オントロジのアップデート管理と
セマンティックグリッドサービスに関する研究

内林　俊洋

平成26年1月

目次

図目次

表目次

第 1 章

序論

本章では、本研究の背景，研究目的，論文構成について述べる．

1.1 背景

インテル社の創立者の 1 人である Gordon Moore がムーアの法則で素子数は約 18 ヶ月ごとに倍増すると提唱しているように，プロセッサやメモリなどのハードウェアは非常に高性能になっている．さらに，パーソナルコンピュータの普及に伴う大量生産により，パーソナルコンピュータは従来に比べ低価格で入手することが可能になった．これらの影響はサーバ向け製品にも表れており，低価格でサーバを構築することが可能になった．

1983 年に DEC 社から VMScluster が出たことを発端とし，クラスタコンピューティングが提唱された．クラスタコンピューティングは，非常に高速なネットワークでコンピュータ間を接続して並列処理を行うことを目的としている [1][2]．それまでのスーパーコンピュータのように 1 台のコンピュータの性能を高めるのではなく，複数のコンピュータ資源をノードとして接続することで全体としての性能を高め性能の向上を図るものである．これは，コンピュータ資源が低価格になったことで実現した技術である．クラスタコンピューティングでは，並列コンピュータを利用するための標準化された規格である MPI (Message Passing Interface) [3] などを使用して並列計算を行うことができる．しかし，高速なネットワークを使用して資源を接続しなければならないため，インターネットが普及するまではキャビネット内など近い場所に資源を配置しなければならないという制約があった．

1990 年代になりインターネットが商用化されネットワークが発達するまでは，高価な専用回線を使用しなければ遠隔地の資源を利用することができず，遠隔地の資源を共有することは非現実的であった．インターネットが発達することでネットワーク速度も向上し，遠隔地の資源を共有する動きが強まりグリッドコンピューティングが登場した．1992 年に米国立スーパーコンピュータ応用研究所 (National Center for Supercomputing Applications)

の Charlie Catlett と Larry Smarr がメタコンピューティング [4] という概念を発表したことで，ネットワーク上に仮想的なコンピュータ環境を構築し大規模な並列処理を行うシステムの研究が加速した．グリッドコンピューティングの初期構想では，スーパーコンピュータやクラスタなどの計算資源のみを接続することが目的であった．しかし，現在のグリッドコンピューティングには計算資源だけでなくデータベースやセンサなどの様々な資源を接続することも可能になっている．グリッドコンピューティングの環境 (以下：グリッド環境) は Globus Toolkit[5] などの共通したミドルウェアを使用することで構築できる．グリッドコンピューティングで遠隔地の資源を共有することができるようになる．グリッド環境は，膨大なデータの保存や計算処理を行うには非常に有効であるが，少量のデータの保存や計算には不向きである．

2006 年に Google の CEO である Eric Emerson Schmidt が発言した “クラウドコンピューティング” という言葉を発端として，クラウドコンピューティングが急速に普及した．クラウドコンピューティングは，利用者が必要なだけの資源を自動的に割り振り，必要であればアップスケーリングを行えることが特徴である．クラウドはサービスとして提供され，SaaS(Software as a Service)，PaaS(Platform as a Service)，IaaS(Infrastructure as a Service) に分類できる．SaaS ではサーバの中にあるソフトウェアをクラウドのサービスとして提供している．PaaS は，プラットフォームを提供するサービスで，利用者にはシステム構築の基盤が提供される．IaaS はサーバやネットワーク機器などのインフラを提供する．利用者はインフラを意識することなく様々な処理を行うことができる．

また，グリッドコンピューティングやクラウドコンピューティングのような分散コンピューティング環境では，様々な資源も分散してネットワーク上に存在し，資源に関する情報の表現も統一されていない． そこで，この問題を解決するための方策の 1 つとしてオントロジと呼ばれる技術の応用が始まっている．オントロジは，データの表現の差異を吸収する目的で構築される．オントロジ自身もネットワーク上に分散して存在しているため，転送の効率化や維持管理が必要となる．

1.2 研究目的

本研究の目的は，オントロジのアップデート管理の効率化と使いやすいグリッドコンピューティング環境を実現することである．

オントロジのアップデート管理では，オントロジの更新に着目した．分散管理している同一のオントロジが更新された場合，オントロジを転送し同一性を保たなければならない．また，転送の効率化を行うためにサブオントロジを抽出した場合は，元のオントロジが更新された際に整合性が取れなくなる．本研究では，これらの問題の解決を行う．

グリッドコンピューティングを用いることでネットワークを介して複数のコンピュータを

結び，利用者がそこから必要な資源を取り出して使用できる．しかし，グリッドを利用するにはノードの情報を知った上でジョブを作成しなければいけないことが問題となる．そこで，利用者がグリッドの使用方法を熟知していなくてもグリッドを利用できるシステムの提案を行う．

1.3 論文構成

第 2 章ではオントロジのアップデート管理について，サブオントロジ抽出，サブオントロジテーラリングの説明とオントロジアップデート，サブオントロジアップデートの提案を行う．第 3 章ではオントロジのアップデート管理のケーススタディとして，クラウドサービス発見システムの提案を行う．第 4 章ではセマンティックグリッドサービスの提案を行う．第 5 章では結論を述べる．

第 2 章

オントロジのアップデート管理

本章では，サブオントロジ抽出，サブオントロジテーラリングの説明，オントロジアップデートの提案とサブオントロジアップデートの提案を行う．

2.1 オントロジ

オントロジとは，対象とする世界に存在するものごとの体系的な分類と，その関係を明示的・形式的に記述したものである [10]．オントロジは，データの表現の差異を吸収する目的で構築される．オントロジという言葉は，古代ギリシャ語の “存在” を意味する動詞の語幹である “オント” と “論” を意味する “ロジ” を組み合わせた “存在論” を意味している．哲学の分野では，存在とは何か存在しているとはどういうことかを問う学問である．しかし情報学では，2000 年に Tim Berners-Lee が提唱したセマンティックウェブ [6] におけるアーキテクチャの 7 階層の中の第 4 層として定義されたオントロジ層を指す (図 2.1)．セマンティックウェブとは，データの意味 (セマンティック) をタグを用いて記述し，オントロジを使用してコンピュータによる自動的な情報の収集を行うことを目的としたものである．

ウェブをはじめとした文書検索において，従来の方法では単語単位での一致，もしくは類義語を含む文書を検索するのが限度であった．しかし，オントロジを用いることで各文書について統一的な付加情報を持たせる事ができ，高度な検索を行うことが可能になる．文書内にメタデータを付加することによって，検索対象となる文書が，単なる単語の集まりとしてではなく，文書全体で大きな意味を持ったデータとして扱われる．

2.2 オントロジの構成

オントロジは，XML を拡張した OWL (Web Ontology Language)[11] で記述を行う．OWL ではクラス，プロパティの集合としてオントロジを記述する．OWL は，人間に対して情報を提示するのではなく，アプリケーションが情報の内容を処理しやすいように設計さ

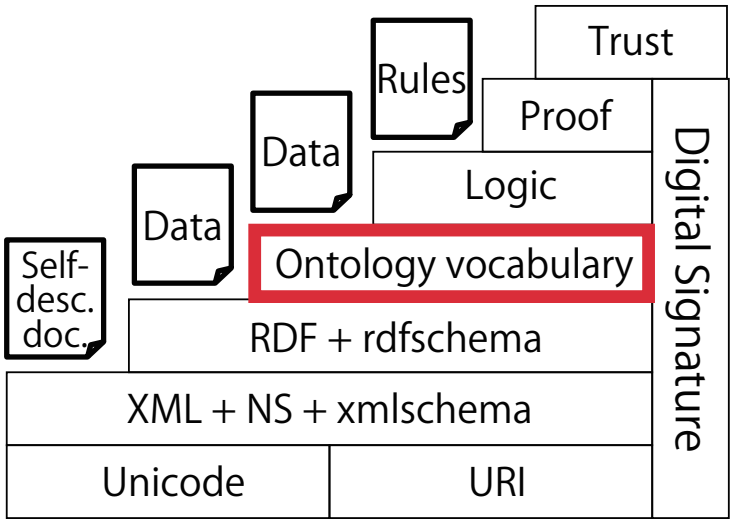


図 2.1 セマンティックウェブのアーキテクチャ

れている．クラスは，類似する特徴を持つものをグループ化したものであり，外延と内包を持つ．外延はそのクラスに属するインスタンスと呼ばれる個体の集合であり，内包はそのクラスの意味となる基礎の概念である．プロパティには，個体と個体の関連付けを行うオブジェクトプロパティと，個体と数値や文字列を関連付けするデータプロパティがある．そして， クラスがどのプロパティに属しているかを表す属性マッピングがある．

2.2.1 クラス

OWL では一般名詞に相当する用語は概念として扱われ，クラス名として定義を行う．owl:Thing クラス及び owl:Nothing クラスの 2 つの OWL のクラスが事前に設定されている．すべての OWL のクラスの最上位クラスは owl:Thing であり，最下位クラスは owl:Nothing となる．クラス記述は，rdf:ID 属性の値にクラス名を持った owl:Class によって表される．次の例では，自動車という名前を持つクラスを記述している．

```
<owl:Class rdf:ID="自動車" />
```

クラスの定義をクラス公理と呼ぶ．別のクラスの記述をクラス公理と結び付けることで詳細なクラスの定義が可能となる．

2.2.1.1 下位クラス公理

RDF スキーマ [12] の構成要素 rdfs:subClassOf を用いて下位クラスの関係を表す．RDF とは，ウェブ上にある資源を記述するための統一された枠組みである．次の例では，トラックが自動車クラスの下位クラスであることが記述されている．#は，すでに定義されている

クラスやプロパティを参照することを意味する．複数のクラスの下位クラスである場合があるため，1 つのクラスに対して下位クラス公理は複数記述してもよい．

```
<owl:Class rdf:ID="トラック">
    <rdfs:subClassOf rdf:resource="#自動車" />
</owl:Class>
```

2.2.1.2 等価クラス公理

owl:equivalentClass を用いて，2 つのクラスが等価であることを表す．次の例では，トラッククラスと Truck クラスが等価であることを示している．

```
<owl:Class rdf:ID="Truck">
    <owl:equivalentClass rdf:resource="#トラック" />
</owl:Class>
```

2.2.1.3 排他クラス公理

owl:disjointWithClass を用いて，2 つのクラスが互いに素であることを表す．次の例では，Track クラスと Truck クラスが互いに素であることを示している．

```
<owl:Class rdf:ID="Track">
    <owl: disjointWithClasss rdf:resource="#Truck" />
</owl:Class>
```

2.2.2 プロパティ

プロパティには，あるクラス定義と別のクラス定義を関連付けるオブジェクトプロパティと，あるクラス定義を数値や文字列と関連付けるデータプロパティがある．オブジェクトプロパティは owl:ObjectProperty を用いて記述する．データプロパティは owl:DataProperty を用いて記述する．これらのプロパティは rdf:Property の下位クラスである．プロパティはプロパティ公理によって，特徴の定義が行われる．

2.2.2.1 下位プロパティ公理

RDF スキーマの構成要素である rdf:subProperty を用いて，あるプロパティが別のプロパティの下位プロパティであることを表す．次の例では，製造会社プロパティは詳細情報プロパティの下位プロパティであると記述している．

```
<owl:ObjectProperty rdf:ID="製造会社">
    <rdfs: subPropertyOf s rdf:resource="#詳細情報" />
```

```
</owl:Class>
```

2.2.2.2 プロパティ定義域公理

RDF スキーマの rdfs:domain を用いて，プロパティとクラスの関係を表す．次の例では，製造会社プロパティがトラッククラスに属していることを示している．

```
<owl:ObjectProperty rdf:ID="製造会社">
    <rdfs: domain>
        <owl:Class rdf:about="#トラック" />
    </ rdfs:domain>
</owl:Class>
```

2.2.2.3 プロパティ値域公理

RDF スキーマの構成要素である rdfs:range を用いて，プロパティとクラスの関係を表す．次の例では，トラッククラスが製造会社プロパティに属していることを示している．

```
<owl:ObjectProperty rdf:ID="製造会社">
    <rdfs: range>
        <owl:Class rdf:about="#トラック" />
    </ rdfs:range>
</owl:Class>
```

2.2.2.4 等価プロパティ公理

owl:equivalentClass を用いて表す．クラスの等価と同様に，2 つのプロパティが等価であることを示す．次の例では，製造会社プロパティとメーカープロパティが等価であることを示している．

```
<owl:ObjectProperty rdf:ID="メーカー">
    <owl:equivalentClass rdf:about="#製造会社" />
</owl:Class>
```

2.2.2.5 逆プロパティ公理

owl:inverseOf を用いて表す．プロパティには定義域から値域への方向性が存在する．両方向に関係を定義するのが有用であることが多いため，プロパティ間の逆の関係を定義するために使用される．次の例では，製造会社プロパティと製造自動車プロパティは逆の関係であると示している．

```
<owl:ObjectProperty rdf:ID="製造自動車">
```

```
    <owl:inverseOf rdf:about="#製造会社" />
</owl:Class>
```

2.2.3 集合

既存のクラスの集合演算を用いてクラスを記述する．クラスの集合演算には複数クラスの積集合，和集合，そして単一クラスの補集合がある．

2.2.3.1 積集合

rdf:parseType 属性の値が Collection である owl:intersectionOf の子要素に積集合をとるクラス記述を列挙することにより，それらのクラスのすべての外延に属する個体からなるクラスを表す．

```
<owl:Class">
    <owl:interSectionOf rdf:parseType="Collection" >
        <owl:Class rdf:about="#c1" />
        <owl:Class rdf:about="#c2" />
    <owl:interSectionOf />
</owl:Class>
```

2.2.3.2 和集合

rdf:parseType 属性の値が Collection である owl:UnionOf の子要素に和集合をとるクラス記述を列挙することにより，それらのクラスの外延のうち少なくとも 1 つに属する個体からなるクラスを表す．

```
<owl:Class">
    <owl:UnionOf rdf:parseType="Collection" >
        <owl:Class rdf:about="#c1" />
        <owl:Class rdf:about="#c2" />
    <owl:interSectionOf />
</owl:Class>
```

2.2.3.3 補集合

owl:complementOf の子要素に 1 つのクラス記述を持たせることにより，そのクラスの外延に属さない個体からなるクラスを表す．

```
<owl:Class">
    < owl:complementOf >
        <owl:Class rdf:about="#c1" />
```

```
    <owl: complementOf />
</owl:Class>
```

2.3 オントロジの管理

多くの場合，個々のオントロジは独立した状態で管理されている．薬や病気などの医療情報を格納した医療オントロジが典型的な例である．医療オントロジを使用することで，ある患者の病気に関して病気情報を検索すると，その病気の情報の他にも治療に有効な薬の情報を知ることができる．病気情報の病気オントロジや薬情報の薬オントロジなどはネットワーク上に分散して保存・管理されている．利用者がオントロジを利用する際には，利用者の手元に転送しなくてはならない．オントロジをそのまま転送するのは効率が悪い．そこで，転送の効率化のためにサブオントロジを用いる技術が提案されている [13]．サブオントロジとは，オントロジの構造を崩さずに必要な部分のみを抽出したオントロジである．サブオントロジ抽出の詳細は 2.4.1 節で述べる．

病気オントロジと薬オントロジを組み合わせて検索するなど，利用者が複数のオントロジを組み合わせて検索する場合がある．その場合，サブオントロジテーラリングを使用する．サブオントロジテーラリングとは，複数箇所に異なるオントロジが分散している場合に，サブオントロジを整合的に結合するという方法である．サブオントロジテーラリングの詳細は 2.4.2 節で述べる．

ネットワーク上に同一内容のオントロジのコピーが存在する場合，同一性を維持する必要がある．この問題を解決するために，本研究ではオントロジアップデートを提案する．オントロジアップデートは，オントロジのコピーの 1 つが更新された場合，オントロジ自体を再度転送して同一性を維持する代わりに，アップデートパッチを転送して同一性を維持する．アップデートパッチは，オントロジの変更内容を記述したものである．

元のオントロジが更新された場合，利用者の持っているサブオントロジは古い情報となってしまう．通常は再度サブオントロジの抽出を行うが，それでは効率が悪い．そこで本研究はサブオントロジアップデートを提案する．サブオントロジアップデートは，アップデートパッチとアップデートメカニズムを使用して更新前のサブオントロジから更新後のサブオントロジを直接生成する．アップデートメカニズムは，オントロジの完全性や一貫性を保つように補完するための仕組みである．2.7 節で，本研究で提案するオントロジアップデートとサブオントロジアップデートの詳細を述べる．

2.4 関連研究

本節では関連研究の説明を行う．

2.4.1 サブオントロジ抽出

Bhatt らが提案しているサブオントロジ抽出 [13] では，オントロジからサブオントロジを抽出する．抽出したサブオントロジを転送することで効率化を図っている．彼らの研究では，MOVE (Materialized Ontology View Extraction) としてサブオントロジ抽出が定義されている．MOVE では，オントロジを構成するクラスとプロパティの ID の値にラベル付けを行う．ラベルの情報を基に抽出されたものがサブオントロジである．サブオントロジは，クラスやプロパティの関係が抽出元のオントロジと同様でなければならない．サブオントロジの抽出過程では，4 つの最適化スキーマ RCOS (Requirements Consistency Optimization Scheme), SCOS (Semantic Completeness Optimization Scheme), WFOS (Well Formedness Optimization Scheme)，TSOS (Total Simplicity Optimization Scheme) を利用することで抽出を行っている．サブオントロジを抽出することで，オントロジを小さくすることができる．小さくすることで，ネットワーク越しにオントロジを転送する際のデータ量を削減ができることが利点である．

一貫性とは，ユーザによるラベル付けに関する基準であり RCOS により操作的に定義されている．完全性はクラスやプロパティに関する基準であり SCOS により操作的に定義されている．WFOS はラベル付けに関する基準であるが，意味が一貫しているとは保証されていない．TSOS は抽出されたサブオントロジをさらに最小化するためのスキーマであるが，必ずしも最小であると保証されていない．

2.4.2 サブオントロジテーラリング

サブオントロジテーラリングとは，2 つのオントロジから 1 つのサブオントロジを生成する方法である．通常は分散しているオントロジをマージしてサブオントロジを抽出する．サブオントロジテーラリングを使用することで，分散しているオントロジを融合せずにサブオントロジを生成することができる．サブオントロジテーラリングの方法として，Flahive の提案した”Reuse, Extract and Add”と”Reuse, Extract and Merge”がある [14]．サブオントロジ抽出では，1 つのオントロジから 1 つのサブオントロジの抽出を行った．しかし，必ずしも 1 つのオントロジ内に必要なすべての情報が含まれているとは限らず，複数のオントロジから 1 つのサブオントロジを生成する場合も想定される．3 つ以上のオントロジがある場合も，上記 2 つの方法を組み合わせることでサブオントロジを生成することができる．

2.4.2.1 サブオントロジテーラリング方法

サブオントロジテーラリングを行うために，元となる 2 つのオントロジのクラスとプロパティにはサブオントロジ抽出と同様に，あらかじめ同じラベルがラベル付けがされている

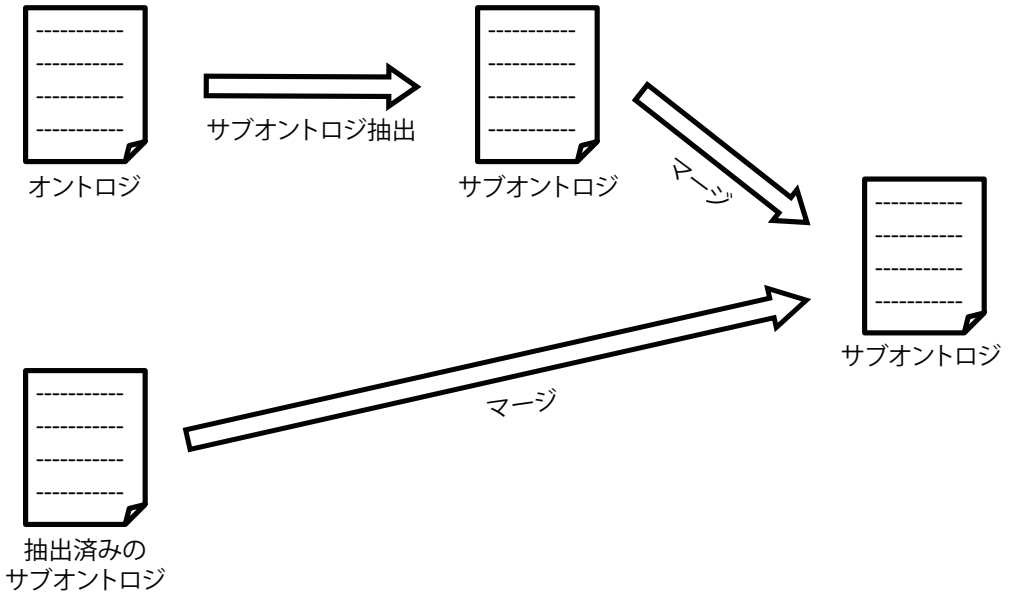


図 2.2 Reuse, Extract and Add

ものとする．サブオントロジテーラリング方法は “Reuse，Extract and Add” と “Reuse, Extract and Merge” の 2 つがある．

2.4.2.2 Reuse, Extract and Add

片方のオントロジからサブオントロジを抽出し，抽出されたサブオントロジと残りの抽出済みのサブオントロジをマージし 1 つのサブオントロジを生成する (図 2.2)．この方法は，オントロジと抽出済みのサブオントロジからサブオントロジを抽出する場合を想定している．

2.4.2.3 Reuse, Extract and Merge

2 つのオントロジのサブオントロジテーラリングを行う． それぞれサブオントロジの抽出を行い，抽出されたサブオントロジをマージして 1 つのサブオントロジの生成を行う (図 2.3)．この方法は，オントロジとオントロジからサブオントロジを抽出する場合を想定している．

2.5 サブオントロジ抽出の改善

本研究は，サブオントロジ抽出を単純化するために保証されていない WFOS と TSOS を削除する [7]．また，RCOS を Consistency Validation Checking (CnV)，SCOS を Completeness Validation Scheme (CmS) とした．

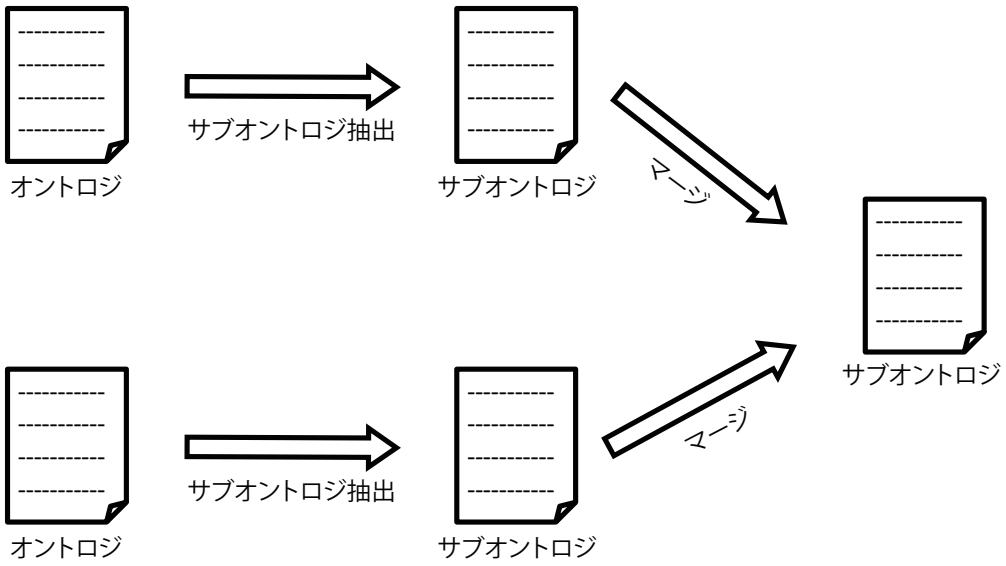


図 2.3 Reuse, Extract and Merge

2.5.1 抽出メカニズム

サブオントロジ抽出処理の流れの例を図 2.4 に示す．サブオントロジ抽出を行う前に，ユーザがあらかじめ抽出したいオントロジの要素にラベル付けを行う．ラベル付けでは，クラスとプロパティの ID の値を要素と呼びラベル付けを行う．サブオントロジに必要な要素であれば “選択” ラベル，サブオントロジに必要でない要素であれば “除外” ラベル，それ以外の要素は “未決定” ラベルとなる．サブオントロジ抽出には，要素の取り出しと要素の補完の 2 つの工程がある．要素の取り出し工程では，“選択” ラベルの要素のみを取り出し，“除外” ラベルの要素と “未決定” ラベルの要素は取り出さない．要素の補完工程では，最適化スキーマを使用し，必要であれば “未決定” ラベルの要素から補完する．最適化スキーマは 2.5.2 で紹介する．これらの工程によって，CmS によるクラスやプロパティの完全性が保たれたサブオントロジが抽出される．そして，“選択” ラベル要素の数にもよるが，サブオントロジ抽出を行うことで元のオントロジと比べて小さくすることができる．

