObjectRef)*>

<!--
An ExternalLink specifies a link from a RegistryEntry and an external URI
-->
<!ELEMENT ExternalLink EMPTY>
<!ATTLIST ExternalLink
        %IntrinsicObjectAttributes;
        externalURI CDATA #IMPLIED
>

<!--
An ExternalIdentifier provides an identifier for a RegistryEntry

The value is the value of the identifier (e.g. the social security number)
-->
```

```
<!ELEMENT ExternalIdentifier EMPTY>
<!ATTLIST ExternalIdentifier
        %IntrinsicObjectAttributes;
        value CDATA #REQUIRED
>

<!--
An Association specifies references to two previously submitted
registry entrys.

The sourceObject is id of the sourceObject in association
The targetObject is id of the targetObject in association
-->
<!ELEMENT Association EMPTY>
<!ATTLIST Association
        %IntrinsicObjectAttributes;
        sourceRole CDATA #IMPLIED
        targetRole CDATA #IMPLIED
        associationType CDATA #REQUIRED
        bidirection (true | false) "false"
        sourceObject IDREF #REQUIRED
        targetObject IDREF #REQUIRED
>

<!--
A Classification specifies references to two registry entrys.

The classifiedObject is id of the Object being classified.
The classificationNode is id of the ClassificationNode classying the object
-->
<!ELEMENT Classification EMPTY>
<!ATTLIST Classification
        %IntrinsicObjectAttributes;
        classifiedObject IDREF #REQUIRED
        classificationNode IDREF #REQUIRED
>

<!--
A Package is a named collection of objects.
-->
<!ELEMENT Package EMPTY>
<!ATTLIST Package
        %IntrinsicObjectAttributes;
>

<!-- Attributes inherited by various types of telephone number elements
                                                    -->
<!ENTITY % TelephoneNumberAttributes " areaCode   CDATA  #REQUIRED
   contryCode CDATA  #REQUIRED
   extension  CDATA  #IMPLIED
   number     CDATA  #REQUIRED
   url        CDATA  #IMPLIED">
<!ELEMENT TelephoneNumber EMPTY>
<!ATTLIST TelephoneNumber
        %TelephoneNumberAttributes;
>
<!ELEMENT FaxNumber EMPTY>
<!ATTLIST FaxNumber
        %TelephoneNumberAttributes;
```

```
>

<!ELEMENT PagerNumber EMPTY>
<!ATTLIST PagerNumber
        %TelephoneNumberAttributes;
>

<!ELEMENT MobileTelephoneNumber EMPTY>
<!ATTLIST MobileTelephoneNumber
        %TelephoneNumberAttributes;
>
<!-- PostalAddress -->
<!ELEMENT PostalAddress EMPTY>
<!ATTLIST PostalAddress
        city CDATA #REQUIRED
        country CDATA #REQUIRED
        postalCode CDATA #REQUIRED
        state CDATA #IMPLIED
        street CDATA #REQUIRED
>
<!-- PersonName -->
<!ELEMENT PersonName EMPTY>
<!ATTLIST PersonName
        firstName CDATA #REQUIRED
        middleName CDATA #IMPLIED
        lastName CDATA #REQUIRED
>

<!-- Organization -->
<!ELEMENT Organization (PostalAddress, FaxNumber?, TelephoneNumber)>
<!ATTLIST Organization
        %IntrinsicObjectAttributes;
        parent IDREF #IMPLIED
        primaryContact IDREF #REQUIRED
>

<!ELEMENT User (PersonName, PostalAddress, TelephoneNumber,
                                            MobileTelephoneNumber?,
                                            FaxNumber?, PagerNumber?)>
<!ATTLIST User
        %ObjectAttributes;
        organization IDREF #IMPLIED
        email CDATA #IMPLIED
        url CDATA #IMPLIED
>

<!ELEMENT AuditableEvent EMPTY>
<!ATTLIST AuditableEvent
        %ObjectAttributes;
        eventType CDATA #REQUIRED
        registryEntry IDREF #REQUIRED
        timestamp CDATA #REQUIRED
        user IDREF #REQUIRED
>

<!--
ClassificationNode is used to submit a Classification tree to the Registry.
```

```
parent is the id to the parent node. code is an optional code value for
                                                a ClassificationNode
often defined by an external taxonomy (e.g. NAICS)
-->
<!ELEMENT ClassificationNode EMPTY>
<!ATTLIST ClassificationNode
        %IntrinsicObjectAttributes;
        parent IDREF #IMPLIED
        code CDATA #IMPLIED
>

<!--
End information model mapping.

Begin Registry Services Interface

<!ELEMENT RequestAcceptedResponse EMPTY>
<!ATTLIST RequestAcceptedResponse
        xml:lang NMTOKEN #REQUIRED
>
<!--

The SubmitObjectsRequest allows one to submit a list of RegistryEntry
elements. Each RegistryEntry element provides metadata for a single
submitted object.  Note that the repository item being submitted is in a
separate document that is not in this DTD. The ebXML Messaging Services
Specfication defines packaging, for submission, of the metadata of a
repository item with the repository item itself. The value of the contentURI
attribute of the ExtrinsicObject element must be the same as the xlink:href
attribute within the Reference element within the Manifest element of the
MessageHeader.
-->
<!ELEMENT SubmitObjectsRequest (RegistryEntryList)>
<!ELEMENT AddSlotsRequest (ObjectRef, SlotList)+>
<!-- Only need name in Slot within SlotList -->
<!ELEMENT RemoveSlotsRequest (ObjectRef, SlotList)+>
<!--
The ObjectRefList is the list of
refs to the registry entrys being approved.
-->
<!ELEMENT ApproveObjectsRequest (ObjectRefList)>
<!--
The ObjectRefList is the list of
refs to the registry entrys being deprecated.
-->
<!ELEMENT DeprecateObjectsRequest (ObjectRefList)>
<!--
The ObjectRefList is the list of
refs to the registry entrys being removed
-->
<!ELEMENT RemoveObjectsRequest (ObjectRefList)>
<!ATTLIST RemoveObjectsRequest
        deletionScope (DeleteAll | DeleteRepositoryItemOnly) "DeleteAll"
>
<!ELEMENT GetRootClassificationNodesRequest EMPTY>
<!--
The namePattern follows SQL-92 syntax for the pattern specified in
LIKE clause. It allows for selecting only those root nodes that match
the namePattern. The default value of '*' matches all root nodes.
```

```
-->
<!ATTLIST GetRootClassificationNodesRequest
        namePattern CDATA "*"
>
<!--
The response includes one or more ClassificationNodes
-->
<!ELEMENT GetRootClassificationNodesResponse ( ClassificationNode+ )>
<!--
Get the classification tree under the ClassificationNode specified
                                                parentRef.

