

特別研究報告書

メタデータを用いた
コラボレーションコンテンツの共有

指導教員 石田 亨 教授

京都大学工学部情報学科

岩部 正明

平成17年2月10日

メタデータを用いたコラボレーションコンテンツの共有

岩部 正明

内容梗概

近年のインターネットの普及に伴い、地理的・時間的制約を意識せずに共同作業を行うことが可能になってきている。このため、地理的に離れた人々で構成されたプロジェクトも増えていく傾向にある。しかし、地理的・時間的制約が取り払われた反面、以下のような問題がプロジェクトメンバ間に生じる。

アノテーションデータの互換性の欠如

プロジェクト内では、文書や画像に代表されるさまざまなコンテンツが共有される。各メンバは、作業を進める際に、これらのコンテンツに対して、「このコンテンツは重要だ」「これは何日までに仕上げる必要がある」といった情報を他のメンバに伝える必要が生じる。このような情報は、コンテンツを分類したり注釈を付加したりすることにより伝えられる。プロジェクトに関係する各種のデータを統一的に扱う枠組みとしてプロジェクトオントロジーが構築されているが、プロジェクトオントロジーはコンテンツの分類や注釈には対応していない。このため、各メンバがさまざまな形式で分類や注釈を行い、メンバ間で分類や注釈に関する情報が共有できなくなるという問題が生じる。

コンテンツ分類間の対応付けが困難な問題

コンテンツ分類は各メンバで異なる。また、コンテンツ分類を行うことにより、各メンバはコンテンツに対して意味づけを行っている。したがって、コンテンツ分類間の対応関係を知ることができれば、「ある人はこのコンテンツをこのように捉えているが、別の人はこのように捉えている」というように、互いのコンテンツに対する意味づけを知ることができる。しかし、メンバが分散しているプロジェクトでは議論の場が限られるため、これには長い時間と手間を必要とする。

そこで、上記の問題に対処するために、本研究ではアノテーションデータのスキーマの共通化を行い、なおかつコンテンツ分類間の対応関係を知るための手法を考えた。

各メンバがコンテンツに対して行う意味づけは、分類や注釈といったメタデータとしてコンテンツに付加される。本研究では、このようなメタデータに着目

し，問題解決のために利用する．

各メンバがコンテンツに付加したメタデータを相互に参照することで，プロジェクトやコンテンツの持つ意味を知ることができる．さらに，メンバ間でこれらのメタデータの対応関係を求めることで，互いのコンテンツに対する意味づけの間に，どのような関係があるかを知ることができる．そこで，問題の解決法として，以下の2点を提案する．

アノテーションデータ形式の共通化

メタデータのスキーマが共通であれば，コンテンツの分類や注釈をメンバ間で共有できる．そこで，本研究では，プロジェクトオントロジーを拡張する形で，コンテンツの分類や注釈に用いるメタデータのスキーマをRDFスキーマ形式で定義する．これにより，共通の語彙を用いてコンテンツの分類や注釈を扱うことが可能となる．さらに，このスキーマに基づくRDFデータを共有のRDFデータベースに格納することで，コンテンツの分類や注釈を共有できるようになる．以上のことから，アノテーションデータの互換性の欠如という問題が解決される．

メタデータ間の対応関係の検出

コンテンツ分類に用いられるメタデータは，階層構造をなす．本研究では，この階層構造をコンテンツに関するオントロジーと捉える．そして，異なる2つのオントロジーを入力として，それらの間のマッピングを出力する問題を定式化する．さらに，この問題の解を求めるアルゴリズムを提案する．本アルゴリズムは，あるメタデータを付加したコンテンツの集合と別のメタデータを付加したコンテンツの集合の間の類似度を計算し，2つのメタデータ間に存在する対応関係を求めるものである．これにより，コンテンツ分類間の対応関係を求めることが可能となる．

本研究では，上記の提案手法を実現するためにツールの開発を行った．開発したツールを用いて，各メンバのコンテンツ分類を相互利用できることを確認し，さらにコンテンツ分類間の対応関係を求めることができた．

Sharing Collaboration Contents Using Metadata

Masaaki IWABU

Abstract

Wide use of the Internet in recent years have enabled us to work collaboratively without being conscious of geographical and time constraints. As a consequence, projects consisting of geographically distant people are increasing. However, the following problems occur between project members in spite of the removal of geographical and time constraints.

The lack of compatibility of annotated data

Various contents represented by documents and images are being shared in projects. When a project member performs tasks, he or she needs to convey information related to the content to other members. Examples of such information include "this is important" or "we need to finish this by what day". They are passed on to others by classifying or annotating the contents. "Project ontology" is developed as a framework for systematically managing various data associated with projects, but it does not support classification or annotation of contents. Hence, each member classifies and annotates in various ways, and we can not share these classification and annotation information.

The difficulty in mapping between content classifications

The classification of content differs among members. Each member imparts meaning to the content by classifying the content. Hence, if we know the relation between content classifications, we can know the meaning of the content defined by two members. For instance, if one manages some content in one way and the other manages the same content in some other way, each member can know what meaning is imparted to the content by the other person. However, in projects consisting of geographically distant people, much time and effort is needed because the place for discussion is restricted.

In this research, we define a common schema for annotated data and propose a way of detecting correspondence between content classifications.

The meaning of contents created by each member is tagged on to the contents as metadata about classification and annotation. In this research, we focus on

such metadata and utilize them for solving problems.

We can know the meaning of projects and contents by referring to the metadata tagged on to the contents by each member. And by detecting correspondence of such metadata between members, we can know the relation between meanings of contents created by each member. Hence, we propose the following two points as the solution to the problems.

Definition of a common schema for annotated data

If a common schema of the metadata is defined, we can share classification and annotation of contents between members. In this research, by extending "project ontology", we define a metadata schema for content classifications and annotations based on RDF Schema. As a result, we can manage content classifications and annotations by using common vocabulary. In addition, we can share them by storing RDF data based on this schema in RDF database. The above solves the problem of the lack of compatibility of annotated data.

Detection of correspondence between metadata

Metadata used in content classifications are structured hierarchically. In this research, we consider this hierarchical structure as an ontology of contents. And then, we define a problem that takes distinct two ontologies as input and that shows mapping between them as output. In addition, we propose an algorithm that solves this problem. In this algorithm, we detect the correspondence between two metadata by calculating the degree of similarity between the set of contents tagged on one metadata and the set of contents tagged on another metadata. Then, we can detect the correspondence between content classifications.

In this research, we developed a tool for achieving these proposals. By using this tool, we verified the mutual use of content classifications created by each member and could detect the correspondence between content classifications.

メタデータを用いたコラボレーションコンテンツの共有

目次

第1章	はじめに	1
第2章	コラボレーション基盤としてのツール	2
2.1	プロジェクトオーガナイザ	3
2.2	パーソナルオーガナイザ	4
第3章	メタデータスキーマの共通化	5
3.1	プロジェクトオントロジー	5
3.2	注釈付けのためのスキーマ	6
3.3	コンテンツ分類のためのスキーマ	7
第4章	メタデータ間の対応関係の検出	8
4.1	検出の方針	9
4.2	問題の定式化	9
4.2.1	知識ベース	9
4.2.2	記法の定義	10
4.2.3	類似概念検出問題	11
4.3	問題の解法	13
4.3.1	アルゴリズム	13
4.3.2	計算量	15
4.3.3	実行例	15
第5章	提案手法の実装	17
5.1	ツールの位置づけ	17
5.2	ツールの概要	18
5.3	コンテンツ分類取得機能	19
5.4	コンテンツ分類間のマッピング機能	20
5.5	設定機能	22
第6章	おわりに	23
	謝辞	25
	参考文献	25

	付録	A-1
A.1	ソースコード	A-1
A.1.1	ABRDFProxy.cs	A-1
A.1.2	Attr.cs	A-9
A.1.3	AttrAttribute.cs	A-9
A.1.4	AttributeItem.cs	A-11
A.1.5	ConfigForm.cs	A-12
A.1.6	Constant.cs	A-19
A.1.7	MainForm.cs	A-21
A.1.8	Pr.cs	A-52
A.1.9	PrAgent.cs	A-53
A.1.10	PrContent.cs	A-54
A.1.11	PrProject.cs	A-55
A.1.12	PrTextDocument.cs	A-56
A.1.13	Relation.cs	A-58
A.1.14	TreeForm.cs	A-59

第1章 はじめに

現在，異文化コラボレーション実験 [1][2] をはじめとした，地理的に離れた人々で構成されたプロジェクトが増えていく傾向にある．これは，近年のインターネットの普及に伴い，地理的・時間的制約を意識せずに共同作業を行うことが可能になってきているためである．しかし，プロジェクトを遂行するための地理的・時間的制約が取り払われたからといって，プロジェクトを遂行する上での問題がなくなったわけではない．実際，以下のような問題が，プロジェクトメンバ間に生じる．

アノテーションデータの互換性の欠如

プロジェクト内では，文書や画像に代表されるさまざまなコンテンツが共有される．各メンバは，作業を進める際に，プロジェクト内で共有されているコンテンツに対して「このコンテンツは重要だ」「これは何日までに仕上げる必要がある」というようにさまざまな意味づけを行う．そして，その情報を他のメンバに知らせる必要があるとき，各メンバはコンテンツを分類したり注釈を付加したりすることにより，情報を伝えようとする．この活動を支援するためには，メンバ間で共通の方法で，コンテンツの分類や注釈に関する情報にアクセスできる必要がある．

これを実現するためには，コンテンツ分類や注釈に関する共通の語彙が必要となる．すでに，プロジェクトに関係する各種のデータを統一的に扱う枠組みとしてプロジェクトオントロジー [3] が存在しているが，現状では，プロジェクトオントロジーはコンテンツの分類や注釈には対応していない．このため，各メンバがさまざまな形式で分類や注釈を行い，メンバ間で分類や注釈に関する情報の共有ができなくなるという問題が生じる．

コンテンツ分類間の対応付けが困難な問題

各メンバは，大量のコンテンツを作業を行いやすいように整理することを目的として，コンテンツを分類している．コンテンツ分類には各メンバのコンテンツに対する意味づけが表れるため，各メンバのコンテンツ分類は互いに異なるものとなる．各メンバのコンテンツ分類を利用して，それらの間の対応関係を知ることができれば「ある人はこのコンテンツをこのように捉えているが，別の人はこのように捉えている」というように，コンテンツに対する互いの意味づけを知ることになる．しかし，メンバが分散

しているプロジェクトにおいては議論の場が限られるため、コンテンツ分類間の対応関係を知るためには、長い時間と手間を必要とする。

各メンバは、共有のコンテンツに対してさまざまな意味づけを行っている。このような意味づけは、コンテンツの分類や注釈に関するメタデータとしてコンテンツに付加される。そこで、上記の問題に対処するために、本研究では、コンテンツに付加されるメタデータに着目する。本研究では、メタデータを有効利用するためにアノテーションデータのスキーマの共通化を図り、なおかつコンテンツ分類間の対応関係を知るための手法を考える。

以降、本稿の構成は次のとおりである。まず、第2章で、プロジェクトで利用されるプロジェクト支援ツールについて概説する。これらのツールは、各メンバのプロジェクトやコンテンツに対する意味づけをメタデータの形でコンテンツに付加することによって、プロジェクト支援を行うものである。次に、第3章で、アノテーションデータの互換性を確保する手法としての、プロジェクトオントロジーの拡張によるメタデータスキーマの共通化について述べる。そして、第4章で、自らのコンテンツ分類と他のメンバのコンテンツ分類の間にはどのような関係があるかを見出すための手法として、類似概念検出問題を定義し、その解法について述べる。さらに、第5章で、提案手法を実現するために開発したツールについて述べる。最後に、第6章にまとめを記し、本稿を締めくくる。

第2章 コラボレーション基盤としてのツール

プロジェクトの支援を目的として、さまざまなツールが開発されている。本章では、プロジェクト支援ツールの中から、プロジェクトやコンテンツに対する意味づけをメタデータの形でコンテンツに付加することのできるツールの例として、プロジェクトオーガナイザとパーソナルオーガナイザを示す。

本研究では、このようなプロジェクト支援ツールによって付加されるメタデータのうち、コンテンツ分類に関するメタデータを利用して、コンテンツ分類間の対応関係を検出することを目的とする。

2.1 プロジェクトオーガナイザ

プロジェクトオーガナイザ¹⁾は、京都大学で開発されたプロジェクト支援ツールである。このツールは、プロジェクト管理者の立場から、プロジェクトやコンテンツに対して意味づけを行うためのツールである。コンテンツの内容を変更することなく、管理者がコンテンツの管理を効率的に行えるようにするための情報を付加することができる。プロジェクトオーガナイザのスクリーンショットを、図 2.1 に示す。このツールは、以下のような機能を備えている。

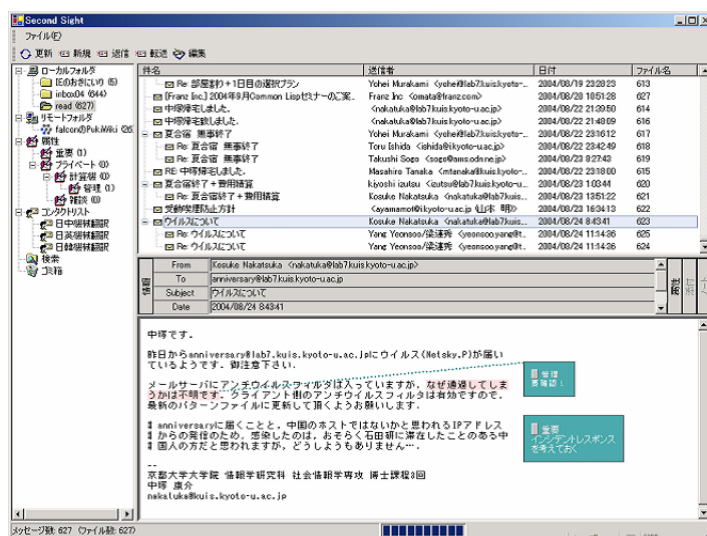


図 2.1: 京都大学で開発されたプロジェクトオーガナイザ

(<http://www.lab7.kuis.kyoto-u.ac.jp/~nakatuka/ss/>)

1. コンテンツの集約: コンテンツの集約のために、ローカルフォルダやネットワークフォルダなどのリポジトリに依存せずに、コンテンツを取得できる。
2. コンテンツの表示: 電子メールやオフィスドキュメントなどのファイルのフォーマットに依存せずに、コンテンツを表示できる。
3. コンテンツの分類: フォルダ分けと同様の操作で、コンテンツを分類できる。ただし、フォルダとは異なり、1つのコンテンツを複数の項目に重複して分類できる。
4. コンテンツへの注釈付け: コンテンツおよびその一部に、注釈を付けることができる。

¹⁾ <http://www.lab7.kuis.kyoto-u.ac.jp/~nakatuka/ss/>

コンテンツの分類や注釈においては，RDF データベースとの連携により，RDF を利用したメタデータの付加が可能である．さらに，RDF による分類や注釈はメンバ間で共有することができる．

2.2 パーソナルオーガナイザ

パーソナルオーガナイザ¹⁾は，和歌山大学で開発されたプロジェクト支援ツールである．このツールは，ネットワーク上のデータベースなどに保存されたコンテンツを画面に2次元的に配置することで，コンテンツの整理を行うツールである．ユーザは，画面上に置かれたコンテンツに対して自由に注釈をつけることもできる．コンテンツの整理や注釈はRDF メタデータとして付加され，このメタデータはRDF データベースに蓄えられる．パーソナルオーガナイザのスクリーンショットを，図 2.2 に示す．

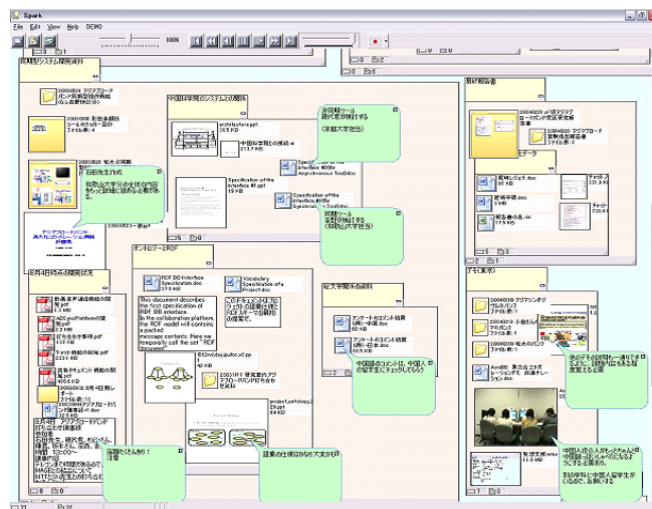


図 2.2: 和歌山大学吉野研究室で開発されたパーソナルオーガナイザ
(<http://yoshino.sys.wakayama-u.ac.jp/spark/>)

このツールは，各メンバが自らの作業に必要なコンテンツを取得し，自身の視点で好きなように整理するのに適している．このツールにより，プロジェクトメンバの発想を支援することができる．

¹⁾ <http://yoshino.sys.wakayama-u.ac.jp/spark/>

第3章 メタデータスキーマの共通化

コンテンツに関する情報を他のメンバにも伝えるために、コンテンツの分類が行われたり、コンテンツに対して注釈付けがなされたりする。しかし、分類や注釈を行うための統一的な枠組みが確立されていないならば、各メンバがさまざまな形式で分類や注釈を行い、メンバ間で分類や注釈に関する情報の共有ができなくなるという問題が生じる。すでに、プロジェクトに関係する各種のデータを統一的に扱う枠組みとして、プロジェクトオントロジー [3] が構築されている。しかし、プロジェクトオントロジーは、コンテンツの分類や注釈には対応していない。そこで、プロジェクトオントロジーを拡張して、分類や注釈を扱うようにすることを考える。