2.5.2 最適化スキーマ

サブオントロジ抽出処理を行う過程で，最適化スキーマを満たすように要素の補完を行う．最適化スキーマを使用することで，サブオントロジの完全性を保証する．最適化スキーマを満たさない場合は完全なサブオントロジではない．

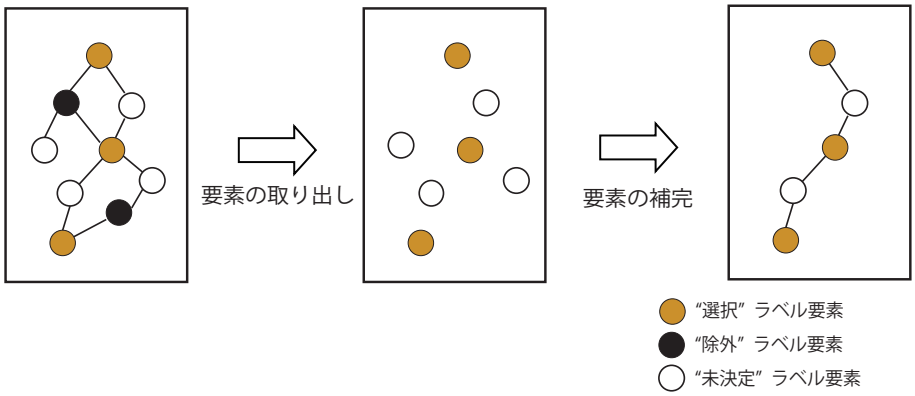


図 2.4 サブオントロジ抽出例

2.5.2.1 Consistency Validation Checking (CnV)

CnV は，ユーザによるオントロジ内のクラスやプロパティのラベル付けに一貫性があるかの確認を行う．ラベル付けの一貫性を保つことによって，サブオントロジ内の最下位のクラスやプロパティの ID から最上位のクラスやプロパティへの経路が存在しない可能性を排除する．CnV は 4 つサブスキーマ CnV(1)-CnV(4) の集合体である．

- CnV(1)
  オントロジ内のラベル付けされた 2 つの要素が等価クラス公理で定義されており，片方の要素が “選択” ラベルでラベル付けされている場合，もう一方の要素も “選択” ラベルでなければならない．

- CnV(2)
  あるクラスが “選択” ラベルでラベル付けされており属性マッピングに属しているなら，属性マッピングのプロパティも”選択”ラベルでなければならない．CnV(2) の規則は CnV(1) と似ているが，クラスがどのプロパティに属しているかという属性マッピングで適用されるということが異なる．

- CnV(3)
  この規則は属性マッピングの特定の特性に要件を課すものである．属性マッピングに “除外” ラベルでラベル付けされたクラスが存在するなら，プロパティも “除外” ラベルでなければならない．

- CnV(4)
  CnV(4) は CnV(1)-CnV(3) より複雑である．CnV(4) では，経路の考え方を利用す

る．クラスに関係が存在する場合，クラスにはプロパティで定義されたクラスとクラスまたはクラスとプロパティの経路が存在する．オントロジの記述で経路の存在は明示されていないが，経路があることで関係が成立している．経路が連鎖することによりオントロジを形成しており，同じ経路はオントロジ内に 1 つしか存在しない．プロパティが”選択”ラベルでラベル付けされており，属するクラスが “除外” ラベルでラベル付けされている場合，“未決定” ラベルのクラスへ経路が存在しなければいけない．

2.5.2.2 Completeness Validation Scheme (CmS)

オントロジ内のクラスやプロパティの関係に完全性があるかの確認を行う．サブオントロジの抽出のためには，“選択” ラベルの要素のための定義要素が必要となる．定義要素とは，そのクラスの意味を示すために不可欠なクラスやプロパティである．定義要素があることで，“選択” ラベルの要素の意味が定義される．例えば，定義要素がなければそのクラスはサブオントロジ内で孤立することになり，不完全なサブオントロジとなってしまう．CmS は 3 つの最適化スキーマ CmS(1)-CmS(3) の集合から成る．

- CmS(1)
  クラスを “選択” ラベルでラベル付けするなら，すべての上位クラスは”選択”ラベルでラベル付けしなければならない．

- CmS(2)
  クラスを “選択” ラベルでラベル付けするなら，積集合や和集合，そして補集合などの集合関係と共に含まれるすべてのクラスを “選択” ラベルでラベル付けしなければいけない．

- CmS(3)
  クラスを “選択” ラベルでラベル付けするなら，そのクラスのすべてのプロパティと属性マッピングを “選択” ラベルでラベル付けしなければいけない．

2.6 オントロジアップデートの提案

2.3 で述べたように，ネットワーク上に同一内容のオントロジのコピーが存在する場合，同一性を維持する必要がある．本研究は，オントロジアップデートを提案する．オントロジ

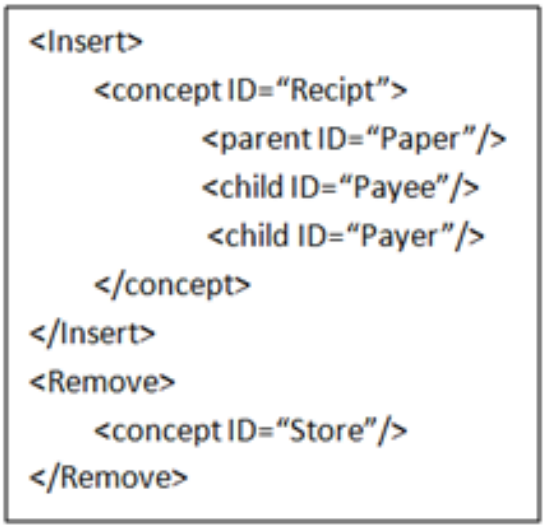


図 2.5 アップデートパッチの例

アップデートは，ネットワーク上のオントロジの同一性の維持を効率的に行う．オントロジ自体を再度転送する代わりにオントロジの変更内容を記述したアップデートパッチを転送する方式である．

アップデートパッチは XML で作成され，クラスとプロパティに関するアップデート内容が記述されている．アップデートパッチの要素は，クラスおよびプロパティの追加 (Insert)・削除 (Remove)・置換 (Replace) となる．それぞれ ID 属性だけがあり，対象を指定する．追加を指定する Insert 要素の子要素はクラスを指定する concept 要素とプロパティを指定する property 要素がある．それぞれの要素の子要素に上位クラスや上位プロパティを指定する parent 要素と下位クラスや下位プロパティを指定する child 要素がある．削除を指定する Remove 要素の子要素は concept 要素と property 要素がある．置換を指定する Replace 要素の子要素は concept 要素と property 要素がある．それぞれの要素の子要素に parent 要素と child 要素がある．図 2.5 に例を示す．オントロジデータ内の追加を示す Insert において，Receipt クラスを Paper クラスの下位に配置し，Payer クラスと Payee クラスの上位クラスとすることが記述してある．オントロジデータ内の削除を示す Remove において Store クラスをオントロジデータから削除することが記述してある．

2.7 サブオントロジアップデートの提案

2.3 で述べたように，元のオントロジが更新された場合，利用者の持っているサブオントロジは古い情報をなってしまう．通常は再度オントロジの抽出を行うが，それでは効率が悪いという問題がある．本研究は，以前抽出したサブオントロジから再度サブオントロジ抽出処理を行わずにアップデート後のサブオントロジを直接生成するアップデートメカニズムを

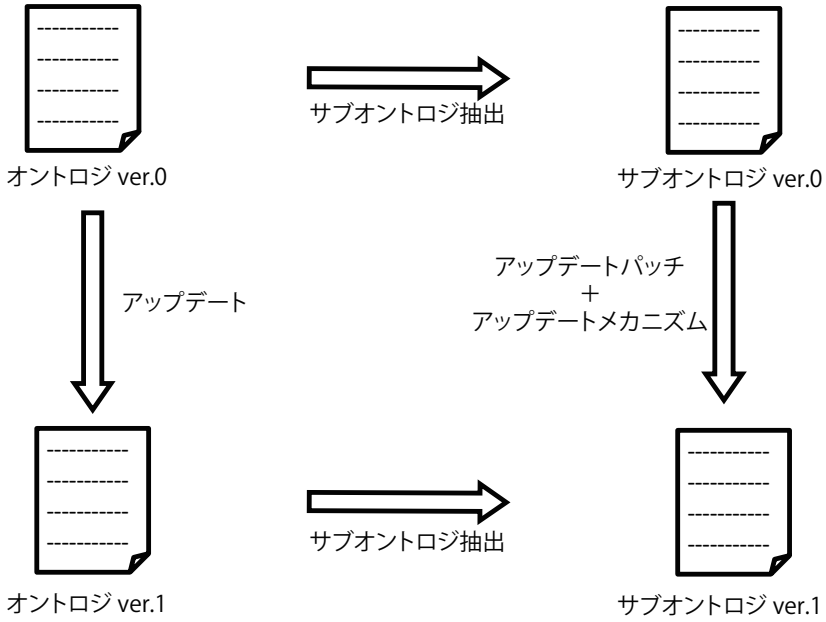


図 2.6 サブオントロジアップデートの流れ

提案する [8][9]．これは，オントロジがアップデートされた際に，サブオントロジ抽出をやり直さなければならないという点に注目したものである．図 2.6 は，元のオントロジがアップデートされた後のサブオントロジの生成の 2 つの流れを示している．従来は元のオントロジがアップデートされた場合，アップデートされたオントロジ ver.1 から再度サブオントロジの抽出を行いサブオントロジ ver.1 を生成しなければならなかった．提案するアップデートメカニズムとアップデートパッチを使用することで，以前抽出したサブオントロジ ver.0 から直接サブオントロジ ver.1 を生成することができる．サブオントロジアップデートを使用することで，サブオントロジ抽出を再度せずに，変更点のみをアップデートすることができる．

2.7.1 概要

2.6 節で提案したアップデートパッチを，以前抽出したサブオントロジに適用するだけではうまくサブオントロジ内の構成が変更されない．例えば，アップデートされた要素がサブオントロジ抽出時に元のオントロジで選択されていない場合，サブオントロジにその要素は存在せず，構造が壊れてしまう可能性がある．

本研究は，アップデートメカニズムを通してアップデートパッチを適用することでその問題を解決する．元のオントロジを $Ontology\_V0$ とする．サブオントロジ抽出処理を $Extract()$ とし，$Ontology\_V0$ から抽出したサブオントロジを $SubOntology\_V0$ とする．

$$SubOntology\_V0 = Extract(Ontology\_V0)$$

アップデート情報をアップデートパッチとして保存したものを $Update\_V1$ とする．$Ontology\_V0$ がアップデートされたオントロジを $Ontology\_V1$ とする．

$$Ontology\_V1 = Ontology\_V0 + Update\_V1$$

提案するアップデートメカニズムの処理を $UpdateMechanism()$ とする．アップデートされたサブオントロジを $SubOntology\_V1$ とすると，次の 2 つの方法がある．

$$SubOntology\_V1 = Extract(Ontology\_V1)$$
$$SubOntology\_V1 = UpdateMechanism(SubOntology\_V0, Update\_V1)$$

サブオントロジアップデートを行うための，アップデートメカニズムの提案を行う．元のオントロジのアップデート内容を記録したアップデートパッチを基に，アップデートメカニズムを使用してアップデート処理を行う．

2.7.2 アップデートメカニズム

アップデートメカニズムは，アップデートパッチを使用してオントロジの構造を維持しつつアップデートを実行するためのメカニズムを提案する．アップデートパッチの内容を読み込み，すべてのアップデートパッチの項目に対して処理を行う．以下にその詳細を示す．

- クラスの削除
  クラスが存在するかチェック．上位クラスがある場合はなにもしない．それ以外の場合は該当クラスを削除.
- プロパティの削除
  プロパティが存在するかチェック．プロパティが存在する場合は該当プロパティを削除.
- クラスの置換
  クラスが存在するかチェック．該当クラスに上位概念クラス，下位クラス共に存在するならば該当クラスを削除し上位クラスの下位に該当クラスの下位クラスを再配置.

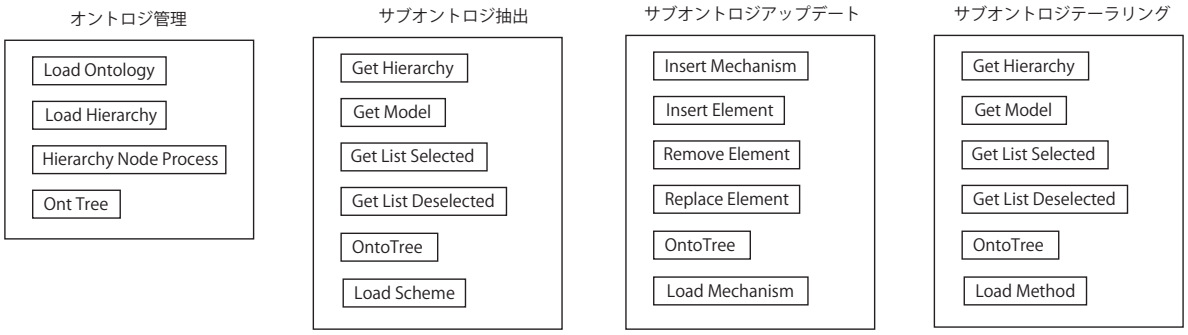


図 2.7 アプリケーションの構造

それ以外の場合はそのまま削除する．アップデートパッチに記述してある置換先に上位クラスか下位クラスのみが存在する場合はなにもしない．上位クラスと下位クラスが共に存在するならば，アップデートパッチに記述してある上位クラスと下位クラスの間に該当クラスを追加する．

- プロパティの置換
  プロパティ内のクラスをアップデートパッチに記述してあるクラスに置き換える．
- クラスの追加
  アップデートパッチに記述してある上位クラスか下位クラスのみが存在する場合はなにもしない．上位クラスと下位クラスが共に存在するならば，アップデートパッチに記述してある上位クラスと下位クラスの間に該当クラスを追加する．
- プロパティの追加
  プロパティ内の概念が存在するならば，該当クラスにプロパティを追加する．

2.8 ケーススタディと実験

サブオントロジ抽出，サブオントロジテーラリングの手法を紹介し，オントロジアップデートとサブオントロジアップデートの提案を行った．以下，サブオントロジ抽出とサブオントロジテーラリングの詳細をケーススタディによって説明し，サブオントロジアップデートに関して実験を行う．図 2.7 はケーススタディと実験に使用するアプリケーションの構造を示している．

アプリケーションの機能は，オントロジ管理，サブオントロジ抽出，サブオントロジアップデート，サブオントロジテーラリングの 4 つに分けられる．

オントロジ管理は，サブオントロジ抽出やサブオントロジアップデートにおけるオントロジの読み込みや書き込み，階層構造への変換などの処理を行う．オントロジの読み込みや書き込み，クラスやプロパティの取得には Apache Jena[15] を使用している．

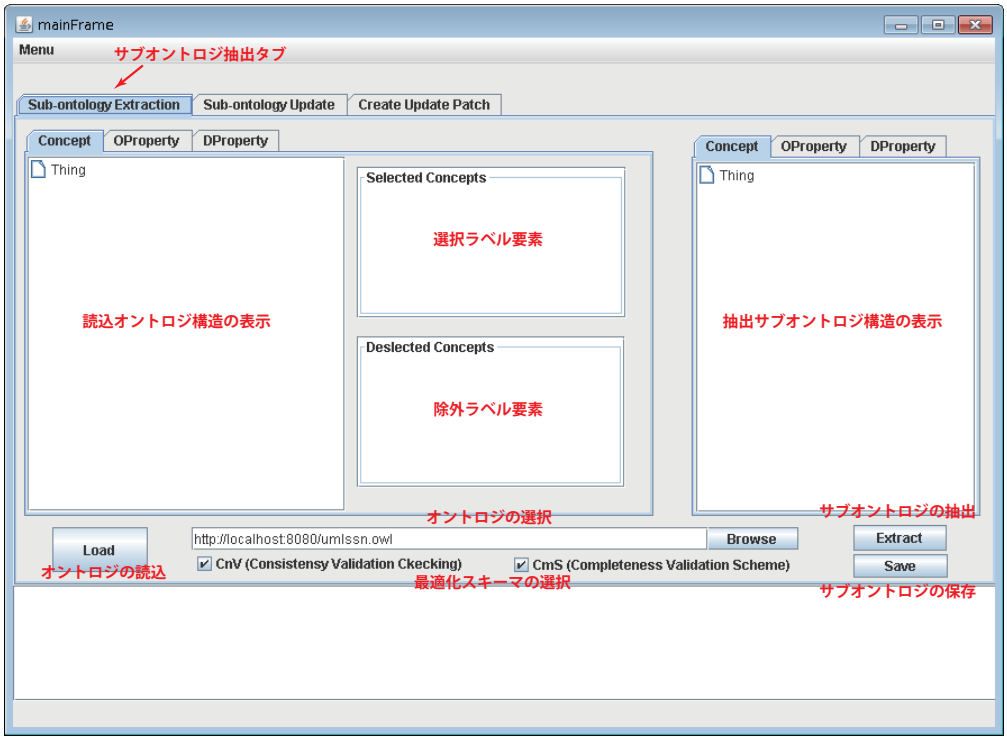


図 2.8 サブオントロジ抽出画面

サブオントロジ抽出とサブオントロジテーラリングでは，アプリケーションの使用者がラベル付けしたリストを基に，最適化スキーマを使用してサブオントロジの抽出や生成を行う．

サブオントロジアップデートには，アップデートパッチ作成とアップデート処理の 2 つの機能がある．アップデートパッチの生成では，階層構造で表示されたオントロジ構造からクラスやプロパティの追加・削除・置換を行うことができる．アップデート処理では，オントロジとアップデートパッチからオントロジのアップデートを行う．

実装したアプリケーションは機能ごとにタブ分けされており，使用したいタブを表示して各機能を使う．サブオントロジ抽出を行う場合は Sub-ontology Extraction タブ，サブオントロジアップデートを行いたい場合は Sub-ontology Update タブ，アップデートパッチの作成は Create Update タブを使用する．また，サブオントロジテーラリングはコマンドラインでのみ実装した．

2.8.1 サブオントロジ抽出

Sub-ontology Extraction タブ (図 2.8) は，サブオントロジ抽出を行うためのタブである．

サブオントロジ抽出を行うために最適化スキーマを満たすための操作の実装を行った．以下にその詳細を示す．

- CnV(1)
  “選択” ラベルのクラスに等価クラス公理が設定されているかを確認．設定されている場合，等価なクラスが “選択” ラベルであるか確認する．“選択” ラベルでない場合は等価なクラスを “選択” ラベルへ変更する．

- CnV(2)
  “選択” ラベルのプロパティに属しているクラスを確認．クラスが “未決定” ラベルあれば “選択” ラベルへ変更する．

- CnV(3)
  “未決定” ラベルのプロパティに属しているクラスを確認．クラスが “除外” ラベルであればプロパティを “除外” ラベルへ変更する．

- CnV(4)
  “選択” ラベルのプロパティに属しているクラスを確認．クラスが “除外” ラベルであれば下位クラスまたは “除外” ラベルのクラスを “選択” ラベルへ変更する．

- CmS(1)
  “選択” ラベルのクラスを確認．上位のクラスを “選択” ラベルへ変更する．

- CmS(2)
  “選択” ラベルのクラスを確認．クラスに集合関係があれば，集合関係にあるクラスを “選択” ラベルに変更する．

- CmS(3)
  “選択” ラベルのクラスを確認．クラスのプロパティと属性マッピングをすべて “選択” ラベルに変更する．

画面下の Browse ボタンでオントロジの場所を指定し Load ボタンでオントロジの読み込みを行う．読み込みが正常に行われると，画面左のエリアに階層状にオントロジが変換されて表示される．階層表示されたクラスやプロパティを右クリックすることでラベル付けをすることができる (図 2.9)．“選択” ラベルのクラスは画面中央の Selected Concept に，“除外” ラベルのクラスは Deselected Concept にリストで表示される．プロパティも同様にラベル付けを行うことができる．画面中央下の CnV と CmS のチェックボックスをチェックすることで，デバッグ用にスキーマ使用の有無を決定することができる．画面右下

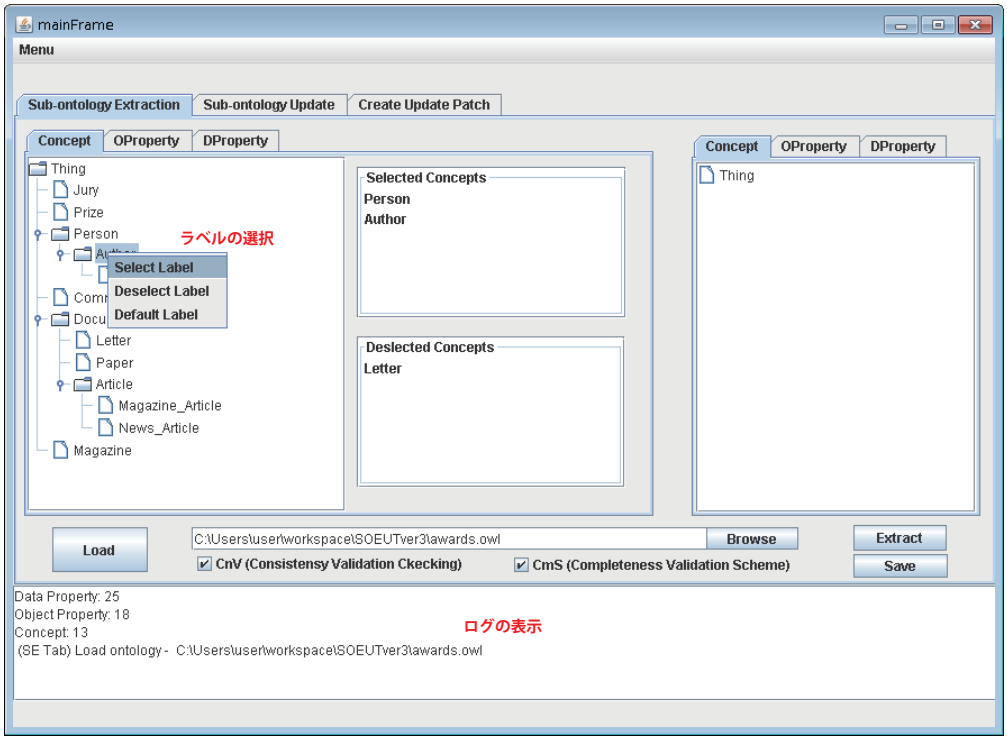


図 2.9 サブオントロジ抽出要素選択画面

の Extract ボタンを押すことでサブオントロジの抽出が行われ，画面右に結果が階層状に表示される．実際にサブオントロジ抽出を行った結果を図 2.10 に示す．

UMLSSN(Unified Medical Language System Semantic Network) オントロジを使用したサブオントロジ抽出の動作の紹介を行う．UMLSSN は，知識の取得プロセスを固定する共通セマンティックフレームワークを提供する生物医学科学のオントロジである．“選択” と “除外” のラベル付けを行ったクラスとプロパティのリストを、表 2.1 に示す．また，UMLSSN のクラスとプロパティ数を表 2.2 に示す．これらの “選択” と “除外” のラベル付けを行ったクラスとプロパティでのサブオントロジ抽出を行った場合の，CnV と CmS の 2 つのスキーマのサブスキーマの処理を STEP ごとに紹介する．

2.8.1.1 CnV の動作

抽出されたサブオントロジが整合するか否かを確認し，そうでない場合は補完を行う．CnV は，CnV(1)～CnV(4) の 4 つのサブスキーマで構成されている．

図 2.11 は，CnV(1) の流れを示している．表 2.1 に基づいて，STEP1 では “選択” ラベルと “除外” ラベルの要素を決定する．STEP2 は，“選択” ラベルのクラスのみ抽出する．STEP3 では，等価クラスが “選択” ラベルで補完されている．結果を図 2.12 に示す．

CnV(2) の処理の流れを図 2.13 に示す．STEP1 では，表 2.1 に基づいて “選択” ラベル

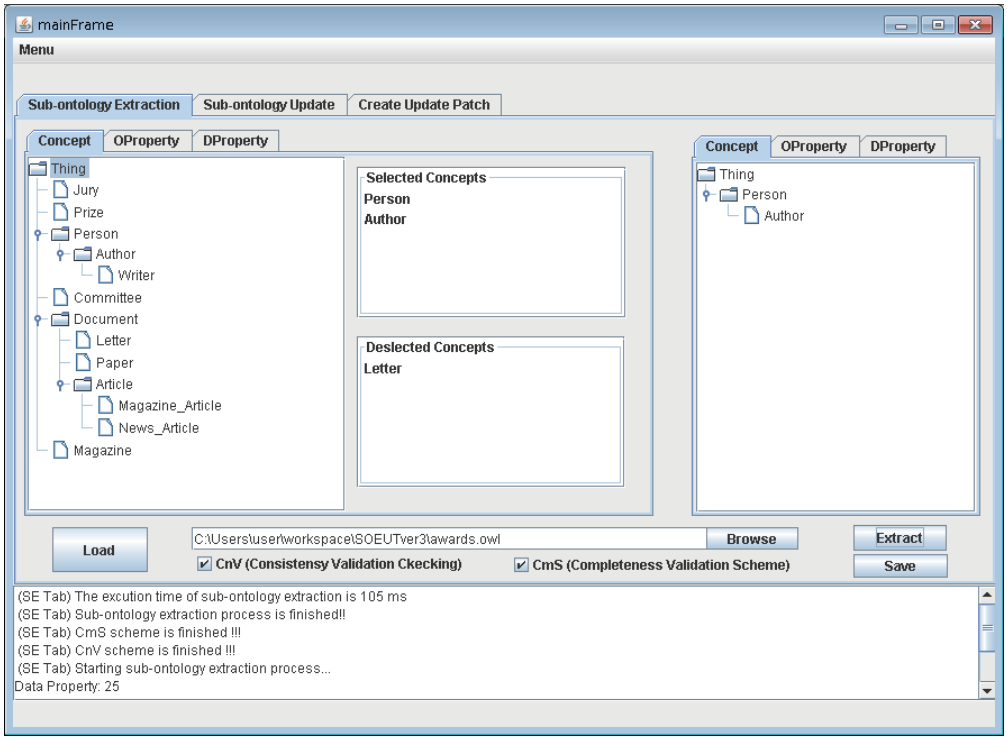


図 2.10 サブオントロジ抽出結果画面

表 2.1 選択・除外を行うクラスとプロパティ

	選択	除外
クラス	Research_Activity Geographic_Area Behavior_T053	Occurs_in_T152 Complicates_T149
プロパティ	Organization_T92 Antibiotic_T195	Treats_T154