If depth is 1 just fetch immediate child
nodes, otherwise fetch the descendant tree upto the specified depth level.
If depth is 0 that implies fetch entire sub-tree
-->
<!ELEMENT GetClassificationTreeRequest EMPTY>
<!ATTLIST GetClassificationTreeRequest
        parent CDATA #REQUIRED
        depth CDATA "1"
>
<!--
The response includes one or more ClassificationNodes which includes only
immediate ClassificationNode children nodes if depth attribute in
GetClassificationTreeRequest was 1, otherwise the decendent nodes
upto specified depth level are returned.
-->
<!ELEMENT GetClassificationTreeResponse ( ClassificationNode+ )>
<!--
Get refs to all registry entrys that are classified by all the
ClassificationNodes specified by ObjectRefList.
Note this is an implicit logical AND operation
-->
<!ELEMENT GetClassifiedObjectsRequest (ObjectRefList)>
<!--
objectType attribute can specify the type of objects that the registry
client is interested in, that is classified by this ClassificationNode.
It is a String that matches a choice in the type attribute of
                                                ExtrinsicObject.
The default value of '*' implies that client is interested in all types
of registry entrys that are classified by the specified ClassificationNode.
-->
<!--
The response includes a RegistryEntryList which has zero or more
RegistryEntrys that are classified by the ClassificationNodes
specified in the ObjectRefList in GetClassifiedObjectsRequest.
-->
<!ELEMENT GetClassifiedObjectsResponse ( RegistryEntryList )>
<!--
An Ad hoc query request specifies a query string as defined by [RS] in the
                                                queryString attribute
-->
<!ELEMENT AdhocQueryRequest (FilterQuery | ReturnRegistryEntry |
                                                ReturnRepositoryItem |
                                                SQLQuery)>
<!ELEMENT SQLQuery (#PCDATA)>
<!--
The response includes a RegistryEntryList which has zero or more
RegistryEntrys that match the query specified in AdhocQueryRequest.
```

```
-->
<!ELEMENT AdhocQueryResponse
 ( RegistryEntryList |
   FilterQueryResult |
   ReturnRegistryEntryResult |
   ReturnRepositoryItemResult )>
<!--
Gets the actual content (not metadata) specified by the ObjectRefList
-->
<!ELEMENT GetContentRequest (ObjectRefList)>
<!--
The GetObjectsResponse will have no sub-elements if there were no errors.
The actual contents will be in the other payloads of the message.
-->
<!ELEMENT GetContentResponse EMPTY >
<!--
Describes the capability profile for the registry and what optional
                                              features
are supported
-->
<!ELEMENT RegistryProfile (OptionalFeaturesSupported)>
<!ATTLIST RegistryProfile
        version CDATA #REQUIRED
>

<!ELEMENT OptionalFeaturesSupported EMPTY>
<!ATTLIST OptionalFeaturesSupported
        sqlQuery (true | false) "false"
        xQuery (true | false) "false"
>
<!-- Begin FilterQuery DTD -->
<!ELEMENT FilterQuery (RegistryEntryQuery | AuditableEventQuery |
                                            ClassificationNodeQuery |
                                            RegistryPackageQuery |
                                            OrganizationQuery)>
<!ELEMENT FilterQueryResult (RegistryEntryQueryResult |
                                            AuditableEventQueryResult |
                                            ClassificationNodeQueryResu
                                            lt |
                                            RegistryPackageQueryResult
                                            | OrganizationQueryResult)>
<!ELEMENT RegistryEntryQueryResult (RegistryEntryView*)>
<!ELEMENT RegistryEntryView EMPTY>
<!ATTLIST RegistryEntryView
        objectURN CDATA #REQUIRED
        contentURI CDATA #IMPLIED
        objectID CDATA #IMPLIED
>
<!ELEMENT AuditableEventQueryResult (AuditableEventView*)>
<!ELEMENT AuditableEventView EMPTY>
<!ATTLIST AuditableEventView
        objectID CDATA #REQUIRED
        timestamp CDATA #REQUIRED
>
<!ELEMENT ClassificationNodeQueryResult (ClassificationNodeView*)>
<!ELEMENT ClassificationNodeView EMPTY>
<!ATTLIST ClassificationNodeView
        objectURN CDATA #REQUIRED
        contentURI CDATA #IMPLIED
```

```
        objectID CDATA #IMPLIED
>
<!ELEMENT RegistryPackageQueryResult (RegistryPackageView*)>
<!ELEMENT RegistryPackageView EMPTY>
<!ATTLIST RegistryPackageView
        objectURN CDATA #REQUIRED
        contentURI CDATA #IMPLIED
        objectID CDATA #IMPLIED
>
<!ELEMENT OrganizationQueryResult (OrganizationView*)>
<!ELEMENT OrganizationView EMPTY>
<!ATTLIST OrganizationView
        orgURN CDATA #REQUIRED
        objectID CDATA #IMPLIED
>

<!ELEMENT RegistryEntryQuery
   (  RegistryEntryFilter?,
      SourceAssociationBranch*,
      TargetAssociationBranch*,
      HasClassificationBranch*,
      SubmittingOrganizationBranch?,
      ResponsibleOrganizationBranch?,
      ExternalIdentifierFilter*,
      ExternalLinkFilter*,
      SlotFilter*,
      HasAuditableEventBranch*          )>

<!ELEMENT SourceAssociationBranch (AssociationFilter?,
                                   RegistryEntryFilter?)>
<!ELEMENT TargetAssociationBranch (AssociationFilter?,
                                   RegistryEntryFilter?)>
<!ELEMENT HasClassificationBranch (ClassificationFilter?,
                                   ClassificationNodeFilter?)>
<!ELEMENT SubmittingOrganizationBranch (OrganizationFilter?,
                                   ContactFilter?)>
<!ELEMENT ResponsibleOrganizationBranch (OrganizationFilter?,
                                   ContactFilter?)>
<!ELEMENT HasAuditableEventBranch (AuditableEventFilter?, UserFilter?,
                                   OrganizationFilter?)>
<!ELEMENT AuditableEventQuery
  (AuditableEventFilter?, RegistryEntryQuery*, InvokedByBranch? )>

<!ELEMENT InvokedByBranch
  ( UserFilter?, OrganizationQuery? )>

<!ELEMENT ClassificationNodeQuery (ClassificationNodeFilter?,
                                   PermitsClassificationBranch
                                   *, HasParentNode?,
                                   HasSubnode*)>
<!ELEMENT PermitsClassificationBranch (ClassificationFilter?,
                                   RegistryEntryQuery?)>
<!ELEMENT HasParentNode (ClassificationNodeFilter?, HasParentNode?)>
<!ELEMENT HasSubnode (ClassificationNodeFilter?, HasSubnode*)>
<!ELEMENT RegistryPackageQuery (PackageFilter?, HasMemberBranch*)>
<!ELEMENT HasMemberBranch (RegistryEntryQuery?)>
```