3.1 プロジェクトオントロジー

コンテンツに付加するメタデータのスキーマが共通化されていないならば、メンバ間でメタデータを相互利用することはできない。共通化されたメタデータのスキーマを構築する上では、各メンバが利用しているプロジェクト支援ツールに依存しない形でスキーマの定義ができる必要がある。RDF は、特定のアプリケーションや知識領域を前提とせずに、相互運用可能な形でリソースを記述するための標準的な枠組みを提供するため、上記の目的にかなっている。そこで、RDF スキーマ [4] に基づいて異文化コラボレーション実験で開発された、プロジェクトオントロジー¹⁾を利用することを考える。しかし、このプロジェクトオントロジーは、これまでの異文化コラボレーション実験で実際に使われたものではないために洗練されたものとは言えず、運用上の問題が存在した。さらに、プロパティの重複や不足といった問題も生じていた。このため、実際の運用に耐えうるように、プロジェクトオントロジーを再構築することで、これらの問題に対処した。再構築後のプロジェクトオントロジーのクラス構造を、表 3.1 に示す。なお、表 3.1 中の pr: は名前空間 <http://ice.kuis.kyoto-u.ac.jp/ont/project/0.1> を表すものである。このプロジェクトオントロジーを基盤として、次節以降でコンテンツの分類や注釈のためのスキーマを定義する。

¹⁾ http://ice.kuis.kyoto-u.ac.jp/project_rdf/rdf/ に、完全なクラス定義およびプロパティ定義が記述されている。

表 3.1: 再構築後のプロジェクトオントロジーのクラス構造

クラス	サブクラス
pr:Project	
pr:Task	
pr:Plan	
pr:Resource	pr:Budget pr:Tool
pr:Role	
pr:Agent	pr:Group pr:Person pr:SoftwareAgent
pr:Interaction	pr:Message pr:LogItem
pr:Content	pr:TextDocument
pr:Context	pr:Log

3.2 注釈付けのためのスキーマ

プロジェクトオントロジーを拡張して、コンテンツへの注釈付けに関するメタデータのスキーマを共通化する。本研究では、コンテンツへの注釈付けのための Annotation クラスを表 3.1 中の TextDocument クラスのサブクラスとして定義した。これは、アノテーションはコンテンツに対するメタコンテンツである [5] ため、コンテンツの一種として捉えることができるためである。

Annotation クラスでは多くのプロパティが利用可能であるが、このうち主要なプロパティは以下のとおりである。

- annotation プロパティ：付加されたアノテーションを表すプロパティ。アノテーションに対するアノテーションに相当する。
- creator プロパティ：アノテーションを付加した人を表すプロパティ。
- reference プロパティ：アノテーションを付加したコンテンツを表すプロパティ。annotation プロパティとは双対の関係にある。
- text プロパティ：アノテーション本体を表すプロパティ。

本研究で定義した Annotation クラスを用いることで、コンテンツへの注釈付けの一例として、図 3.1 のような表現が可能である。図 3.1 では、「nakatuka」というエージェントが「iwabu」というエージェントの作成した「spec.html」というコンテンツに対して、「大至急」と注釈付けした様子が示されている。

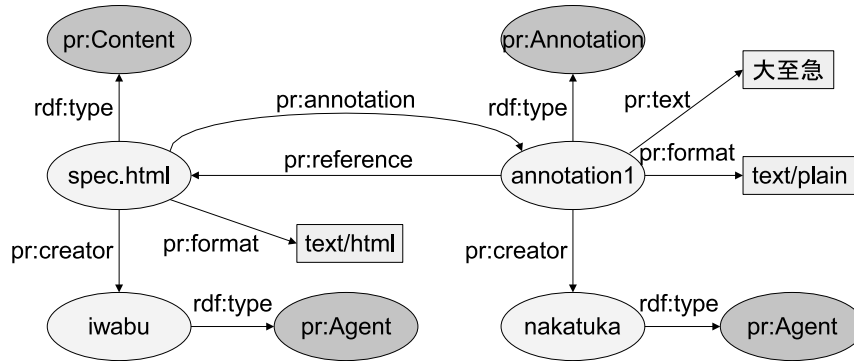


図 3.1: 注釈付けの表現例

3.3 コンテンツ分類のためのスキーマ

コンテンツへの注釈付けのためのスキーマに加えて，さらにコンテンツ分類に関するメタデータのスキーマを共通化する．本研究では，表 3.2 に表される Attribute クラスを，コンテンツ分類のためのスキーマとして定義した．なお，表 3.2 中の attr: は名前空間 <http://ice.kuis.kyoto-u.ac.jp/ont/attr/0.1> を表すものである．

表 3.2: コンテンツ分類のための Attribute クラス

Class	Property	Range
attr:Attribute	attr:subAttribute	attr:Attribute
	pr:content	pr:Content
	pr:creator	pr:Agent
	pr:name	rdfs:Literal

Attribute クラスは，以下のプロパティを持つ．

- subAttribute プロパティ：コンテンツ分類に用いるメタデータの階層構造を決定するためのプロパティ．
- content プロパティ：分類されるコンテンツを表すプロパティ．
- creator プロパティ：コンテンツ分類を行った人を表すプロパティ．
- name プロパティ：コンテンツ分類に用いるメタデータの名前を表すプロパティ．

本研究で定義した Attribute クラスを用いることで，コンテンツ分類の一例として，図 3.2 のような表現が可能である．図 3.2 では，「iwabu」というエージェ

ントが、「29日の発表資料」という名前の分類を用意し、関連するコンテンツ (iwabu.ppt と iwabu.pdf) をひとまとめにすることでコンテンツを分類した様子が示されている。

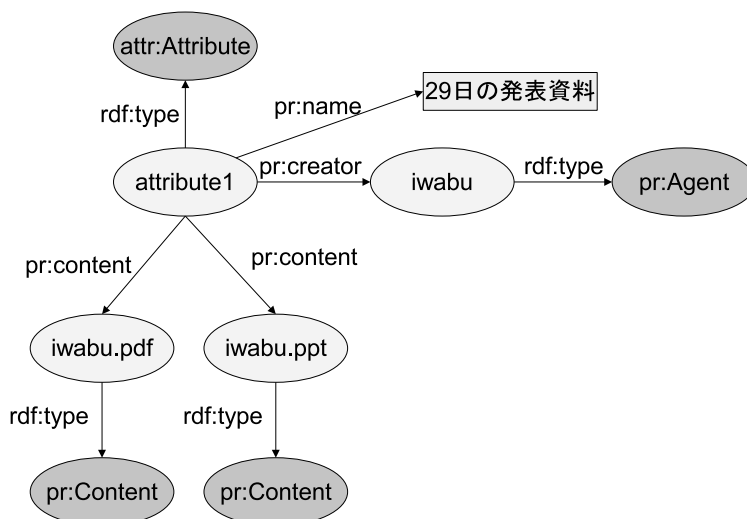


図 3.2: コンテンツ分類の表現例

第4章 メタデータ間の対応関係の検出

各メンバは、大量のコンテンツを効率よく利用するために、コンテンツを分類・整理している。コンテンツ分類には各メンバのコンテンツに対する意味づけが表れるため、各メンバのコンテンツ分類は互いに異なるものとなる。プロジェクト内で共有されるコンテンツの作成や利用の過程では、コンテンツに対して各メンバが付加した意味を知る必要がある。このために、コンテンツ分類間の対応関係の検出を行う。コンテンツ分類間の対応関係が分かれば、各メンバがコンテンツに対して付加した意味の間の対応関係を知ることができるためである。

コンテンツ分類間の対応関係を検出するためには、コンテンツ分類に利用されるメタデータが相互利用できる必要がある。このことを実現するために、第3章ではメタデータのスキーマを共通化した。さらに、本章では、メンバ間で相互に利用できるようになったメタデータを利用して、コンテンツ分類に利用されるメタデータ間の対応関係を検出する手法について述べる。

4.1 検出の方針

本研究では，コンテンツ分類のために付加したメタデータの階層構造を階層的概念で作られたオントロジーとみなす．そして，異なる2つのオントロジー間の対応関係を求めることを考える．ここでの対応関係とは，一方のオントロジー中の特定の概念が，他方のオントロジー中のどの概念に相当するかという関係のことである．

すでに，このような対応関係の検出を支援するシステムとして，Chimaera[6]やPROMPT[7]が提案されている．これらのシステムは，概念名の類似性などを利用して概念間の対応関係を検出し，人間に提示する．このため，実際は対応関係にあるにもかかわらず，概念名がまったく異なるようなものを発見することはできない．また，検出結果は利用する類語辞書に依存する．

また，インスタンスに着目して対応関係を検出する例として，TF/IDF法によりテキストをベクトル空間にマッピングすることで，概念間の対応関係を求める方法[8]がある．しかし，テキストでないインスタンスに対して，この方法を適用することはできない．

そこで，本研究では，対応関係を求める際に概念の持つ意味を考慮せず，インスタンスが概念によってどのように分類されるかという点にのみ着目し，概念間の対応関係を見出す手法を提案する．具体的には，「ある概念に分類されているインスタンスの多くが別の概念のインスタンスでもあるならば，その2つの概念は対応関係にある」と考える．インスタンスに関する形式的な情報のみを用いることで，上記の問題の影響がなくなると考えられるためである．

次節以降で，メタデータ間の対応関係を検出するための問題を定式化し，解法を述べる．

4.2 問題の定式化

メタデータ間の対応関係を検出する問題を解くために，本研究では類似概念検出問題として問題の定式化を行う．

4.2.1 知識ベース

本問題では，それぞれのメンバによって作られた各コンテンツ分類を，知識ベース $C = (I, S, E_{IS}, E_{SS})$ と捉える．知識ベースの概念図を，図4.1に示す．

I, S, E_{IS}, E_{SS} とは，それぞれ以下に定義される．

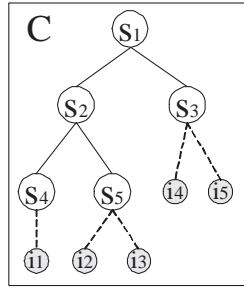


図 4.1: 知識ベース C の概念図

- I : インスタンス i_k の集合．コンテンツに対応する．図 4.1 では，インスタンスを小さな円で表している．
- S : 概念 s_k の集合．コンテンツ分類のために用いられるメタデータに相当する．図 4.1 では，概念を大きな円で表している．
- E_{IS} : $E_{IS} = \{(i, s) | i \in I, s \in S\}$ であり，概念とインスタンスを結ぶ枝の集合を表す．コンテンツがどのメタデータによって分類されたかということに対応する．図 4.1 では，点線で表されている．
- E_{SS} : $E_{SS} = \{(s_1, s_2) | s_1, s_2 \in S, s_1 \text{ は } s_2 \text{ の上位概念}\}$ であり，2 つの概念を結ぶ枝の集合を表す．コンテンツ分類のために用いられるメタデータの階層構造に対応する．図 4.1 では，実線で表されている．

なお，本問題で用いる知識ベースに含まれる概念の階層構造は，木構造であると仮定する．つまり， $G = (S, E_{SS})$ により表されるグラフは，連結で閉路を持たないグラフであると仮定する．なぜなら，木構造はさまざまなコンテンツ分類に用いられている基本的な構造であるためである．2.1 で紹介したプロジェクトオーガナイザや，2.2 で紹介したパーソナルオーガナイザで作られるコンテンツ分類の階層構造も同様に，木構造となる．本研究では，これらのプロジェクト支援ツールで構築されるコンテンツ分類に対してマッピングをとることを目的としているため，木構造を前提として問題を考えることにする．ただし，複数の上位概念を持っている概念は，図 4.2 のように展開することによって，木構造として扱う．

4.2.2 記法の定義

以下で，問題およびその解法の表現のために必要となる記法の定義を行う．

- $root(S)$: 木構造の根からなる集合．たとえば，図 4.1 では， $root(S) = \{s_1\}$ ．
- $dsucc(s)$: 概念 s の直下の概念．たとえば，図 4.1 では， $dsucc(s_1) = \{s_2, s_3\}$ ．

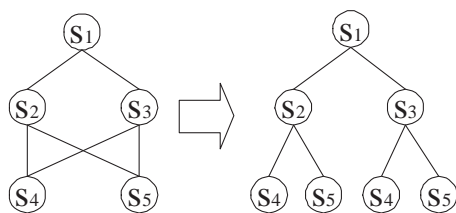


図 4.2: 木構造への変換

- $succ(s)$: 概念 s の下位概念全体 . たとえば , 図 4.1 では , $succ(s_1) = \{s_2, s_3, s_4, s_5\}$.
- I_s : 概念 s に属するインスタンスの集合 . I_s は , 概念 $s' \in dsucc(s)$ に属するインスタンスの集合 $I_{s'}$ を包含する . たとえば , 図 4.1 では , $I_{s_2} = \{i_1, i_2, i_3\}$ であり , $I_{s_4} \subset I_{s_2}$, $I_{s_5} \subset I_{s_2}$ である .

4.2.3 類似概念検出問題

$C = (I_1, S_1, E_{I_1 S_1}, E_{S_1 S_1})$, $D = (I_2, S_2, E_{I_2 S_2}, E_{S_2 S_2})$ という 2 つの知識ベースを用いることにより , 2 つのオントロジー間のマッピングを求める問題を考えることができる . マッピングとは , 類似概念の組からなる集合である . まず , 類似概念の定義を以下に示す .

定義 1 (類似概念). 以下の 2 条件を満たすとき , $s_1 \in S_1$ と $s_2 \in S_2$ は類似概念であり , $s_1 \simeq s_2$ と記述する .

1. 集合 I_{s_1} と集合 I_{s_2} が類似している .
2. I_{s_1} は I_{s_2} と最も類似した集合の 1 つであり , I_{s_2} は I_{s_1} と最も類似した集合の 1 つである .

これを数学的に記述すると , 以下のようになる .

1. $sim(s_1, s_2) \geq \alpha$ (α は閾値を表すパラメータで $0 \leq \alpha \leq 1$ を満たす) .
2. $\forall s'_1 \in S_1, \forall s'_2 \in S_2$ について , $sim(s_1, s_2) \geq sim(s'_1, s_2)$ かつ $sim(s_1, s_2) \geq sim(s_1, s'_2)$

ここで , $sim(s_1, s_2)$ は類似度の尺度を表すものである . 本研究では , 4.1 で述べたとおり , インスタンス分類の類似性に基づいて概念間の対応関係を求める . そこで , 類似度の尺度として , 集合の類似度を用いることにする . 集合の類似度としては , 以下に示すものが提案されている [9] .

1. Jaccard 係数

$$sim(s_1, s_2) = \frac{|I_{s_1} \cap I_{s_2}|}{|I_{s_1} \cup I_{s_2}|}$$

2. Dice 係数

$$sim(s_1, s_2) = \frac{2|I_{s_1} \cap I_{s_2}|}{|I_{s_1}| + |I_{s_2}|}$$

3. Cosine 係数

$$sim(s_1, s_2) = \frac{|I_{s_1} \cap I_{s_2}|}{\sqrt{|I_{s_1}| |I_{s_2}|}}$$

4. Simpson 係数

$$sim(s_1, s_2) = \frac{|I_{s_1} \cap I_{s_2}|}{\min(|I_{s_1}|, |I_{s_2}|)}$$

上記の尺度には、それぞれ特徴がある。Jaccard 係数、Dice 係数、Cosine 係数は、 $|I_{s_1}|$ と $|I_{s_2}|$ の値に大きな開きがあるとき、類似度の値が低くなりやすいという特徴がある。たとえば、 $|I_{s_1}| = 100$ 、 $|I_{s_2}| = 10$ 、 $|I_{s_1} \cap I_{s_2}| = 10$ とする。 s_2 に属するインスタンスは全て s_1 にも属しているにもかかわらず、このとき Jaccard 係数は 0.1、Dice 係数は 0.18、Cosine 係数は 0.32 という小さな値をとる。これに対して、Simpson 係数は分母で min をとっているために、この欠点は解消される。その代わりに、 $|I_{s_1}|$ と $|I_{s_2}|$ のうち一方の値が小さいときには、類似度の値が高くなりやすい。たとえば、 $|I_{s_1}| = 100$ 、 $|I_{s_2}| = 1$ 、 $|I_{s_1} \cap I_{s_2}| = 1$ とする。このときの Simpson 係数の値は、1 である。これは、Simpson 係数が、共通なインスタンスの数が極端に少ない場合でも、 s_1 と s_2 が類似と判断してしまう可能性があることを示している。

以降で提案するアルゴリズムにおいては、これらの係数のうち、どれを用いるかということは特定しない。したがって、ユーザがそれぞれの目的に応じた尺度を用いることができる。

さらに、定義 1 に表される類似概念の定義を用いて、知識ベース間のマッピングを以下のように定義する。

定義 2 (マッピング). 2 つの知識ベース $C = (I_1, S_1, E_{I_1 S_1}, E_{S_1 S_1})$, $D = (I_2, S_2, E_{I_2 S_2}, E_{S_2 S_2})$ の間のマッピング M とは、以下に定義される集合のことである。

$$M = \{(s_1, s_2) | s_1 \simeq s_2, s_1 \in S_1, s_2 \in S_2\}$$

図 4.3 に正しいマッピングの概念図を、図 4.4 にマッピングでない例を示す。図 4.3 は、マッピングの 2 条件を満たしているが、図 4.4 は、以下の理由によりマッピングといえない。

- 集合 I_{t_2} に最も類似した集合は、 I_{s_5} ではない

- 類似度の尺度として上記のどれを用いても, $sim(s_4, t_4) = 0$ であり, これは明らかに類似度の閾値に満たない

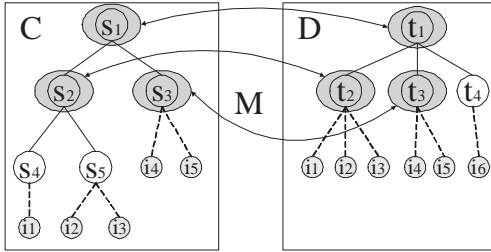


図 4.3: マッピング M の概念図

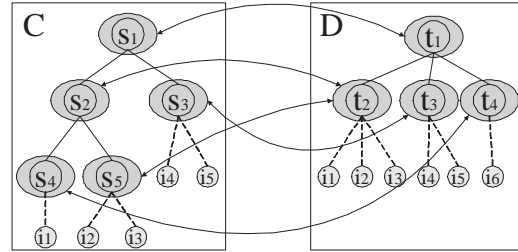


図 4.4: マッピングでない例

2つのオントロジー間のマッピングを求めることは, 定義 1 に定められる類似概念の組からなる集合を求めることと同義である.