表 2.2 UMLSSN の構成

クラス	オブジェクトプロパティ	データプロパティ
135	0	65

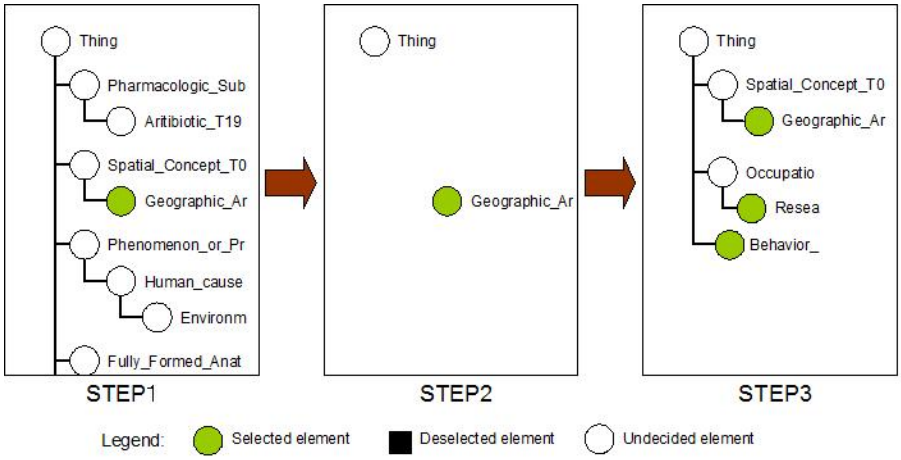


図 2.11 CnV(1) の処理の流れ

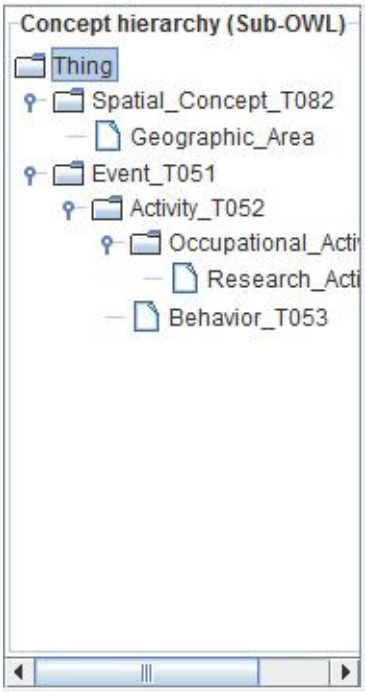


図 2.12 CnV(1) の適用結果

と “除外” ラベルの要素を決定する．STEP2 では，“選択” ラベルのクラスが持つ属性マッピングのクラスが取り出される．STEP3 では，“選択” ラベルのクラスが持つ属性マッピングのクラスが “選択” ラベルが付けられている．結果を図 2.14 に示す．

CnV(3) の処理の流れを図 2.15 に示す．STEP1 では，表 2.1 に基づいて “選択” ラベルと “除外” ラベルの要素が決定される．STEP2 では，“除外” ラベルのクラスの属性をチェックする．STEP3 では，“除外” ラベルのクラスの属性に属するクラスを “除外” ラベルとし

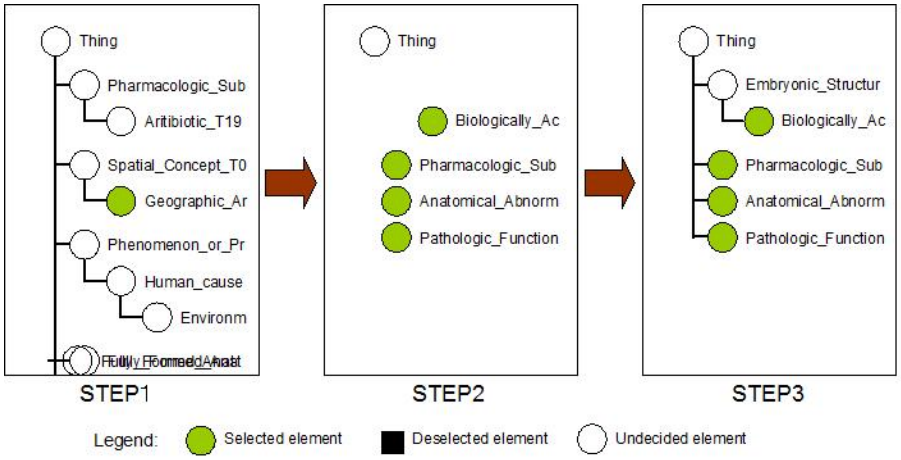


図 2.13 CnV(2) の処理の流れ

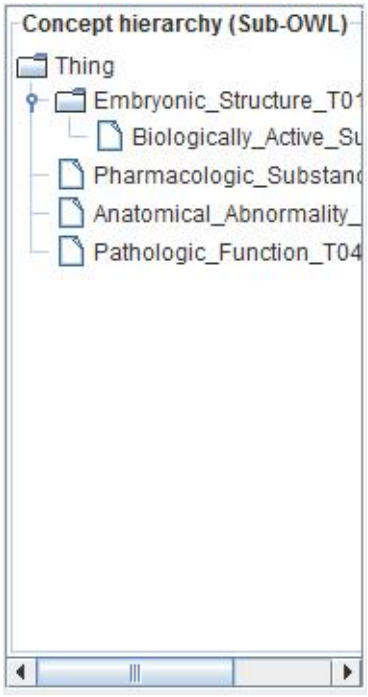


図 2.14 CnV(2) の適用結果

てラベル付けし，“選択” ラベルのクラスと “未決定” ラベルのクラスを残している．結果を図 2.16 に示す．

CnV(4) の処理の流れを図 2.17 に示す．STEP1 では，“選択” ラベルと “除外” ラベルの要素が決定される．STEP2 では，“選択” ラベルのクラスが抽出される．STEP3 では，“選択” ラベルのクラスがすべての “選択” ラベルのプロパティに属している場合は処理が終了する．そうでない場合は，“選択” ラベルのプロパティに属する “未決定” ラベルのクラスを

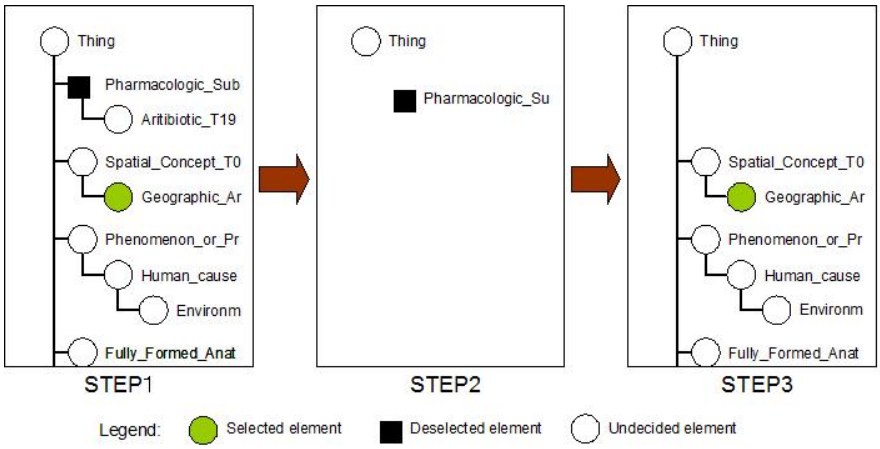


図 2.15 CnV(3) の処理の流れ

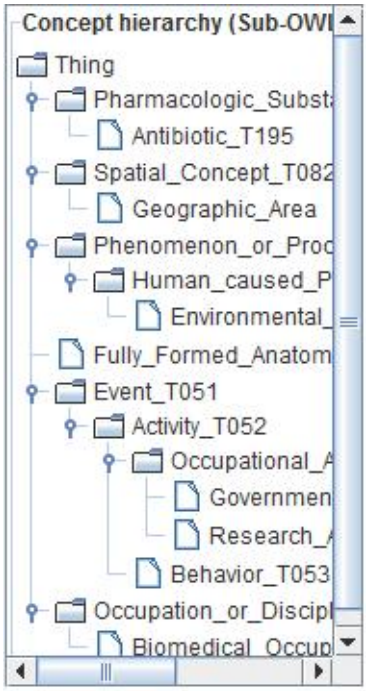


図 2.16 CnV(3) の適用結果

“選択” ラベルにする．

2.8.1.2 CmS の動作

完全性の検証チェックは，抽出されたサブオントロジが完全か否かを確認し補完する．CmS は，CmS(1)〜CmS(3) の 3 つのサブスキーマで構成される．

CmS(1) の処理の流れを図 2.18 に示す．STEP1 では，“選択” ラベルと “除外” ラベルの

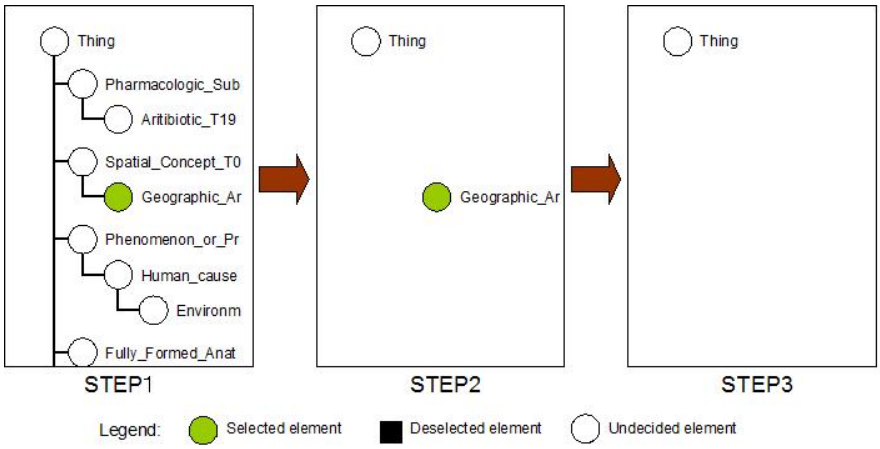


図 2.17 CnV(4) の処理の流れ

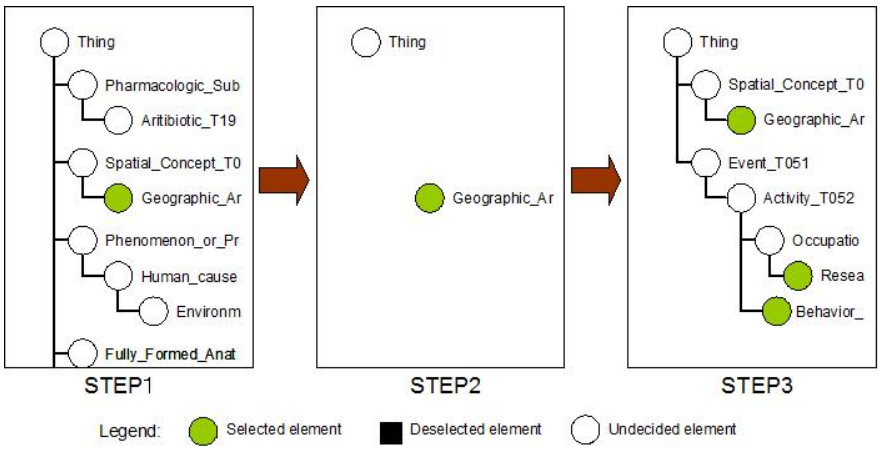


図 2.18 CmS(1) の処理の流れ

要素が決定される．STEP2 では，“選択” ラベルのクラスが抽出される．STEP3 は，“選択” ラベルのクラスの上位クラスを “選択” ラベルにする．結果を図 2.19 に示す．

図 2.20 は，CmS(2) の処理の流れを示している．STEP1 では，表 2.1 に基づいて “選択” ラベルと “除外” ラベルの要素が決定される．STEP2 では，“選択” ラベルのクラスが抽出される．STEP3 では，“選択” ラベルと集合関係のクラスがあればすべてのクラスを “選択” ラベルとする．

図 2.21 は，最終段階の結果を示している．STEP1 では，“選択” ラベルと “除外” ラベルの要素が決定される．STEP2 では，“選択” ラベルのクラスと “選択” ラベルのプロパティを抽出する．STEP3 では，それぞれが一致しているかを確認する．結果を図 2.22 に示す．図 2.23 は，すべてのサブスキーマが使用されたサブオントロジを示している．以上の過程を経てサブオントロジ抽出を行う．

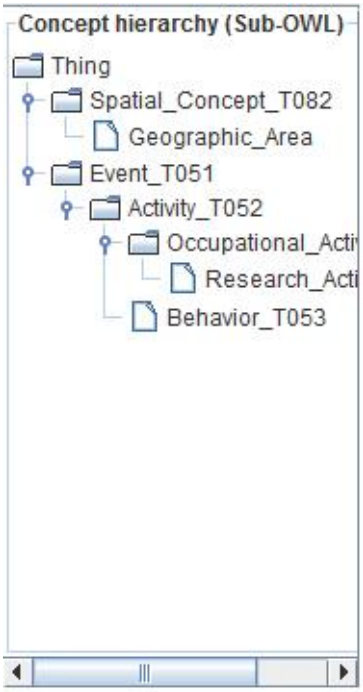


図 2.19 CmS(1) の適用結果

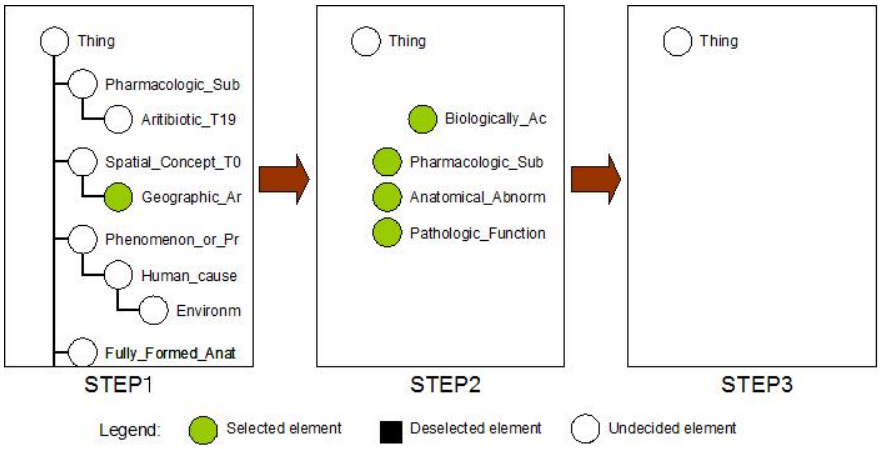


図 2.20 CmS(2) の処理の流れ

2.8.2 サブオントロジテーラリング

UMLSSN オントロジを元に抜粋された 2 つのオントロジを使用してテーラリング処理の紹介を行う．図 2.24 は UMLSSN-1 オントロジの要素を示している．21 のクラスと 65 のデータプロパティを持っており，4 つのクラスに “選択” ラベルがラベル付けされ，2 つのクラスが “除外” ラベルでラベル付けされた状態である．図 2.25 は UMLSSN-2 を示してお

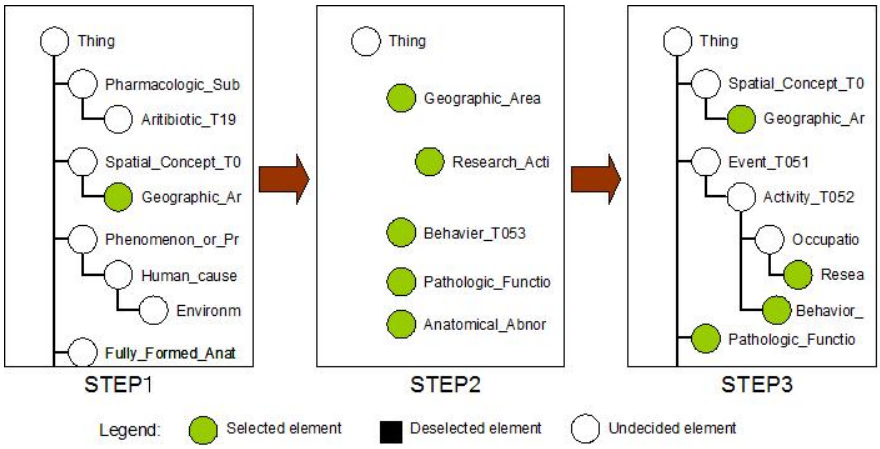


図 2.21 CmS(3) の処理の流れ

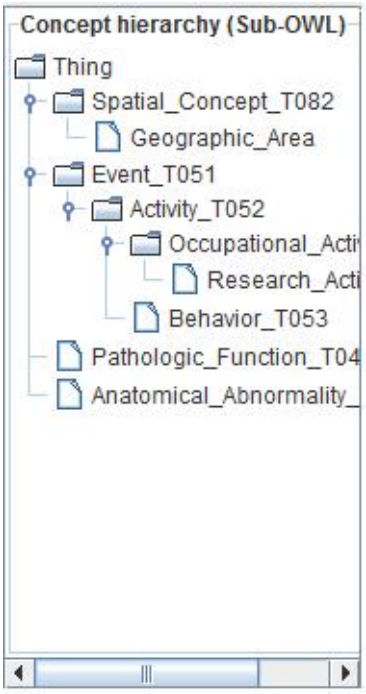


図 2.22 CmS(3) の適用結果

り，27 のクラスと 65 のデータプロパティを持ち，5 つのクラスが “選択” ラベルでラベル付けされ 2 つのクラスが “除外” ラベルでラベル付けされている．

UMLSSN-1 オントロジから抽出された Sub-UMLSSN-1 サブオントロジを図 2.26 に示す．14 のクラスと 65 のデータプロパティを持っており，元のオントロジよりも 14 のクラスが減少している．図 2.27 は UMLSSN-2 オントロジから抽出された Sub-UMLSSN-2 サブオントロジを示している．こちらは元のオントロジから 11 のクラスが減少している．

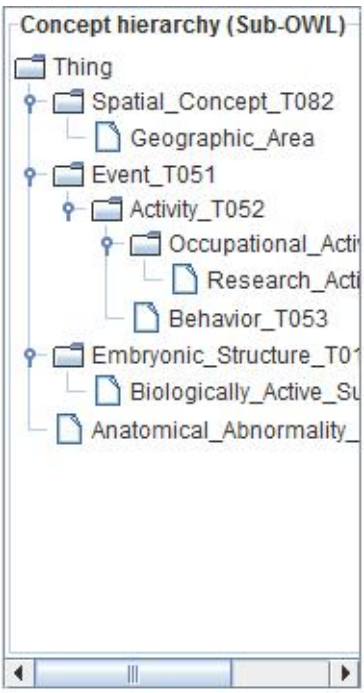


図 2.23 すべてのサブスキーマ適用結果

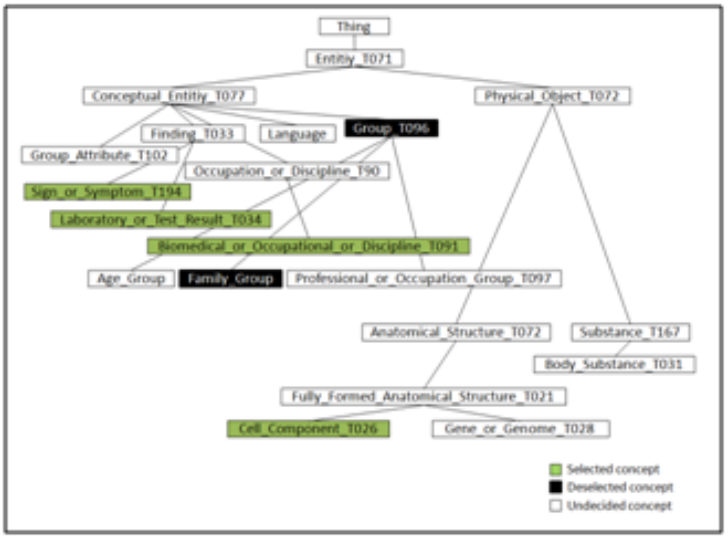


図 2.24 UMLSSN-1 オントロジ

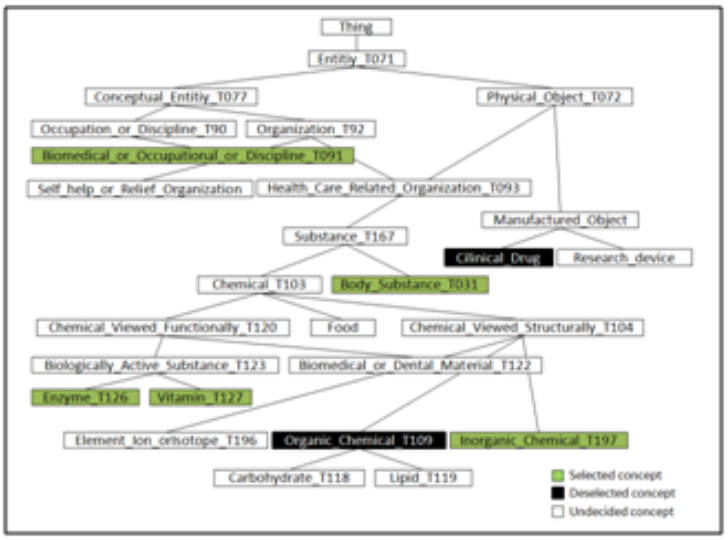


図 2.25 UMLSSN-2 オントロジ

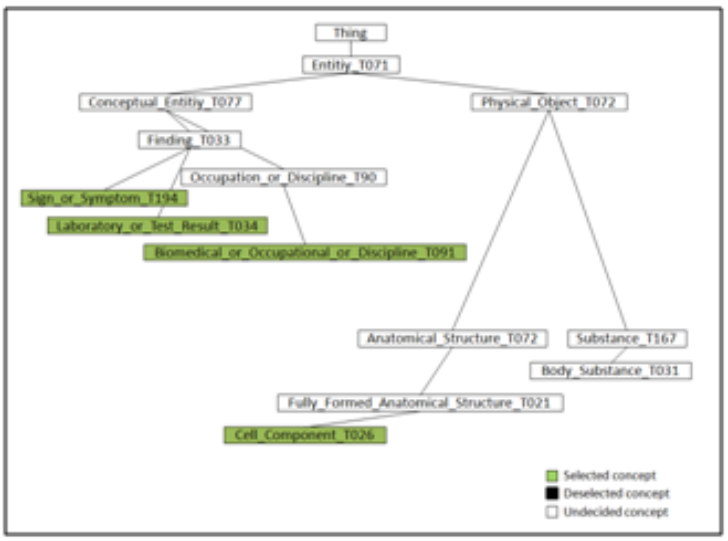


図 2.26 Sub-UMLSSN-1 サブオントロジ

最後に，2 つの方法でテーラリングを行った際のサブオントロジの結果をそれぞれ示す．Reuse, Extract and Add 方法で，Sub-UMLSSN-1 サブオントロジと UMLSSN-2 オントロジをテーラリングした結果のサブオントロジを図 2.28 に示す．このサブオントロジは 35 のクラスと 65 のデータプロパティを持っている．Reuse, Extract and Merge 方法で，Sub-UMLSSN-1 サブオントロジと Sub-UMLSSN-2 サブオントロジをテーラリングした結果のサブオントロジを図 2.29 に示す．このサブオントロジは 22 のクラスと 65 のデータプロパティを持っている．