```
<!ELEMENT OrganizationQuery (OrganizationFilter?, SubmitsRegistryEntry*,
                                                HasParentOrganization?,
                                                InvokesEventBranch*,
                                                ContactFilter*)>
<!ELEMENT SubmitsRegistryEntry (RegistryEntryQuery?)>
<!ELEMENT HasParentOrganization (OrganizationFilter?,
                                                HasParentOrganization?)>
<!ELEMENT InvokesEventBranch (UserFilter?, AuditableEventFilter?,
                                                RegistryEntryQuery?)>
<!ELEMENT ReturnRegistryEntry (RegistryEntryQuery, WithClassifications?,
                                                WithSourceAssociations?,
                                                WithTargetAssociations?,
                                                WithAuditableEvents?,
                                                WithExternalLinks?)>
<!ELEMENT WithClassifications (ClassificationFilter?)>
<!ELEMENT WithSourceAssociations (AssociationFilter?)>
<!ELEMENT WithTargetAssociations (AssociationFilter?)>
<!ELEMENT WithAuditableEvents (AuditableEventFilter?)>
<!ELEMENT WithExternalLinks (ExternalLinkFilter?)>
<!ELEMENT ReturnRegistryEntryResult (RegistryEntryMetadata*)>
<!ELEMENT RegistryEntryMetadata (RegistryEntry, Classification*,
                                                SourceAssociations?,
                                                TargetAssociations?,
                                                AuditableEvent*,
                                                ExternalLink*)>
<!ELEMENT SourceAssociations (Association*)>
<!ELEMENT TargetAssociations (Association*)>
<!ELEMENT ReturnRepositoryItem (RegistryEntryQuery,
                                                RecursiveAssociationOption?,
                                                WithDescription?)>
<!ELEMENT RecursiveAssociationOption (AssociationType+)>
<!ATTLIST RecursiveAssociationOption
        depthLimit CDATA #IMPLIED
>
<!ELEMENT AssociationType EMPTY>
<!ATTLIST AssociationType
        role CDATA #REQUIRED
>
<!ELEMENT WithDescription EMPTY>
<!ELEMENT ReturnRepositoryItemResult (RepositoryItem*)>
<!ELEMENT RepositoryItem (RegistryPackage | ExtrinsicObject |
                                                WithdrawnObject |
                                                ExternalLink)>
<!ATTLIST RepositoryItem
        identifier CDATA #REQUIRED
        name CDATA #REQUIRED
        contentURI CDATA #REQUIRED
        objectType CDATA #REQUIRED
        status CDATA #REQUIRED
        stability CDATA #REQUIRED
        description CDATA #IMPLIED
>
<!ELEMENT RegistryPackage EMPTY>
<!ELEMENT WithdrawnObject EMPTY>
<!ELEMENT ExternalLinkItem EMPTY>
<!ELEMENT ObjectFilter (Clause)>
<!ELEMENT RegistryEntryFilter (Clause)>
<!ELEMENT IntrinsicObjectFilter (Clause)>
<!ELEMENT ExtrinsicObjectFilter (Clause)>
```

```
<!ELEMENT PackageFilter (Clause)>
<!ELEMENT OrganizationFilter (Clause)>
<!ELEMENT ContactFilter (Clause)>
<!ELEMENT ClassificationNodeFilter (Clause)>
<!ELEMENT AssociationFilter (Clause)>
<!ELEMENT ClassificationFilter (Clause)>
<!ELEMENT ExternalLinkFilter (Clause)>
<!ELEMENT SlotFilter (Clause)>
<!ELEMENT ExternalIdentifierFilter (Clause)>
<!ELEMENT AuditableEventFilter (Clause)>
<!ELEMENT UserFilter (Clause)>

<!--
The following lines define the XML syntax for Clause.
-->
<!ELEMENT Clause (SimpleClause | CompoundClause)>
<!ELEMENT SimpleClause (BooleanClause | RationalClause | StringClause)>
<!ATTLIST SimpleClause
        leftArgument CDATA #REQUIRED
>
<!ELEMENT CompoundClause (Clause, Clause+)>
<!ATTLIST CompoundClause
        connectivePredicate (And | Or) #REQUIRED
>
<!ELEMENT BooleanClause EMPTY>
<!ATTLIST BooleanClause
        booleanPredicate (true | false) #REQUIRED
>
<!ELEMENT RationalClause (IntClause | FloatClause)>
<!ATTLIST RationalClause
        logicalPredicate (LE | LT | GE | GT | EQ | NE) #REQUIRED
>
<!ELEMENT IntClause (#PCDATA)>
<!ATTLIST IntClause
        e-dtype NMTOKEN #FIXED "int"
>
<!ELEMENT FloatClause (#PCDATA)>
<!ATTLIST FloatClause
        e-dtype NMTOKEN #FIXED "float"
>
<!ELEMENT StringClause (#PCDATA)>
<!ATTLIST StringClause
        stringPredicate
      (contains | -contains |
         startswith | -startswith |
         equal | -equal |
         endswith | -endswith) #REQUIRED
>
<!-- End FilterQuery DTD -->

<!-- Begin RegistryError definition -->
<!-- The RegistryErrorList is derived from the ErrorList element from the
  ebXML Message Service Specification -->
<!ELEMENT RegistryErrorList ( RegistryError+ )>
<!ATTLIST RegistryErrorList
   highestSeverity  ( Warning | Error ) ‘Warning’ >

<!ELEMENT RegistryError  (#PCDATA) >
<!ATTLIST RegistryError
```

```
   codeContext   CDATA   #REQUIRED
   errorCode     CDATA   #REQUIRED
   severity  ( Warning | Error ) 'Warning'
   location      CDATA   #IMPLIED
   xml:lang     NMTOKEN  #IMPLIED>

<!ELEMENT RegistryResponse
 (( AdhocQueryResponse  |
    GetContentResponse  |
    GetClassificationTreeResponse  |
    GetClassifiedObjectsResponse  |
    GetRootClassificationNodesResponse  )?,
   RegistryErrorList?  )>
<!ATTLIST RegistryResponse
   status  (success | failure )  #REQUIRED >

<!-- The contrived root node -->

<!ELEMENT RootElement
  ( SubmitObjectsRequest |
    ApproveObjectsRequest |
    DeprecateObjectsRequest |
    RemoveObjectsRequest |
    GetRootClassificationNodesRequest |
    GetClassificationTreeRequest |
    GetClassifiedObjectsRequest |
    AdhocQueryRequest |
    GetContentRequest |
    AddSlotsRequest |
    RemoveSlotsRequest |
    RegistryResponse |
    RegistryProfile) >