4.3 問題の解法

4.2 により定式化される問題は, 以下を入出力とする.

入力 2つの知識ベース $C = (I_1, S_1, E_{I_1 S_1}, E_{S_1 S_1}), D = (I_2, S_2, E_{I_2 S_2}, E_{S_2 S_2})$

出力 2つのオントロジー間のマッピング M

本節では, この問題を解く手順について述べる.

4.3.1 アルゴリズム

以下に, 類似概念検出問題を解くアルゴリズムを示す. ただし, $\overline{sim(s_1, s_2)}$ は $sim(s_1, s_2)$ の上限を与えるものであり, 以下の2条件を満たす.

- $|I_{s_2}|$ について単調増加な関数
- $sim(s_1, s_2) \leq \overline{sim(s_1, s_2)}$

アルゴリズム

1. スタック $stack$, 集合 $elem$, $cand1$, $cand2$, 解の集合 M を空にする.
2. $\forall s_1 \in S_1$ について, 以下を行う.
 - (a) $stack$ に $root(S_2)$ を push.
 - (b) $stack$ が空になるまで, 以下を行う.
 - i. $stack$ の要素を pop し, これを s_2 とする.
 - ii. $\overline{sim(s_1, s_2)} \geq \alpha$ のとき,
 - A. $stack$ に $\forall s'_2 \in dsucc(s_2)$ を push.
 - B. $sim(s_1, s_2) \geq \alpha$ であれば, $elem \leftarrow elem \cup \{(s_1, s_2)\}$,

3. $\forall s_1 \in S_1$ について，以下を行う．
 - (a) 集合 $\{(s_1, b) | (s_1, b) \in elem\}$ から， $sim(s_1, b)$ が最大のものを選ぶ
 - (b) 選ばれた (s_1, b) を， $cand1$ に追加
4. $\forall s_2 \in S_2$ について，以下を行う．
 - (a) 集合 $\{(a, s_2) | (a, s_2) \in elem\}$ から， $sim(a, s_2)$ が最大のものを選ぶ
 - (b) 選ばれた (a, s_2) を， $cand2$ に追加
5. $M \leftarrow cand1 \cap cand2$ とし， M を出力する．

上記手順は，定義 1 に対応したものとなる．定義 1 とアルゴリズムの間には，以下のような対応関係がある．

- 手順 2 は，定義 1 の条件 1 に相当
- 手順 3 は，定義 1 の条件 2 中の $sim(s_1, s_2) \geq sim(s_1, s'_2)$ の部分に相当
- 手順 4 は，定義 1 の条件 2 中の $sim(s_1, s_2) \geq sim(s'_1, s_2)$ の部分に相当

これら全てを満たす概念の組を求めることは，定義 1 に示される類似概念の組を求めることに等しい．

また，上記の手順 2(b)iiA では，枝刈りを行って不要な類似度計算を省略している．なお，この手順で用いた類似度の上限 $\overline{sim(s_1, s_2)}$ の具体的な式は，それぞれ以下のとおりである．

1. Jaccard 係数の上限

$$\overline{sim(s_1, s_2)} = \frac{|I_{s_2}|}{|I_{s_1}|}$$

2. Dice 係数の上限

$$\overline{sim(s_1, s_2)} = \frac{2|I_{s_2}|}{|I_{s_1}| + |I_{s_2}|}$$

3. Cosine 係数の上限

$$\overline{sim(s_1, s_2)} = \sqrt{\frac{|I_{s_2}|}{|I_{s_1}|}}$$

概念階層の下位になるにつれて，属するインスタンスの個数は少なくなるため，もし $\overline{sim(s_1, s_2)} \leq \alpha$ ならば， $\forall s'_2 \in succ(s_2)$ について $\overline{sim(s_1, s'_2)} \leq \alpha$ である．これは，ある概念 $s_2 \in S_2$ について，概念 $s_1 \in S_1$ との類似度の上限が閾値に満たないことが分かれば，それより下位の概念 $s'_2 \in succ(s_2)$ について，概念 s_1 との類似度が閾値を上回ることがないことを意味している．この点に着目して，本アルゴリズムでは不要な類似度計算を省略する．

4.3.2 計算量

4.3.1のアルゴリズムの計算量について、考察する。ここで、 $|S_1| = O(n)$, $|S_2| = O(n)$, $|I_1| = O(k)$, $|I_2| = O(k)$ とする。

まず、時間計算量について述べる。4.3.1のアルゴリズムで最も時間を要する箇所は、手順2における類似度計算の部分である。手順2においては、 S_1 中の概念1個につき最大 $O(n)$ 回の類似度計算を必要とする。このため、類似度計算の合計回数は多くとも $O(n^2)$ 回である。類似度計算を行うためには、和集合や積集合を求める操作が必要となる。これはインスタンスの個数に比例する時間を要するので、1回の類似度計算に要する時間は $O(k)$ である。以上より、類似度計算には $O(kn^2)$ の時間を要する。他の手順は、条件を満たす概念の組を選択する操作であるので、 $O(kn^2)$ より短い時間で可能である。よって、手順2以外の操作は、時間計算量のオーダーに影響を及ぼさない。以上より、4.3.1のアルゴリズムの時間計算量は $O(kn^2)$ である。

次に、空間計算量について述べる。手順2では、前順による木の走査を行っている。この操作に必要な空間計算量は、 $O(n)$ である。これに加えて、類似概念の候補を格納しておく空間が必要である。これに必要な空間計算量は、多くとも $O(n^2)$ である。以上より、4.3.1のアルゴリズムの空間計算量は $O(n^2)$ である。

4.3.3 実行例

4.3.1のアルゴリズムの実行例を示す。入力として、図4.5に示される概念図に相当する2つの知識ベース C , D を与える。また、類似度の尺度にJaccard係数を、類似度の閾値に0.7を用いることとする。

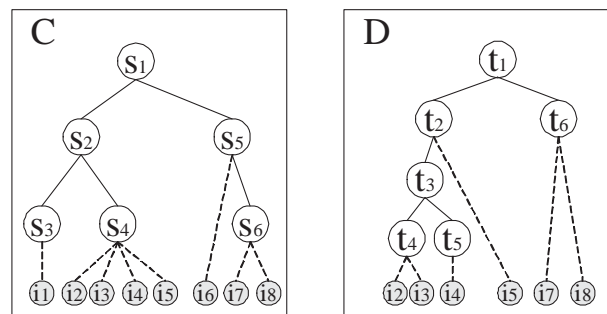


図 4.5: マッピングを求める知識ベース C , D の概念図

上記の入力に対する、アルゴリズムの実行過程を示す。

まず，手順 2 により， $sim(a, b) \geq \alpha$ となる組 (a, b) を全て求める．類似度が閾値を超える概念の組は，以下のとおりである．

$$elem = \{(s_1, t_1), (s_2, t_2), (s_4, t_2), (s_4, t_3), (s_6, t_6)\}$$

次に，手順 3 で， C 中の概念それぞれについて，最も類似する概念を D 中から探す．ここで， C 中の概念 a と D 中の概念 b の類似度は， $sim(a, b) \geq \alpha$ を満たさなければならないので，手順 3 で求める概念の組は，集合 $elem$ の要素である．集合 $elem$ の要素のうち，条件を満たすものからなる集合 $cand1$ は，以下のとおりである．

$$cand1 = \{(s_1, t_1), (s_2, t_2), (s_4, t_2), (s_6, t_6)\}$$

そして，手順 4 で， D 中の概念それぞれについて，最も類似する概念を C 中から探す．ここで， C 中の概念 a と D 中の概念 b の類似度は， $sim(a, b) \geq \alpha$ を満たさなければならないので，手順 4 で求める概念の組は，集合 $elem$ の要素である．集合 $elem$ の要素のうち，条件を満たすものからなる集合 $cand2$ は，以下のとおりである．

$$cand2 = \{(s_1, t_1), (s_4, t_2), (s_4, t_3), (s_6, t_6)\}$$

集合 $cand1 \cup cand2$ は，類似概念の組の候補からなる集合に相当する．ここまでで求められた類似概念の組の候補を概念図で表すと，図 4.6 のようになる．

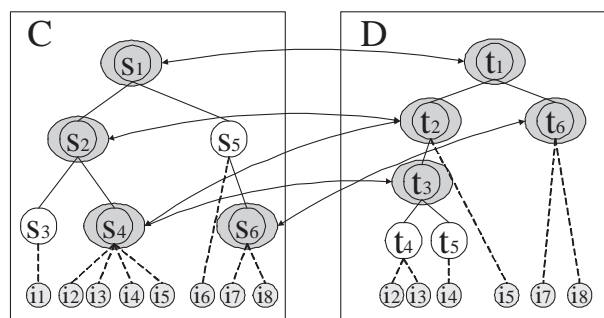


図 4.6: 類似概念の組となる候補の概念図

最後に，手順 5 を行う．

$$M = cand1 \cap cand2 = \{(s_1, t_1), (s_4, t_2), (s_6, t_6)\}$$

であり，これがアルゴリズムの出力となる．

以上の操作により，図 4.7 に示されるマッピングを求めることができる．

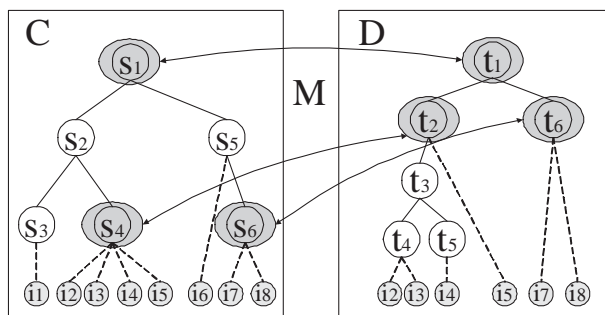


図 4.7: 得られたマッピングの概念図

第5章 提案手法の実装

本章では、第3章、第4章で提案した手法を実現するために実装したツールについて述べる。

5.1 ツールの位置づけ

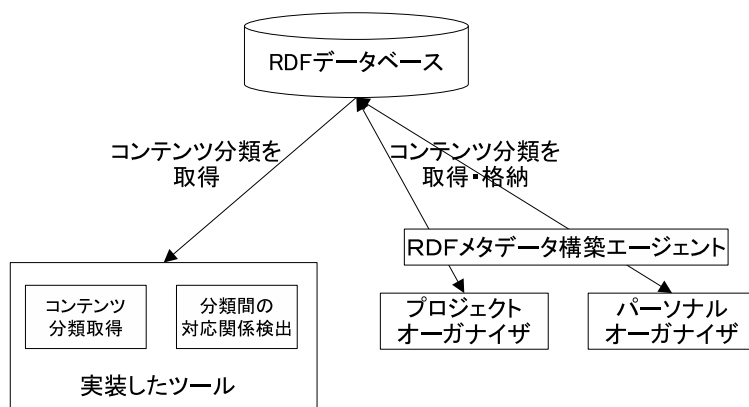


図 5.1: ツールの位置づけ

本ツールは、Microsoft Visual Studio .NET 2003 開発環境を用いて C# で実装したものである。本ツールの位置づけは、図 5.1 に表される。RDF データベースに蓄えられているプロジェクトオーガナイザ (2.1 参照) やパーソナルオーガナイザ (2.2 参照) が構築したコンテンツ分類を取得し、コンテンツ分類間の対応関係の検出を実現する。なお、RDF データベースには、Jena¹⁾を用いて Java で実装されたサーバを用いる。RDF データベースとの通信には、XML-RPC[10]

¹⁾ <http://jena.sourceforge.net/>

を用いる。XML-RPC は、インターネットを介してリモートプロシージャコールを実行するためのプロトコルである。Java、Microsoft .NET など様々な環境に対応したライブラリが用意されているため、今回の実装に採用した。

5.2 ツールの概要

本ツールの構成図を、図 5.2 に示す。本ツールは、メイン処理部・通信管理部・アルゴリズム部・描画処理部に分けられる。これに加えて、コンテンツ分類間の対応関係を視覚的に捉えたり、アルゴリズムに必要な設定を行ったりするためのインタフェースを持つ。

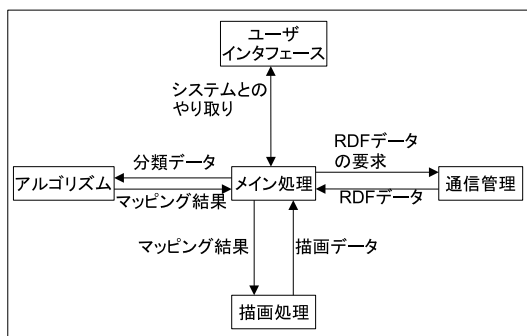


図 5.2: ツールの構成図

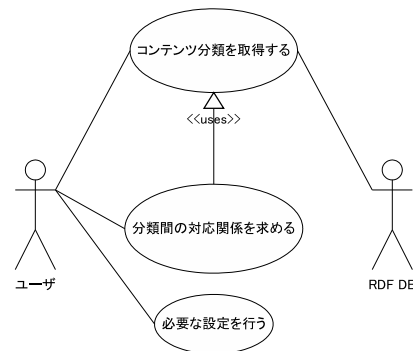


図 5.3: ツールのユースケース図

本ツールの各部分では、以下の処理が行われる。

1. メイン処理部：全体的な処理を行う。
2. 通信管理部：RDF データベースとの通信を管理する。
3. アルゴリズム部：4.3 で述べたアルゴリズムに基づいて、コンテンツ分類間の対応関係を検出する。
4. 描画処理部：アルゴリズムの実行結果を利用して、メタデータの階層構造の描画を行う。対応関係にあるメタデータは、重なって表示されるように描画する。
5. ユーザインタフェース：コンテンツ分類を閲覧する画面、コンテンツ分類間の対応関係を可視化するための画面、アルゴリズムの実行に必要な設定を行う画面からなる。

本ツールの機能をまとめたユースケース図を、図 5.3 に示す。ユーザに提供されている機能として、

- メンバのコンテンツ分類を取得する機能
- コンテンツ分類間の対応関係を検出する機能
- アルゴリズムの実行に必要な設定を行う機能

の3つがある．それぞれの機能について，次節以降で述べる．

5.3 コンテンツ分類取得機能

プロジェクトメンバのコンテンツ分類を取得するために，ユーザはプロジェクトとそれに参加するメンバを指定する必要がある．この操作を受けて，本ツールはRDFデータベースから必要なデータを取得し，コンテンツ分類の表示を行う．

図5.4は，ユーザがログインしてから所望のコンテンツ分類を取得するまでのシーケンス図である．この過程では，以下の処理が行われる．

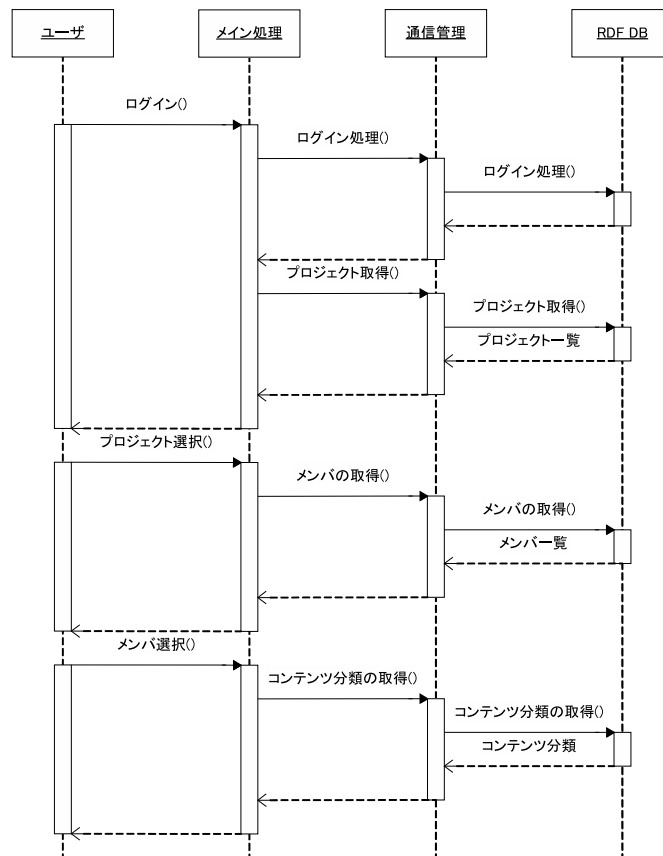


図 5.4: コンテンツ分類の取得までのシーケンス図

1. RDF データベースへログインしてセッションが確立されると、本ツールは RDF データベースに格納されている全てのプロジェクトを取得する。そして、取得されたプロジェクト名を画面に表示する。
2. ユーザは、画面に表示されたプロジェクトの中から、適当なプロジェクトを選択する。このとき、本ツールはそのプロジェクトのメンバに関するデータを取得する。そして、取得されたメンバを画面に表示する。この様子を図 5.5 に示す。
3. ユーザは、画面に表示されたメンバの中から適当な人を選択する。このとき、本ツールは、選択されたメンバによって構築されたコンテンツ分類に関するデータを取得する。そして、そのデータからコンテンツ分類を再構築し、画面に表示する。この様子を図 5.6 に示す。コンテンツ分類の表示は、画面の左側にメタデータの階層構造を、右側にそれらのメタデータによって分類されたコンテンツの一覧を表示することによってなされるものとした。なお、本ツールでは異なる 2 人のメンバのコンテンツ分類を比較対照できるように、画面構成の設計を行った。