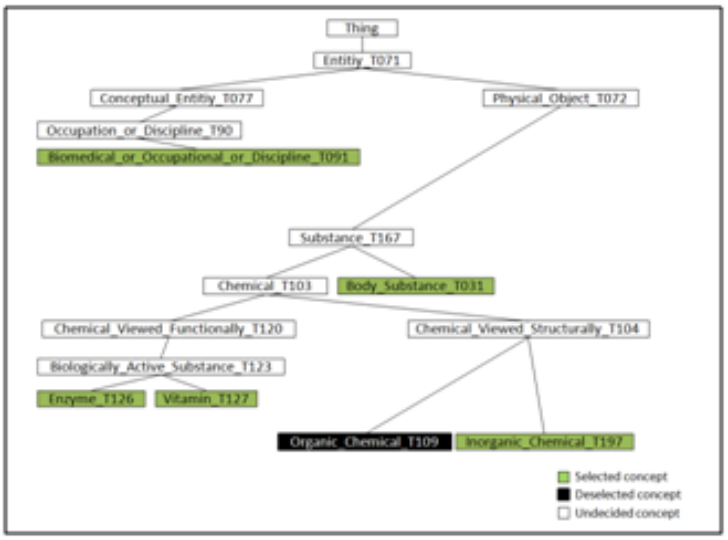


図 2.27 Sub-UMLSSN-2 サブオントロジ

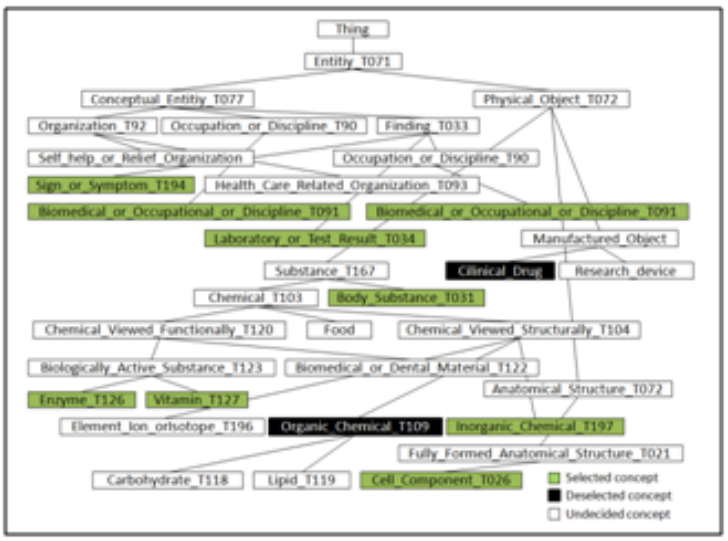


図 2.28 サブオントロジ (Reuse, Extract and Add)

2.8.3 アップデートパッチ作成

Create Update タブ (図 2.30) は，オントロジのアップデートを行いアップデートパッチを生成するタブである．手動でアップデートパッチを記述する方法もあるが，本アプリケーションを用いることで視覚的にオントロジのアップデートを行える．画面下の Browse ボタンでオントロジの場所を指定して Load ボタンでオントロジの読み込みを行う．読み込みが正常に行われると，画面左のエリアに階層状にオントロジが変換されて表示される．階層

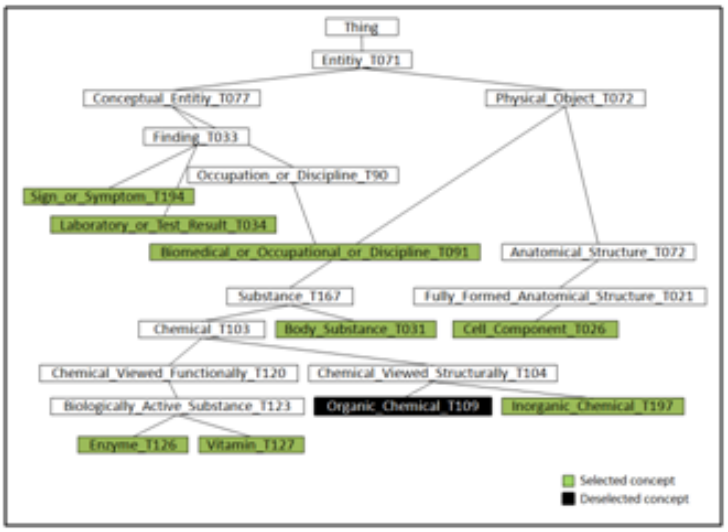


図 2.29 サブオントロジ (Reuse, Extract and Merge)

表示されたクラスやプロパティなどの要素を右クリックすることで追加・削除・置換を行うことができる (図 2.31)．これらの操作は記録され，画面右下の Create ボタンを押すことで XML ファイルでアップデートパッチが作成され，画面に表示され保存される (図 2.32)．

2.8.4 サブオントロジアップデートに関する実験

Sub-ontology Update タブ (図 2.33) は，アップデートパッチを使用してサブオントロジのアップデートを行うタブである．画面下の Browse ボタンでオントロジの場所とアップデートパッチの場所を指定して Load ボタンで読み込みを行う．読み込みが行われると画面左にオントロジが階層状に表示され，画面中央にアップデートパッチが表示される (図 2.34)．画面右下の Load ボタンを押すことでアップデートが実行され，更新後のオントロジが画面右に表示される．サブオントロジアップデートを行った結果を図 2.35 に示す．

サブオントロジの再抽出とサブオントロジアップデートを使用した方法でサブオントロジの生成の実験を行う．サブオントロジの再抽出で生成されるサブオントロジと提案したサブオントロジアップデートの結果のサブオントロジを比較し，サブオントロジアップデートの有効性を確認する．

元のオントロジは AWARDS オントロジを使用した（図 2.36）．AWARDS オントロジは，13 のクラスと 18 のオブジェクトプロパティ，25 のデータプロパティで構成されている．元のオントロジの “Author” クラスと “Writer” クラスの間に “Novelist” クラスを追加し，アップデートしたオントロジを図 2.37 に示す．サブオントロジの抽出・生成の際には，どちらの方法も共通し “Committee” と “Writer” の 2 つのクラスを選択するものとする．

図 2.37 のアップデート後のオントロジから再度サブオントロジの抽出を行う．選択する

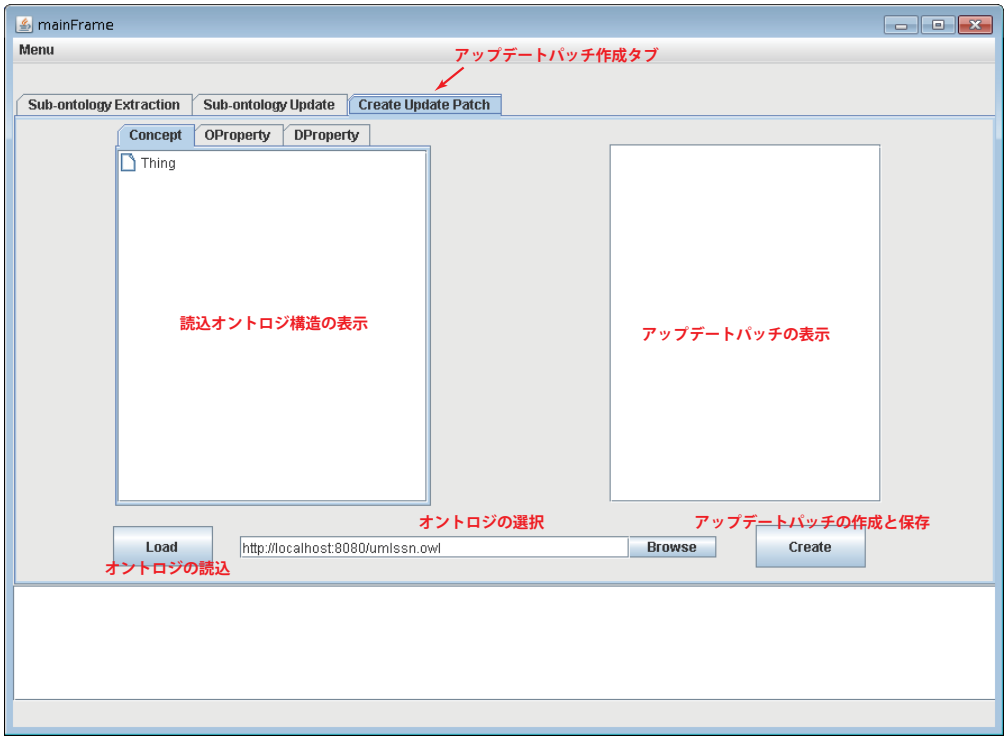


図 2.30 アップデートパッチ生成画面

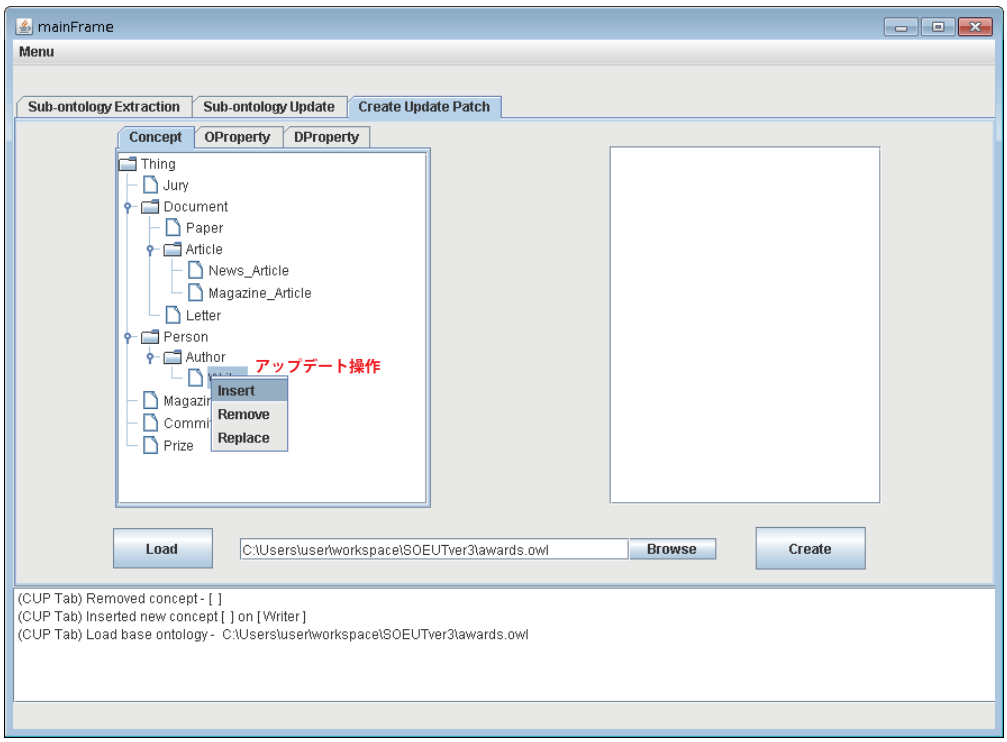


図 2.31 アップデートパッチ作成画面

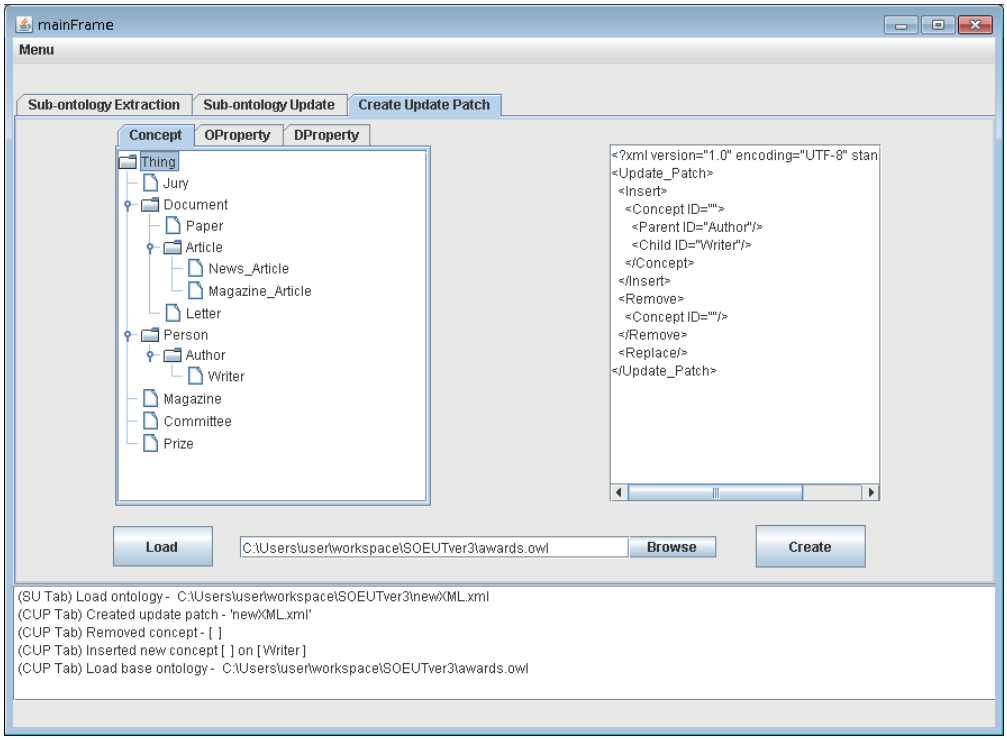


図 2.32 アップデートパッチ生成結果画面

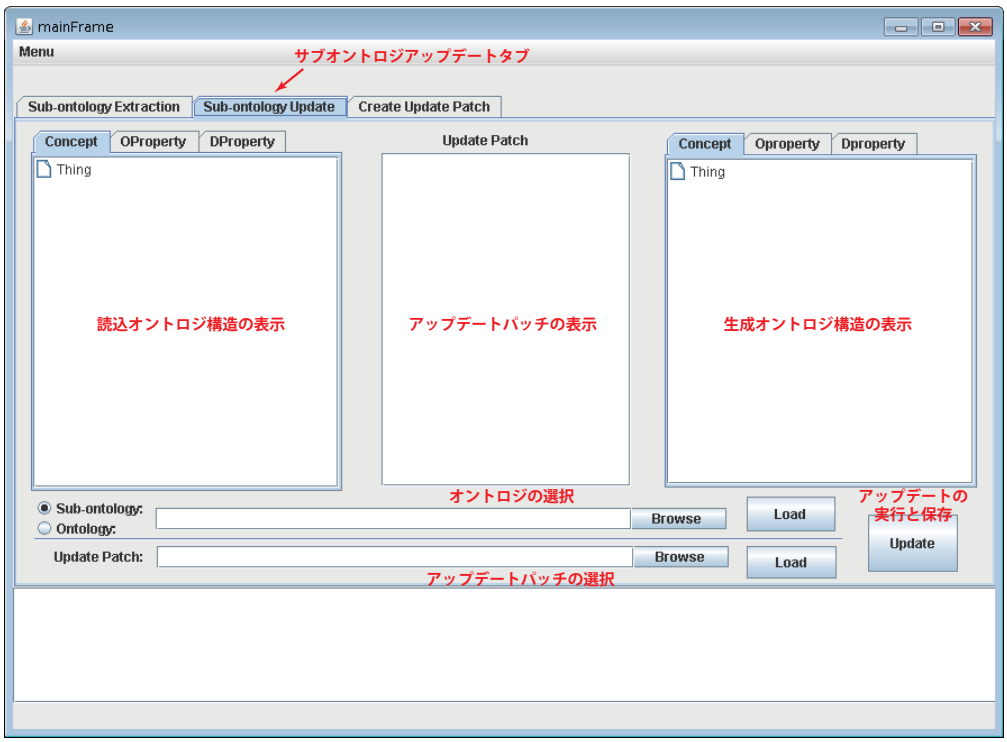


図 2.33 サブオントロジアップデート画面

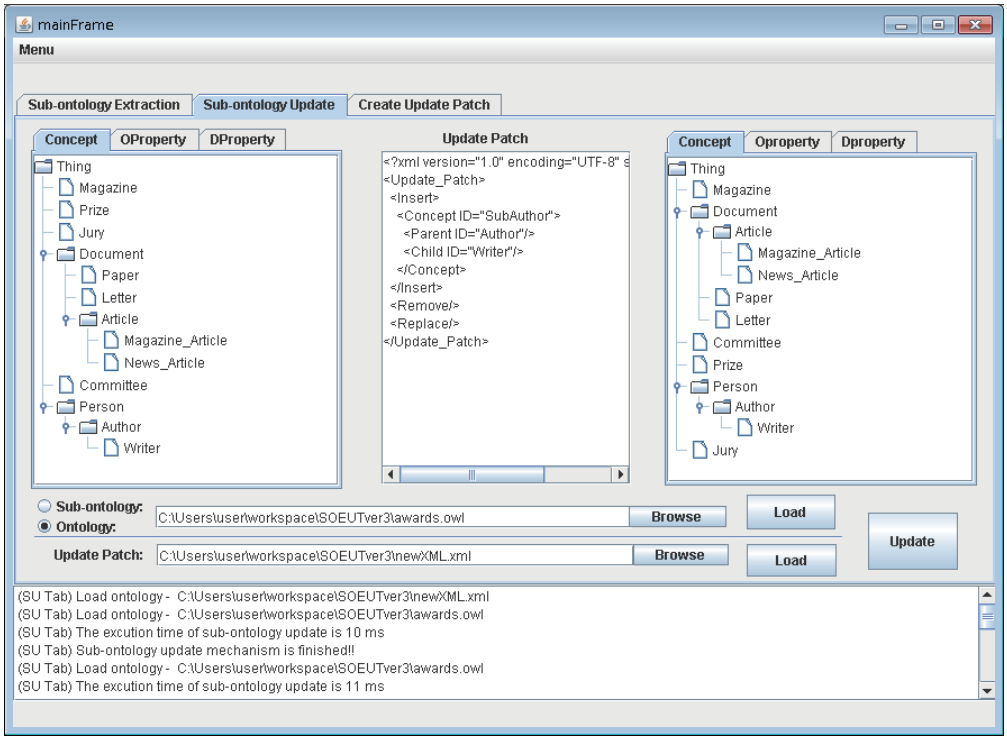


図 2.34 サブオントロジアップデート読込画面

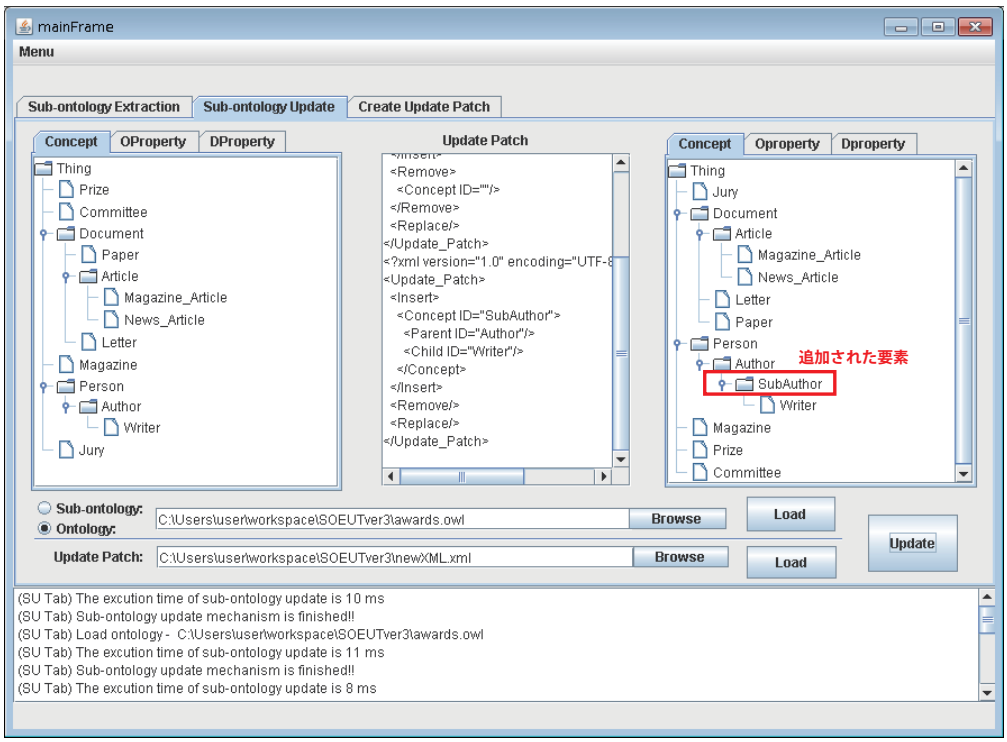


図 2.35 サブオントロジアップデート結果画面

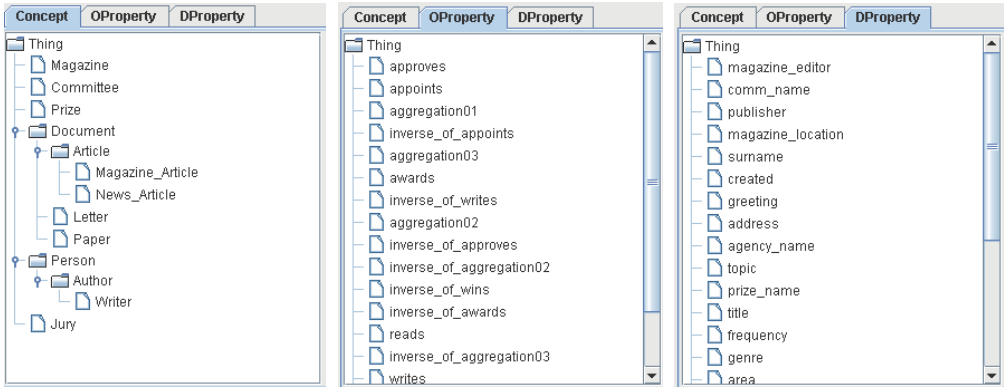


図 2.36 元のオントロジ

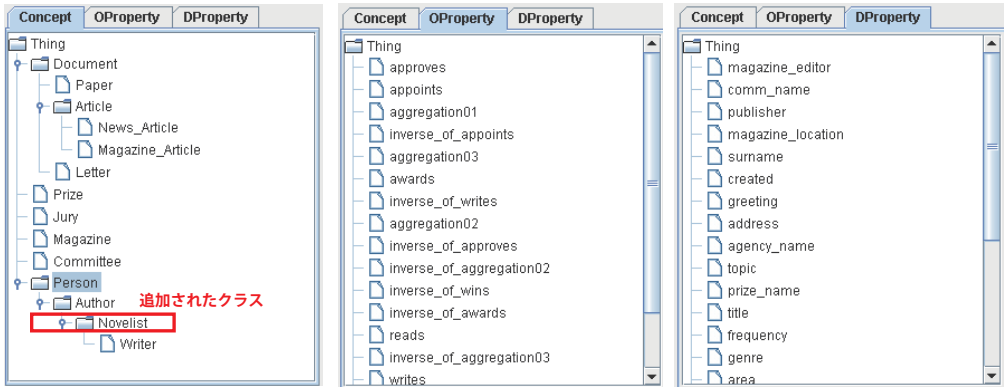


図 2.37 アップデート後のオントロジ

クラスはアップデート前のサブオントロジ抽出と同条件で行うものとする．抽出されたサブオントロジを図 2.38 に示す．

元のオントロジから抽出したアップデート前のサブオントロジを図 2.39 に示す．このサブオントロジにアップデートメカニズムを使用して生成したアップデート後のサブオントロジを図 2.40 に示す．

以前抽出したサブオントロジにアップデートメカニズムを通してアップデートパッチを適用することで，再度サブオントロジの抽出をすることなく更新後のサブオントロジを生成するアップデートメカニズムを提案を行った．再度サブオントロジの抽出を行った図 2.38 とアップデートメカニズムを使用して生成した図 2.40 は同じサブオントロジである．よって，サブオントロジアップデートを使用することで，再抽出と同様のサブオントロジが生成出来た．

次に，上記の処理を 20 回行い処理速度の検証を行う．表 2.3 に実行環境を示す．再度サブオントロジの抽出を行った場合は平均で 17ms であった．サブオントロジアップデートを使用すると平均で 11ms であった．よって，サブオントロジアップデートを使用することで

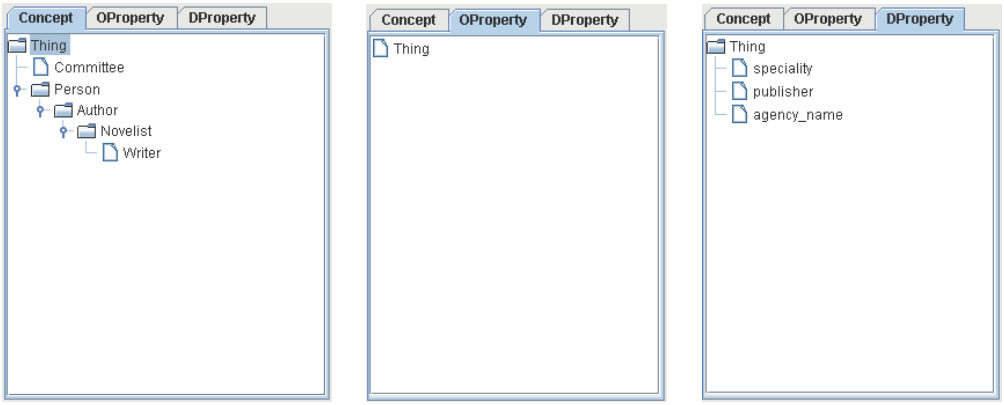


図 2.38 アップデート後のサブオントロジ

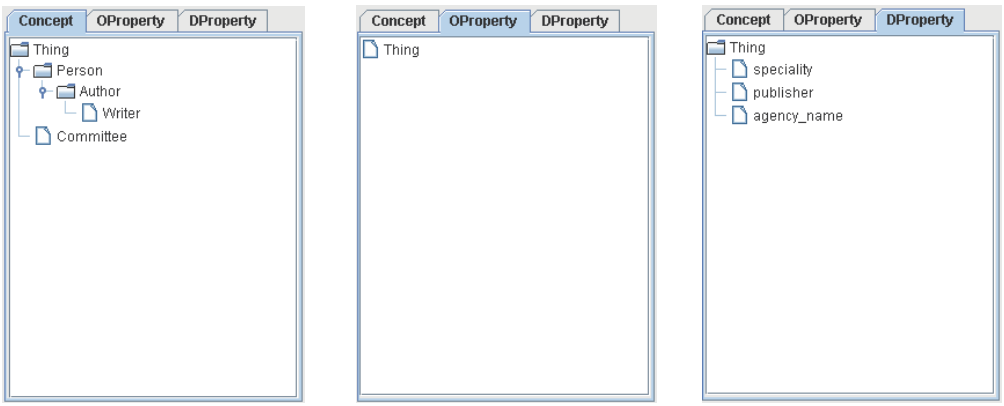


図 2.39 アップデート前のサブオントロジ

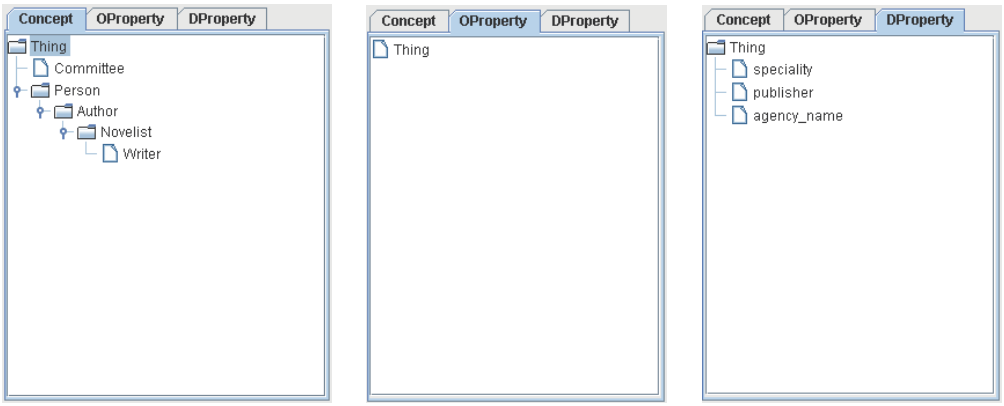


図 2.40 アップデート後のサブオントロジ

表 2.3 実行環境

CPU 性能	Core-i 7 2.4GHz
メモリ	16GB
ストレージ	500GB
OS	Windows 7

表 2.4 実行結果

	ファイルサイズ	実行時間
AWARD オントロジ	19,568 バイト	
サブオントロジ再抽出	2,200 バイト	17ms
サブオントロジアップデート	2,200 バイト	11ms

6ms 速くなった (表 2.4)．要素数の少ないオントロジで検証を行ったのであまり差はなかったため，今後は様々なオントロジを使用して評価をする必要がある．

2.9 まとめ

利用者がオントロジを利用して検索をするためには，利用者の手元へオントロジを転送しなければならない．そこで，サブオントロジの抽出を行うことでデータ転送量の削減を目的としたサブオントロジ抽出を紹介した．さらに，2 つのオントロジから 1 つのサブオントロジを生成するサブオントロジテーラリングを紹介した．

本研究では，ネットワーク上のオントロジの同一性の維持を効率的に行うオントロジアップデートを提案した．XML 形式のアップデートパッチを転送することで，オントロジ本体を転送することなく更新が行える．

元のオントロジが更新された場合，サブオントロジを再抽出せずに以前抽出したサブオントロジの更新を行えるサブオントロジアップデートの提案を行った．従来はサブオントロジを再抽出する必要があり時間がかかった．サブオントロジアップデートを行うことで同一のサブオントロジが生成でき，従来の再抽出に比べて速く更新を行うことができた．

第 3 章

オントロジのクラウドコンピューティングへの適用

本章では，2 章のオントロジのアップデート管理とエージェント技術を使用したクラウドサービス発見システムの提案を行う [18][19].