<!ELEMENT Href  (#PCDATA )>

<!ELEMENT XMLDocumentErrorLocn  (DocumentId , Xpath )>

<!ELEMENT DocumentId  (#PCDATA )>

<!ELEMENT Xpath (#PCDATA)>
```

付録 B UML 図の解釈

本節では、UML で ebXML ビジネスプロセス記述を定義するために使用する規約を概念の観点から説明する。

B.1 UML クラス図

UML クラス図は、ebXML のレジストリサービスとクライアントを実装するために必要なサービスインタフェース（[ebCPP]で定義）を記述するために使用する。例については、14 ページの図 2 を参照のこと。UML クラス図には以下の要素が含まれる。

1. UML インタフェースのコレクション。インタフェースはそれぞれレジストリサービスに対するサービスインタフェースを表す。

2. 各インタフェース上のメソッドの表記述。メソッドはそれぞれ UML インタフェースを表すサービスインタフェース内のアクション（[ebCPP]によって定義される）を表す。

3. UML インタフェース内の各メソッドは、1 つまたは複数のパラメータを指定する。ここで、各メソッドの引数の型はメソッドに対応するアクションの一部として交換する ebXML メッセージ型を表す。引数が複数あることは、対応する ebXML メッセージの本文に含まれる搬送内容文書が複数あることを意味する。

B.2 UML シーケンス図

UML シーケンス図は、レジストリ固有の ebXML ビジネスプロセスに対する UML インタフェース間の相互作用を表すビジネスプロコトルを指定するために使用される。UML シーケンス図は、[ebCPP]の規定に従ってメッセージの順序、要求と応答の関係のほか、要求とエラー応答の関係を決定するのに必要な情報を提供する。

メソッドは要求元から応答先に呼び出されるため、シーケンス図はそれぞれ特定の会話プロトコルのシーケンスを示す。メソッドの呼び出しは UML 表記のどのリンクの矢印を使用するかによって同期または非同期とすることができる。半分の矢印は非同期通信を表す。完全な矢印は同期通信を表す。

それぞれのメソッドの呼び出しの次に、前の要求に対する ResponseName を指定するために応答先から要求元への応答メソッドの呼び出しを続けることができる。可能なエラー応答は要求元から応答先への条件付き応答メソッドの呼び出しで指定される。例は、20 ページの図 4 を参照のこと。

付録 C SQL クエリ仕様

C.1 SQL クエリの構文仕様

本節では、SQL クエリ構文を SQL-92 のサブセットとして定義する規則を指定する。かぎ括弧で括られる用語は[SQL]または[SQL/PSM]で定義される。SQL クエリ構文は下記の制限を除いて<query specification>に適合する。

1. <select list>には最大で 1 つの<select sublist>を含むことができる。

2. <select list>には、<from clause>内のテーブルに由来するデータ型が UUID のカラムが一つなければならない。

3. <derived column>には<as clause>を使用できない。

4. <table expression>にはオプションの<group by clause>および<having clause>の文節は含まない。

5. <table reference>の要素は<table name>および<correlation name>のみとしなければならない。

6. <table reference>には<table name>と<correlation name>との間にオプションの AS がない。

7. <from clause>には一つの<table reference>のみを使用できる。

8. サブクエリの制限付き使用は次の構文によって許可される。すなわち、<in predicate>では、<in predicate>の右側を前記の制限付き<query specification>に限定できる。

9. <where clause>内の<search condition>には<query expression>を含めることができない。

10. SQL クエリ構文では、[SQL/PSM]からの<sql invoked routines>の呼び出しを<in predicate>の RHS として使用できる。

C.2 クエリ構文法に関する非規範的 BNF

次の BNF はレジストリクエリの構文のための文法を例示する。この例は実装担当者の参考のためにここに収録される。この BNF は[SQL]に準拠するものではないため、非規範的構文として示される。規範的な構文規則については、付録 C.1 を参照のこと。

```
/*****************************************************************
 * The Registry Query (Subset of SQL-92) grammar starts here
 *****************************************************************/

RegistryQuery = SQLSelect [“;”]

SQLSelect = "SELECT" SQLSelectCols "FROM" SQLTableList [ SQLWhere ]

SQLSelectCols =  ID

SQLTableList = SQLTableRef

SQLTableRef = ID

SQLWhere = "WHERE" SQLOrExpr

SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr)*

SQLAndExpr = SQLNotExpr ("AND" SQLNotExpr)*

SQLNotExpr = [ "NOT" ] SQLCompareExpr

SQLCompareExpr =
    (SQLColRef "IS") SQLIsClause
  | SQLSumExpr [ SQLCompareExprRight ]


SQLCompareExprRight =
    SQLLikeClause
  | SQLInClause
  | SQLCompareOp SQLSumExpr

SQLCompareOp =
    "="
  | "<>"
  | ">"
  | ">="
  | "<"
  | "<="

SQLInClause = [ "NOT" ] "IN" "(" SQLLValueList ")"

SQLLValueList = SQLLValueElement ( "," SQLLValueElement )*

SQLLValueElement = "NULL" | SQLSelect

SQLIsClause =  SQLColRef "IS" [ "NOT" ] "NULL"

SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern

SQLPattern = STRING_LITERAL
```

```
SQLLiteral =
    STRING_LITERAL
  | INTEGER_LITERAL
  | FLOATING_POINT_LITERAL

SQLColRef = SQLLvalue

SQLLvalue =  SQLLvalueTerm

SQLLvalueTerm = ID ( "."  ID )*

SQLSumExpr = SQLProductExpr (( "+" | "-" ) SQLProductExpr )*

SQLProductExpr = SQLUnaryExpr (( "*" | "/" ) SQLUnaryExpr )*

SQLUnaryExpr = [ ( "+" | "-") ] SQLTerm

SQLTerm = "(" SQLOrExpr ")"
  | SQLColRef
  | SQLLiteral

INTEGER_LITERAL = (["0"-"9"])+

FLOATING_POINT_LITERAL =
         (["0"-"9"])+ "." (["0"-"9"])+ (EXPONENT)?
       | "." (["0"-"9"])+ (EXPONENT)?
       | (["0"-"9"])+ EXPONENT
       | (["0"-"9"])+ (EXPONENT)?

EXPONENT = ["e","E"] (["+","-"])? (["0"-"9"])+

STRING_LITERAL: "'" (~["'"])* ( "'" (~["'"])* )* "'"

ID = ( <LETTER> )+ ( "_" | "$" | "#" | <DIGIT> | <LETTER> )*
LETTER = ["A"-"Z", "a"-"z"]
DIGIT = ["0"-"9"]
```

C.3 SQL クエリの関係スキーマ

```
--SQL Load file for creating the ebXML Registry tables


--Minimal use of SQL-99 features in DDL is illustrative and may be easily mapped to SQL-92


CREATE TYPE ShortName AS VARCHAR(64) NOT FINAL;
CREATE TYPE LongName AS VARCHAR(128) NOT FINAL;
CREATE TYPE FreeFormText AS VARCHAR(256) NOT FINAL;

CREATE TYPE UUID UNDER ShortName FINAL;
CREATE TYPE URI UNDER LongName FINAL;

CREATE TABLE ExtrinsicObject (

--RegistryObject Attributes
  id                                                    UUID PRIMARY KEY NOT NULL,
  name                                                  LongName,
  description                                       FreeFormText,
  accessControlPolicy             UUID NOT NULL,

--Versionable attributes
  majorVersion                                      INT DEFAULT 0 NOT NULL,
  minorVersion                                      INT DEFAULT 1 NOT NULL,

--RegistryEntry attributes
  status                                                INT DEFAULT 0 NOT NULL,
  userVersion                                       ShortName,
  stability                                             INT      DEFAULT 0 NOT NULL,
  expirationDate                              TIMESTAMP,