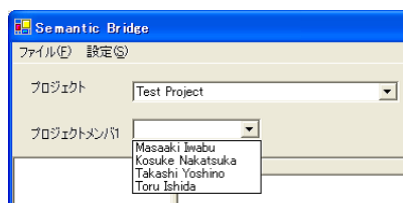


図 5.5: メンバの取得

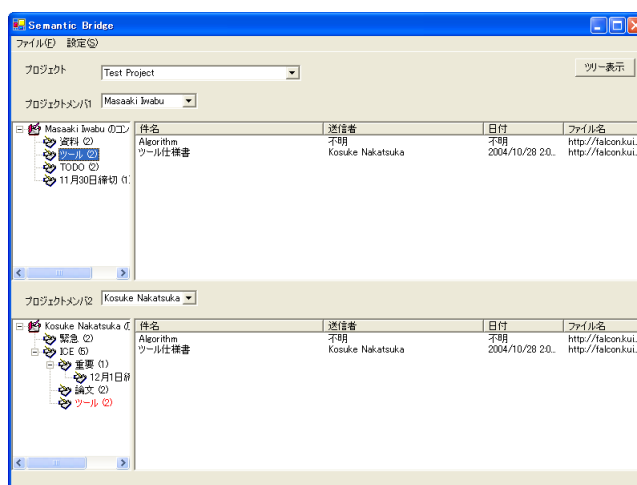


図 5.6: コンテンツ分類取得機能

5.4 コンテンツ分類間のマッピング機能

ユーザは、マウスクリックすることで、画面左側に表示されるメタデータ (図 5.6 参照) を選択することができる。画面上に、異なる 2 人のメンバのコンテ

ツ分類が表示されているとき，選択されたメタデータに対応するメタデータをもう1人のコンテンツ分類から検出する．検出手法として，4.3で述べたアルゴリズムを用いる．そして，対応関係にあるメタデータを赤字で強調表示する．この様子を，図5.7に示す．



図 5.7: コンテンツ分類間のマッピング機能

また，2つのコンテンツ分類間のマッピングを視覚的に捉えるための可視化機能を実装した．コンテンツ分類間のマッピングは，図5.8のように2つの木構造を重ね合わせることで表現される．このとき，対応関係にあるメタデータは重なって表示される．

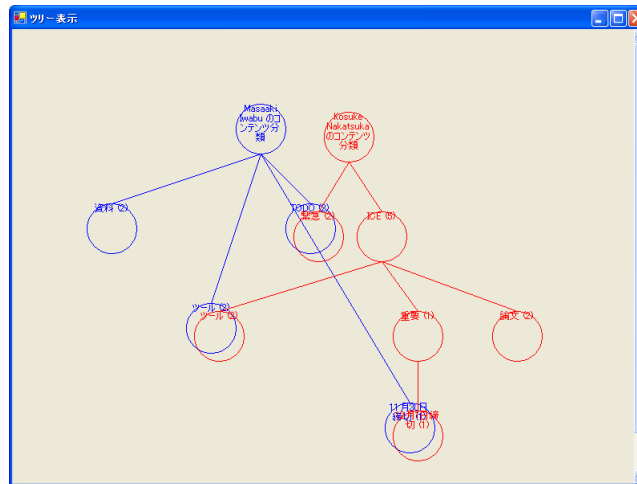


図 5.8: コンテンツ分類間のマッピングの可視化

異なる2人のメンバのコンテンツ分類を取得して，マッピング可視化画面を表示させるまでのシーケンス図を，図5.9に示す．この過程では，以下の処理が行われる．

1. 異なる2人のメンバのコンテンツ分類が取得された後，4.3で述べたアルゴ

リズムにより，コンテンツ分類間のマッピングを検出するモジュールが呼び出される．そして，アルゴリズムの実行により得られたマッピングを記憶する．

2. ユーザがマッピング可視化画面を表示させるための操作を行うと，描画処理モジュールが呼び出される．ここでは，得られたマッピングにしたがって，対応関係にあるメタデータは重なって表示されるように，メタデータの描画位置を決定する仕組みを実装している．

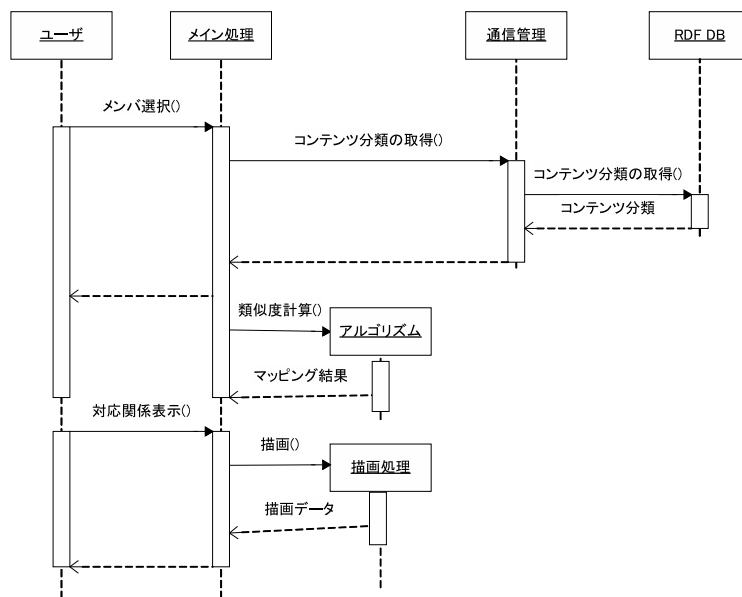


図 5.9: コンテンツ分類間のマッピング検出のシーケンス図

なお，描画機能は他に 2 つの特徴を持つ．メタデータの階層構造が大きく全体が一画面に収まりきらない場合などに，マウスの中ボタンを操作することによって描画サイズを自由に拡大縮小できるようにしている．また，画面内でマウスをドラッグアンドドロップすることにより，描画されたツリーを移動させることができる．

5.5 設定機能

プロジェクトに関するさまざまなデータを格納している RDF データベースにアクセスするためには，データベースへのログインを行うための設定をする必要がある．また，4.3 に記したアルゴリズムでは，類似度の閾値 α の値や類似度

の尺度を自由に設定できるようになっているため、これらに関する設定を行う必要がある。このような設定を行うためのインタフェースを、図 5.10 に示す。

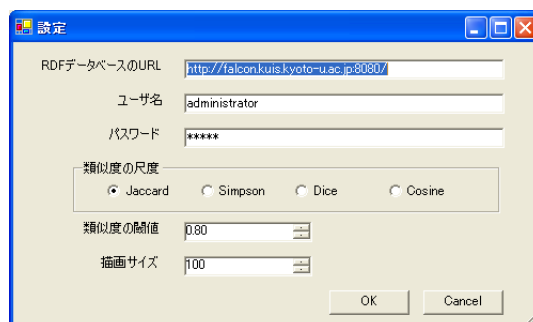


図 5.10: 設定画面

図 5.10 中の RDF データベースの URL・ユーザ名・パスワードの 3 つが、RDF データベースへのアクセスを行えるようにするためのものである。これらを正しく設定することにより、ユーザが明示的に RDF データベースへのログイン作業を行わなくても、自動的に RDF データベースへログインすることが可能となる。

図 5.10 中の類似度の尺度・類似度の閾値の 2 つは、アルゴリズムの実行に必要な設定である。ユーザは、類似度の尺度を Jaccard 係数・Simpson 係数・Dice 係数・Cosine 係数の 4 つから選択することができる。また、0 以上 1 以下の数値で類似度の閾値を自由に設定できる。ユーザがこれらの値を設定しない場合は、類似度の尺度として Jaccard 係数を選択し、類似度の閾値を 0.8 としてアルゴリズムが実行される。

図 5.10 中の最下行は、コンテンツ分類間のマッピングを可視化する際に必要な設定である。この数値を変更することにより、ツリーの描画サイズを拡大縮小できる。

第 6 章 おわりに

本稿では、ネットワークを介して地理的に離れた人々同士が構成するプロジェクトにおいて生じる以下のような問題を指摘した。

アノテーションデータの互換性の欠如

コンテンツに関する情報を各メンバに伝えるために、コンテンツの分類や

注釈付けがなされる。しかし、各メンバがさまざまな形式で分類や注釈を行うことで、メンバ間で分類や注釈に関する情報の共有ができなくなるという問題が生じる。

コンテンツ分類間の対応付けが困難な問題

コンテンツ分類には各メンバのコンテンツに対する意味づけが表れるため、コンテンツ分類間の対応関係を知ることは、互いのコンテンツに対する意味づけを知ることにつながる。しかし、メンバが分散しているプロジェクトにおいては議論の場が限られるため、これには長い時間と手間を必要とする。

そして、上記の問題を解決するために、本研究ではコンテンツに付加されるメタデータに着目して、以下の2つの提案を行った。

アノテーションデータ形式の共通化

プロジェクトオントロジーを拡張する形で、コンテンツの分類や注釈に用いるメタデータのスキーマを RDF スキーマ形式で定義した。これにより、共通の語彙でコンテンツの分類や注釈を扱うことが可能となり、アノテーションデータの互換性の欠如という問題が解決された。

メタデータ間の対応関係の検出

コンテンツ分類に用いられるメタデータの階層構造をコンテンツに関するオントロジーと捉え、オントロジー2つを入力としてそれらの間のマッピングを出力する問題を、類似概念検出問題として定式化した。そして、この問題の解を求めるアルゴリズムを提案した。本アルゴリズムは、メタデータによるコンテンツ分類の類似性に着目して、2つのメタデータ間に対応関係があるかどうかを求めるものであった。これにより、コンテンツ分類間の対応関係を求めることが可能となった。

さらに、本研究では、上記の提案手法を実現するためにツールの開発を行い、そのツールの詳細について述べた。開発したツールにより、各メンバのコンテンツ分類を相互利用できることを確認し、さらにコンテンツ分類間の対応関係を求めることができた。

今後の課題としては、類似概念検出問題の計算効率を向上させることがあげられる。大規模なプロジェクトになるとコンテンツの量は膨大になるために、類似概念の検出を実用的な時間内で行えなくなる可能性がある。このような状況にも耐えうる解法を考えることが必要である。

謝辞

本研究を行う機会と環境を与えて下さり、明晰で熱意あるご指導をしていただいた石田亨教授に厚く御礼を申し上げます。また、プロジェクト支援ツールの開発にご協力いただいている、和歌山大学の吉野孝助教授に深謝いたします。そして、研究のみならず様々な助言や協力をいただきました中塚康介氏をはじめ、石田研究室の皆様方に心より感謝いたします。

参考文献

- [1] 野村早恵子, 石田亨, 船越要, 安岡美佳, 山下直美: アジアにおける異文化コラボレーション実験 2002: 機械翻訳を介したソフトウェア開発, 情報処理, Vol. 44, No. 5, pp. 503–511 (2003).
- [2] 野村早恵子, 船越要, 山下直美, 安岡美佳, 石田亨: 機械翻訳を介したオープンソースソフトウェア開発: Intercultural Collaboration Experiment 2002, 情報処理学会第 65 回全国大会, 2T6-3 (2003).
- [3] Funakoshi, K., Sugiyama, K., Ishida, T., Yoshino, T., Munemori, J., Zhang, H. and Shi, Z.: Semantic Interoperability in Tools for Intercultural Collaboration, *Third International Conference on Active Media Technology (AMT-05)* (2005).
- [4] Brickley, D. and Guha, R. V.: RDF Vocabulary Description Language 1.0: RDF Schema, Technical report, W3C (2004).
- [5] Nagao, K., Shirai, Y. and Squire, K.: Semantic Annotation and Transcoding: Making Web Content More Accessible, *IEEE MultiMedia*, Vol. 8, No. 2, pp. 69–81 (2001).
- [6] McGuinness, D. L., Fikes, R., Rice, J. and Wilder, S.: An Environment for Merging and Testing Large Ontologies, *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning* (2000).
- [7] Noy, N. F. and Musen, M. A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment, *Proceedings of the 17th National Conference on Artificial Intelligence* (2000).
- [8] Lacher, M. S. and Groh, G.: Facilitating the Exchange of Explicit Knowl-

- edge through Ontology Mappings, *Proceedings of the 14th International Florida Artificial Intelligence Research Society Conference*, AAAI Press, pp. 305–309 (2001).
- [9] van Rijsbergen, C. J.: *Information Retrieval 2nd edition*, Butterworths, London (1979).
- [10] Winer, D.: XML-RPC Specification, Technical report, XML-RPC (1999).

付録

A.1 ソースコード

A.1.1 ABRDFProxy.cs

```
using System;
using System.Collections;
using ABAPI;
using CookComputing.XmlRpc;

namespace SemanticBridge
{
    /// <summary>
    /// ABAPI を使って、RDF リポジトリに接続します
    /// </summary>
    public class ABRDFProxy
    {
        /// <summary>
        /// RDF リポジトリの URI
        /// </summary>
        private string mUrl;
        /// <summary>
        /// ユーザ名
        /// </summary>
        private string mUserName;
        /// <summary>
        /// パスワード
        /// </summary>
        private string mPassword;
        /// <summary>
        /// プロキシ
        /// </summary>
        private ABAPIProxy mABProxy;
        /// <summary>
        /// セッション
        /// </summary>
        private string mSession;

        /// <summary>
        /// セッション
        /// </summary>
        public string Session
        {
            // 読み取り専用
            get
            {
                return mSession;
            }
        }

        /// <summary>
        /// コンストラクタ
    }
}
```

```

/// </summary>
/// <param name="url">RDF リポジトリの URI</param>
/// <param name="username">ユーザ名</param>
/// <param name="password">パスワード</param>
public ABRDFProxy(string url, string username, string password)
{
    mUrl = url;
    mUserName = username;
    mPassword = password;
    mABProxy = null;
    mSession = "";
}

/// <summary>
/// AB サーバにログインします
/// </summary>
public void Login()
{
    // 既にログインされていればログアウト
    Logout();
    // ログイン
    try
    {
        mABProxy = new ABAPIProxy();
        mABProxy.Url = mUrl;
        mSession = mABProxy.loginUser(mUserName, mPassword);
    }
    catch(Exception ex)
    {
        // ログインが失敗した
        mABProxy = null;
        mSession = null;
        // Exception を投げなおす
        throw ex;
    }
}

/// <summary>
/// AB サーバからログアウトします
/// </summary>
public void Logout()
{
    if(mABProxy!=null && mSession!="")
    {
        try
        {
            mABProxy.logoutUser(mSession);
        }
        catch
        {
            // ログアウトが失敗しても構わないので無視
        }
    }
}

```

```

    mABProxy = null;
    mSession = "";
}

/// <summary>
/// トランザクションを開始します
/// </summary>
/// <returns>トランザクション ID</returns>
public string BeginTransaction()
{
    string tid;
    try
    {
        tid = mABProxy.begin(mSession);
    }
    catch (ABAPI.IllegalSessionException)
    {
        // トランザクションを開始できないときは、再ログイン
        mSession = mABProxy.loginUser(mUserName, mPassword);
        tid = mABProxy.begin(mSession);
    }
    return tid;
}

/// <summary>
/// RDQL クエリを AB サーバに投げます
/// </summary>
/// <param name="tid">トランザクション ID</param>
/// <param name="rdql">RDQL 文字列</param>
/// <returns>クエリ結果</returns>
public XmlRpcStruct[] SendRDQLQuery(string tid, string rdql)
{
    Object[] vec = mABProxy.searchRDFNode(mSession, tid, rdql);
    if(vec==null)
    {
        throw new XmlRpcException("no results");
    }
    // Object[] を XmlRpcStruct[] にする
    XmlRpcStruct[] ret = new XmlRpcStruct[vec.Length];
    for(int i=0; i<vec.Length; i++)
    {
        ret[i] = (XmlRpcStruct)vec[i];
    }
    return ret;
}

/// <summary>
/// すべてのプロジェクトを取得します
/// </summary>
/// <returns>プロジェクトクラスの配列</returns>
public PrProject[] GetAllProjects()
{
    // トランザクション ID を取得

```

```

string tid = BeginTransaction();
// プロジェクト名とプロジェクト URI を取得
// NOTE: クエリ仕様
// rdf:type の検索はできなかったので,
// <リポジトリ URI> - pr:project -> <プロジェクト URI>
// <プロジェクト URI> - pr:name -> プロジェクト名
// で取得する
XmlRpcStruct[] results = SendRDQLQuery(tid,
    "SELECT ?proj ?name " +
    "WHERE (<" + mUrl + "> pr:project ?proj) (?proj pr:name ?name)" +
    "USING pr FOR <" + Pr.pr + ">");
// 返り値用 ArrayList
ArrayList alret = new ArrayList();
PrProject project;
foreach(XmlRpcStruct xrs in results)
{
    // PrProject オブジェクトを生成
    project = new PrProject();
    project.Name = (string)xrs["name"];
    project.URI = (string)xrs["proj"];
    alret.Add(project);
}
// PrProject [] 形式に変換
PrProject[] ret = (PrProject[])alret.ToArray(typeof(PrProject));
return ret;
}

/// <summary>
/// 属性 attribute が付加されたすべてのコンテンツを取得します
/// </summary>
/// <param name="attribute">属性の URI</param>
/// <returns>コンテンツの配列</returns>
public PrContent[] GetAllContentByAttribute(string attribute)
{
    // トランザクション ID を取得
    string tid = BeginTransaction();
    // まずはコンテンツの URI 取得
    XmlRpcStruct[] results = SendRDQLQuery(tid,
        "SELECT ?object " +
        "WHERE (<" + attribute + "> pr:content ?object)" +
        "USING pr FOR <" + Pr.pr + ">");
    // 返り値用 ArrayList
    ArrayList alret = new ArrayList();
    PrContent prcontent;
    foreach(XmlRpcStruct xrs in results)
    {
        // PrContent オブジェクトを生成
        prcontent = new PrContent();
        string resourceuri = (string)xrs["object"];
        XmlRpcStruct[] results2;
        // このコンテンツの持つプロパティを全て取得
        try
        {