3.1 クラウドコンピューティング

クラウドコンピューティングは，膨大なインターネットのネットワークを雲に例えたものである．ネットワークの向こう側にあるコンピュータを用いて様々な作業を行うことができる．従来は，ユーザや企業内でコンピュータを管理するか高価なサーバを借りる必要があった．しかし，クラウドコンピューティングを利用することで容易に必要な資源を必要な時間のみ使うことが可能になった.

“クラウドコンピューティング” という言葉は，2006 年に Google の CEO である Eric Emerson Schmidt が発言したのが初めてだとされている．しかし，その基本的な考え方は 1960 年代の集中処理ネットワークでの端末とホストコンピュータの関係と似ている．1990 年代のクライアント・サーバシステムを経て，クラウドコンピューティングへと変化してきた中で捉えると，コンピュータシステムの動向が集中から分散，そして再び集中へと変わっていることが伺える．また，クラウドには様々な種類があり，代表的なものにパブリッククラウド，プライベートクラウド，ハイブリッドクラウドの 3 種類がある.

パブリッククラウドは，不特定多数の一般企業や個人向けに提供されているクラウドサービスである．現在，様々なクラウドサービス提供企業が，Amazon Web Services[20] や Windows Azure[21] に代表されるような多様なパブリッククラウドサービスの提供を行っている．クラウドの考え方が登場したころにはクラウドといえばパブリッククラウドであったが，プライベートクラウドが登場したことにより区別するために用いられるように

なった．

プライベートクラウドは，特定の企業により自社内の部署やグループ企業などのためのサービスとして提供され，限定されたネットワークでのみ利用されるものである．企業の業務や状況に合わせ使いやすいクラウドを構築でき，パブリッククラウドに比べて高いセキュリティが確保できる利点がある．クラウドサービスが爆発的に普及し，多くの企業や研究機関が自社内などにプライベートクラウドを構築するまでになっている．CloudStack[22] や Eucalyptus[23] などの OSS(Open Source Software) のクラウド構築ソフトウェアも充実しプライベートクラウドを構築する環境も整ってきている．

ハイブリッドクラウドは，パブリッククラウドとプライベートクラウドを組み合わせたクラウドである [24][25][26][27]．秘匿性の高い重要なデータはプライベートクラウドに置いておき，アクセスの多いであろうフロント部分のシステムはパブリッククラウドを使用するような手法や，通常はプライベートクラウドでシステムを運用し，処理量が増大した場合にパブリッククラウドを利用するような手法がある．

また，クラウドの提供形態には，SaaS (Software as a Service)，PaaS (Platform as a Service)，IaaS (Infrastructure as a Service) がある．

SaaS ではサーバの中にあるソフトウェアをクラウドのサービスとして提供している．企業向けの大規模なものから個人向けの小規模なものまで様々なサービスが存在する．ソフトウェアのアップデートは利用者が行う必要がなく，常に最新のソフトウェアを利用できる．

PaaS は，プラットフォームを提供するサービスで，利用者にはシステム構築の基盤が提供される．提供された環境の中にソフトウェアをインストールしたり，外部に向けてネットサービスを提供することもできる．ソフトウェアの開発環境は整えられており，利用者はシステム構築のみに作業を集中できる利点がある．

IaaS はサーバやネットワーク機器などのインフラを提供するサービスである．利用者は提供されたインフラ上にシステムを構築することができ，システムの利用規模などに応じて柔軟に処理性能を向上させることもできる．ハードウェアのメンテナンスや障害対応などをクラウドサービス提供企業に任せることができることが利点である．しかし，様々な仮想計算機が提供されており，適切なクラウドサービス提供企業を選ぶのは困難である．

そこで本研究は，パブリッククラウドサービスの提供形態の中で IaaS に焦点を当てる．

3.2 現状と問題点

IaaS で仮想マシンを起動する場合，インスタンスタイプと OS を選択する．インスタンスタイプとは，クラウドサービスが提供する仮想マシンの性能のことである．これらの組み合わせで利用料金が決まる．パブリッククラウドサービス事業者によって様々な組み合わせが存在し，利用者がすべてを把握するのは困難である．利用者へ適切なクラウドサービスを

検索する必要があるが，インスタンスタイプに用いられる項目も事業者によって名称が異なるため，差異を解消し選択しなければならない．

そこで，オントロジを使用して表現の統一を行う．インタークラウドのように，パブリッククラウド間でのリソースの共通化が注目されている．インタークラウドとは，異なるパブリッククラウドを組み合わせて使うクラウドである．将来的にパブリッククラウドにオントロジを配置することで，パブリッククラウド間の表現の統一が行われると期待される．パブリッククラウドごとにオントロジを管理しているため，検索にオントロジを使用するためには手元にオントロジを保管しなければならない．オントロジが更新された場合は再度オントロジ全体を転送する必要があるが，それでは効率が悪い．そこで第 2 章のオントロジアップデートを使用して効率化を行う．

3.3 クラウドサービス発見システムの提案

利用者が適切なクラウドサービスを選択するための，オントロジを利用したクラウドサービス発見システムを提案する．利用者は，自身の所有するプライベートクラウドとパブリッククラウドのインスタンスタイプを検索する．クラウドサービスによって異なるインスタンスタイプの表現の差異を吸収するためにオントロジを使用する．オントロジは各クラウドサービスに分散配置し，このオントロジの管理に 2.6 節で提案したアップデート管理手法を適用する．

本研究のシステムの環境は，クラウド上で動くエージェントシステムとブローカサーバで構成される．図 3.1 は，プライベートクラウドとパブリッククラウドが存在する環境の例である．プライベートクラウドは，利用者が所有しているクラウドサービスである．パブリッククラウドは，不特定多数にクラウドサービスを提供する事業者である．ブローカサーバは，オントロジやインスタンスタイプの情報を集約・管理する．

クラウドサービス発見システムは，“資源情報の収集” と “ユーザによる検索” と “オントロジの更新” の 3 つの機能をもつ．“資源情報の収集” を行う機能では，プライベートクラウドとパブリッククラウドを監視する．変更があればブローカサーバへ変更内容を伝えることで常に最新の状態をブローカサーバへ集約する．“ユーザによる検索” の機能では，ブローカサーバに集約・管理されているクラウドサービスの情報を検索する．利用者が仮想マシンを起動したい際に検索項目を指定することで適切なクラウドサービスの検索が行える．“オントロジの更新” の機能では，プライベートクラウドとパブリッククラウドのオントロジが更新された場合，ブローカサーバのオントロジへ変更を反映する．

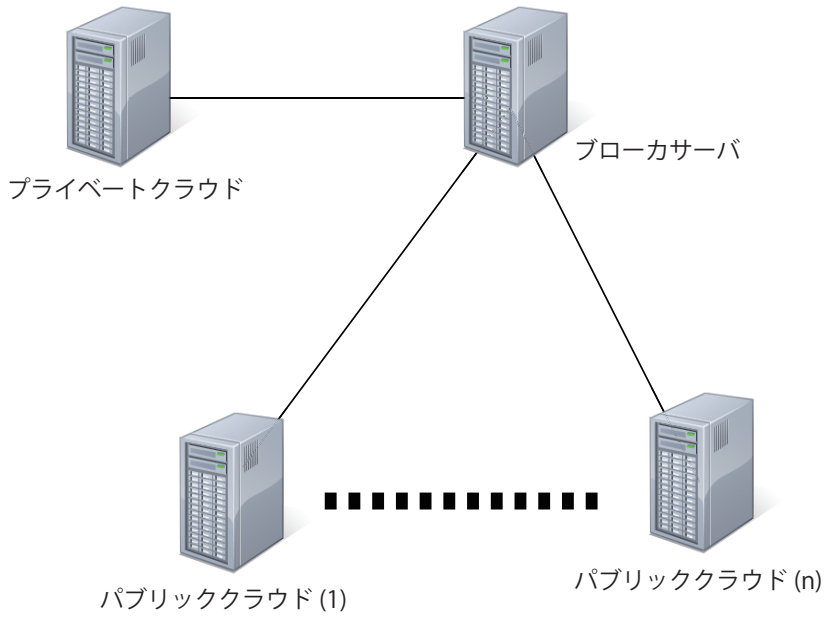


図 3.1 クラウドサービス発見システムの構成

3.3.1 資源情報の収集

資源情報の収集には，エージェントを使用する．エージェントを使用することで，各プラットフォームで柔軟な処理を行うことができる．また，CPU 使用率などの動的な情報を取得することができる．エージェントは役割ごとに存在し，複数のエージェントがコミュニケーションを取ることで様々な処理を行う．利用者はブローカサーバに蓄積されたクラウドサービスの情報をオントロジを使って検索する．よって，ブローカサーバに蓄積されるクラウドサービスの情報は常に最新のものでなければならない．そのために，プライベートクラウドとパブリッククラウドにはエージェントが常駐し監視を行う．主な監視対象は，インスタンスタイプの表現が記述されたオントロジとインスタンスタイプが格納されているデータベースである．オントロジとデータベースは更新があれば，エージェントを介してブローカサーバへ転送される．(図 3.2).

図 3.3 はプライベートクラウド，ブローカサーバ，そしてパブリッククラウド上のエージェントの配置図である．これらのエージェントは，個々が独立して動作する．情報のやり取りが必要な場合は他のエージェントと通信を行う．

図 3.3 の 5 種類のエージェントの詳細を以下に示す．

- MA(Master Agent)

  MA は各プラットフォームに必ず常駐しており，すべてのエージェントは MA を介して連携する． DCA とはデータベースのデータの受け渡しを行い，OBA とはオ

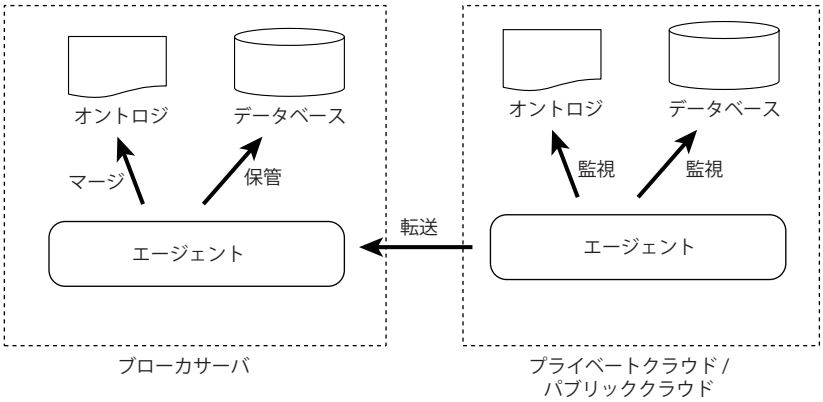


図 3.2 プライベートクラウド/パブリッククラウド情報の監視と収集

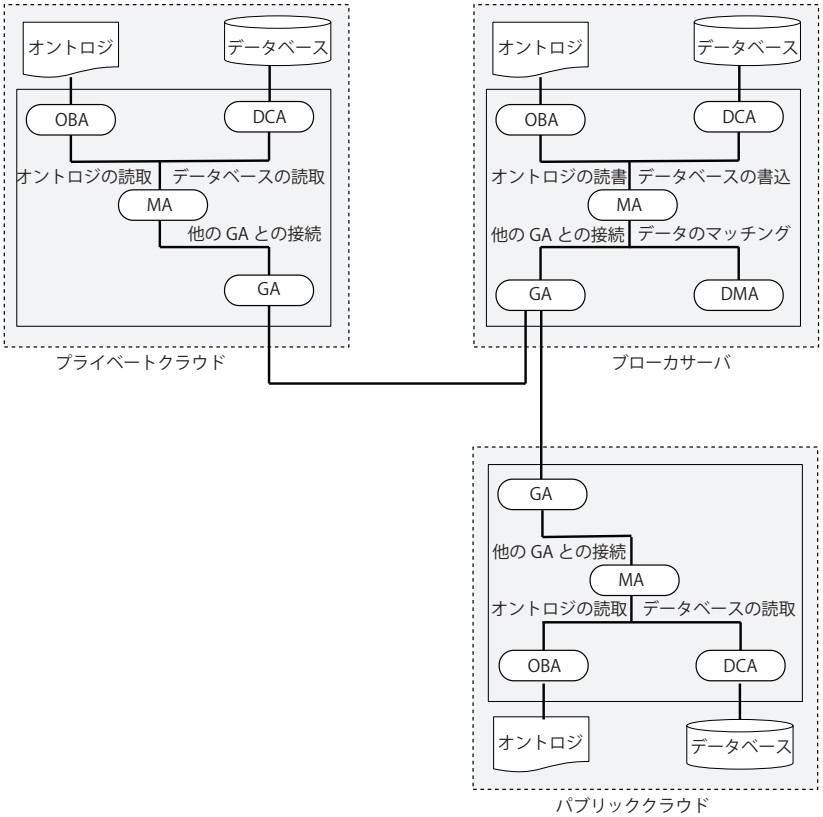


図 3.3 エージェント配置図

ントロジデータの受け渡しを行う．DMA からはマッチングの結果を受け取る．他のプラットフォームの MA との連携は GA を通して行われる．

- DCA(Database Connect Agent)
  DCA はデータベースに保管されているインスタンスタイプのデータを読み書きする際に利用される．データベースを利用するためのドライバを管理しており，データベースへの接続はこのエージェントを介して行われる．また，データベースに変更がないか監視をしている．
- OBA(Ontology Broker Agent)
  OBA は保管されているオントロジデータを読み込み MA へ渡したり，ブローカサーバでは他のエージェントが収集した情報を MA を介して受け取り既存のオントロジとマージする．また，オントロジに変更がないか監視をしている．
- DMA(Data Matching Agent)
  ブローカサーバのみに存在し，ユーザの要求とマッチングを行う．
- GA(Gate Agent)
  各プラットフォームの窓口となる．プラットフォーム間の MA のやり取りはすべてこのエージェントを通して行われる．

3.3.2 ユーザによる検索

利用者の要求する情報と，プライベートクラウドやパブリッククラウドのインスタンスタイプのマッチングを行う．インスタンスタイプの項目は，CPU 数，CPU 性能，メモリ容量，ストレージ容量，オペレーティングシステム，初期費用，月額費用である．ユーザの検索は GUI からプライベートクラウド上のエージェントを介してブローカサーバに伝えられ，結果が返ってくる．

図 3.4 はプライベートクラウド上にユーザインターフェースがある例を示している．ユーザの要求はプライベートクラウド内の MA へ渡す．プライベートクラウド内では，MA → GA → [ブローカサーバ] となり，ブローカサーバでは [プライベートクラウド] → GA → MA → OBA → MA → DCA → MA → DMA → MA → GA → [プライベートクラウド] という流れで処理を行う．結果は，[ブローカサーバ] → GA → MA の流れで返す．

3.3.3 オントロジの更新

プライベートクラウドやパブリッククラウドのオントロジは，それぞれの管理者が更新を行う．オントロジが更新された場合，OBA を介してブローカサーバにオントロジが転送される．しかし，オントロジ全体を転送するのは効率が悪い．そこで本研究では，2.6 節で提

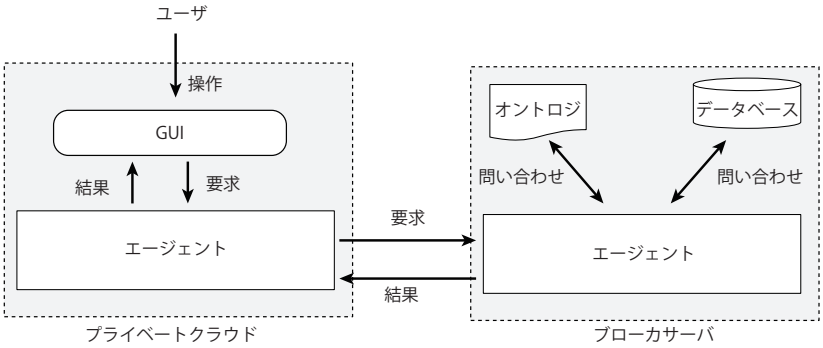


図 3.4 ユーザの検索要求

案したオントロジアップデートを適用することで問題の解決を図る．

プライベートクラウドやパブリッククラウドの管理者は，オントロジを更新する際にアップデートパッチを作成する．アップデートパッチをブローカサーバに転送することで，オントロジ全体の転送に比べて帯域の削減ができる．図 3.5 は通常の方法とオントロジアップデートを適用した場合の違いを示している．(A) オントロジを転送する場合は，オントロジ全体を転送しブローカサーバのオントロジとマージしている．(B) アップデートパッチを転送する場合は，アップデートパッチのみ転送し，ブローカサーバのオントロジとオントロジアップデートを行っている．また，アップデートパッチを使用したオントロジのアップデートは OBA が行う．

3.4 プロトタイプシステムの実装

提案したプロトタイプシステムを実装した．表 3.1 にパブリッククラウド A のインスタンス例を，表 3.2 にパブリッククラウド B のインスタンス例を示す．同じ CPU の性能を示す項目だが，CPU と ECU のようにパブリッククラウドによって名称が異なる．名称が異なればコンピュータは同じものを示しているとは認識できず検索を行うことができない．そこでオントロジを利用することで，表現の差異を吸収し検索を行うことができる．

3.4.1 プライベートクラウドの構築

プライベートクラウドを構築するための様々な基盤構築ソフトウェアがある．以下で代表的な 4 つソフトウェアである Eucalyptus, OpenNebula [28], OpenStack [29], CloudStack の紹介を行う．

- Eucalyptus
  “プライベートクラウドをなるべく簡単に構築できるようにする” というコンセプ

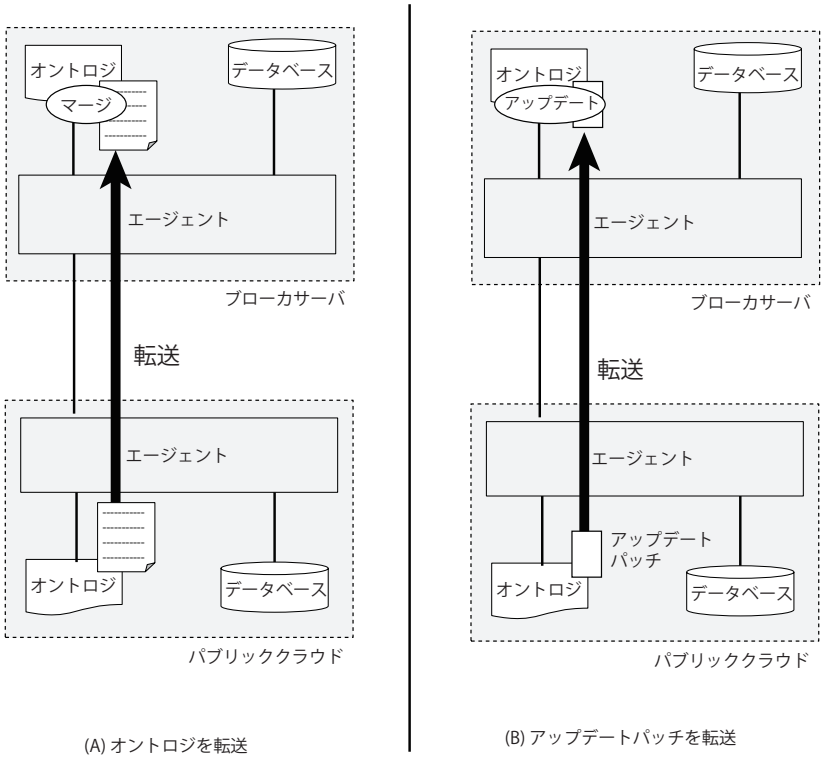


図 3.5 オントロジの更新

表 3.1 パブリッククラウド A のインスタンス例

タイプ	CPU_Num	CPU (GHz)	Memory (MB)	Storage(GB)
mini1	1	1	512	30
small	1	1	512	30
small2	1	3	2000	30
small4	1	3	4000	30
medium	2	3	2000	30
large	4	3	4000	30

トで作られたプライベートクラウド基盤であり，Amazon EC2 (Amazon Elastic Compute Cloud) および Amazon S3 (Amazon Simple Storage Service) と互換性がある．カルフォルニア大学サンタバーバラ校のコンピュータサイエンス学科の研究プロジェクトとして開発された．

- OpenNebula
  もともとグリッドコンピューティングに関する研究から始まった．2005 年に Ignacio M. Llorente と Ruben S. Montero の調査プロジェクトとして開始された．

表 3.2 パブリッククラウド B のインスタンス例

タイプ	vCPU	ECU (GHz)	メモリ (MB)	ストレージ (GB)
XS	1	0.8	500	15
S	1	1.6	2000	15
M	2	1.6	4000	15
L	4	1.6	4000	15
XL	8	2.4	120000	15

- OpenStack
  完全にオープンな開発スタイルをとっており，クローズなバージョンを作らないことを宣言している．仮想ネットワーク制御 “Neuton”，ハイパーバイザ制御 “Nova”，イメージ管理 “Glace”，ブロックストレージ “Cinder”，オブジェクトストレージ “Cinder”，統合認証 “Keystone” から構成される．
- CloudStack
  Cloud.com 社が提供しているプライベートクラウド基盤．Web ブラウザによる運用管理および一般利用といった幅広い WebUI が実装されており，ハイパーバイザに依存しない幅広い仮想化基盤もサポートしている．

我々はこれらのプライベートクラウド構築基盤の中で，AWS と互換性のあり，多くの実績がある Eucalyptus を使用してプライベートクラウドの構築を行った．Eucalyptus は複数の機能が組み合わさることでクラウド環境を構築している．そのため，機能によってコンポーネントと呼ばれる単位でソフトウェアが分割されており，それぞれのコンポーネントは単体で動作するように設計されている．