--ExtrinsicObject attributes
```

```
  contentURI                                          URI,
  mimeType                                                ShortName,
  objectType                                          INT DEFAULT 0 NOT NULL,
  opaque                                                  BOOLEAN DEFAULT false NOT
NULL

);

CREATE PROCEDURE RegistryEntry_associatedObjects(registryEntryId) {
--Must return a collection of UUIDs for related RegistryEntry instances
}

CREATE PROCEDURE RegistryEntry_auditTrail(registryEntryId) {
--Must return an collection of UUIDs for AuditableEvents related to the RegistryEntry.
--Collection must be in ascending order by timestamp
}

CREATE PROCEDURE RegistryEntry_externalLinks(registryEntryId) {
--Must return a collection of UUIDs for ExternalLinks annotating this RegistryEntry.
}

CREATE PROCEDURE RegistryEntry_externalIdentifiers(registryEntryId) {
--Must return a collection of UUIDs for ExternalIdentifiers for this RegistryEntry.
}

CREATE PROCEDURE RegistryEntry_classificationNodes(registryEntryId) {
--Must return a collection of UUIDs for ClassificationNodes classifying this RegistryEntry.
}

CREATE PROCEDURE RegistryEntry_packages(registryEntryId) {
--Must return a collection of UUIDs for Packages that this RegistryEntry belongs to.
}

CREATE TABLE Package (

--RegistryObject Attributes
  id                                                      UUID PRIMARY KEY NOT NULL,
  name                                                    LongName,
  description                                         FreeFormText,
  accessControlPolicy             UUID NOT NULL,

--Versionable attributes
  majorVersion                                        INT DEFAULT 0 NOT NULL,
  minorVersion                                        INT DEFAULT 1 NOT NULL,

--RegistryEntry attributes
  status                                                  INT DEFAULT 0 NOT NULL,
  userVersion                                         ShortName,
  stability                                               INT      DEFAULT 0 NOT NULL,
  expirationDate                            TIMESTAMP,

--Package attributes
);

CREATE PROCEDURE Package_memberbjects(packageId) {
--Must return a collection of UUIDs for RegistryEntrys that are memebers of this Package.
}

CREATE TABLE ExternalLink (

--RegistryObject Attributes
  id                                                      UUID PRIMARY KEY NOT NULL,
  name                                                    LongName,
  description                                         FreeFormText,
  accessControlPolicy             UUID NOT NULL,

--Versionable attributes
  majorVersion                                        INT DEFAULT 0 NOT NULL,
  minorVersion                                        INT DEFAULT 1 NOT NULL,

--RegistryEntry attributes
  status                                                  INT DEFAULT 0 NOT NULL,
  userVersion                                         ShortName,
  stability                                               INT      DEFAULT 0 NOT NULL,
```

```
  expirationDate                            TIMESTAMP,

--ExternalLink attributes
  externalURI                                       URI NOT NULL

);

CREATE PROCEDURE ExternalLink_linkedObjects(registryEntryId) {
--Must return a collection of UUIDs for objects in this relationship
}

CREATE TABLE ExternalIdentifier (

--RegistryObject Attributes
  id                                                        UUID PRIMARY KEY NOT NULL,
  name                                                      LongName,
  description                                       FreeFormText,
  accessControlPolicy             UUID NOT NULL,

--Versionable attributes
  majorVersion                                      INT DEFAULT 0 NOT NULL,
  minorVersion                                      INT DEFAULT 1 NOT NULL,

--RegistryEntry attributes
  status                                                    INT DEFAULT 0 NOT NULL,
  userVersion                                       ShortName,
  stability                                                 INT      DEFAULT 0 NOT NULL,
  expirationDate                            TIMESTAMP,

--ExternalIdentifier attributes
  value                                                     ShortName NOT NULL

);


--A SlotValue row represents one value of one slot in some
--RegistryEntry
CREATE TABLE SlotValue (

--RegistryObject Attributes
  registryEntry                             UUID     PRIMARY KEY NOT NULL,

--Slot attributes
  name                                              LongName NOT NULL PRIMARY KEY NOT
NULL,
  value                                             ShortName NOT NULL

);

CREATE TABLE Association (
--RegistryObject Attributes
  id                                                        UUID PRIMARY KEY NOT NULL,
  name                                                      LongName,
  description                                       FreeFormText,
  accessControlPolicy             UUID NOT NULL,

--Versionable attributes
  majorVersion                                      INT DEFAULT 0 NOT NULL,
  minorVersion                                      INT DEFAULT 1 NOT NULL,

--RegistryEntry attributes
  status                                                    INT DEFAULT 0 NOT NULL,
  userVersion                                       ShortName,
  stability                                                 INT      DEFAULT 0 NOT NULL,
  expirationDate                            TIMESTAMP,

--Association attributes
  associationType                           INT NOT NULL,
  bidirectional                                     BOOLEAN DEFAULT false NOT NULL,
  sourceObject                                      UUID NOT NULL,
  sourceRole                                        ShortName,
  label                                                     ShortName,
  targetObject                                      UUID NOT NULL,
  targetRole                                        ShortName
```

```
);

--Classification is currently identical to Association
CREATE TABLE Classification (
--RegistryObject Attributes
  id                                                    UUID PRIMARY KEY NOT NULL,
  name                                                  LongName,
  description                                   FreeFormText,
  accessControlPolicy             UUID NOT NULL,

--Versionable attributes
  majorVersion                                  INT DEFAULT 0 NOT NULL,
  minorVersion                                  INT DEFAULT 1 NOT NULL,

--RegistryEntry attributes
  status                                                INT DEFAULT 0 NOT NULL,
  userVersion                                   ShortName,
  stability                                             INT      DEFAULT 0 NOT NULL,
  expirationDate                          TIMESTAMP,

--Classification attributes. Assumes not derived from Association
  sourceObject                                  UUID NOT NULL,
  targetObject                                  UUID NOT NULL,
);


CREATE TABLE ClassificationNode (
--RegistryObject Attributes
  id                                                    UUID PRIMARY KEY NOT NULL,
  name                                                  LongName,
  description                                   FreeFormText,
  accessControlPolicy             UUID NOT NULL,

--Versionable attributes
  majorVersion                                  INT DEFAULT 0 NOT NULL,
  minorVersion                                  INT DEFAULT 1 NOT NULL,

--RegistryEntry attributes
  status                                                INT DEFAULT 0 NOT NULL,
  userVersion                                   ShortName,
  stability                                             INT      DEFAULT 0 NOT NULL,
  expirationDate                          TIMESTAMP,

--ClassificationNode attributes
  parent                                                UUID,
  path                                                  VARCHAR(512) NOT NULL,

  code                                                  ShortName
);

CREATE PROCEDURE ClassificationNode_classifiedObjects(classificationNodeId) {
--Must return a collection of UUIDs for RegistryEntries classified by this ClassificationNode
}

--Begin Registry Audit Trail tables

CREATE TABLE AuditableEvent (
--RegistryObject Attributes
  id                                                    UUID PRIMARY KEY NOT NULL,
  name                                                  LongName,
  description                                   FreeFormText,
  accessControlPolicy             UUID NOT NULL,