```

```

    results2 = SendRDQLQuery(tid,
        "SELECT ?predicate ?object " +
        "WHERE (<" + resourceuri + "> ?predicate ?object)");
}
catch(Exception)
{
    // 例外を捕まえた場合は無視して継続
    continue;
}
prcontent.Resource = resourceuri;
// プロパティ毎に PrContent オブジェクトに入れる
foreach(XmlRpcStruct xrs2 in results2)
{
    string key = (string)xrs2["predicate"];
    string val = (string)xrs2["object"];
    if(key==Pr.format)
    {
        prcontent.Format = val;
    }
    else if(key==Pr.name)
    {
        prcontent.Name = val;
    }
    else if(key==Pr.created)
    {
        prcontent.Created = val;
    }
    else if(key==Pr.creator)
    {
        prcontent.Creator = val;
    }
    else if(key==Pr.modified)
    {
        prcontent.Modified = val;
    }
    else if(key==Pr.text)
    {
        // 今の所、型がとれないので、text があれば TextDocument と解釈
        // prcontent は PrContent 型なので、PrTextDocument 型に変えないといけない
        // ダウンキャストを強行
        prcontent = (PrContent)new PrTextDocument(prcontent);
        // テキストを入れる
        ((PrTextDocument)prcontent).Text = val;
    }
}
// 値を設定し終わったら追加
alret.Add(prcontent);
}
// PrContent [] 形式に変換
PrContent[] ret = (PrContent[])alret.ToArray(typeof(PrContent));
return ret;
}

```

```

/// <summary>
/// プロジェクト名から、そのプロジェクトに参加するすべてのエージェントを取得します
/// </summary>
/// <param name="project">プロジェクトの URI</param>
/// <returns>エージェントの配列</returns>
public PrAgent[] GetAllAgentByProject(string project)
{
    // トランザクション ID を取得
    string tid = BeginTransaction();
    // まずはエージェントの URI 取得
    XmlRpcStruct[] results = SendRDQLQuery(tid,
        "SELECT ?object " +
        "WHERE (<" + project + "> pr:agent ?object)" +
        "USING pr FOR <" + Pr.pr + ">");
    // 返り値用 ArrayList
    ArrayList alret = new ArrayList();
    PrAgent pragent;
    foreach(XmlRpcStruct xrs in results)
    {
        // PrAgent オブジェクトを生成
        pragent = new PrAgent();
        string resourceuri = (string)xrs["object"];
        XmlRpcStruct[] results2;
        // このエージェントの持つプロパティを全て取得
        try
        {
            {
                results2 = SendRDQLQuery(tid,
                    "SELECT ?predicate ?object " +
                    "WHERE (<" + resourceuri + "> ?predicate ?object)");
            }
            catch
            {
                // 例外を捕まえた場合は無視して継続
                continue;
            }
            pragent.Resource = resourceuri;
            // プロパティ毎に PrAgent オブジェクトに入れる
            foreach(XmlRpcStruct xrs2 in results2)
            {
                string key = (string)xrs2["predicate"];
                string val = (string)xrs2["object"];
                if(key==Pr.fullname)
                {
                    pragent.Fullname = val;
                }
            }
            // 値を設定し終わったら追加
            alret.Add(pragent);
        }
        // PrAgent [] 形式に変換
        PrAgent[] ret = (PrAgent[])alret.ToArray(typeof(PrAgent));
        return ret;
    }
}

```

```

/// <summary>
/// あるプロジェクトメンバが作成した属性を取得します
/// </summary>
/// <param name="project">プロジェクトの URI</param>
/// <param name="agent">エージェントの URI</param>
/// <returns>属性の配列</returns>
public AttrAttribute[] GetAllAttributeByProjectAndAgent(string project, string agent)
{
    // トランザクション ID を取得
    string tid = BeginTransaction();
    // まずは属性の URI 取得
    XmlRpcStruct[] results = SendRDQLQuery(tid,
        "SELECT ?object " +
        "WHERE (<" + project + "> attr:attribute ?object) (?object pr:creator <" + agent + ">)"
" +
        "USING pr FOR <" + Pr.pr + ">" + "attr FOR <" + Attr.attr + ">");
    // 返り値用 ArrayList
    ArrayList alret = new ArrayList();
    AttrAttribute attrattribute;
    foreach(XmlRpcStruct xrs in results)
    {
        // AttrAttribute オブジェクトを生成
        attrattribute = new AttrAttribute();
        string resourceuri = (string)xrs["object"];
        XmlRpcStruct[] results2, results3;
        try
        {
            // この属性が持つさまざまなプロパティを取得
            results2 = SendRDQLQuery(tid,
                "SELECT ?predicate ?object " +
                "WHERE (<" + resourceuri + "> ?predicate ?object)");
            // この属性の上位属性を取得
            results3 = SendRDQLQuery(tid,
                "SELECT ?subject " +
                "WHERE (?subject attr:subAttribute <" + resourceuri + ">)" +
                "USING attr FOR <" + Attr.attr + ">");
        }
        catch
        {
            // 例外を捕まえた場合は無視して継続
            continue;
        }
        attrattribute.Resource = resourceuri;
        // 今作成中の AttrAttribute オブジェクトの下位属性を格納する ArrayList
        ArrayList subalret = new ArrayList();
        // プロパティ毎に AttrAttribute オブジェクトに入れる
        foreach(XmlRpcStruct xrs2 in results2)
        {
            string key = (string)xrs2["predicate"];
            string val = (string)xrs2["object"];
            if(key==Pr.name)
            {

```

```

        attrattribute.Name = val;
    }
    else if(key==Attr.subAttribute)
    {
        // 今作成中の AttrAttribute オブジェクトの下位属性に関する情報を取得
        AttrAttribute subattrattribute = GetSubAttribute(val);
        // 下位属性は一般に複数あるので、とりあえず ArrayList に格納
        subalret.Add(subattrattribute);
    }
}
// subAttribute プロパティが完成したので、追加 (ArrayList->AttrAttribute[])
attrattribute.SubAttributes = (AttrAttribute[])subalret.ToArray(typeof(AttrAttribute))
;

// 上位属性の情報を取得
// TODO: 現在は URI のみの取得
foreach(XmlRpcStruct xrs3 in results3)
{
    string val = (string)xrs3["subject"];
    attrattribute.ParentAttribute = new AttrAttribute();
    attrattribute.ParentAttribute.Resource = val;
}
// 値を設定し終わったら追加
alret.Add(attrattribute);
}
// AttrAttribute[] 形式に変換
AttrAttribute[] ret = (AttrAttribute[])alret.ToArray(typeof(AttrAttribute));
return ret;
}

/// <summary>
/// 下位属性のプロパティを取得
/// </summary>
/// <param name="attribute">下位属性の URI</param>
/// <returns>下位属性オブジェクト</returns>
public AttrAttribute GetSubAttribute(string attribute)
{
    // 返り値用下位属性オブジェクト
    AttrAttribute ret = new AttrAttribute();
    ret.Resource = attribute;
    // トランザクション ID を取得
    string tid = BeginTransaction();
    // 下位属性の持つプロパティを全て取得
    XmlRpcStruct[] results = SendRDQLQuery(tid,
        "SELECT ?predicate ?object " +
        "WHERE (<" + attribute + "> ?predicate ?object)");
    // 下位属性の下位属性を仮に格納する ArrayList
    ArrayList subalret = new ArrayList();
    // プロパティ毎に AttrAttribute オブジェクトに入れる
    foreach(XmlRpcStruct xrs in results)
    {
        string key = (string)xrs["predicate"];
        string val = (string)xrs["object"];
        if(key==Pr.name)

```

```

    {
        ret.Name = val;
    }
    else if(key==Attr.subAttribute)
    {
        // 下位属性の情報を再帰的に取得
        AttrAttribute subret = GetSubAttribute(val);
        subalret.Add(subret);
    }
}
// 下位属性を全て追加 ( ArrayList->AttrAttribute[] )
ret.SubAttributes = (AttrAttribute[])subalret.ToArray(typeof(AttrAttribute));
return ret;
}
}
}

```

A.1.2 Attr.cs

```

using System;

namespace SemanticBridge
{
    /// <summary>
    /// attr 空間内のタグ定義
    /// </summary>
    public class Attr
    {
        /// <summary>
        /// attr 空間の URI
        /// </summary>
        public const string attr = "http://ice.kuis.kyoto-u.ac.jp/ont/attr/0.1/";
        /// <summary>
        /// subAttribute プロパティ
        /// </summary>
        public const string subAttribute = attr + "subAttribute";

        /// <summary>
        /// インスタンス生成を行わせないための処理
        /// </summary>
        private Attr()
        {
        }
    }
}

```

A.1.3 AttrAttribute.cs

```

using System;
namespace SemanticBridge
{
    /// <summary>
    /// Project Vocabulary に付随する属性クラス
    public class AttrAttribute
    {
        /// <summary>
        /// 属性の URI
        /// </summary>

```

```

private string mResource;
/// <summary>
/// 属性を付加したコンテンツ
/// </summary>
private PrContent[] mContents;
/// <summary>
/// 属性の定義者
/// </summary>
private PrAgent mCreator;
/// <summary>
/// 属性名
/// </summary>
private string mName;
/// <summary>
/// 下位の属性
/// </summary>
private AttrAttribute[] mSubAttributes;
/// <summary>
/// 上位の属性
/// </summary>
private AttrAttribute mParentAttribute;

/// <summary>
/// 属性の URI
/// </summary>
public string Resource
{
    get
    {
        return mResource;
    }
    set
    {
        mResource = value;
    }
}

/// <summary>
/// 属性を付加したコンテンツ
/// </summary>
public PrContent[] Contents
{
    get
    {
        return mContents;
    }
    set
    {
        mContents = value;
    }
}

/// <summary>
/// 属性の定義者
/// </summary>
public PrAgent Creator
{
    get
    {
        return mCreator;
    }
    set
    {
        mCreator = value;
    }
}

/// <summary>
/// 属性名
/// </summary>
public string Name
{
    get
    {
        return mName;
    }
    set
    {
        mName = value;
    }
}

/// <summary>
/// 下位の属性
/// </summary>
public AttrAttribute[] SubAttributes
{
    get
    {
        return mSubAttributes;
    }
    set
    {
        mSubAttributes = value;
    }
}

/// <summary>
/// 上位の属性
/// </summary>
public AttrAttribute ParentAttribute
{
    get
    {
        return mParentAttribute;
    }
}

```

```

        set
        {
            mParentAttribute = value;
        }
    }

    /// <summary>
    /// 属性クラスのコンストラクタ
    /// </summary>
    public AttrAttribute()
    {
        mResource = "";
        mContents = new PrContent[0];
        mCreator = null;
        mName = "";
        mSubAttributes = new AttrAttribute[0];
        mParentAttribute = null;
    }
}

```

A.1.4 AttributeItem.cs

```

using System;

namespace SemanticBridge
{
    /// <summary>
    /// 属性のデータ構造クラス
    /// </summary>
    public class AttributeItem
    {
        /// <summary>
        /// 属性の URI
        /// </summary>
        private string mResource;
        /// <summary>
        /// 属するコンテンツ
        /// </summary>
        private PrContent[] mContents;
        /// <summary>
        /// 属性ノードの x 座標 (可視化時に利用)
        /// </summary>
        private int mx;
        /// <summary>
        /// 属性ノードの y 座標 (可視化時に利用)
        /// </summary>
        private int my;

        /// <summary>
        /// 属性の URI
        /// </summary>
        public string Resource
        {
            get
            {
                return mResource;
            }
            set
            {
                mResource = value;
            }
        }

        /// <summary>
        /// 属するコンテンツ
        /// </summary>
        public PrContent[] Contents
        {
            get
            {
                return mContents;
            }
            set
            {
                mContents = value;
            }
        }

        /// <summary>
        /// 属性ノードの x 座標 (可視化時に利用)
        /// </summary>
        public int x
        {
            get
            {
                return mx;
            }
            set
            {
                mx = value;
            }
        }

        /// <summary>
        /// 属性ノードの y 座標 (可視化時に利用)
        /// </summary>
        public int y
        {
            get
            {

```

```

        return my;
    }
    set
    {
        my = value;
    }
}

/// <summary>
/// コンストラクタ

```

```

/// </summary>
public AttributeItem()
{
    mResource = "";
    mContents = new PrContent[0];
    mx = 0;
    my = 0;
}
}
}

```

A.1.5 ConfigForm.cs

```

using System;
using System.Xml.Serialization;
using System.IO;

namespace SemanticBridge
{
    /// <summary>
    /// 設定ウィンドウ用クラス
    /// </summary>
    public class ConfigForm : System.Windows.Forms.Form
    {
        /// <summary>
        /// 設定データ
        /// </summary>
        private Constant mConfig;

        /// <summary>
        /// 設定データ
        /// </summary>
        public Constant Config
        {
            // 読み取り専用
            get
            {
                return mConfig;
            }
        }

        private System.Windows.Forms.Label labelRDFDB;
        private System.Windows.Forms.Label labelUserName;
        private System.Windows.Forms.Label labelPassword;
        private System.Windows.Forms.Label labelDrawSize;
        private System.Windows.Forms.TextBox textBoxRDFDB;
        private System.Windows.Forms.TextBox textBoxUserName;
        private System.Windows.Forms.TextBox textBoxPassword;
        private System.Windows.Forms.NumericUpDown numericUpDownDrawSize;
        private System.Windows.Forms.Button buttonOK;
        private System.Windows.Forms.Button buttonCancel;
        private System.Windows.Forms.Label labelSim;
        private System.Windows.Forms.NumericUpDown numericUpDownSim;
    }
}

```

```

private System.Windows.Forms.GroupBox groupBoxSim;
private System.Windows.Forms.RadioButton radioButtonJaccard;
private System.Windows.Forms.RadioButton radioButtonSimpson;
private System.Windows.Forms.RadioButton radioButtonDice;
private System.Windows.Forms.RadioButton radioButtonCosine;
/// <summary>
/// 必要なデザイナ変数です。
/// </summary>
private System.ComponentModel.Container components = null;

/// <summary>
/// 設定ウィンドウのコンストラクタ
/// </summary>
public ConfigForm()
{
    //
    // Windows フォーム デザイナ サポートに必要です。
    //
    InitializeComponent();
    // 設定を読み込んで、ダイアログボックスに表示
    mConfig = LoadSetting();
    this.textBoxRDFDB.Text = mConfig.RDFDB;
    this.textBoxUserName.Text = mConfig.UserName;
    this.textBoxPassword.Text = mConfig.Password;
    this.numericUpDownSim.Value = (decimal)mConfig.Sim;
    this.numericUpDownDrawSize.Value = mConfig.Distance;
    if(mConfig.Type==Constant.JACCARD)
    {
        this.radioButtonJaccard.Checked = true;
    }
    else if(mConfig.Type==Constant.SIMPSON)
    {
        this.radioButtonSimpson.Checked = true;
    }
    else if(mConfig.Type==Constant.DICE)
    {
        this.radioButtonDice.Checked = true;
    }
    else if(mConfig.Type==Constant.COSINE)
    {
        this.radioButtonCosine.Checked = true;
    }
}

/// <summary>
/// 使用されているリソースに後処理を実行します。
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if(components != null)
        {

```

```

        components.Dispose();
    }
}
base.Dispose( disposing );
}

#region Windows フォーム デザイナで生成されたコード
/// <summary>
/// デザイナ サポートに必要なメソッドです。このメソッドの内容を
/// コード エディタで変更しないでください。
/// </summary>
private void InitializeComponent()
{
    this.labelRDFDB = new System.Windows.Forms.Label();
    this.labelUserName = new System.Windows.Forms.Label();
    this.labelPassword = new System.Windows.Forms.Label();
    this.labelSim = new System.Windows.Forms.Label();
    this.labelDrawSize = new System.Windows.Forms.Label();
    this.textBoxRDFDB = new System.Windows.Forms.TextBox();
    this.textBoxUserName = new System.Windows.Forms.TextBox();
    this.textBoxPassword = new System.Windows.Forms.TextBox();
    this.numericUpDownSim = new System.Windows.Forms.NumericUpDown();
    this.numericUpDownDrawSize = new System.Windows.Forms.NumericUpDown();
    this.buttonOK = new System.Windows.Forms.Button();
    this.buttonCancel = new System.Windows.Forms.Button();
    this.groupBoxSim = new System.Windows.Forms.GroupBox();
    this.radioButtonJaccard = new System.Windows.Forms.RadioButton();
    this.radioButtonSimpson = new System.Windows.Forms.RadioButton();
    this.radioButtonDice = new System.Windows.Forms.RadioButton();
    this.radioButtonCosine = new System.Windows.Forms.RadioButton();
    ((System.ComponentModel.ISupportInitialize)(this.numericUpDownSim)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.numericUpDownDrawSize)).BeginInit();
    this.groupBoxSim.SuspendLayout();
    this.SuspendLayout();
    //
    // labelRDFDB
    //
    this.labelRDFDB.Location = new System.Drawing.Point(24, 16);
    this.labelRDFDB.Name = "labelRDFDB";
    this.labelRDFDB.Size = new System.Drawing.Size(120, 16);
    this.labelRDFDB.TabIndex = 0;
    this.labelRDFDB.Text = "RDF データベースの URL";
    //
    // labelUserName
    //
    this.labelUserName.Location = new System.Drawing.Point(96, 48);
    this.labelUserName.Name = "labelUserName";
    this.labelUserName.Size = new System.Drawing.Size(48, 16);
    this.labelUserName.TabIndex = 1;
    this.labelUserName.Text = "ユーザ名";
    //
    // labelPassword
    //