Cloud Controller (CLC) は，クラウド全体の情報を管理し，クラウド利用者に対して API や Web 管理画面を提供するコンポーネントである．Cluster Controller (CC) は，Node Controller とインスタンスのネットワークを管理する．Node Controller (NC) は，Cluster Controller からの要求を受け，ハイパーバイザにインスタンスの起動を命令する．Storage Controller (SC) は，インスタンスに対して Amazon EC2 での EBS 相当の機能を提供する．最後に Walrus は，Amazon S3 互換の API を提供し，仮想マシンイメージなどは Walrus が管理する．

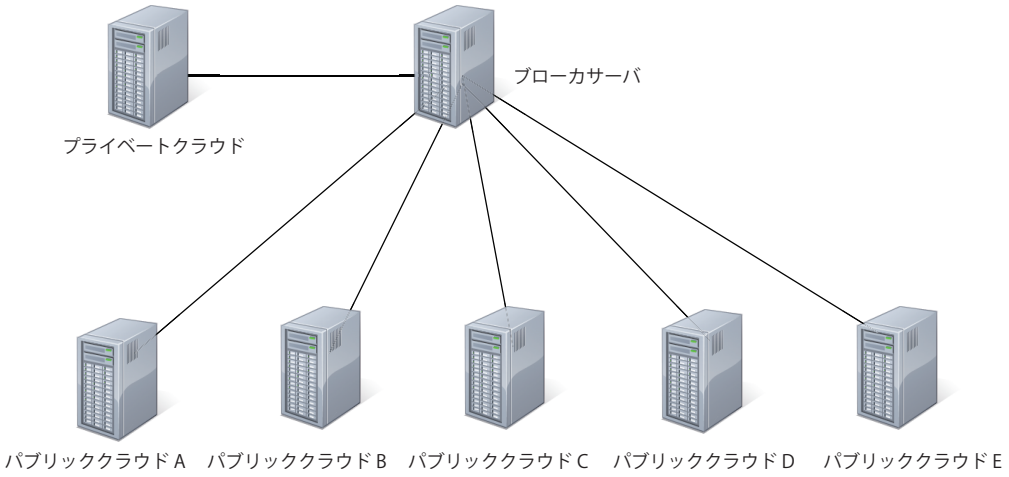


図 3.6 クラウドサービス発見システム実装環境

表 3.3 プラットフォームの性能

CPU 性能	Core-i 7 2.4GHz
メモリ	16GB
ストレージ	500GB
OS	ContOS 6
クラウド基盤	Eucalyptus

3.4.2 実装環境

図 3.6 はクラウドサービス発見システムの実装環境を示す．プライベートクラウドを 1 つ，パブリッククラウドを 5 つ，ブローカサーバを 1 つ配置するパブリッククラウドとブローカサーバ間のネットワークには 130ms の遅延を持たせている．プライベートクラウドとパブリッククラウドのデータベースには 2〜112 個のインスタンスタイプを格納している．

また，すべてのプラットフォームの構築に使用する計算機の性能を表 3.3 に示す．プライベートクラウドとパブリッククラウドは共に Eucalyptus で構築する．エージェントは JADE を使用して構築する．オントロジの読み書きは Apache Jena を使用する．

各エージェントにはどのタイミングで命令を実行するかを設定できる．あらかじめ 4 種類の Behavior が用意されている．常に命令を実行し続ける “SimpleBehavior”，一回のみ命令を実行する “OneShotBehavior”，指定ミリ秒ごとに命令を実行する “TickerBehavior”，指定した時刻に命令を実行する “WakerBehavior” がある．

クラウドサービス発見システムの 5 つのエージェントは，プライベートクラウドやパブ

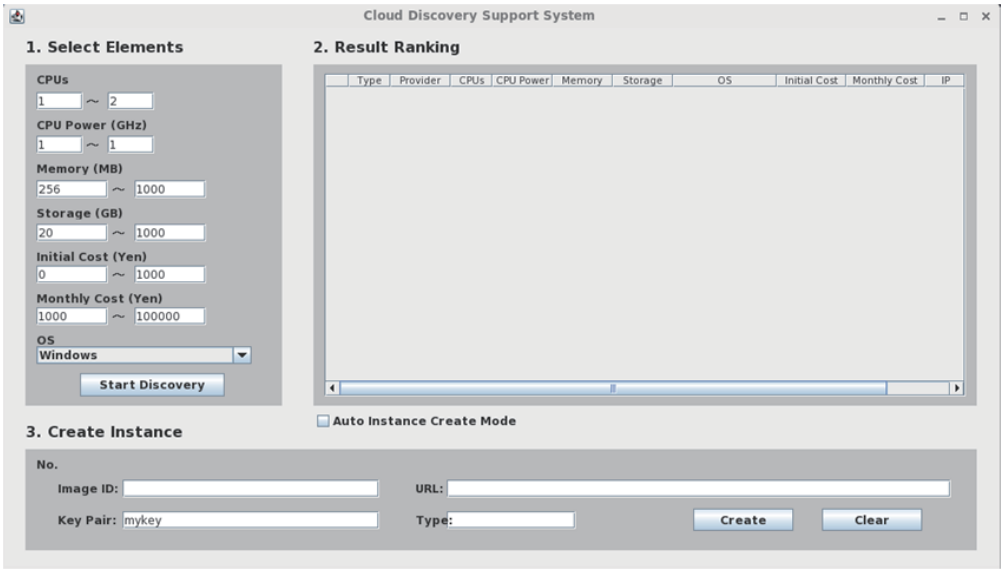


図 3.7 検証用検索フォーム

リッククラウドのオントロジとデータベースの監視には “TickerBehavior” を使用する．5 分ごとに OBA がオントロジを，DCA がデータベースを監視する．エージェント間の通信は，JADE に用意されているメッセージ通信機能を利用する．すべてのエージェントは，“SimpleBehavior” を使用して他のエージェントからの通信の待機を行う．

データベースが更新された場合，DCA がデータベースを読み込む．読み込んだデータをメッセージ通信でブローカサーバの DCA へリレーする．ブローカサーバの DCA が DCA のデータベースへデータを格納する．オントロジが更新された場合，アップデートパッチを OBA が読み込む．読み込んだアップデートパッチをメッセージ通信でブローカサーバの OBA へリレーする．OBA でブローカサーバのオントロジをアップデートパッチを使用してオントロジアップデートを行う．

3.4.3 ユーザによる検索要求の確認

ユーザから要求が出され結果が返ってくるまでに，ユーザのプライベートクラウド上の 3 つのエージェント，ブローカサーバ上の 4 つのエージェントを介す．本論文では，図 3.7 のような検索項目の入力フォームを持つ GUI を用いる．GUI は GUI が動作している計算機上の MA と連携を行う．プライベートクラウド上で GUI が動作している場合，検索処理の流れは，プライベートクラウド【MA → GA】→ブローカサーバ【GA → MA → OBA → MA → DCA → MA → DMA → MA → GA】→プライベートクラウド【GA → MA】となる (図 3.8).

ユーザは CPU 数 1〜2，CPU 性能 1GHz，メモリ 256〜1000MB，ストレージ 20〜

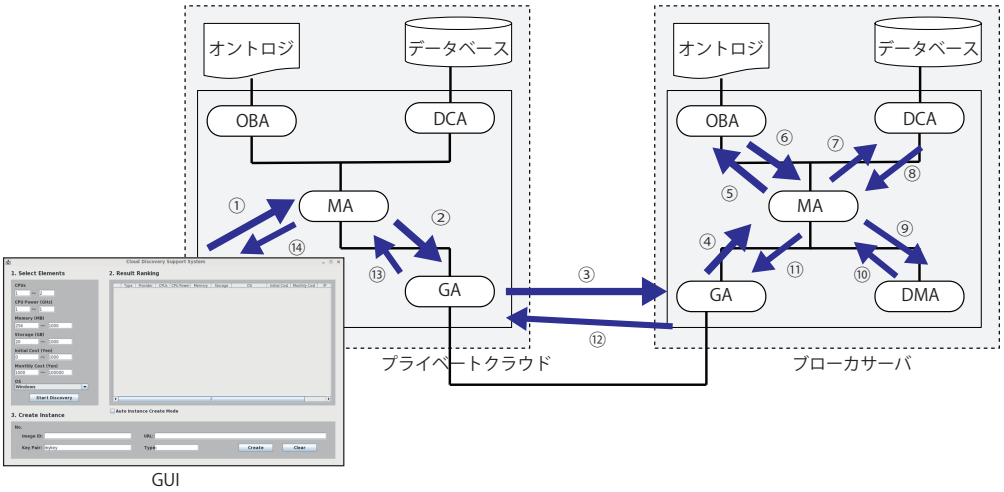


図 3.8 ユーザの検証処理の流れ

1000GB，初期費用 0〜1000 円，月額費用 1000〜100000 円，OS Windows で検索を行った (図 3.9)．その結果を図 3.10 に示す．

ユーザが要求を出し結果が返されるまでに全体で 1316 ms であった．ただし，検索内容によって実行時間は多少前後する．検索フォームからプライベートクラウドの MA を経てブローカサーバの GA までは 7ms、ブローカサーバ内のエージェントの通信はすべて 2ms であった．これは同一マシン上にエージェントが存在するためである．OBA でブローカサーバ内にあるオントロジの読み込みは 130ms であり，DCA ではパブリックサーバの情報を読み込むのに 60ms であった．DMA での検索には 1100ms 要している．このことから，結果が返ってくるまでの約 84 %の時間が DMA での処理であった．

3.4.4 オントロジ更新の評価

オントロジの更新時間の比較を行う．オントロジ更新時のエージェントの処理の流れを図 3.11 に示す．

オントロジをブローカサーバへ転送して既存のブローカサーバのオントロジとマージを行う場合の検証を行うパブリッククラウドの CPU の下位プロパティを has_CPUNumber から has_CPUNum へ変更する．図 3.12 にブローカサーバのオントロジのプロパティを示す．次にパブリッククラウドのオントロジのプロパティを示す (図 3.13)．図 3.12 のオントロジと図 3.13 のオントロジをマージすることでオントロジがブローカサーバに生成される (図 3.14)．以前のブローカサーバのオントロジにはなかったプロパティが追加されていることがわかる．よって，正しく動いているといえる．

次に，パブリッククラウドのオントロジ全体をブローカサーバへ転送する場合の時間を計測する．パブリッククラウドが保持しているオントロジデータは 12KB である．オントロ

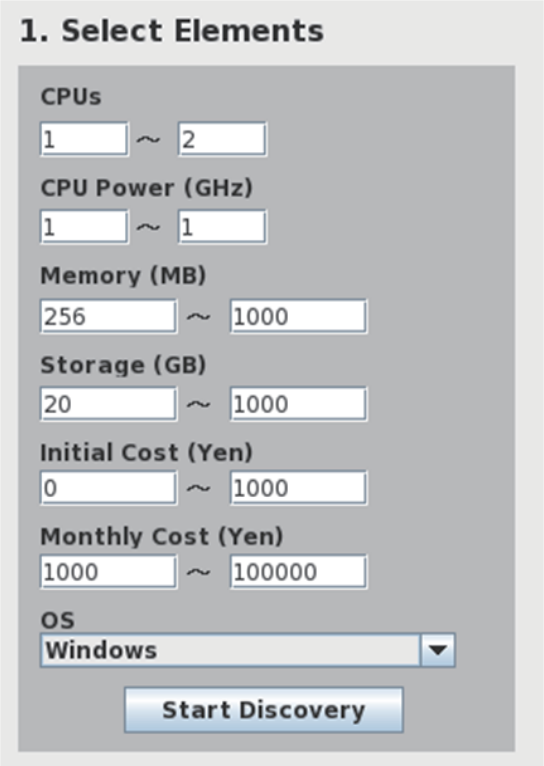
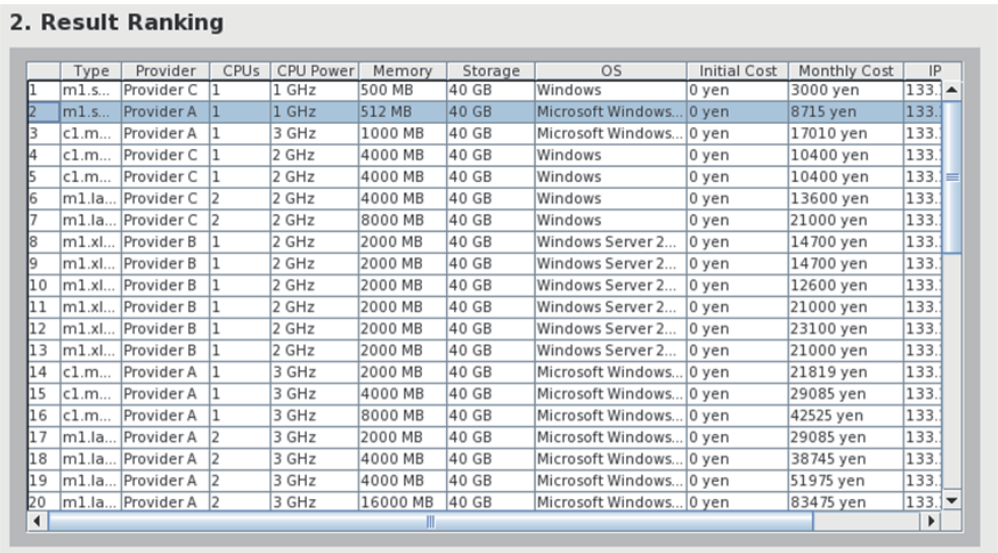


図 3.9 ユーザの検索項目



	Type	Provider	CPUs	CPU Power	Memory	Storage	OS	Initial Cost	Monthly Cost	IP
1	m1.s...	Provider C	1	1 GHz	500 MB	40 GB	Windows	0 yen	3000 yen	133.
2	m1.s...	Provider A	1	1 GHz	512 MB	40 GB	Microsoft Windows...	0 yen	8715 yen	133.
3	c1.m...	Provider A	1	3 GHz	1000 MB	40 GB	Microsoft Windows...	0 yen	17010 yen	133.
4	c1.m...	Provider C	1	2 GHz	4000 MB	40 GB	Windows	0 yen	10400 yen	133.
5	c1.m...	Provider C	1	2 GHz	4000 MB	40 GB	Windows	0 yen	10400 yen	133.
6	m1.la...	Provider C	2	2 GHz	4000 MB	40 GB	Windows	0 yen	13600 yen	133.
7	m1.la...	Provider C	2	2 GHz	8000 MB	40 GB	Windows	0 yen	21000 yen	133.
8	m1.xl...	Provider B	1	2 GHz	2000 MB	40 GB	Windows Server 2...	0 yen	14700 yen	133.
9	m1.xl...	Provider B	1	2 GHz	2000 MB	40 GB	Windows Server 2...	0 yen	14700 yen	133.
10	m1.xl...	Provider B	1	2 GHz	2000 MB	40 GB	Windows Server 2...	0 yen	12600 yen	133.
11	m1.xl...	Provider B	1	2 GHz	2000 MB	40 GB	Windows Server 2...	0 yen	21000 yen	133.
12	m1.xl...	Provider B	1	2 GHz	2000 MB	40 GB	Windows Server 2...	0 yen	23100 yen	133.
13	m1.xl...	Provider B	1	2 GHz	2000 MB	40 GB	Windows Server 2...	0 yen	21000 yen	133.
14	c1.m...	Provider A	1	3 GHz	2000 MB	40 GB	Microsoft Windows...	0 yen	21819 yen	133.
15	c1.m...	Provider A	1	3 GHz	4000 MB	40 GB	Microsoft Windows...	0 yen	29085 yen	133.
16	c1.m...	Provider A	1	3 GHz	8000 MB	40 GB	Microsoft Windows...	0 yen	42525 yen	133.
17	m1.la...	Provider A	2	3 GHz	2000 MB	40 GB	Microsoft Windows...	0 yen	29085 yen	133.
18	m1.la...	Provider A	2	3 GHz	4000 MB	40 GB	Microsoft Windows...	0 yen	38745 yen	133.
19	m1.la...	Provider A	2	3 GHz	4000 MB	40 GB	Microsoft Windows...	0 yen	51975 yen	133.
20	m1.la...	Provider A	2	3 GHz	16000 MB	40 GB	Microsoft Windows...	0 yen	83475 yen	133.

図 3.10 検索結果

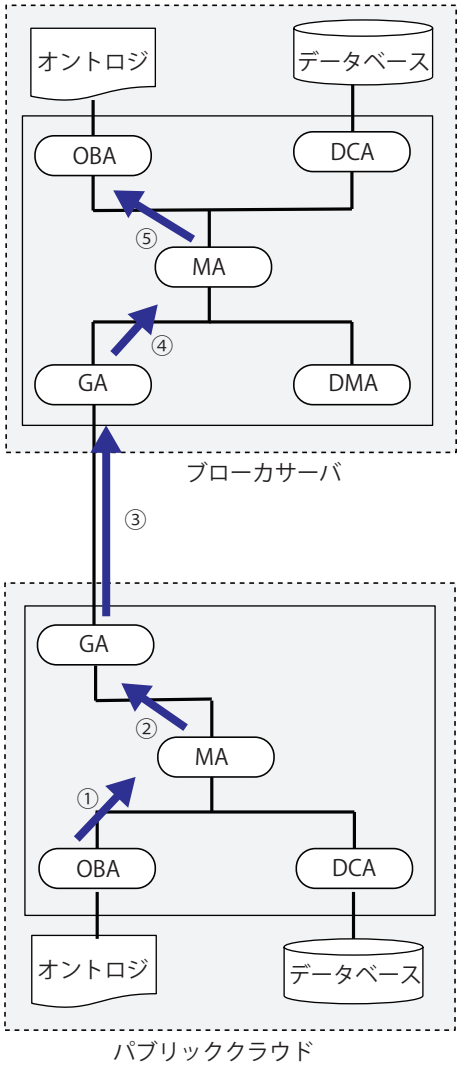


図 3.11 オントロジ更新時のエージェントの処理の流れ

ジマージ処理は全体で 1639ms かかった．ブローカサーバから追加されるプロバイダへのオントロジデータ要求に 139ms かかり，パブリッククラウドからブローカサーバへのオントロジデータの転送に 158ms かかった．パブリッククラウドでのオントロジデータの読み込みに 112ms かかり，ブローカサーバでのオントロジのマージに 1230ms かかった．

通信全体にかかった時間とオントロジの読み込みとマージの時間の割合を図 3.15 に示す．オントロジ関連の処理が全体の 82 %を占めていることがわかる．このことからオントロジマージ処理のボトルネックは通信ではなくマージ処理にあり，マージ処理のアルゴリズムを改善することで処理全体にかかる時間を短縮することが可能であると考えられる．

同様に，アップデートパッチを使用するオントロジの更新の検証を行う．パブリッククラウドのオントロジの更新はマージと同様のものとする．アップデートパッチのサイズは 1KB である．パブリッククラウドのオントロジの変更が検知され，ブローカサーバのオン

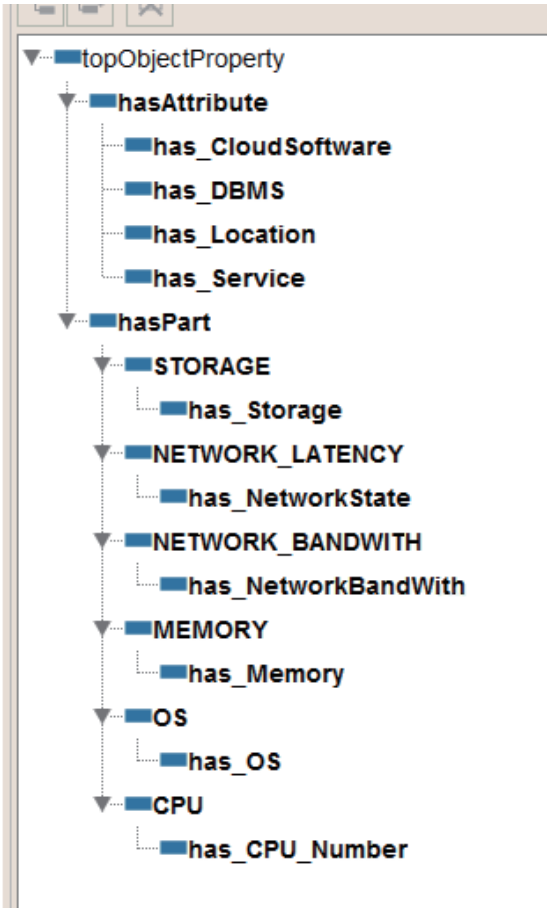


図 3.12 ブローカのオントロジのプロパティ

トロジに更新が適用されるまでに全体で 230ms であった．パブリッククラウドからブローカサーバへの転送に 138ms かかり，ブローカサーバでのアップデート処理には 92ms かかった．

2 つの方法の更新時間の比較を行った．オントロジ全体を転送しマージする場合は 1230ms であり，アップデートパッチを用いる場合は 230ms であった．処理時間は 20 %を下回る結果となった．よって，アップデートパッチを用いてブローカサーバのオントロジを更新する方法は有効といえる．

3.5 まとめ

オントロジとエージェント技術を使用したクラウドサービス発見システムの提案を行った．プライベートクラウドやパブリッククラウドのインスタンスタイプをオントロジを用いて表現の統一を行った．また，プライベートクラウドやパブリッククラウドのオントロジやデータベースが更新された場合，エージェントを用いてブローカサーバのオントロジやデー

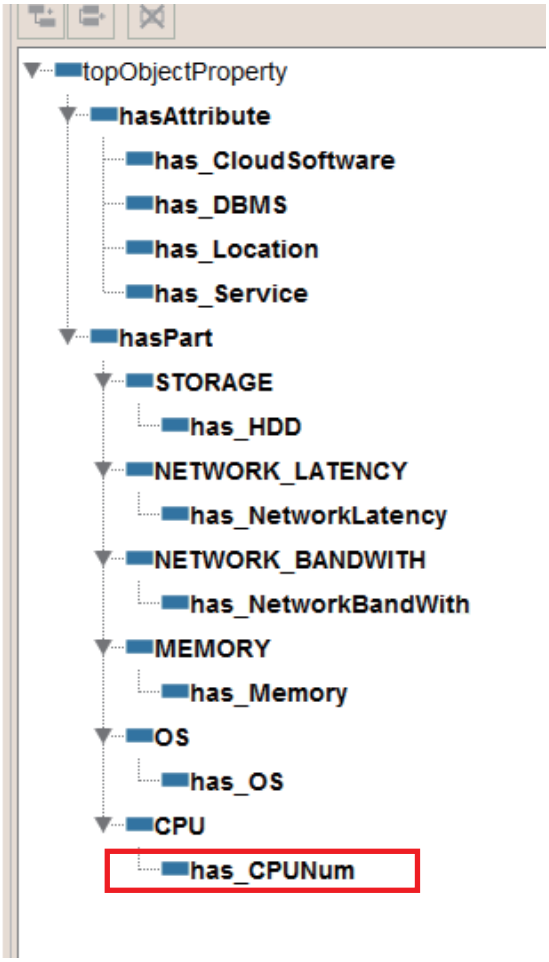


図 3.13 パブリッククラウドのオントロジのプロパティ

タベース更新を行った．オントロジの更新には本研究で提案したオントロジアップデートを使用することで効率化を行うことができた．

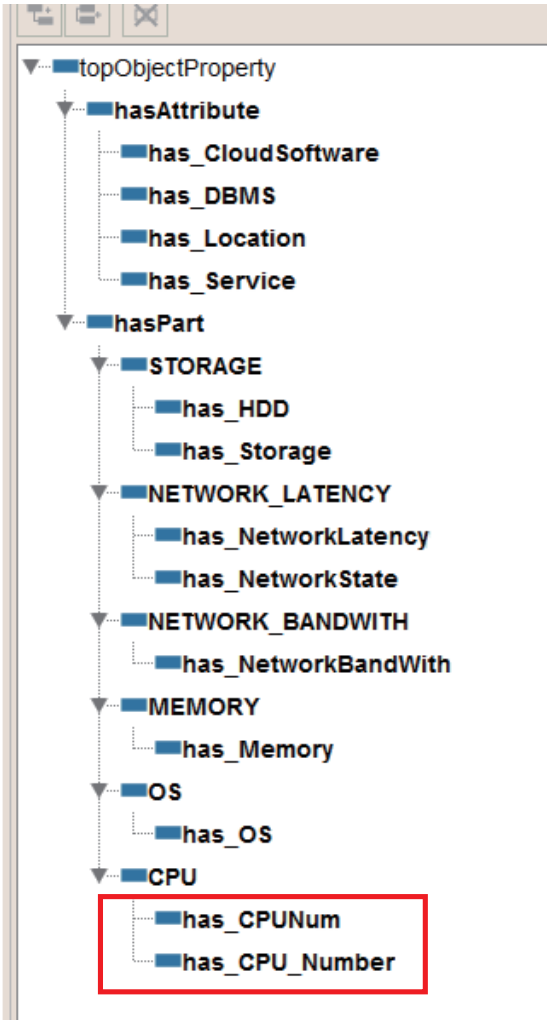


図 3.14 マージされた結果

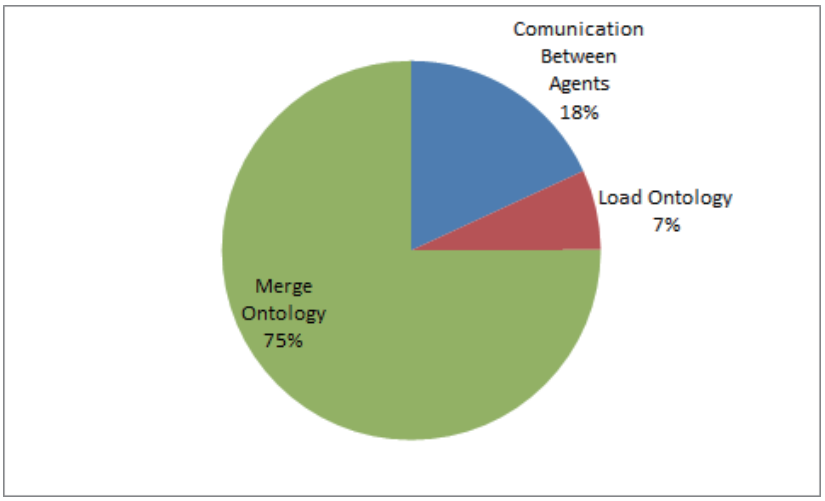


図 3.15 検索結果

第 4 章

セマンティックグリッドサービスの提案

本章では，セマンティックグリッドサービスを提案を行う．

4.1 はじめに

グリッドコンピューティングを用いることで，ネットワークを介して複数のコンピュータを結び，利用者がそこから必要な資源を取り出して使用することができるようになる．しかし，グリッドを利用するには様々な知識が必要となる．そこで，利用者がグリッドの使用方法を熟知していなくてもグリッドを利用できるグリッドサービスの提案を行う．グリッドサービスは，ウェブアプリケーションからファイルの転送や計算処理などのグリッド上でのジョブを実行できるようにしたサービスである．グリッドサービスを利用するにはノード情報を知っておかなければならない．そこで，セマンティックグリッドを使用する．セマンティックグリッドは，ノード情報についてのオントロジを利用することで，正確なノード情報を知らなくても不完全な情報からノードを特定することができる．このグリッドサービスとセマンティックグリッドとエージェント技術を組合わせたセマンティックグリッドサービスを提案する．セマンティックグリッドサービスを利用することにより，アプリケーションが最適なノードを選択しジョブの記述を行うため，利用者の負担を減らすことができる．