--AuditableEvent attributes
  user                            UUID,
  eventType                                             INT DEFAULT 0 NOT NULL,

  registryEntry                                 UUID NOT NULL,
  timestamp                                             TIMESTAMP NOT NULL,
);



CREATE TABLE User (
```

```
--RegistryObject Attributes
  id                                                    UUID PRIMARY KEY NOT NULL,
  name                                                  LongName,
  description                                   FreeFormText,
  accessControlPolicy             UUID NOT NULL,

--User attributes
  organization                            UUID NOT NULL

--address attributes flattened
  address_city                            ShortName,
  address_country                 ShortName,
  address_postalCode              ShortName,
  address_state                           ShortName,
  address_street                  ShortName,

  email                                         ShortName,

--fax attribute flattened
  fax_areaCode                            VARCHAR(4) NOT NULL,
  fax_countryCode                 VARCHAR(4),
  fax_extension                           VARCHAR(8),
  fax_umber                                     VARCHAR(8) NOT NULL,
  fax_url                                       URI

 --mobilePhone attribute flattened
  mobilePhone_areaCode                    VARCHAR(4) NOT NULL,
  mobilePhone_countryCode         VARCHAR(4),
  mobilePhone_extension                   VARCHAR(8),
  mobilePhone_umber                             VARCHAR(8) NOT NULL,
  mobilePhone_url                               URI

--name attribute flattened
  name_firstName                  ShortName,
  name_middleName                 ShortName,
  name_lastName                           ShortName,

--pager attribute flattened
  pager_areaCode                  VARCHAR(4) NOT NULL,
  pager_countryCode               VARCHAR(4),
  pager_extension                 VARCHAR(8),
  pager_umber                             VARCHAR(8) NOT NULL,
  pager_url                                     URI

--telephone attribute flattened
  telephone_areaCode                      VARCHAR(4) NOT NULL,
  telephone_countryCode                   VARCHAR(4),
  telephone_extension                     VARCHAR(8),
  telephone_umber                               VARCHAR(8) NOT NULL,
  telephone_url                                         URI,

  url                                                           URI,

);

CREATE TABLE Organization (
--RegistryObject Attributes
  id                                                    UUID PRIMARY KEY NOT NULL,
  name                                                  LongName,
  description                                   FreeFormText,
  accessControlPolicy             UUID NOT NULL,

--Versionable attributes
  majorVersion                                  INT DEFAULT 0 NOT NULL,
  minorVersion                                  INT DEFAULT 1 NOT NULL,

--RegistryEntry attributes
  status                                                INT DEFAULT 0 NOT NULL,
  userVersion                                   ShortName,
  stability                                             INT      DEFAULT 0 NOT NULL,
  expirationDate                          TIMESTAMP,

--Organization attributes
```

```
--Organization.address attribute flattened
  address_city                                  ShortName,
  address_country                       ShortName,
  address_postalCode              ShortName,
  address_state                                 ShortName,
  address_street                        ShortName,

--primary contact for Organization, points to a User.
--Note many Users may belong to the same Organization
  contact                                                 UUID NOT NULL,

--Organization.fax attribute falttened
  fax_areaCode                          VARCHAR(4) NOT NULL,
  fax_countryCode                 VARCHAR(4),
  fax_extension                         VARCHAR(8),
  fax_umber                                     VARCHAR(8) NOT NULL,
  fax_url                                       URI,

--Organization.parent attribute
  parent                                        UUID,

--Organization.telephone attribute falttened
  telephone_areaCode                    VARCHAR(4) NOT NULL,
  telephone_countryCode                 VARCHAR(4),
  telephone_extension                   VARCHAR(8),
  telephone_umber                               VARCHAR(8) NOT NULL,
  telephone_url                                         URI
);


--Note that the ebRIM security view is not visible through the public query mechanism
--in the current release


--The RegistryEntry View allows polymorphic queries over all ebRIM classes derived
--from RegistryEntry

CREATE VIEW RegistryEntry (
--RegistryObject Attributes
  id,
  name,
  description,
  accessControlPolicy,

--Versionable attributes
  majorVersion,
  minorVersion,

--RegistryEntry attributes
  status,
  userVersion,
  stability,
  expirationDate

) AS
 SELECT
--RegistryObject Attributes
  id,
  name,
  description,
  accessControlPolicy,

--Versionable attributes
  majorVersion,
  minorVersion,

--RegistryEntry attributes
  status,
  userVersion,
  stability,
  expirationDate

 FROM ExtrinsicObject
 UNION
```

```
 SELECT
--RegistryObject Attributes
  id,
  name,
  description,
  accessControlPolicy,

--Versionable attributes
  majorVersion,
  minorVersion,

--RegistryEntry attributes
  status,
  userVersion,
  stability,
  expirationDate
 FROM (Registry)Package
 UNION

 SELECT
--RegistryObject Attributes
  id,
  name,
  description,
  accessControlPolicy,

--Versionable attributes
  majorVersion,
  minorVersion,

--RegistryEntry attributes
  status,
  userVersion,
  stability,
  expirationDate
FROM ClassificationNode;
```

付録 D アドホッククエリに基づく非規範的コンテンツ

Registry SQL クエリ機能は、コンテンツをカタログ化するメタデータに基づくコンテンツの検索ばかりでなく、コンテンツ自身に含まれるデータに基づく検索機能もサポートしている。たとえば、クライアントは、CPP 文書自身の RoleName 要素内で " seller " と名付けられた役割を定義するすべてのコラボレーションプロトコルプロファイルを検索するクエリを申請できる。現在のところ、コンテンツに基づくクエリ機能は、XML コンテンツに制限されている。

D.1.1 XML コンテンツの自動分類

コンテンツに基づくクエリは、Registry によってサポートされる既存の分類機構を通して、間接的にサポートされる。

申請組織は、任意の XML スキーマまたは DTD について、それが申請されるときに論理インデックスを定義できる。こうした論理インデックスのインスタンスは、XML 文書ツリー内の特定の属性または要素ノードと、レジストリ内の分類スキームにある ClassificationNode との間のリンクを定義する。

レジストリは、このインデックスを利用して、スキーマのインスタンスである文書を文書インスタンスが申請されるときに自動的に分類する。このような文書は文書自体に含まれるデータに従って分類される。

このように自動的に分類されたコンテンツについては、後でクライアントが Registry の既存

の分類ベース検索機構と ObjectQueryManager のクエリ機能を用いて検索できる。

[注]この手法はデータベースが索引付き検索をサポートする方法と概念的に類似する。DBA はスキーマ内のテーブルについてインデックスを定義する。データがテーブルに追加されるときに、データに自動的にインデックスが付けられる。

D.1.2 インデックス定義

本節では、Registry DTD で定義される SubmittedObject 要素で、論理インデックスを定義する方法を説明する。完全な Registry DTD については付録 A で指定している。

スキーマまたは DTD に対する SubmittedObject 要素は、オプションの ClassificationIndexList 要素で ClassificationIndexes のコレクションを定義できる。申請されるコンテンツが SCHEMA objectType でない場合は、ClassificationIndexList は無視される。

ClassificationIndex 要素は、[ebRIM]における基本クラス Registry Object の属性を継承する。したがって、この要素は専門属性を次のように定義する。

1. classificationNode:この属性は、特定の ClassificationNode をその ID によって参照する。

2. contentIdentifier:この属性は、[XPT]によって定義される XPATH 式を用いて、スキーマの文書インスタンスに含まれる特定のデータ要素を識別する。

D.1.3 インデックス定義の例

CPP の RoleName 要素内で定義される役割に基づいて、CPP を自動的に分類するインデックスを定義するには、次のインデックスを CPP スキーマまたは DTD について定義しなければならない。