```

```

this.labelPassword.Location = new System.Drawing.Point(88, 80);
this.labelPassword.Name = "labelPassword";
this.labelPassword.Size = new System.Drawing.Size(56, 16);
this.labelPassword.TabIndex = 2;
this.labelPassword.Text = "パスワード";
//
// labelSim
//
this.labelSim.Location = new System.Drawing.Point(64, 168);
this.labelSim.Name = "labelSim";
this.labelSim.Size = new System.Drawing.Size(80, 16);
this.labelSim.TabIndex = 3;
this.labelSim.Text = "類似度の閾値";
//
// labelDrawSize
//
this.labelDrawSize.Location = new System.Drawing.Point(80, 200);
this.labelDrawSize.Name = "labelDrawSize";
this.labelDrawSize.Size = new System.Drawing.Size(64, 16);
this.labelDrawSize.TabIndex = 4;
this.labelDrawSize.Text = "描画サイズ";
//
// textBoxRDFDB
//
this.textBoxRDFDB.Location = new System.Drawing.Point(160, 16);
this.textBoxRDFDB.Name = "textBoxRDFDB";
this.textBoxRDFDB.Size = new System.Drawing.Size(300, 19);
this.textBoxRDFDB.TabIndex = 5;
this.textBoxRDFDB.Text = "";
//
// textBoxUserName
//
this.textBoxUserName.Location = new System.Drawing.Point(160, 48);
this.textBoxUserName.Name = "textBoxUserName";
this.textBoxUserName.Size = new System.Drawing.Size(300, 19);
this.textBoxUserName.TabIndex = 6;
this.textBoxUserName.Text = "";
//
// textBoxPassword
//
this.textBoxPassword.Location = new System.Drawing.Point(160, 80);
this.textBoxPassword.Name = "textBoxPassword";
this.textBoxPassword.PasswordChar = '*';
this.textBoxPassword.Size = new System.Drawing.Size(300, 19);
this.textBoxPassword.TabIndex = 7;
this.textBoxPassword.Text = "";
//
// numericUpDownSim
//
this.numericUpDownSim.DecimalPlaces = 2;
this.numericUpDownSim.Increment = new System.Decimal(new int[] {
    1,
    0,

```

```

        0,
        131072});
this.numericUpDownSim.Location = new System.Drawing.Point(160, 168);
this.numericUpDownSim.Maximum = new System.Decimal(new int[] {
    1,
    0,
    0,
    0});
this.numericUpDownSim.Name = "numericUpDownSim";
this.numericUpDownSim.TabIndex = 9;
//
// numericUpDownDrawSize
//
this.numericUpDownDrawSize.Location = new System.Drawing.Point(160, 200);
this.numericUpDownDrawSize.Maximum = new System.Decimal(new int[] {
    1000,
    0,
    0,
    0});
this.numericUpDownDrawSize.Minimum = new System.Decimal(new int[] {
    1,
    0,
    0,
    0});
this.numericUpDownDrawSize.Name = "numericUpDownDrawSize";
this.numericUpDownDrawSize.TabIndex = 10;
this.numericUpDownDrawSize.Value = new System.Decimal(new int[] {
    1,
    0,
    0,
    0});
//
// buttonOK
//
this.buttonOK.DialogResult = System.Windows.Forms.DialogResult.OK;
this.buttonOK.Location = new System.Drawing.Point(296, 232);
this.buttonOK.Name = "buttonOK";
this.buttonOK.TabIndex = 11;
this.buttonOK.Text = "OK";
this.buttonOK.Click += new System.EventHandler(this.buttonOK_Click);
//
// buttonCancel
//
this.buttonCancel.DialogResult = System.Windows.Forms.DialogResult.Cancel;
this.buttonCancel.Location = new System.Drawing.Point(384, 232);
this.buttonCancel.Name = "buttonCancel";
this.buttonCancel.TabIndex = 12;
this.buttonCancel.Text = "Cancel";
this.buttonCancel.Click += new System.EventHandler(this.buttonCancel_Click);
//
// groupBoxSim
//
this.groupBoxSim.Controls.Add(this.radioButtonCosine);

```

```

this.groupBoxSim.Controls.Add(this.radioButtonDice);
this.groupBoxSim.Controls.Add(this.radioButtonSimpson);
this.groupBoxSim.Controls.Add(this.radioButtonJaccard);
this.groupBoxSim.Location = new System.Drawing.Point(56, 112);
this.groupBoxSim.Name = "groupBoxSim";
this.groupBoxSim.Size = new System.Drawing.Size(400, 48);
this.groupBoxSim.TabIndex = 8;
this.groupBoxSim.TabStop = false;
this.groupBoxSim.Text = "類似度の尺度";
//
// radioButtonJaccard
//
this.radioButtonJaccard.Location = new System.Drawing.Point(32, 16);
this.radioButtonJaccard.Name = "radioButtonJaccard";
this.radioButtonJaccard.Size = new System.Drawing.Size(80, 24);
this.radioButtonJaccard.TabIndex = 0;
this.radioButtonJaccard.Text = "Jaccard";
//
// radioButtonSimpson
//
this.radioButtonSimpson.Location = new System.Drawing.Point(120, 16);
this.radioButtonSimpson.Name = "radioButtonSimpson";
this.radioButtonSimpson.Size = new System.Drawing.Size(80, 24);
this.radioButtonSimpson.TabIndex = 1;
this.radioButtonSimpson.Text = "Simpson";
//
// radioButtonDice
//
this.radioButtonDice.Location = new System.Drawing.Point(208, 16);
this.radioButtonDice.Name = "radioButtonDice";
this.radioButtonDice.Size = new System.Drawing.Size(80, 24);
this.radioButtonDice.TabIndex = 2;
this.radioButtonDice.Text = "Dice";
//
// radioButtonCosine
//
this.radioButtonCosine.Location = new System.Drawing.Point(296, 16);
this.radioButtonCosine.Name = "radioButtonCosine";
this.radioButtonCosine.Size = new System.Drawing.Size(80, 24);
this.radioButtonCosine.TabIndex = 3;
this.radioButtonCosine.Text = "Cosine";
//
// ConfigForm
//
this.AutoScaleBaseSize = new System.Drawing.Size(5, 12);
this.ClientSize = new System.Drawing.Size(492, 266);
this.Controls.Add(this.groupBoxSim);
this.Controls.Add(this.buttonCancel);
this.Controls.Add(this.buttonOK);
this.Controls.Add(this.numericUpDownDrawSize);
this.Controls.Add(this.numericUpDownSim);
this.Controls.Add(this.textBoxPassword);
this.Controls.Add(this.textBoxUserName);

```

```

        this.Controls.Add(this.textBoxRDFDB);
        this.Controls.Add(this.labelDrawSize);
        this.Controls.Add(this.labelSim);
        this.Controls.Add(this.labelPassword);
        this.Controls.Add(this.labelUserName);
        this.Controls.Add(this.labelRDFDB);
        this.Name = "ConfigForm";
        this.Text = "設定";
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDownSim)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDownDrawSize)).EndInit();
        this.groupBoxSim.ResumeLayout(false);
        this.ResumeLayout(false);
    }
#endregion

/// <summary>
/// 設定ファイルから設定を読み込む
/// </summary>
/// <returns>設定データ</returns>
private Constant LoadSetting()
{
    Constant c = new Constant();
    try
    {
        XmlSerializer serializer = new XmlSerializer(typeof(Constant));
        FileStream reader = new FileStream(Constant.Filename, FileMode.Open);
        c = (Constant)serializer.Deserialize(reader);
        reader.Close();
    }
    catch(FileNotFoundException)
    {
        // 設定ファイルがなければ、初期値を利用
    }
    return c;
}

/// <summary>
/// 設定ファイルに設定を書き込む
/// </summary>
/// <param name="c">設定データ</param>
private void SaveSetting(Constant c)
{
    XmlSerializer serializer = new XmlSerializer(typeof(Constant));
    StreamWriter writer = new StreamWriter(Constant.Filename);
    serializer.Serialize(writer, c);
    writer.Close();
}

/// <summary>
/// キャンセルボタンが押されたときの処理
/// </summary>
/// <param name="sender"></param>

```

```

/// <param name="e"></param>
private void buttonCancel_Click(object sender, System.EventArgs e)
{
    this.Close();
}

/// <summary>
/// OK ボタンが押されたときの処理
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void buttonOK_Click(object sender, System.EventArgs e)
{
    // 設定データを取得
    mConfig.RDFDB = this.textBoxRDFDB.Text;
    mConfig.UserName = this.textBoxUserName.Text;
    mConfig.Password = this.textBoxPassword.Text;
    mConfig.Sim = (double)this.numericUpDownSim.Value;
    mConfig.Distance = (int)this.numericUpDownDrawSize.Value;
    if(this.radioButtonJaccard.Checked)
    {
        mConfig.Type = Constant.JACCARD;
    }
    else if(this.radioButtonSimpson.Checked)
    {
        mConfig.Type = Constant.SIMPSON;
    }
    else if(this.radioButtonDice.Checked)
    {
        mConfig.Type = Constant.DICE;
    }
    else if(this.radioButtonCosine.Checked)
    {
        mConfig.Type = Constant.COSINE;
    }
    // 設定データを保存
    SaveSetting(mConfig);
    this.Close();
}
}
}

```

A.1.6 Constant.cs

```

using System;

namespace SemanticBridge
{
    /// <summary>
    /// RDF データベースの URL
    /// </summary>
    private string mRDFDB;
    /// <summary>
    /// ユーザ名
    /// </summary>
    private string mUserName;
    /// <summary>

```

```

/// パスワード
/// </summary>
private string mPassword;
/// <summary>
/// Jaccard 係数の閾値
/// </summary>
private double mSim;
/// <summary>
/// 属性ノード間の距離
/// </summary>
private int mDistance;
/// <summary>
/// どの類似度尺度を使うかを示す
/// </summary>
private int mType;

/// <summary>
/// 設定保存ファイルの名前
/// </summary>
public const string Filename="setting.xml";
/// <summary>
/// 類似度の尺度を表す整数 ( Jaccard 係数 )
/// </summary>
public const int JACCARD = 1;
/// <summary>
/// 類似度の尺度を表す整数 ( Simpson 係数 )
/// </summary>
public const int SIMPSON = 2;
/// <summary>
/// 類似度の尺度を表す整数 ( Dice 係数 )
/// </summary>
public const int DICE = 3;
/// <summary>
/// 類似度の尺度を表す整数 ( Cosine 係数 )
/// </summary>
public const int COSINE = 4;
/// <summary>
/// 閾値つき Simpson 係数の閾値
/// </summary>
public const int THRESHOLD = 0;

/// <summary>
/// RDF データベースの URL
/// </summary>
public string RDFDB
{
    get
    {
        return mRDFDB;
    }
    set
    {
        mRDFDB = value;
    }
}

}

}

/// <summary>
/// ユーザ名
/// </summary>
public string UserName
{
    get
    {
        return mUserName;
    }
    set
    {
        mUserName = value;
    }
}

}

/// <summary>
/// パスワード
/// </summary>
public string Password
{
    get
    {
        return mPassword;
    }
    set
    {
        mPassword = value;
    }
}

}

/// <summary>
/// Jaccard 係数の閾値
/// </summary>
public double Sim
{
    get
    {
        return mSim;
    }
    set
    {
        mSim = value;
    }
}

}

/// <summary>
/// 属性ノード間の距離
/// </summary>
public int Distance
{

```

```

    get
    {
        return mDistance;
    }
    set
    {
        mDistance = value;
    }
}

/// <summary>
/// この類似度尺度を使うかを示す
/// </summary>
public int Type
{
    get
    {
        return mType;
    }
    set
    {
        mType = value;
    }
}

/// <summary>
/// コンストラクタ
/// </summary>
public Constant()
{
    mRDFDB = "";
    mUserName = "";
    mPassword = "";
    mSim = 0.8;
    mDistance = 100;
    mType = JACCARD;
}
}
}

```

A.1.7 MainForm.cs

```

using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using CookComputing.XmlRpc;

namespace SemanticBridge
{
    /// <summary>
    /// SemanticBridge 本体
    /// </summary>
    public class MainForm : System.Windows.Forms.Form
    {
        /// <summary>
        /// プロジェクトの URI を格納
        /// </summary>
        private ArrayList projectURI = new ArrayList();
        /// <summary>
        /// プロジェクトメンバの URI を格納
        /// </summary>
        private ArrayList agentURI = new ArrayList();
        /// <summary>
        /// プロジェクトメンバの URI と名前の対応関係を示す表
        /// </summary>
        private Hashtable agentTable = new Hashtable();
        /// <summary>
        /// プロジェクトメンバ 1 のコンテンツ分類を格納
        /// </summary>
        private Hashtable mem1;
        /// <summary>

```

```

/// プロジェクトメンバ 2 のコンテンツ分類を格納
/// </summary>
private Hashtable mem2;
/// <summary>
/// 属性ノードの描画位置を決めるために使用する変数 ( 葉ノードの個数 )
/// </summary>
private int elem;
/// <summary>
/// 属性ノードの深さの最大値
/// </summary>
private int MaxDepth;
/// <summary>
/// 定数を格納
/// </summary>
private Constant c;
/// <summary>
/// 類似属性の候補を格納
/// </summary>
private Hashtable candidate = new Hashtable();
/// <summary>
/// 類似属性を格納 ( キーはプロジェクトメンバ 1 の属性ノード )
/// </summary>
private Hashtable result1 = new Hashtable();
/// <summary>
/// 類似属性を格納 ( キーはプロジェクトメンバ 2 の属性ノード )
/// </summary>
private Hashtable result2 = new Hashtable();

private System.Windows.Forms.MainMenu mainMenu;
private System.Windows.Forms.MenuItem menuItemFile;
private System.Windows.Forms.MenuItem menuItemExit;
private System.Windows.Forms.Panel panel1;
private System.Windows.Forms.Splitter splitter1;
private System.Windows.Forms.Panel panel2;
private System.Windows.Forms.Splitter splitter2;
private System.Windows.Forms.Panel panel3;
private System.Windows.Forms.Splitter splitter3;
private System.Windows.Forms.Panel panel4;
private System.Windows.Forms.Splitter splitter4;
private System.Windows.Forms.Splitter splitter5;
private System.Windows.Forms.ComboBox comboBoxForProject;
private System.Windows.Forms.Label labelProject;
private System.Windows.Forms.ComboBox comboBoxForMember1;
private System.Windows.Forms.Label labelMember1;
private System.Windows.Forms.TreeListView treeListViewForMember1;
private System.Windows.Forms.TreeView treeViewForMember1;
private System.Windows.Forms.ComboBox comboBoxForMember2;
private System.Windows.Forms.Label labelMember2;
private System.Windows.Forms.ColumnHeader columnHeaderSubject1;
private System.Windows.Forms.ColumnHeader columnHeaderFrom1;
private System.Windows.Forms.ColumnHeader columnHeaderDate1;
private System.Windows.Forms.ColumnHeader columnHeaderFilename1;
private System.Windows.Forms.ImageList imageListFolder;

```

```

private System.Windows.Forms.StatusBar statusBar;
private System.Windows.Forms.Panel panel5;
private System.Windows.Forms.TreeView treeViewForMember2;
private System.Windows.Forms.Splitter splitter6;
private System.Windows.Forms.TreeListView treeListViewForMember2;
private System.Windows.Forms.ColumnHeader columnHeaderSubject2;
private System.Windows.Forms.ColumnHeader columnHeaderFrom2;
private System.Windows.Forms.ColumnHeader columnHeaderDate2;
private System.Windows.Forms.ColumnHeader columnHeaderFilename2;
private System.Windows.Forms.Button DrawTreeButton;
private System.Windows.Forms.MenuItem menuItemSettingFolder;
private System.Windows.Forms.MenuItem menuItemSetting;
private System.ComponentModel.IContainer components;

/// <summary>
/// SemanticBridge のコンストラクタ
/// </summary>
public MainForm()
{
    //
    // Windows フォーム デザイナ サポートにが必要です。
    //
    InitializeComponent();
}

/// <summary>
/// 使用されているリソースに後処理を実行します。
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

#region Windows フォーム デザイナで生成されたコード
/// <summary>
/// デザイナ サポートに必要なメソッドです。このメソッドの内容を
/// コード エディタで変更しないでください。
/// </summary>
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    System.Resources.ResourceManager resources = new System.Resources.ResourceManager(typeof(
MainForm));
    this.mainMenu = new System.Windows.Forms.MainMenu();
    this.menuItemFile = new System.Windows.Forms.MenuItem();
    this.menuItemExit = new System.Windows.Forms.MenuItem();
}