4.2 関連研究

本節では関連研究を説明する．

表 4.1 Globus Tookit の機能比較

	Globus Toolkit 4	Globus Toolkit 5
GRAM	○	○
MDS	○	×
GridFTP	○	○
GSI	○	○

4.2.1 グリッドコンピューティング

グリッドコンピューティングとは，ネットワークを介して複数のコンピュータを結ぶことで，利用者がそこから必要な資源を取り出して使用するサービスの仕組みである．グリッドには，計算資源だけでなくデータベースやセンサなどの様々な資源を接続することができる．グリッドの名称の由来は，電力網 (Power Grid) からきている．電源プラグをコンセントに挿すと電力網から送電される電力を簡単に使えるように，グリッドの利用者はネットワーク上の計算資源を必要なときに必要なだけ電気のように簡単に利用することが出来ることを目指した．

グリッドコンピューティングの環境は，ミドルウェアによって実現される．ミドルウェアとは，アプリケーションと OS の中間に位置するソフトウェアで，OS やハードウェアの仕様の違いを吸収し，それらを利用するための情報を統一的に提供する．グリッドミドルウェアには Globus Alliance が提供している Globus Toolkit[16] がある．Globus Toolkit は，複数のノードを管理しそれぞれにジョブを投入 (GRAM)，プロセッサ使用率など各ノードの情報を収集 (MDS)，ノード間でのデータ転送 (GridFTP)，セキュリティを確保 (GSI) などの機能を提供している．しかし，最新版の Globus Toolkit 5 以降では各ノードの情報を収集する MDS は外されており，情報を収集する手段を別途用意しなければならない (表 4.1)．

4.2.2 セマンティックグリッド

通常，利用者はどのノードがどのような処理を行えるかなどの情報を把握しておく必要があり，利用時にどのノードを利用するか明記しなければならない．ノード情報をまったく知らない利用者がグリッドを利用することは困難である．

この問題をセマンティックグリッドが解決する．セマンティックグリッドは，オントロジとグリッドを合わせたものである．ノード情報を表現する資源オントロジを利用することで，従来のグリッドと比べて柔軟なノードの検索が可能になる利点がある．

4.3 グリッドサービス

本研究では，ウェブアプリケーションとして処理を登録し，ジョブの記述を生成する仕組みの提案を行う．グリッドサービスは，ウェブアプリケーションからグリッドミドルウェアへアクセスし，ファイルの転送や計算処理などのジョブを実行できるようにしたサービスと本論文では定義する．

一般的にグリッド環境を利用する際には，利用者がジョブの記述を行ってグリッド環境を利用する．下記のジョブは，Globus Toolkit のジョブを投入する機能である GRAM を使用した例である．この例では，ノード grid.example.org 上で SubOntExt.jar を実行してサブオントロジを抽出している．サブオントロジ抽出時の引数を arguments で指定している．その後，抽出されたサブオントロジを GridFTP を使用してローカルのディレクトリへ転送ている．GridFTP は Globus Toolkit の FTP 転送機能であり，利用時には gsiftp というプロトコルを指定する．これらのジョブの記述は利用者が行わなければならず，利用者への負担が大きい．

```
globusrun -s -r grid.example.org:2119/jobmanager-pbs
"&(executable=/home/griduser/bin/SubOntExt.jar)
(arguments=umlssn.owl, Physical Object T072, connected to T174 )
(file\_stage\_in = (gsiftp://grid.example.org:2119/opt/owl/sub-umlssn.owl
/tmp/subont))"
```

本研究で提案するグリッドサービスでは，管理者または使用者がグリッドネットワークを利用する処理をあらかじめ定義しておくことで，使用者はウェブブラウザでウェブアプリケーションにアクセスし容易にグリッド環境を利用できる．図 4.1 はグリッドサービスの構成を示している．インデックスサービスノードは，同じグリッドネットワーク上のノード情報をデータベースにインデックス情報として保管している．

グリッドサービスは，グリッドネットワーク上にグリッドポータルサーバが存在する．グリッドポータルサーバではウェブサーバとグリッド API が連携しており，ウェブアプリケーションを提供している (図 4.2)．あらかじめ，管理者がジョブをウェブアプリケーションに登録しておくことで，ウェブアプリケーションとグリッド API を使用した処理が行える．その結果，ウェブブラウザ経由でグリッドを簡単に使用することができるようになる．同じ系統の処理であれば使用者が新たにジョブを作成することなく，ジョブを容易に生成することができる．

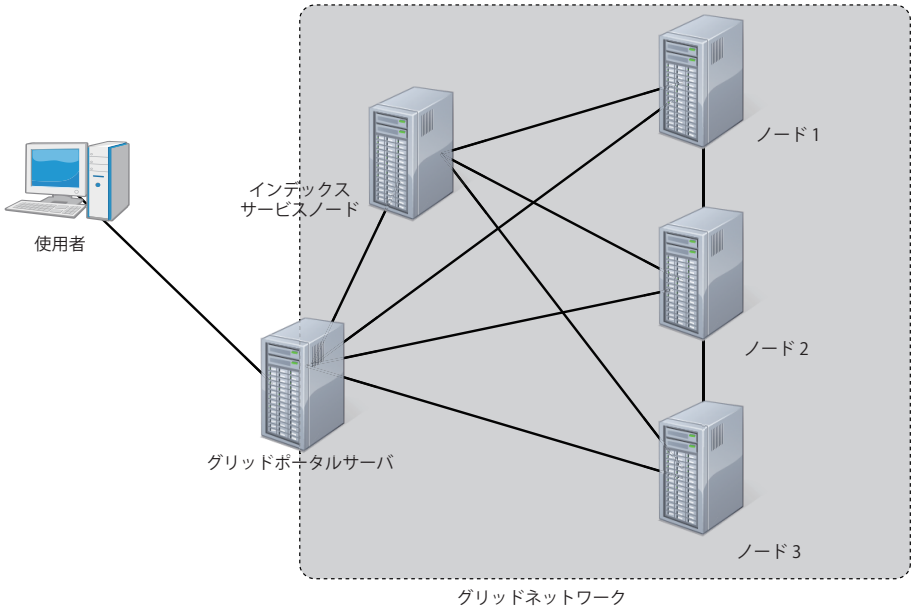


図 4.1 グリッドサービス環境

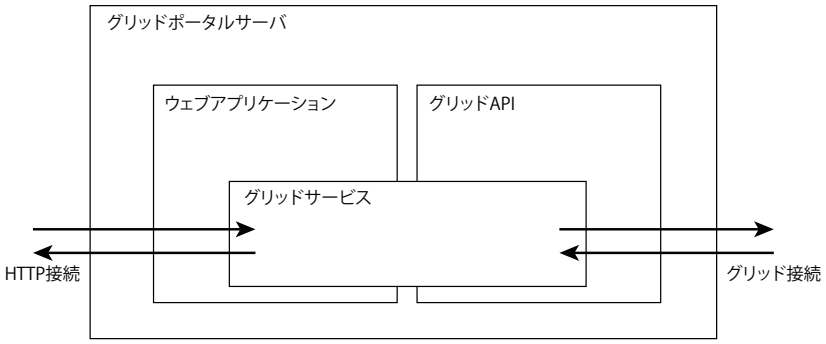


図 4.2 グリッドポータルサーバの構成

4.4 エージェントの利用

グリッドでは動的なノード情報の取得は行えず，Globus Toolkit の MDS を用いてもあらかじめ登録された静的なプロセッサ性能などしか取得することができなかった．SNMP[32] などで動的な情報をとることもできるが，状況に応じて柔軟に情報を得ることは難しかった．

そこで本研究では，ノード情報の取得にエージェントを使用する．エージェントを使用しノードへ常駐させることで，ノードの動的な情報を取得できる．また，どのような情報を取得するかをエージェントへ指示できるため，状況に応じた情報の収集も行える．

ノードの動的な情報は 1 つのサーバに集約し，ノード情報検索の際はオントロジの静的な情報と合わせて検索を行う．図 4.3 は，ノード情報収集のエージェントの構成を示してい

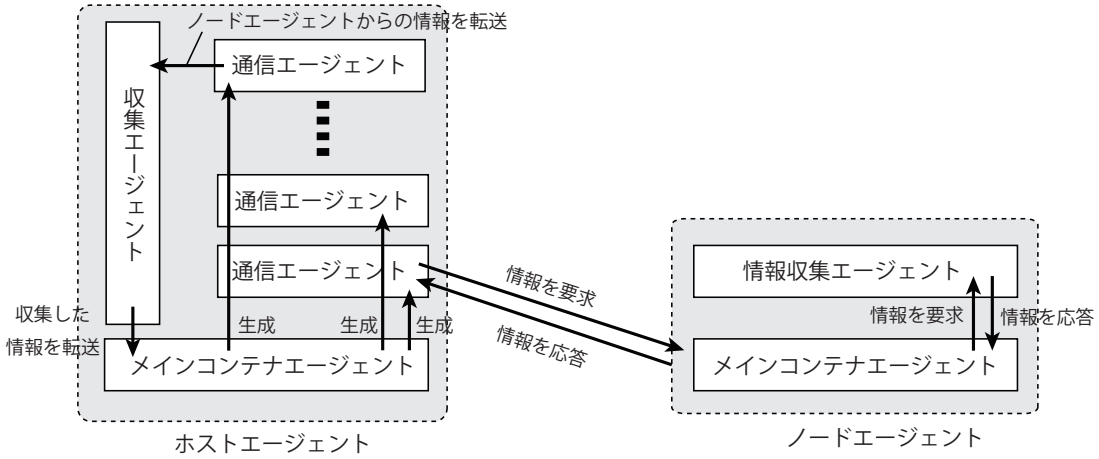


図 4.3 エージェントの構成

る．エージェントはホストエージェントとノードエージェントに分類される．ホストエージェントは 1 つのグリッド環境に 1 つしか存在しない．ノードエージェントはノード上で動作するエージェントの集まりであり，ノードの数だけ存在する．

ホストエージェントのメインコンテナエージェントは，通信エージェントの生成や収集エージェントからデータの受け取りを行う．通信エージェントは，ノードエージェントへの動的情報の要求と受け取りを行い，受け取った動的な情報を収集エージェントへ渡す．動的な情報の受け取りが終われば通信エージェントは破棄される．

ノードエージェントのメインコンテナエージェントは，ホストエージェントの窓口となる．ホストエージェントから要求があった場合，情報収集エージェントへ動的な情報を要求する．情報収集エージェントは動的な情報の収集を行うエージェントである．

エージェントを使用することで，現在のノードの状態といった動的な情報を取得できるようになる．さらに，オントロジで静的情報に意味付けを行うことで推論検索が行えるようになる．

4.5 セマンティックグリッドサービスの提案

本研究は，セマンティックグリッドサービスの提案を行う [30][31]．セマンティックグリッドサービスは，処理に使用するノードやアプリケーションの選択をウェブアプリケーションのサービスとして組み込む．アプリケーションごとの，オントロジを利用したノード発見や選別をウェブサーバで行うようにしたものである．不完全な要求でも対応するためにセマンティックウェブを組み合わせたセマンティックグリッドを利用することで，資源情報をまったく知らない利用者でもグリッドを使用することができる．図 4.4 はセマンティックグリッドサービスの構成を示す．

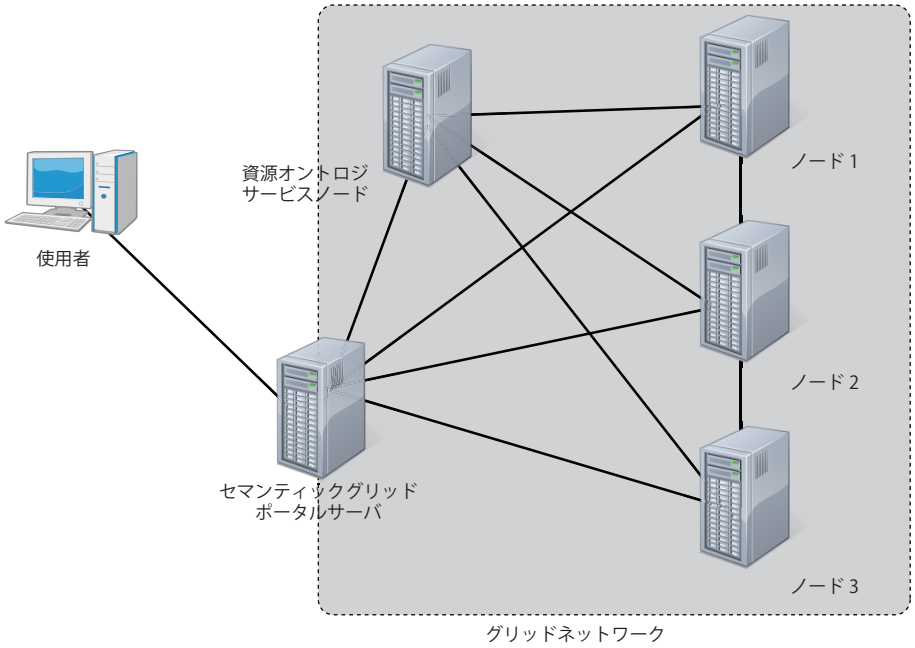


図 4.4 セマンティックグリッドサービス環境

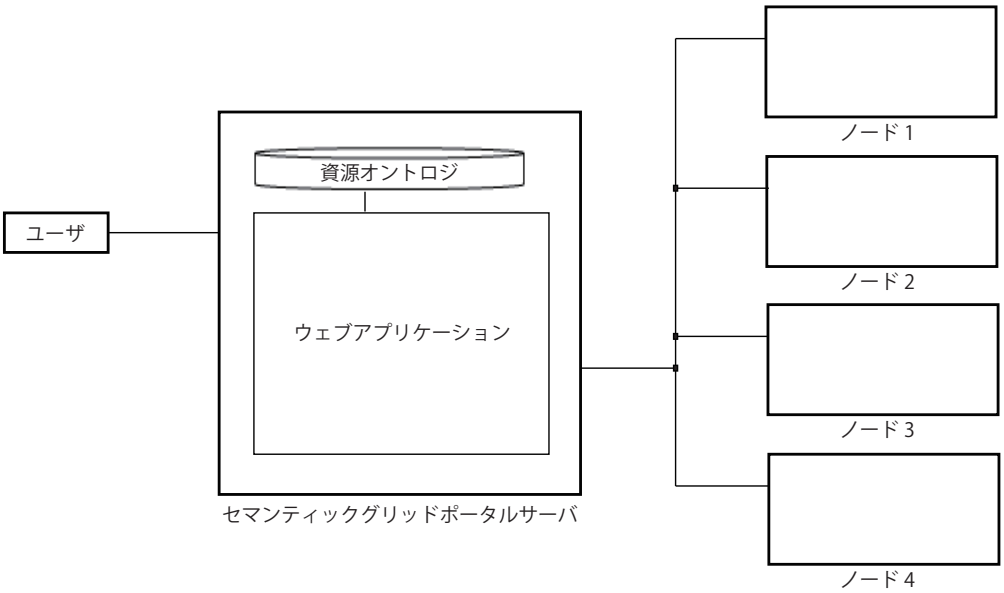


図 4.5 セマンティックグリッドサービスの構成

セマンティックグリッドサービスは，図 4.2 のグリッドポータルサーバの代わりに，ノード情報を保管する資源オントロジを組み込んだセマンティックグリッドポータルサーバを配置する．セマンティックグリッドサービスの構成を図 4.5 に示す．セマンティックグリッドサービスシステムは，セマンティックグリッドポータルサーバとノードで構成される．セマンティックグリッドポータルサーバとすべてのノードにはエージェントが配置される．

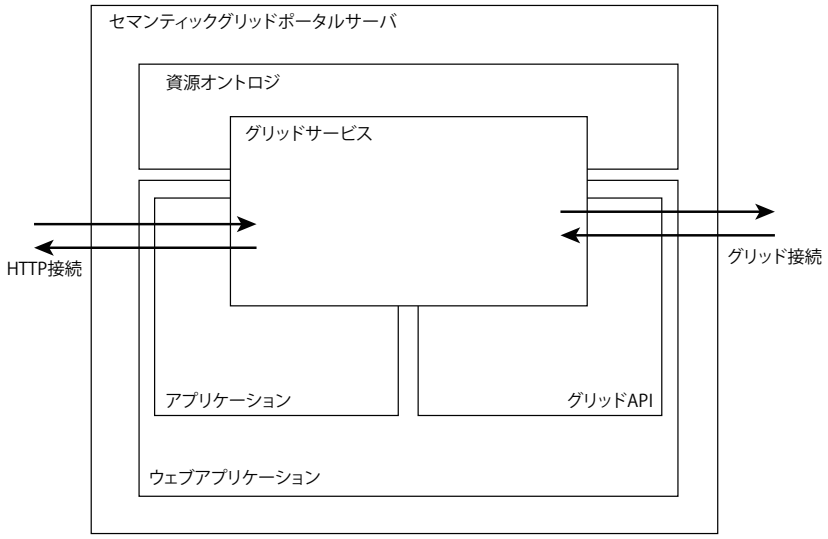


図 4.6 セマンティックグリッドポータルサーバの構成

図 4.6 は，セマンティックグリッドサービスのセマンティックグリッドポータルサーバの構成を示している．ウェブアプリケーションでこの資源オントロジを利用する処理を記述することで，どのノードを使用するかをサーバ側で自動的に決定することを可能としている．従来のグリッドとは異なり，グリッドサービスによって利用者は細かなジョブを指定せずに利用でき，セマンティックグリッドによって利用者はどのノードを利用するかを意識せずに処理を行うことができる．

4.6 ケーススタディ

分散したデータを管理するためにグリッドを使用する．グリッドを使用するには，利用者によるノード情報の把握とジョブの記述が必要である．ジョブはノードの利用方法を決めるために必要であり，必ず記述しなければならない．そのことが利用者がグリッドを使用する妨げとなっている．

そこで，分散しているオントロジのデータを管理するために，提案したセマンティックグリッドサービスを使用する．すべてのノードの計算機性能を表 4.2 に示す．実装した環境の構成を図 4.7 に示す．4 つのノードと 1 つのセマンティックグリッドポータルサーバで構成する．利用者はセマンティックグリッドポータルサーバへウェブブラウザでアクセスし，グリッドを利用する．図 4.8 にウェブアプリケーションの画面を示す．キーワードを入力しセマンティックグリッドサービスを使用する．

また，実装の詳細を図 4.9 に示す．セマンティックグリッドポータルサーバと資源は，エージェントとグリッドでつながっている．ウェブアプリケーションの実行環境は Tomcat

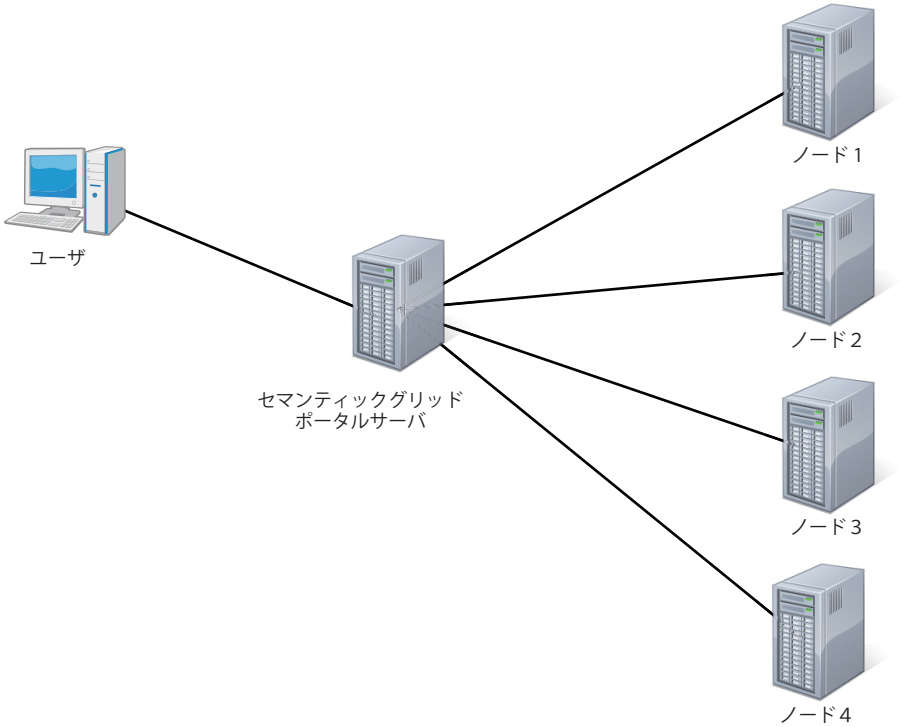


図 4.7 実装した環境の構成

図 4.8 サブオントロジ抽出アプリケーション画面

表 4.2 ノードの性能

CPU	Core-i 7 2.67GHz
MEMORY	12GB
STORAGE	500GB
OS	Ubuntu 10.04

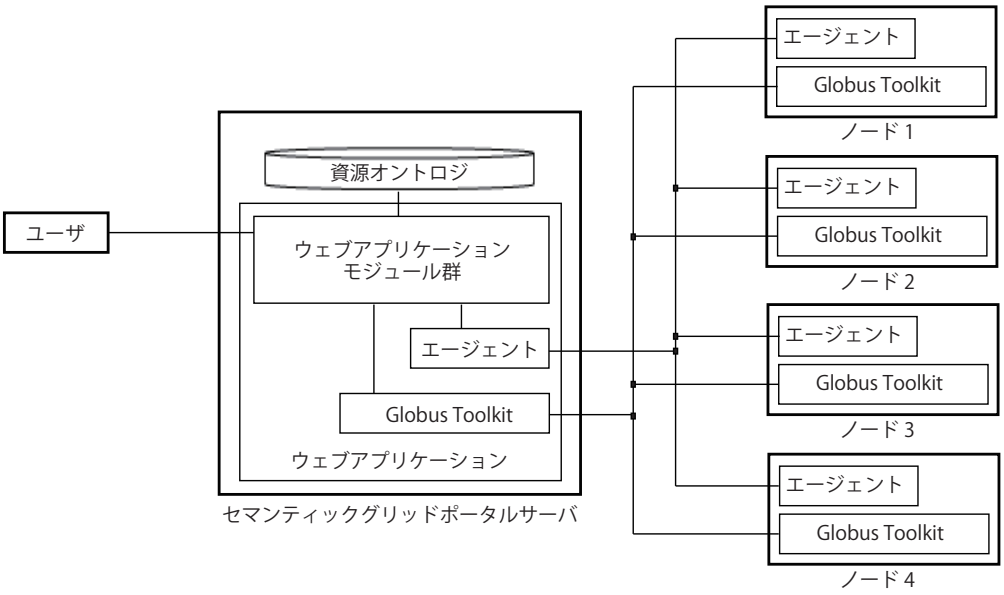


図 4.9 実装環境

6 で構築し，JSP と Servlet で動作の記述を行った．エージェントは JADE[17] を使用し，グリッドは Globus Toolkit 5.0.2 で構築する．Servlet でのオントロジの読み込み書き込みは Jena を利用する．