```
<ClassificationIndex
       classificationNode='id-for-role-classification-scheme'
       contentIdentifier='/Role//RoleName'
/>
```

D.1.4 推奨される XML 定義

```
<!--
A ClassificationIndexList is specified on ExtrinsicObjects of objectType
'Schema' to define an automatic Classification of instance objects of the
schema using the specified classificationNode as parent and a
ClassificationNode created or selected by the object content as selected
by the contentIdentifier
-->
<!ELEMENT ClassificationIndex EMPTY>
<!ATTLIST ClassificationIndex
        %ObjectAttributes;
        classificationNode IDREF #REQUIRED
        contentIdentifier CDATA #REQUIRED
>

<!-- ClassificationIndexList contains new ClassificationIndexes  -->
<!ELEMENT ClassificationIndexList (ClassificationIndex)*>
```

D.1.5 自動分類の例

「seller」と「buyer」としての 2 つの役割を定義する CPP が申請されると仮定する。その CPP が申請されるとき、ClassificationIndex の classificationNode 属性において指定される ClassificationNode の子（たとえば、Role という名前のノード）である「seller」と「buyer」という名前の 2 つの ClassificationNodes によって、CPP は自動的に分類される。「seller」と「buyer」という名前の 2 つの ClassificationNodes がどちらも以前に存在しなかった場合は、ObjectManager が自動的にこれらの ClassificationNode を作成することに注意すること。

付録 E セキュリティ実装ガイドライン

本節では、セキュリティ処理をレジストリに実装できる方法に関する青写真の案を示す。本書で説明するるデフォルトのセキュリティ上の役割と権限付与規則がサポートされるかぎり、レジストリは異なる実装を選択してよい。

E.1 認証

1. メッセージ受信後の最初の作業は認証である。本人オブジェクトが作成される。

2. メッセージに署名がある場合、その署名が確認され(証明書の有効期限を含む)、証明書の DN が本人の身元となる。次に、レジストリで本人が検索され、それが見つかると、役割とグループが埋められる。

3. メッセージに署名がない場合、空の本人が役割 RegistryGuest を使って作成される。このステップは対称性を確保するため、そして残りの処理を切り離すためである。

4. 次に、コマンドとそのコマンドの実行対象であるオブジェクトに関してメッセージが処理される。

E.2 許可

オブジェクトごとに、アクセスコントローラはそのオブジェクトを含む AccessControlPolicy オブジェクトすべてで反復して実行し、要求されたメソッドが本人に対して許可されていることを確認するため、許可オブジェクトに共通するチェーンがあるかどうかを調べる。そのオブジェクトが関係付けられる許可オブジェクトのいずれかに本人と共通の役割、同一性、またはグループがある場合、アクションが許可される。

E.3 レジストリブートストラップ

レジストリが新規に登録されるときは、役割を RegistryAdmin とする Registry Admin の証明書 DN の ID を使ってデフォルトの Principal オブジェクトを作成する必要がある。このようにして、Registry Admin によって署名されたメッセージはすべての特権を取得する。

レジストリが新規に登録されるときは、AccessControlPolicy の単一インスタンスがデフォルト AccessControlPolicy として作成される。これには必要な Permission インスタンスのほか、特権および特権属性の作成が含まれる。

E.4 コンテンツの申請 - クライアントの責任

レジストリクライアントは申請の前にコンテンツに署名しなければならない。署名がない

場合、コンテンツは却下される。

E.5 コンテンツの申請 - レジストリの責任

1. 他の要求の場合と同様に、クライアントが最初に認証される。この場合、Principal オブジェクトは証明書から DN を取得する。

2. メッセージ内の要求に従って、RegistryEntry が作成される。

3. RegistryEntry に単一のデフォルト AccessControlPolicy が割り当てられる。

4. SO の ID を持つ本人が見つからない場合は、SO の DN を持つ ID オブジェクトが作成される。

5. この ID を持つ本人が作成される。

E.6 コンテンツの削除/廃止 - クライアントの責任

レジストリクライアントは認証のために申請前に搬送内容(メッセージ全体ではない)に署名しなければならない。それがないと、要求は却下される。

E.7 コンテンツの削除/廃止 - レジストリの責任

1. 他の要求の場合と同様に、クライアントが最初に認証される。この場合、Principal オブジェクトは証明書から DN を取得する。レジストリにはこの ID を持つ本人がいるため、Principal オブジェクトはそのオブジェクトからすべての役割を取得する。

2. メッセージ内の要求(削除または廃止)に従って、オブジェクト内の該当するメソッドを評価する。

3. アクセスコントローラは、単一のデフォルト AccessControlPolicy を通じてこのオブジェクトと関係付けられる Permission オブジェクトで繰り返しを実行することにより権限付与を実行する。

4. 権限付与に成功すると、アクションが許可される。さもなければ、エラー応答が適切な AuthorizationException エラーメッセージを付して返送される。

付録 F ネイティブ言語のサポート（NLS）

F.1 定義

本節では文字セットと言語のみについて説明するが、以下の用語を明確に定義しておく必要がある。

F.1.1 コード化文字セット（CCS）

CCS は抽象文字のセットから整数のセットへのマッピングである[RFC 2130]。CCS の例は、ISO-10646、US-ASCII、ISO-8859-1 などである。

F.1.2 文字コード化スキーム（CES）

CES は 1 つの(または複数の)CCS からオクテットのセットへのマッピングである[RFC 2130]。CES の例は ISO-2022、UTF-8 である。

F.1.3 文字セット（charset）

charset はオクテットのシーケンスから文字シーケンスへのマッピングである[RFC 2277], [RFC 2278]。文字セットの例は ISO-2022-JP、EUC-KR である。

登録済み文字セットの一覧は[IANA]に記載されている。

F.2 NLS および要求/応答メッセージ

レジストリクライアントとレジストリサービスの両方でデータ処理を正確に行うためには、使用している文字セットを知ることが重要である。トランザクションの本体部には xml 符号化宣言のなかに charset が含まれる場合があるが、レジストリクライアントとレジストリサービスはテキスト/xml を使用するときに MIME ヘッダで charset パラメータを指定するものとする。[RFC 3023]で定義されているように、charset パラメータを省略してテキスト/xml エンティティを受け取った場合、MIME プロセッサおよび XML プロセッサはデフォルトの charset 値"us-ascii"を使用しなければならない。

Ex. Content-Type: text/xml; charset=ISO-2022-JP

また、アプリケーション/xml エンティティが使用される場合、charset パラメータはオプションであり、レジストリクライアントとレジストリサービスは、その場合について直接規定する[REC-XML]の節 4.3.3 の要件に従わなければならない。

別の Content-Type を使用する場合、charset の使用は[RFC 3023]に従わなければならない。

F.3 NLS および RegistryEntry の格納

本節ではレジストリが RegistryEntry インスタンスを格納すべき方法に関する NLS ガイドラインを示す。

F.3.1 RegistryEntry の文字セット

RegistryEntry の格納に使用される実際の文字セットはインタフェースに影響しないため、これは基本的には実装の問題である。しかし、さまざまな言語を取り扱うためには、UTF-16 または UTF-8 を使用することが強く推奨される。

F.3.2 RegistryEntry の言語情報

言語は xml:lang 属性で指定できる（節 2.12 [REC-XML]）。xml:lang 属性が指定される場合には、レジストリはその言語コードを RegistryEntry 内の名前言語を持ち、sloType が nls である特別なスロットの値として使用できる。値は[RFC 1766]に適合しなければならない。スロットは[ebRIM]で定義される。

F.4 NLS およびリポジトリ項目の格納

本節ではレジストリがリポジトリ項目を格納すべき方法に関する NLS ガイドラインを示す。