```

```

this.menuItemSettingFolder = new System.Windows.Forms.MenuItem();
this.menuItemSetting = new System.Windows.Forms.MenuItem();
this.panel1 = new System.Windows.Forms.Panel();
this.DrawTreeButton = new System.Windows.Forms.Button();
this.comboBoxForProject = new System.Windows.Forms.ComboBox();
this.labelProject = new System.Windows.Forms.Label();
this.splitter1 = new System.Windows.Forms.Splitter();
this.panel2 = new System.Windows.Forms.Panel();
this.comboBoxForMember1 = new System.Windows.Forms.ComboBox();
this.labelMember1 = new System.Windows.Forms.Label();
this.splitter2 = new System.Windows.Forms.Splitter();
this.panel3 = new System.Windows.Forms.Panel();
this.treeListViewForMember1 = new System.Windows.Forms.TreeListView();
this.columnHeaderSubject1 = new System.Windows.Forms.ColumnHeader();
this.columnHeaderFrom1 = new System.Windows.Forms.ColumnHeader();
this.columnHeaderDate1 = new System.Windows.Forms.ColumnHeader();
this.columnHeaderFilename1 = new System.Windows.Forms.ColumnHeader();
this.splitter5 = new System.Windows.Forms.Splitter();
this.treeViewForMember1 = new System.Windows.Forms.TreeView();
this.imageListFolder = new System.Windows.Forms.ImageList(this.components);
this.splitter3 = new System.Windows.Forms.Splitter();
this.panel4 = new System.Windows.Forms.Panel();
this.comboBoxForMember2 = new System.Windows.Forms.ComboBox();
this.labelMember2 = new System.Windows.Forms.Label();
this.splitter4 = new System.Windows.Forms.Splitter();
this.statusBar = new System.Windows.Forms.StatusBar();
this.panel5 = new System.Windows.Forms.Panel();
this.treeListViewForMember2 = new System.Windows.Forms.TreeListView();
this.columnHeaderSubject2 = new System.Windows.Forms.ColumnHeader();
this.columnHeaderFrom2 = new System.Windows.Forms.ColumnHeader();
this.columnHeaderDate2 = new System.Windows.Forms.ColumnHeader();
this.columnHeaderFilename2 = new System.Windows.Forms.ColumnHeader();
this.splitter6 = new System.Windows.Forms.Splitter();
this.treeViewForMember2 = new System.Windows.Forms.TreeView();
this.panel1.SuspendLayout();
this.panel2.SuspendLayout();
this.panel3.SuspendLayout();
this.panel4.SuspendLayout();
this.panel5.SuspendLayout();
this.SuspendLayout();
//
// mainMenu
//
this.mainMenu.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
    this.menuItemFile,
    this.menuItemSettingFolder});
//
// menuItemFile
//
this.menuItemFile.Index = 0;
this.menuItemFile.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
    this.menuItemExit});
this.menuItemFile.Text = "ファイル (&F)";

```

```

//
// menuItemExit
//
this.menuItemExit.Index = 0;
this.menuItemExit.Text = "終了 (&X)";
this.menuItemExit.Click += new System.EventHandler(this.menuItemExit_Click);
//
// menuItemSettingFolder
//
this.menuItemSettingFolder.Index = 1;
this.menuItemSettingFolder.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
    this.menuItemSetting});
this.menuItemSettingFolder.Text = "設定 (&S)";
//
// menuItemSetting
//
this.menuItemSetting.Index = 0;
this.menuItemSetting.Text = "設定 (&S)";
this.menuItemSetting.Click += new System.EventHandler(this.menuItemSetting_Click);
//
// panel1
//
this.panel1.Controls.Add(this.DrawTreeButton);
this.panel1.Controls.Add(this.comboBoxForProject);
this.panel1.Controls.Add(this.labelProject);
this.panel1.Dock = System.Windows.Forms.DockStyle.Top;
this.panel1.Location = new System.Drawing.Point(0, 0);
this.panel1.Name = "panel1";
this.panel1.Size = new System.Drawing.Size(792, 40);
this.panel1.TabIndex = 0;
//
// DrawTreeButton
//
this.DrawTreeButton.Location = new System.Drawing.Point(704, 8);
this.DrawTreeButton.Name = "DrawTreeButton";
this.DrawTreeButton.TabIndex = 2;
this.DrawTreeButton.Text = "ツリー表示";
this.DrawTreeButton.Click += new System.EventHandler(this.DrawTreeButton_Click);
//
// comboBoxForProject
//
this.comboBoxForProject.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDownList;
this.comboBoxForProject.Location = new System.Drawing.Point(112, 16);
this.comboBoxForProject.Name = "comboBoxForProject";
this.comboBoxForProject.Size = new System.Drawing.Size(250, 20);
this.comboBoxForProject.TabIndex = 1;
this.comboBoxForProject.SelectedIndexChanged += new System.EventHandler(this.comboBoxFor
Project_SelectedIndexChanged);
//
// labelProject
//
this.labelProject.Location = new System.Drawing.Point(16, 16);
this.labelProject.Name = "labelProject";

```

```

this.labelProject.Size = new System.Drawing.Size(64, 16);
this.labelProject.TabIndex = 0;
this.labelProject.Text = "プロジェクト";
//
// splitter1
//
this.splitter1.Dock = System.Windows.Forms.DockStyle.Top;
this.splitter1.Location = new System.Drawing.Point(0, 40);
this.splitter1.Name = "splitter1";
this.splitter1.Size = new System.Drawing.Size(792, 3);
this.splitter1.TabIndex = 1;
this.splitter1.TabStop = false;
//
// panel2
//
this.panel2.Controls.Add(this.comboBoxForMember1);
this.panel2.Controls.Add(this.labelMember1);
this.panel2.Dock = System.Windows.Forms.DockStyle.Top;
this.panel2.Location = new System.Drawing.Point(0, 43);
this.panel2.Name = "panel2";
this.panel2.Size = new System.Drawing.Size(792, 40);
this.panel2.TabIndex = 2;
//
// comboBoxForMember1
//
this.comboBoxForMember1.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDownList;
this.comboBoxForMember1.Enabled = false;
this.comboBoxForMember1.Location = new System.Drawing.Point(112, 8);
this.comboBoxForMember1.Name = "comboBoxForMember1";
this.comboBoxForMember1.Size = new System.Drawing.Size(121, 20);
this.comboBoxForMember1.TabIndex = 2;
this.comboBoxForMember1.SelectedIndexChanged += new System.EventHandler(this.comboBoxFor
Member1_SelectedIndexChanged);
//
// labelMember1
//
this.labelMember1.Location = new System.Drawing.Point(16, 16);
this.labelMember1.Name = "labelMember1";
this.labelMember1.Size = new System.Drawing.Size(96, 16);
this.labelMember1.TabIndex = 0;
this.labelMember1.Text = "プロジェクトメンバ 1";
//
// splitter2
//
this.splitter2.Dock = System.Windows.Forms.DockStyle.Top;
this.splitter2.Location = new System.Drawing.Point(0, 83);
this.splitter2.Name = "splitter2";
this.splitter2.Size = new System.Drawing.Size(792, 3);
this.splitter2.TabIndex = 3;
this.splitter2.TabStop = false;
//
// panel3
//

```

```

this.panel3.Controls.Add(this.treeListViewForMember1);
this.panel3.Controls.Add(this.splitter5);
this.panel3.Controls.Add(this.treeViewForMember1);
this.panel3.Dock = System.Windows.Forms.DockStyle.Top;
this.panel3.Location = new System.Drawing.Point(0, 86);
this.panel3.Name = "panel3";
this.panel3.Size = new System.Drawing.Size(792, 200);
this.panel3.TabIndex = 4;
//
// treeListViewForMember1
//
this.treeListViewForMember1.Columns.AddRange(new System.Windows.Forms.ColumnHeader[] {
    this.columnHeaderSubject1,
    this.columnHeaderFrom1,
    this.columnHeaderDate1,
    this.columnHeaderFilename1});
this.treeListViewForMember1.Dock = System.Windows.Forms.DockStyle.Fill;
this.treeListViewForMember1.Location = new System.Drawing.Point(153, 0);
this.treeListViewForMember1.Name = "treeListViewForMember1";
this.treeListViewForMember1.Size = new System.Drawing.Size(639, 200);
this.treeListViewForMember1.TabIndex = 2;
//
// columnHeaderSubject1
//
this.columnHeaderSubject1.Text = "件名";
this.columnHeaderSubject1.Width = 235;
//
// columnHeaderFrom1
//
this.columnHeaderFrom1.Text = "送信者";
this.columnHeaderFrom1.Width = 200;
//
// columnHeaderDate1
//
this.columnHeaderDate1.Text = "日付";
this.columnHeaderDate1.Width = 100;
//
// columnHeaderFilename1
//
this.columnHeaderFilename1.Text = "ファイル名";
this.columnHeaderFilename1.Width = 100;
//
// splitter5
//
this.splitter5.Location = new System.Drawing.Point(150, 0);
this.splitter5.Name = "splitter5";
this.splitter5.Size = new System.Drawing.Size(3, 200);
this.splitter5.TabIndex = 1;
this.splitter5.TabStop = false;
//
// treeViewForMember1
//
this.treeViewForMember1.Dock = System.Windows.Forms.DockStyle.Left;

```

```

this.treeViewForMember1.ImageList = this.imageListFolder;
this.treeViewForMember1.Location = new System.Drawing.Point(0, 0);
this.treeViewForMember1.Name = "treeViewForMember1";
this.treeViewForMember1.Size = new System.Drawing.Size(150, 200);
this.treeViewForMember1.TabIndex = 0;
this.treeViewForMember1.AfterSelect += new System.Windows.Forms.TreeViewEventHandler(this.s.treeViewForMember1_AfterSelect);
//
// imageListFolder
//
this.imageListFolder.ImageSize = new System.Drawing.Size(16, 16);
this.imageListFolder.ImageStream = ((System.Windows.Forms.ImageListStreamer)(resources.GetObject("imageListFolder.ImageStream")));
this.imageListFolder.TransparentColor = System.Drawing.Color.Transparent;
//
// splitter3
//
this.splitter3.Dock = System.Windows.Forms.DockStyle.Top;
this.splitter3.Location = new System.Drawing.Point(0, 286);
this.splitter3.Name = "splitter3";
this.splitter3.Size = new System.Drawing.Size(792, 3);
this.splitter3.TabIndex = 5;
this.splitter3.TabStop = false;
//
// panel4
//
this.panel4.Controls.Add(this.comboBoxForMember2);
this.panel4.Controls.Add(this.labelMember2);
this.panel4.Dock = System.Windows.Forms.DockStyle.Top;
this.panel4.Location = new System.Drawing.Point(0, 289);
this.panel4.Name = "panel4";
this.panel4.Size = new System.Drawing.Size(792, 40);
this.panel4.TabIndex = 6;
//
// comboBoxForMember2
//
this.comboBoxForMember2.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDownList;
this.comboBoxForMember2.Enabled = false;
this.comboBoxForMember2.Location = new System.Drawing.Point(112, 8);
this.comboBoxForMember2.Name = "comboBoxForMember2";
this.comboBoxForMember2.Size = new System.Drawing.Size(121, 20);
this.comboBoxForMember2.TabIndex = 2;
this.comboBoxForMember2.SelectedIndexChanged += new System.EventHandler(this.comboBoxForMember2_SelectedIndexChanged);
//
// labelMember2
//
this.labelMember2.Location = new System.Drawing.Point(16, 16);
this.labelMember2.Name = "labelMember2";
this.labelMember2.Size = new System.Drawing.Size(96, 16);
this.labelMember2.TabIndex = 0;
this.labelMember2.Text = "プロジェクトメンバー 2";
//

```

```

// splitter4
//
this.splitter4.Dock = System.Windows.Forms.DockStyle.Top;
this.splitter4.Location = new System.Drawing.Point(0, 329);
this.splitter4.Name = "splitter4";
this.splitter4.Size = new System.Drawing.Size(792, 3);
this.splitter4.TabIndex = 7;
this.splitter4.TabStop = false;
//
// statusBar
//
this.statusBar.Location = new System.Drawing.Point(0, 523);
this.statusBar.Name = "statusBar";
this.statusBar.Size = new System.Drawing.Size(792, 22);
this.statusBar.TabIndex = 8;
//
// panel5
//
this.panel5.Controls.Add(this.treeListViewForMember2);
this.panel5.Controls.Add(this.splitter6);
this.panel5.Controls.Add(this.treeViewForMember2);
this.panel5.Dock = System.Windows.Forms.DockStyle.Fill;
this.panel5.Location = new System.Drawing.Point(0, 332);
this.panel5.Name = "panel5";
this.panel5.Size = new System.Drawing.Size(792, 191);
this.panel5.TabIndex = 9;
//
// treeListViewForMember2
//
this.treeListViewForMember2.Columns.AddRange(new System.Windows.Forms.ColumnHeader[] {
    this.columnHeaderSubject2,
    this.columnHeaderFrom2,
    this.columnHeaderDate2,
    this.columnHeaderFilename2});
this.treeListViewForMember2.Dock = System.Windows.Forms.DockStyle.Fill;
this.treeListViewForMember2.Location = new System.Drawing.Point(153, 0);
this.treeListViewForMember2.Name = "treeListViewForMember2";
this.treeListViewForMember2.Size = new System.Drawing.Size(639, 191);
this.treeListViewForMember2.TabIndex = 3;
//
// columnHeaderSubject2
//
this.columnHeaderSubject2.Text = "件名";
this.columnHeaderSubject2.Width = 235;
//
// columnHeaderFrom2
//
this.columnHeaderFrom2.Text = "送信者";
this.columnHeaderFrom2.Width = 200;
//
// columnHeaderDate2
//
this.columnHeaderDate2.Text = "日付";

```

```

this.columnHeaderDate2.Width = 100;
//
// columnHeaderFilename2
//
this.columnHeaderFilename2.Text = "ファイル名";
this.columnHeaderFilename2.Width = 100;
//
// splitter6
//
this.splitter6.Location = new System.Drawing.Point(150, 0);
this.splitter6.Name = "splitter6";
this.splitter6.Size = new System.Drawing.Size(3, 191);
this.splitter6.TabIndex = 2;
this.splitter6.TabStop = false;
//
// treeViewForMember2
//
this.treeViewForMember2.Dock = System.Windows.Forms.DockStyle.Left;
this.treeViewForMember2.ImageList = this.imageListFolder;
this.treeViewForMember2.Location = new System.Drawing.Point(0, 0);
this.treeViewForMember2.Name = "treeViewForMember2";
this.treeViewForMember2.Size = new System.Drawing.Size(150, 191);
this.treeViewForMember2.TabIndex = 1;
this.treeViewForMember2.AfterSelect += new System.Windows.Forms.TreeViewEventHandler(this.s.treeViewForMember2_AfterSelect);
//
// MainForm
//
this.AutoScaleBaseSize = new System.Drawing.Size(5, 12);
this.ClientSize = new System.Drawing.Size(792, 545);
this.Controls.Add(this.panel5);
this.Controls.Add(this.statusBar);
this.Controls.Add(this.splitter4);
this.Controls.Add(this.panel4);
this.Controls.Add(this.splitter3);
this.Controls.Add(this.panel3);
this.Controls.Add(this.splitter2);
this.Controls.Add(this.panel2);
this.Controls.Add(this.splitter1);
this.Controls.Add(this.panel1);
this.Menu = this.mainMenu;
this.Name = "MainForm";
this.Text = "Semantic Bridge";
this.Load += new System.EventHandler(this.MainForm_Load);
this.panel1.ResumeLayout(false);
this.panel2.ResumeLayout(false);
this.panel3.ResumeLayout(false);
this.panel4.ResumeLayout(false);
this.panel5.ResumeLayout(false);
this.ResumeLayout(false);
}
#endregion

```

```

/// <summary>
/// アプリケーションのメイン エントリ ポイントです。
/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new MainForm());
}

/// <summary>
/// 終了メニューが選択された場合の処理
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuItemExit_Click(object sender, System.EventArgs e)
{
    // プログラムを終了する
    this.Close();
}

/// <summary>
/// 選ばれたプロジェクトに属するプロジェクトメンバを取得
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void comboBoxForProject_SelectedIndexChanged(object sender, System.EventArgs e)
{
    // 取得終了まで操作を禁止
    this.comboBoxForProject.Enabled = false;
    // マウスカーソルを砂時計に変更
    Cursor.Current = Cursors.WaitCursor;
    // AB サーバに接続
    ABRDFProxy proxy = getProxy();
    // AB サーバから全てのプロジェクトメンバを取得
    PrAgent[] agents = proxy.GetAllAgentByProject(projectURI[this.comboBoxForProject.SelectedIndex].ToString());
    // AB サーバから切断
    proxy.Logout();
    // 表示されているものをクリア
    ClearItems();
    // エージェントリストをクリア
    agentURI.Clear();
    agentTable.Clear();
    foreach(PrAgent agent in agents)
    {
        // ドロップダウンリストに追加
        this.comboBoxForMember1.Items.Add(agent.Fullname);
        this.comboBoxForMember2.Items.Add(agent.Fullname);
        // エージェントリストに追加
        agentURI.Add(agent.Resource);
        agentTable.Add(agent.Resource, agent.Fullname);
    }
}

```

```

// 操作可能に
this.comboBoxForProject.Enabled = true;
// プロジェクトメンバを選択可能に
this.comboBoxForMember1.Enabled = true;
this.comboBoxForMember2.Enabled = true;
// マウスカーソルを元に戻す
Cursor.Current = Cursors.Default;
}

/// <summary>
/// プロジェクトメンバ 1 が作った属性を取得
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void comboBoxForMember1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    comboBoxForMember_SelectedIndexChanged(this.comboBoxForMember1, this.comboBoxForMember2,
this.treeViewForMember1, 1);
}

/// <summary>
/// プロジェクトメンバ 2 が作った属性を取得
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void comboBoxForMember2_SelectedIndexChanged(object sender, System.EventArgs e)
{
    comboBoxForMember_SelectedIndexChanged(this.comboBoxForMember2, this.comboBoxForMember1,
this.treeViewForMember2, 2);
}

/// <summary>
/// 選んだ属性に属するコンテンツを表示 (プロジェクトメンバ 1 用)
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void treeViewForMember1_AfterSelect(object sender, System.Windows.Forms.TreeViewEv
entArgs e)
{
    treeViewForMember_AfterSelect(this.treeViewForMember1, this.treeViewForMember2, this.tre
eListViewForMember1, this.comboBoxForMember2, 1);
}

/// <summary>
/// 選んだ属性に属するコンテンツを表示 (プロジェクトメンバ 2 用)
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void treeViewForMember2_AfterSelect(object sender, System.Windows.Forms.TreeViewEv
entArgs e)
{
    treeViewForMember_AfterSelect(this.treeViewForMember2, this.treeViewForMember1, this.tre
eListViewForMember2, this.comboBoxForMember1, 2);
}