図 4.9 のセマンティックグリッドポータルサーバのウェブアプリケーションの構成を図 4.10 に示す．6 つのモジュールで構成されている．ログインモジュールはログインを管理する．利用者はユーザ ID とパスワードを入力しログインモジュールを介してグリッドへのログインを行う．グリッドポータルモジュールは，グリッドアプリケーションを利用するための窓口を提供する．アプリケーションモジュールは，登録されたウェブアプリケーションが保管されている．資源発見モジュールは，エージェントとオントロジを使用してノード発見を行いジョブマネージャモジュールへ渡す．ジョブマネージャモジュールは，ジョブの流れを管理し，各モジュールへ指示を行う中心的なモジュールである．グリッド接続モジュールは，グリッドとの通信を行う．

資源発見モジュールは，資源オントロジとエージェントと通信を行う．資源オントロジを

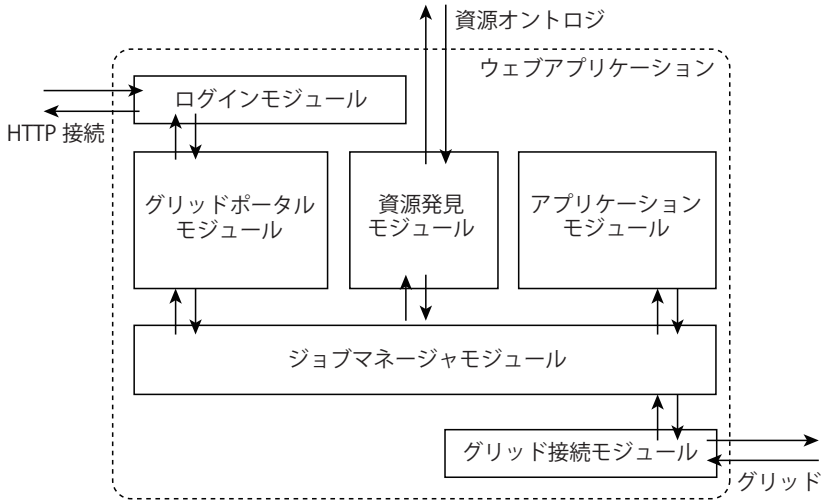


図 4.10 セマンティックグリッドポータルサーバの構成

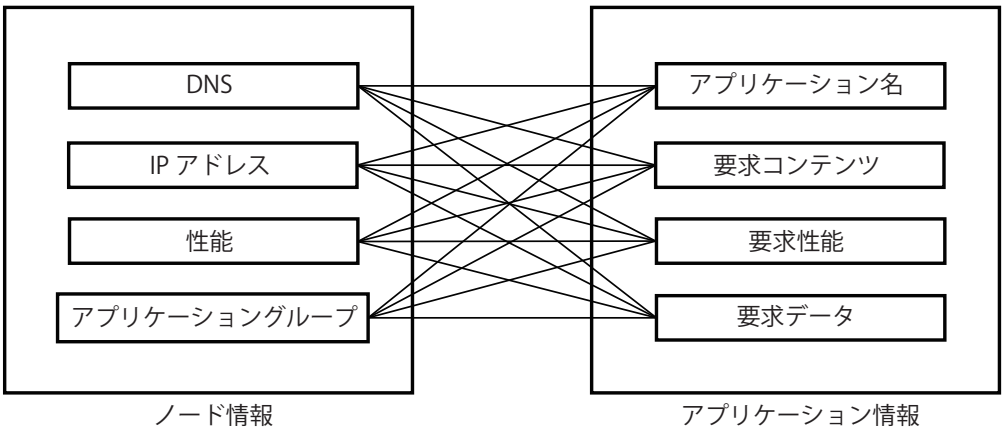


図 4.11 資源オントロジの構造

図 4.11 に示す．資源オントロジの要素は主にノード情報とアプリケーション情報の 2 つに分けられる．これらの情報が関連性を含んで記録されている．例えば，アプリケーション名だけを検索条件にした場合，アプリケーションを利用できるノードの IP 一覧が結果として返ってくる．資源オントロジを使用することで，このような柔軟なノードの検索を行えるようなサービスが構築できる．

以下で，セマンティックグリッドサービス上にアプリケーションを実装した場合の動作例を説明する．

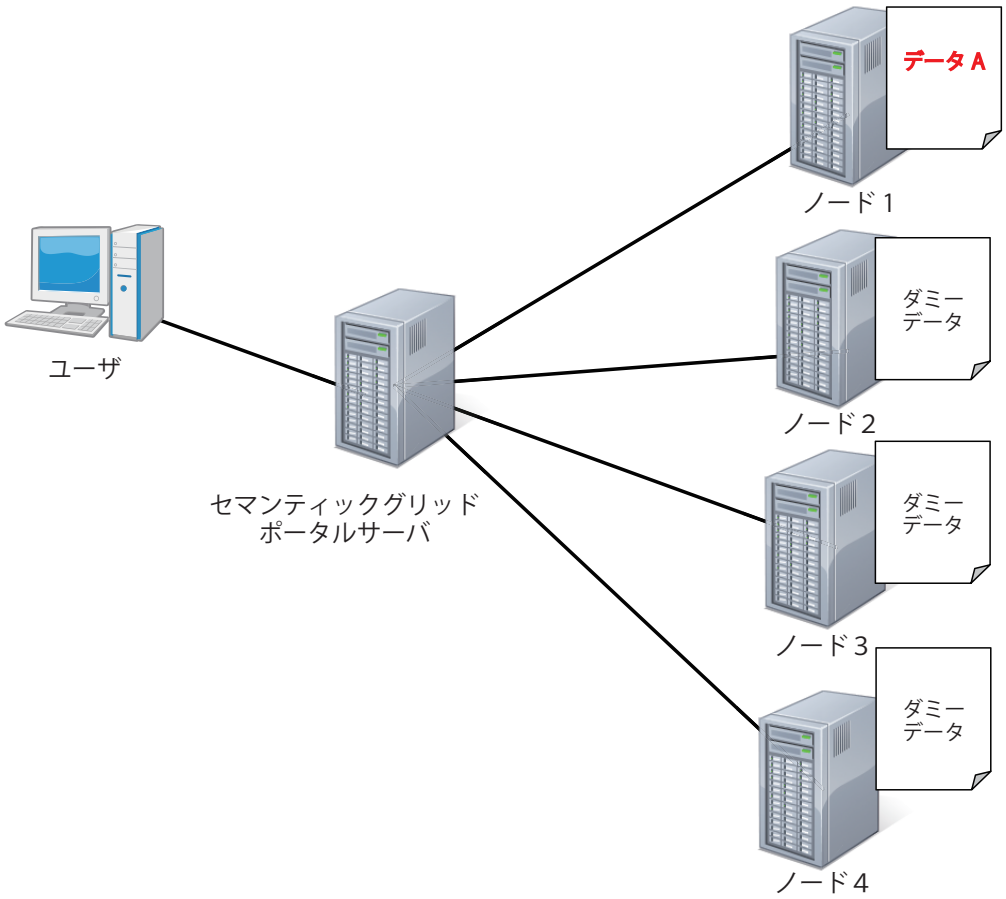


図 4.12 データの要求アプリケーションの環境

4.6.1 データの要求アプリケーション

ノード 1 上にデータ A を配置する．他のノードにはダミーデータを配置する．図 4.12 に環境を示す．利用者はデータ A を利用するために，セマンティックグリッドサービスへデータ A を要求した場合の例を示す．セマンティックグリッドポータルサーバの資源オントロジには，どのノードにどのデータがあるか記述している．

従来のグリッドでは，どのノードに要求するデータ A が配置されているか利用者は把握しなければいけない．さらにデータを要求するジョブを作成し，グリッドでジョブを実行する．

セマンティックグリッドサービスでは，利用者はセマンティックグリッドポータルサーバへアクセスし，データ要求のアプリケーションを利用する．利用者はデータ A をウェブアプリケーションに要求することで，処理が始まる．まず初めに，要求はセマンティックグリッドポータルサーバは，ジョブマネージャモジュールを介して資源発見モジュールへ伝え

られ，資源オントロジを検索する．

資源発見モジュールは，データ A とデータ要求のアプリケーション情報を基に資源オントロジを検索し，ノード 1 の IP アドレスが結果として返ってくる．エージェントでノード 1 の動的な情報を調べる．資源発見モジュールは，ノード 1 の IP アドレスをジョブマネージャへ渡す．

ジョブマネージャモジュールでは，ノード 1 のデータ A をブローカサーバへ転送するジョブの生成が行われる．生成が完了しジョブを実行することで，ブローカサーバにデータ A が転送されてくる．

最後に，転送されてきたデータ A を利用者へ返す．

データ A とアプリケーションを指定するだけで，グリッド上のノードを検索しデータを転送するというジョブが生成される．利用者はグリッドのジョブやノードの情報を知らずともグリッドを利用できる．

4.6.2 2 つのセマンティックグリッドサービスでのデータ要求

ここでは，2 つのセマンティックグリッドサービスが存在する環境を用いてデータの転送を行う．異なるセマンティックグリッドサービスに属するノードにそれぞれデータ A とデータ B が配置されている．他のノードにはダミーデータを配置する．それぞれのセマンティックグリッドサービスの環境を VO-A と VO-B とする．図 4.13 に環境を示す．VO-A のノード 1 にデータ A を配置する．残りのノードにダミーデータを置く．VO-B のノード 1 にデータ B を配置する．残りのノードにダミーデータを置く．

図 4.14 は利用者がデータを要求した場合の例を示している．利用者の要求に対して 2 つのノードが選択された例である．

利用者は，VO-B のセマンティックグリッドポータルサーバへウェブブラウザで接続する．アプリケーションと要求するデータのデータ A とデータ B を入力して実行すると，ノードの検索が行われる．

VO-B のセマンティックグリッドポータルサーバは，VO-A のセマンティックグリッドポータルサーバへキューを送る．それぞれのセマンティックグリッドポータルサーバでノードの検索を行い，エージェントで動的な情報を調べ結果を VO-B のセマンティックグリッドポータルサーバへ集約する．ノードの検索結果を基にジョブが生成される．

データは VO-A と VO-B のノード 1 からそれぞれのセマンティックグリッドポータルサーバへ転送される．VO-A のセマンティックグリッドポータルサーバは転送されてきたデータ A を VO-B のセマンティックグリッドポータルサーバへ転送する．VO-B のセマンティックグリッドポータルサーバへ集められたデータ A とデータ B を返す．

2 つ以上のセマンティックグリッドサービスが存在する環境では，セマンティックグリッ

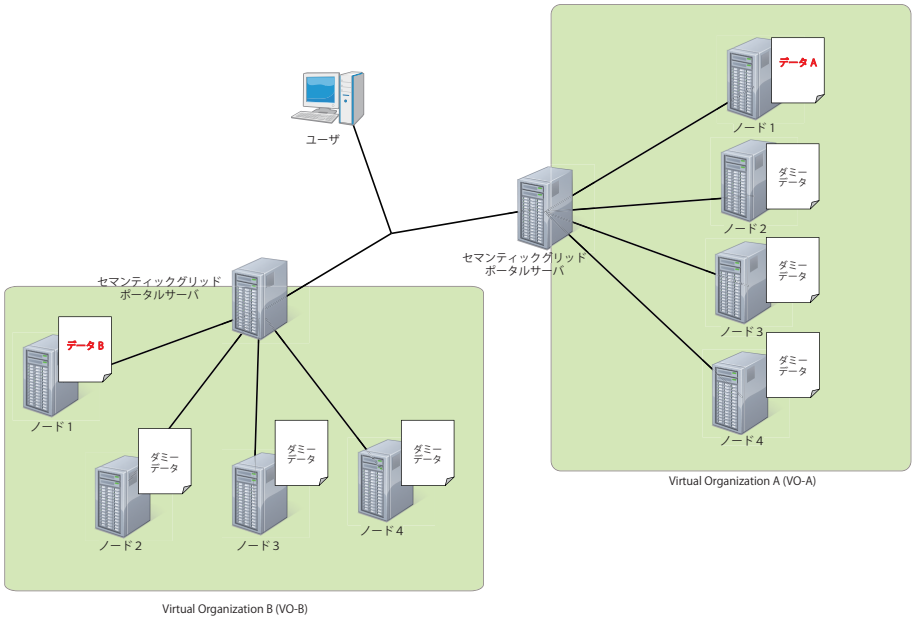


図 4.13 2 つのセマンティックグリッドサービスが存在する環境

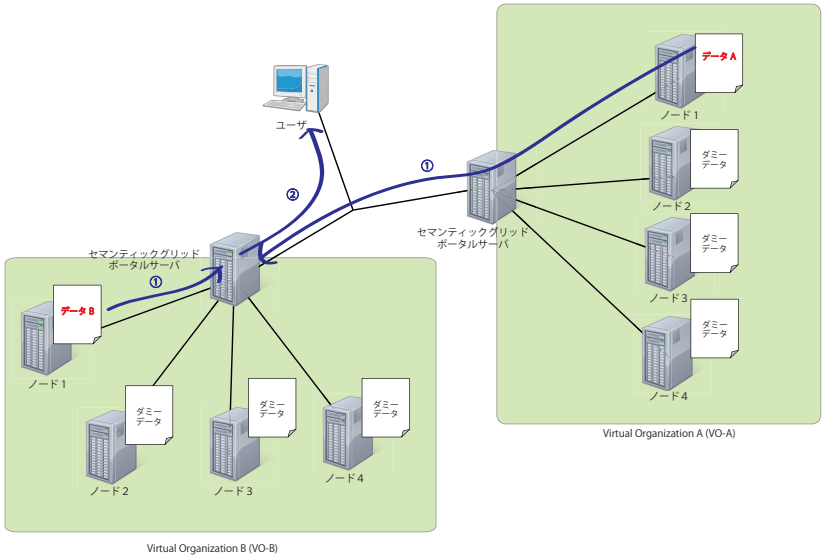


図 4.14 オントロジの配置と流れ

ドポータルサーバ間で連携を行うことができる．

4.7 まとめ

グリッドを使用するには利用者によるノード情報の把握とジョブの記述が必要である．そこでセマンティックグリッドサービスを提案した．

セマンティックグリッドの資源オントロジを利用したノードの検索と，グリッドサービスのジョブの生成を合わせることで利用者の負担を減らすことができる．ケーススタディでは，1 つのセマンティックグリッドサービス環境におけるデータ要求と 2 つのセマンティックグリッドサービス環境におけるデータ要求の例を示した．

第 5 章

結論

本研究では，オントロジのアップデート管理とクラウドコンピューティングへの適用，そしてセマンティックグリッドサービスの提案を行った．

オントロジのアップデート管理は，オントロジアップデートとサブオントロジアップデートの提案を行った．オントロジアップデートでは，同一内容のオントロジのコピーがネットワーク上に存在している場合もあり，同一性を維持管理する必要がある．オントロジ自体を再度転送する代わりにオントロジの変更内容を記述したアップデートパッチを転送する方式を提案した．サブオントロジアップデートでは，元のオントロジが更新された場合に，既に抽出され転送されたサブオントロジを効率的にアップデートする方法として，再度サブオントロジを抽出するのではなくアップデートパッチを転送しサブオントロジを直接更新する．再度抽出したサブオントロジとサブオントロジアップデートを使用して更新したサブオントロジは，同一のサブオントロジが生成できた．また，サブオントロジアップデートを使用することで，再度サブオントロジを抽出するよりも速く更新後のサブオントロジを生成することができた．

クラウドコンピューティングに，オントロジアップデート管理で提案したオントロジアップデートを適用した．利用者へ適切なクラウドサービスを提示するクラウドサービス発見システムの提案を行った．多数のクラウド提供企業から様々なクラウドサービスが提供されているため，利用者が適切なクラウドサービスを選択するのは困難である．さらに，各クラウドサービスによりインスタンスタイプの性能の表現が異なっている．そこでオントロジを使用することで表現の差異を吸収する．オントロジは各クラウドサービスに分散配置し，変更があればブローカサーバへ転送しマージする．このオントロジ管理に本研究で提案したオントロジアップデートを適用した．オントロジ全体をブローカサーバへ転送しマージする場合の処理時間と比較を行った．オントロジアップデートを使用するほうが処理時間が速くなることが確認できた．

セマンティックグリッドサービスでは，セマンティックグリッドとエージェント技術とグ

リッドサービスを組み合わせたグリッドコンピューティング環境の提案を行った．グリッドコンピューティングとは，ネットワークを介して複数のコンピュータを結ぶことで，利用者がそこから必要な資源を取り出して使用するサービスの仕組みである．グリッドコンピューティング環境を効率的に利用するためには，実行したい処理内容をジョブとして個別に毎回記述する必要がある．そこでグリッドサービスを提案した．グリッドサービスでは，ウェブアプリケーションからファイルの転送や計算処理などのグリッド上でのジョブを実行する．利用者がグリッド上で実行したい処理について毎回個別にジョブの内容を記述する必要性を軽減する．グリッドサービスを利用するには分散している資源情報を事前に知っている必要がある．そこで，セマンティックグリッドでこれらの問題の解決を行った．提案したセマンティックグリッドサービスを使用することで，利用者がグリッドの使用方法を熟知していなくてもグリッドを利用できる．

今後は，実際のパブリッククラウドを使用して提案システムの検証を行いたい．そして，パブリッククラウドとプライベートクラウドを用いたハイブリッドクラウドの研究を行っていきたいと思う．

謝辞

九州産業大学情報科学部情報科学科の仲隆教授には，主査として多くのコメントやご指導を頂き深く感謝いたします．本研究にあたり日頃からご指導を賜りました九州産業大学情報科学部情報科学科のアプドゥハン・ベーナディ教授に深く感謝致します．九州産業大学情報科学部情報科学科の成凱教授には副査として御助言を頂き深く感謝いたします．

九州産業大学情報科学部情報科学科の下川俊彦教授には多くの御助言を頂き深く感謝いたします．本研究を進めるにあたって多数のご教示を頂きました九州産業大学情報科学部情報科学科の安武芳紘准教授に深く感謝いたします．早稲田大学の白鳥則郎教授には研究に関する多数のご助言を頂き深く感謝いたします．九州産業大学情報科学部情報科学科の神屋郁子助手には御助言を頂きました，深く感謝いたします．九州産業大学情報科学部情報科学科の林政喜助手には御助言を頂きました，深く感謝いたします．

ラトロボ大学のウェニー・ラハユ准教授とモナシュ大学のデビッド・タニエール准教授にはオントロジ技術の専門知識に関して貴重なご助言をいただきました深く感謝いたします (I would like to express my heartfelt gratitude to Prof. W. J. Rahayu of La Trobe University, Australia and Prof. D. Taniar of Monash University, Australia for sharing their ontology technology expertise and valuable advices)．

また，これまで支えてくださった父と母，そして祖母に感謝いたします．

参考文献

[1] G. Pfister, In Search of Clusters. 2nd Edition, Prentice-Hall,Englewood Cliffs, NJ, 1998.

[2] R. Buyya, High Performance Cluster Computing. Architectures and Systems, Volume 1, 1999.

[3] Message Passing Interface Forum (online), available from (`http://www.mpi-forum.org/`)

[4] C. Catlett and L. Smarr, Metacomputing, Communications of the ACM, Vol35, No.6, pp44-52,1992.

[5] I. Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp.2-13, 2006.

[6] T. Berners-Lee, J. Hendler, and O. Lassila: The Semantic Web. Scientific American. 284(5):pp.34-43, May 2001.

[7] T. Uchibayashi, B. O. Apduhan, W. J. Rahayu, D. Taniar, and N. Shiratori: Towards Workflow Framework for Sub-ontology Extraction in Semantic Grid, The 3rd International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS2009), pp.1205-1210, 2009.

[8] 内林 俊洋, アプドゥハン・ベーナディ: サブオントロジー抽出におけるアップデートメカニズムの提案と検証, 電子情報通信学会技術研究報告 110(167), pp.79-84, 2010.

[9] T. Uchibayashi, B.O. Apduhan, N. Shiratori, W. Rahayu, D. Taniar: Verification of a Sub-ontology Update Mechanism for the Semantic Grid, The 2nd International Symposium on Multidisciplinary Emerging Networks and Systems (MENS 2010), pp.380-385, 2010.

[10] 神崎正英，セマンティック・ウェブのための RDF/OWL 入門，森北出版株式会社，2005 年.

[11] W3C: OWL Web Ontology Language Overview (online), available from (`http://www.w3.org/TR/owl-features/`)

[12] RDF Scheme (online), available from (http://www.w3.org/TR/rdf-schema/)

[13] M. Bhatt, A. Flahive, C. Wouters, J.W. Rahayu and D. Taniar,: MOVE: A Distributed Framework for Materialized Ontology View Extraction, Algorithmica 45(3), pp457-481, 2006.

[14] A. Flahive, D. Taniar, J.W. Rahayu, B.O. Apduhan, Ontology Tailoring in the Semantic Grid. Computer Standards & Interfaces 31(5), pp870-885, 2009

[15] Apache Jena (online), available from (http://jena.apache.org/)

[16] Globus Toolkit (online), available from (http://www.globus.org/toolkit/)

[17] Jade - Java Agent DEvelopment Framework (online), available from (http://jade.tilab.com/)

[18] T. Uchibayashi, B. O. Apduhan, N. Shiratori, and Yoshihiro Yasutake: A Framework of an Agent-based Support System for IaaS Service Discovery, The 2013 International Conference on Computational Science and Its Applications (ICCSA 2013), pp28-32, 2013.

[19] T. Uchibayashi, B. O. Apduhan, and N. Shiratori: An Ontology Update Mechanism in IaaS Service Discovery System, International Journal of Web Information Systems, Emerald Group Publishing Limited, Vol.9, No.4, pp.330-343, 2013.

[20] Amazon Web Services, Cloud Computing: Compute, Storage, Database(online), available from (http://aws.amazon.com/)

[21] Windows Azure: Microsoft's Cloud Platform — Cloud Hosting — Cloud Services (online), available from (http://www.windowsazure.com/)

[22] Cloud Stack (online), available from (http://cloudstack.apache.org/)

[23] Open Source Private and Hybrid Clouds from Eucalyptus — Open Source. AWS-compatible. Private Clouds(online), available from (http://www.eucalyptus.com/)

[24] Strebel, J and Stage , A.: An Economic Decision Model for Business Software Application Deployment on Hybrid Cloud Environments, Multikonferenz Wirtschaftsinformatik 2010, pp.195-206, 2010.

[25] Mell, P. and Grance, T.: Effectively and Securely Using the Cloud Computing Paradigm, National Institute of Standards and Technology, Information Technology Laboratory (2009).

[26] Palwe, R., Kulkarni, G. and Dongare, A.: A New Approach to Hybrid Cloud, International Journal of Computer Science and Engineering Research and Development (IJCSERD), Vol.2, No.1, pp.01-06, 2012.

[27] Gupta, A. K. and Gupta, M. K.: A New Era of Cloud Computing in Private and Public Sector Organization, International Archive of Applied Sciences and

Technology, Vol.3, pp.80-85, 2012.

[28] OpenNebula (online), available from (http://opennebula.org/)

[29] OpenStack Open Source Cloud Computing Software (online), available from (http://www.openstack.org/)

[30] T. Uchibayashi, B. O. Apduhan, and N. Shiratori: Construction and Analysis of a Semantic Grid Service for Large-scale Environment, The International Multi-Conference of Engineers and Computer Scientists 2012 (IMECS 2012), pp229-234, 2012.

[31] T. Uchibayashi, B. O. Apduhan, and N. Shiratori: A Domain Specific Sub-ontology Derivation End-user Tool for the Semantic Grid, Telecommunication Systems Journal, DOI 10.1007/s11235-013-9757-3, Published online: 23 October 2013.

[32] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin: Simple Network Management Protocol (SNMP), RFC 1157, 1990.

[33] H. Tangmunarunkit, S. Decker, and C. Kesselman: Ontology-Based Resource Matching in the Grid - The Grid Meets the Semantic Web. International Semantic Web Conference, volume 2870 of Lecture Notes in Computer Science, pp.706-721, 2003.

[34] O. Corcho, P. Alper, I. Kotsiopoulos, P. Missier, S. Bechhofer, and C. A. Goble: An Overview of S-OGSA: A Reference Semantic Grid Architecture. Web Sem. 4(2), pp.102-115, 2006.

[35] T. S. Somasundaram, R. A. Balachandar, V. Kandasamy, R. Buyya, R. Raman, N. Mohanram, and S. Varun: Semantic-based Grid Resource Discovery and its Integration with the Grid Service Broker. Future Generation Computer Systems archive, Volume 24, Issue 8, pp.806-812, 2008.

[36] Sheila A. McIlraith, Tran Cao Son, Honglei Zeng: Semantic Web Services, IEEE Intelligent Systems, vol. 16, no. 2, pp46-53, Mar./Apr. 2001.

[37] XML (online), available from (http://www.w3.org/XML/)

[38] RDF (online), available from (http://www.w3.org/RDF/)

[39] A. Flahive, B.O. Apduhan, J.W. Rahayu, D. Taniar, Large Scale Ontology Tailoring and Simulation in the Semantic Grid Environment. IJMSO 1(4), pp265-281, 2006.

[40] The UMLS Semantic Network (online), available from (http://semanticnetwork.nlm.nih.gov/)

[41] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 2001.