F.4.1 リポジトリ項目の文字セット

RegistryEntry の文字セットと異なり、リポジトリ項目の charset はもともとトランザクションで指定されたものであるため、それを保存しなければならない。レジストリは、該当するリポジトリ項目の charset を格納するために、名前が repositoryItemCharset、sloType が nls の特別なスロットを RegistryEntry に対して使用できる。値は[RFC 2277]、[RFC 2278]で定義済みのものでなければならない。リポジトリ項目がすべて repositoryItemCharset を必要とするわけではないため、repositoryItemCharset はオプションである。

F.4.2 リポジトリ項目の言語情報

リポジトリ項目で使用されている言語を知るには、文字セットを 1 つ指定するだけでは十分ではない。レジストリは名前が repositoryItemLang、sloType が nls の特別なスロットを使用できる。リポジトリ項目がすべてこの属性を必要とするわけではないため、この属性はオプションである。値は[RFC 1766]に準拠するものでなければならない。

現在、本書は文字セットと言語の情報を送信する方法と、その情報がレジストリに格納される方法のみを指定する。しかし、言語情報はフランス語で書かれた DTD のみの検索など、クエリ基準の一つとして使用できる。さらに、レジストリクライアントがレジストリサービスからのメッセージにお気に入りの言語を指定するなど、言語ネゴシエーションプロシージャを本文書の将来のバージョンの追加機能とすることができる。

付録 G 用語の対応

他の作業で使用される用語と同じ用語を使用すべくあらゆる努力がなされてきたものの、一部に用語の違いがある。

下表は本仕様と他の仕様およびワーキンググループで使用される用語との間の用語の対応を示す。

本書	OASIS	ISO 11179
「リポジトリ項目」	Registered Object	
RegistryEntry	Registry Item	Administered Component
ExternalLink	Related Data	なし
Object.id	RegEntryId, orgId, etc.	
ExtrinsicObject.uri	objectURL	
ExtrinsicObject.objectType	defnSource, objectType	
RegistryEntry.name	CommonName	
Object.description	ShortDescription, Description	
ExtrinsicObject.mimeType	ObjectType=”mime” FileType=”mime type”	
Versionable.majorVersion	userVersion only	
Versionable.minorVersion	userVersion only	
RegistryEntry.status	registrationStatus	

表 1: 用語対応表

参考文献

[Bra97] Keywords for use in RFCs to Indicate Requirement Levels.
[GLS] ebXML Glossary, http://www.ebxml.org/documents/199909/terms_of_reference.htm
[TA] ebXML Technical Architecture
http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf
[OAS] OASIS Information Model
http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html
[ISO] ISO 11179 Information Model
http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument
[ebRIM] ebXML Registry Information Model
http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf
[ebBPM] ebXML Business Process Specification Schema
http://www.ebxml.org/specdrafts/Busv2-0.pdf
[ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
http://www.ebxml.org/project_teams/trade_partner/private/
[ebXML-UDDI] Using UDDI to Find ebXML Reg/Reps
http://lists.ebxml.org/archives/ebxml-regrep/200104/msg00104.html
[CTB] Context table informal document from Core Components
[ebMS] ebXML Messaging Service Specification, Version 0.21
http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf
[ERR] ebXML TRP Error Handling Specification
http://www.ebxml.org/project_teams/transport/ebXML_Message_Service_Specification_v-0.8_001110.pdf
[SEC] ebXML Risk Assessment Technical Report, Version 3.6
http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html
[XPT] XML Path Language (XPath) Version 1.0
http://www.w3.org/TR/xpath
[SQL] Structured Query Language (FIPS PUB 127-2)
http://www.itl.nist.gov/fipspubs/fip127-2.htm

[SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules (SQL/PSM)
[ISO/IEC 9075-4:1996]

[IANA] IANA (Internet Assigned Numbers Authority).
Official Names for Character Sets, ed. Keld Simonsen et al.
ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets

[RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:
Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html

[RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:
IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html

[RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:
IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html

[RFC 3023] IETF (Internet Engineering Task Force). RFC 3023:
XML Media Types, ed. M. Murata. 2001.
ftp://ftp.isi.edu/in-notes/rfc3023.txt

[REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)

http://www.w3.org/TR/REC-xml

[UUID]　DCE 128 bit Universal Unique Identifier
http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
http://www.opengroup.org/publications/catalog/c706.htmttp://www.w3.org/TR/REC-xml

免責

本書で表現されたビューおよび仕様は、著者のビューおよび仕様であり、必ずしも雇用者のビューおよび仕様を示すものではない。著者およびその雇用者は、この設計の正確なまたは不正確な実装または使用から生じるどのような問題に対しても責任を負わないことを明言する。

連絡先

チーム・リーダー
名前: スコット・ニーマン
会社: ノースタン・コンサルティング
住所: 5101 Shady Oak Road
住所: Minnetonka, MN 55343
国: USA
電話番号: 952.352.5889
電子メール: Scott.Nieman@Norstan

副チーム・リーダー
名前: ユタカ・ヨシダ
会社: サン・マイクロシステムズ
住所: 901 San Antonio Road, MS UMPK17-102
住所: Palo Alto, CA 94303
国: USA
電話番号: 650.786.5488
電子メール: Yutaka.Yoshida@eng.sun.com

編集者
名前: Farrukh S. Najmi
会社: サン・マイクロシステムズ
住所: 1 Network Dr., MS BUR02-302
住所: Burlington, MA, 01803-0902
国: USA
電話番号: 781.442.0703
電子メール: najmi@east.sun.com

著作権について

Copyright Statement