```

```

}

/// <summary>
/// ツリー表示ボタンが押されたときの処理
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void DrawTreeButton_Click(object sender, System.EventArgs e)
{
    // 2人分のビューが表示されている場合のみ描画
    if(this.comboBoxForMember1.SelectedItem!=null && this.comboBoxForMember2.SelectedItem!=n
ull)
    {
        // マウスカーソルを砂時計に変更
        Cursor.Current = Cursors.WaitCursor;
        Point pt = new Point();
        // 描画のための変数 elem および MaxDepth を初期化
        elem = 0;
        MaxDepth = 1;
        int depth = MaxDepth;
        // 描画位置の初期化
        InitDrawPosition(this.treeViewForMember1.TopNode);
        InitDrawPosition(this.treeViewForMember2.TopNode);
        // プロジェクトメンバ1のツリーノードの描画位置を計算
        pt = DecideDrawPosition(this.treeViewForMember1.TopNode, depth);
        ((AttributeItem)this.treeViewForMember1.TopNode.Tag).x = pt.X;
        ((AttributeItem)this.treeViewForMember1.TopNode.Tag).y = pt.Y;
        // 類似属性は重ねて配置
        DecideDrawPositionOfSimilarNode(this.treeViewForMember1.TopNode);
        // プロジェクトメンバ2のツリーノードの描画位置を計算
        pt = DecideDrawPosition(this.treeViewForMember2.TopNode, depth);
        ((AttributeItem)this.treeViewForMember2.TopNode.Tag).x = pt.X;
        ((AttributeItem)this.treeViewForMember2.TopNode.Tag).y = pt.Y;
        // プロジェクトメンバ2のツリーで親ノードの上に子ノードが来ないようにする
        CheckTree(this.treeViewForMember2.TopNode);
        // マウスカーソルを元に戻す
        Cursor.Current = Cursors.Default;
        // 全てのノードが描画されるだけの大きさを持つ描画ウィンドウを用意
        TreeForm treeForm = new TreeForm(this.treeViewForMember1, this.treeViewForMember2, (el
em+1)*c.Distance, (MaxDepth+1)*c.Distance);
        // 描画ウィンドウを表示
        treeForm.Show();
    }
    else
    {
        MessageBox.Show("2人分のビューを表示した後で、このボタンを押してください。", "エラー",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

/// <summary>
/// 設定メニューがクリックされたときの処理
/// </summary>

```

```

/// <param name="sender"></param>
/// <param name="e"></param>
private void menuItemSetting_Click(object sender, System.EventArgs e)
{
    // 設定ダイアログを用意し、開く
    ConfigForm configForm = new ConfigForm();
    DialogResult result = configForm.ShowDialog(this);
    // 設定が保存されれば
    if(result==DialogResult.OK)
    {
        // 設定内容の反映
        MainForm_Load(null, null);
    }
}

/// <summary>
/// ロード時の処理 (初期設定)
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MainForm_Load(object sender, System.EventArgs e)
{
    // 設定読み込み
    ConfigForm configForm = new ConfigForm();
    c = configForm.Config;
    // AB サーバに接続
    ABRDFProxy proxy = getProxy();
    // proxy の取得に失敗
    if(proxy==null)
    {
        // エラーメッセージを表示し、設定を促す
        MessageBox.Show("RDF データベースにログインできません。 \n 設定を行ってください。", "エラー", MessageBoxButtons.OK, MessageBoxIcon.Error);
        DialogResult result = configForm.ShowDialog(this);
        if(result==DialogResult.OK)
        {
            // 設定内容を反映
            MainForm_Load(sender, e);
        }
        if(result==DialogResult.Cancel)
        {
            // 設定が正しく行えなかった
            MessageBox.Show("RDF データベースの設定が行えませんでした。 \n 設定ウィンドウより再設定してください。", "エラー", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    // proxy の取得に成功
    else
    {
        // AB サーバからプロジェクトを取得
        PrProject[] projects = proxy.GetAllProjects();
        // AB サーバから切断
        proxy.Logout();
    }
}

```

```

// 表示されているものを全てクリア
this.comboBoxForProject.Items.Clear();
ClearItems();
// プロジェクトリストをクリア
projectURI.Clear();
// プロジェクトメンバを選択不可能に
this.comboBoxForMember1.Enabled = false;
this.comboBoxForMember2.Enabled = false;
foreach(PrProject project in projects)
{
    // ドロップダウンリストに追加
    this.comboBoxForProject.Items.Add(project.Name);
    // プロジェクトリストに追加
    this.projectURI.Add(project.URI);
}
}
}

/// <summary>
/// 属性ノードの描画位置を計算
/// </summary>
/// <param name="node">描画位置を求める属性ノード</param>
/// <param name="depth">属性ノードの深さ</param>
/// <returns>属性ノードの座標 (x, y)</returns>
private Point DecideDrawPosition(TreeNode node, int depth)
{
    Point pt = new Point();
    int newdepth = depth + 1;
    // 描画位置が決まっていない場合のみ、処理を実行
    if(((AttributeItem)node.Tag).x==0)
    {
        // y 座標を決定
        pt.Y = depth * c.Distance;
        // 深さの最大値を記憶
        if(depth>MaxDepth)
        {
            MaxDepth = depth;
        }
        // 子ノードが存在しない場合
        if(node.Nodes.Count==0)
        {
            // そのノードの描画領域を確保し、
            elem++;
            // その x 座標を返す
            pt.X = elem * c.Distance;
            return pt;
        }
        // 子ノードが存在する場合
        else
        {
            int temp = 0;
            // 子ノードそれぞれに対して
            foreach(TreeNode tn in node.Nodes)

```

```

    {
        Point pt2 = new Point();
        // 子ノードの座標を求める
        pt2 = DecideDrawPosition(tn, newdepth);
        ((AttributeItem)tn.Tag).x = pt2.X;
        ((AttributeItem)tn.Tag).y = pt2.Y;
        temp += ((AttributeItem)tn.Tag).x;
    }
    // 子ノードの x 座標の平均値を描画位置として返す
    pt.X = temp / node.Nodes.Count;
    return pt;
}
}
else
{
    // 描画位置が決まっている場合は、その子ノードの処理へ移る
    foreach(TreeNode tn in node.Nodes)
    {
        Point pt2 = new Point();
        pt2 = DecideDrawPosition(tn, newdepth);
        ((AttributeItem)tn.Tag).x = pt2.X;
        ((AttributeItem)tn.Tag).y = pt2.Y;
    }
    // 自分自身を返す
    pt.X = ((AttributeItem)node.Tag).x;
    pt.Y = ((AttributeItem)node.Tag).y;
    return pt;
}
}

/// <summary>
/// node で示される属性に含まれる全てのコンテンツを取得
/// 下位属性のコンテンツは上位属性のコンテンツでもある
/// </summary>
/// <param name="node">属性ノード</param>
/// <returns>コンテンツの配列</returns>
private PrContent[] GetAllContentBelongsToAttr(TreeNode node)
{
    ArrayList al = new ArrayList();
    AttributeItem item = (AttributeItem)node.Tag;
    // 直結するコンテンツを取得
    foreach(PrContent ptemp in item.Contents)
    {
        al.Add(ptemp);
    }
    // 下位属性のコンテンツを取得
    foreach(TreeNode tn in node.Nodes)
    {
        PrContent[] pctemps = GetAllContentBelongsToAttr(tn);
        foreach(PrContent ptemp in pctemps)
        {
            al.Add(ptemp);
        }
    }
}

```

```

    }
    // その合計が所属コンテンツ全て
    PrContent[] ret = (PrContent[])a1.ToArray(typeof(PrContent));
    return ret;
}

/// <summary>
/// tn に対応する属性に含まれるコンテンツを全てハッシュテーブルに入れる
/// </summary>
/// <param name="tn">着目しているツリーノード</param>
/// <param name="i">プロジェクトメンバ番号</param>
private void MakeHashtable(TreeNode tn, int i)
{
    // 直結するコンテンツ
    AddHashtable(tn, i);
    foreach(TreeNode tn2 in tn.Nodes)
    {
        // 下位属性に含まれるコンテンツ
        MakeHashtable(tn2, i);
    }
}

/// <summary>
/// tn に対応する属性を付加されたコンテンツを全てハッシュテーブルに入れる
/// </summary>
/// <param name="tn">着目しているツリーノード</param>
/// <param name="i">プロジェクトメンバ番号</param>
private void AddHashtable(TreeNode tn, int i)
{
    AttributeItem aitem = (AttributeItem)tn.Tag;
    Hashtable temp = new Hashtable();
    PrContent[] pcs = GetAllContentBelongsToAttr(tn);
    // コンテンツを追加
    foreach(PrContent pc in pcs)
    {
        try
        {
            temp.Add(pc.Resource, true);
        }
        catch(System.ArgumentException)
        {
            // すでに追加されている場合は何もしない
        }
    }
    try
    {
        // ハッシュテーブルに追加
        // プロジェクトメンバ1 のとき
        Hashtable content = new Hashtable();
        if(i==1)
        {
            mem1.Add(aitem.Resource, temp);
            content = (Hashtable)mem1[aitem.Resource];
        }
    }
}

```

```

    }
    // プロジェクトメンバ 2 のとき
    else if(i==2)
    {
        mem2.Add(aitem.Resource, temp);
        content = (Hashtable)mem2[aitem.Resource];
    }
    // tn にコンテンツの個数を示す文字列を付加
    tn.Text += " (" + content.Count.ToString() + ")";
}
catch(System.ArgumentException)
{
    // すでに追加されている場合は、何もしない
}
}

/// <summary>
/// AB サーバに接続する
/// </summary>
/// <returns>proxy</returns>
private ABRDFProxy getProxy()
{
    // AB サーバにログイン
    ABRDFProxy proxy;
    try
    {
        proxy = new ABRDFProxy(c.RDFDB, c.UserName, c.Password);
        proxy.Login();
    }
    catch(Exception)
    {
        // ログインに失敗
        return null;
    }
    return proxy;
}

/// <summary>
/// プロジェクトメンバの属性ビューの初期化
/// </summary>
/// <param name="cb">tv に対応するコンボボックス</param>
/// <param name="tv">初期化する属性ビュー</param>
/// <param name="attrs">属性一覧</param>
private void InitTreeView(ComboBox cb, TreeView tv, AttrAttribute[] attrs)
{
    // まずは属性ビューを綺麗にする
    tv.Nodes.Clear();
    // ルートノードの追加
    TreeNode node = new TreeNode(cb.SelectedItem.ToString() + " のコンテンツ分類");
    node.Tag = new AttributeItem();
    tv.Nodes.Add(node);
    // 属性の更新
    // 属性ノードをすべて消して

```

```

tv.Nodes[0].Nodes.Clear();
int i = 0;
// 属性 1 つ 1 つに対して
foreach(AttrAttribute attr in attrs)
{
    // 最上位属性ならば
    if(attr.ParentAttribute==null)
    {
        TreeNode tn = CreateNode(attr);
        // 作ったノードを追加
        tv.Nodes[0].Nodes.Add(tn);
        // 下位属性も追加する
        UpdateTreeViewForMember(tv.Nodes[0].Nodes[i], attr);
        i++;
    }
}
// すべて展開する
tv.ExpandAll();
}

/// <summary>
/// 親ノードと親アイテムを与えて、プロジェクトメンバの属性ビューを更新。
/// 再帰的に呼び出される。
/// </summary>
/// <param name="node">親ノード</param>
/// <param name="attr">親アイテム</param>
private void UpdateTreeViewForMember(TreeNode node, AttrAttribute attr)
{
    // 親アイテムの子アイテムに対して
    foreach(AttrAttribute childattr in attr.SubAttributes)
    {
        TreeNode tn = CreateNode(childattr);
        // 親ノードに子ノード追加
        node.Nodes.Add(tn);
        // 現在のノードに対して再帰的に呼び出し
        UpdateTreeViewForMember(tn, childattr);
    }
}

/// <summary>
/// 属性ノードを生成する
/// </summary>
/// <param name="attr">属性データ</param>
/// <returns>属性ノード</returns>
private TreeNode CreateNode(AttrAttribute attr)
{
    // AB サーバに接続
    ABRDFProxy proxy = getProxy();
    // ノードを形成
    AttributeItem item = new AttributeItem();
    item.Resource = attr.Resource;
    item.Contents = proxy.GetAllContentByAttribute(attr.Resource);
    // AB サーバから切断

```

```

proxy.Logout();
TreeNode tn = new TreeNode(attr.Name);
tn.Tag = item;
tn.ImageIndex = 1;
tn.SelectedImageIndex = 1;
return tn;
}

/// <summary>
/// ツリーノードの色を再帰的にリセットする
/// </summary>
/// <param name="node">色をリセットするツリーノード</param>
private void ResetTreeColor(TreeNode node)
{
    // 色をリセット
    node.ForeColor = SystemColors.WindowText;
    foreach(TreeNode tn in node.Nodes)
    {
        // 子ノードに対して再帰的に実行
        ResetTreeColor(tn);
    }
}

/// <summary>
/// プロジェクトメンバ 2 のツリーで、親ノードの上に子ノードが来ないように描画位置を変更
/// </summary>
/// <param name="node2">プロジェクトメンバ 2 のツリーノード</param>
private void CheckTree(TreeNode node2)
{
    // 子ノードを持つ場合
    if(node2.Nodes.Count!=0)
    {
        // その子ノードそれぞれに対して
        foreach(TreeNode tn2 in node2.Nodes)
        {
            AttributeItem parent = (AttributeItem)node2.Tag;
            AttributeItem child = (AttributeItem)tn2.Tag;
            // 親ノードの描画位置と子ノードの描画位置が逆転している場合
            if(parent.y>=child.y)
            {
                // その差を求め
                int diffdepth = (parent.y-child.y) / c.Distance + 1;
                // 描画位置を修正
                child.y = parent.y + c.Distance;
                // 描画領域が大きくなるので、あふれないように
                if(child.y/c.Distance>MaxDepth)
                {
                    MaxDepth = child.y / c.Distance;
                }
                // プロジェクトメンバ 1 に対応する属性の描画位置をずらす
                ModifyTree(tn2, diffdepth);
            }
            // 子ノードの子ノードに対して再帰的に実行

```

```

        CheckTree(tn2);
    }
}

/// <summary>
/// node2 に対応するノード (プロジェクトメンバ 1 のツリー中) 以下の描画位置を修正
/// </summary>
/// <param name="node2">プロジェクトメンバ 2 のツリーノード</param>
/// <param name="diffdepth">どれだけ描画位置をずらすかを示す値</param>
private void ModifyTree(TreeNode node2, int diffdepth)
{
    // 類似属性があれば
    if(result2[((AttributeItem)node2.Tag).Resource]!=null)
    {
        string key = ((Relation)result2[((AttributeItem)node2.Tag).Resource]).Node1;
        // 対応する属性ノードを探し出し
        TreeNode node = SearchNode(key, this.treeViewForMember1.TopNode);
        // そのノード以下の描画位置を修正
        ModifyTreeMain(node, diffdepth);
    }
}

/// <summary>
/// node 以下の属性ノードの描画位置を修正
/// </summary>
/// <param name="node">修正対象のノード</param>
/// <param name="diffdepth">どれだけずらすかを示す値</param>
private void ModifyTreeMain(TreeNode node, int diffdepth)
{
    // 当該ノードの描画位置をずらす
    ((AttributeItem)node.Tag).y += diffdepth * c.Distance;
    // 描画領域が大きくなってもあふれないように
    if(((AttributeItem)node.Tag).y/c.Distance>MaxDepth)
    {
        MaxDepth = ((AttributeItem)node.Tag).y / c.Distance;
    }
    // 子ノードも再帰的にずらす
    foreach(TreeNode tn in node.Nodes)
    {
        ModifyTreeMain(tn, diffdepth);
    }
}

/// <summary>
/// key に対応するツリーノードを検索
/// </summary>
/// <param name="key">ツリーノードを検索するためのキー</param>
/// <param name="node">検索対象のツリーノード集合</param>
/// <returns>該当するツリーノード</returns>
private TreeNode SearchNode(string key, TreeNode node)
{
    // 該当のものが見つければ、それを返す

```

```

        if(((AttributeItem)node.Tag).Resource==key)
        {
            return node;
        }
        else
        {
            // 子ノード以下を検索し、該当のものを求めようとする
            foreach(TreeNode tn in node.Nodes)
            {
                if(SearchNode(key, tn)!=null)
                {
                    return SearchNode(key, tn);
                }
            }
            // 何も見つからないとき
            return null;
        }
    }
}

/// <summary>
/// 属性ノードの描画位置を再帰的に初期化する
/// </summary>
/// <param name="node">初期化する属性ノード</param>
private void InitDrawPosition(TreeNode node)
{
    // 初期化
    ((AttributeItem)node.Tag).x = 0;
    ((AttributeItem)node.Tag).y = 0;
    // 再帰的に実行
    foreach(TreeNode tn in node.Nodes)
    {
        InitDrawPosition(tn);
    }
}

/// <summary>
/// 類似属性は重なって表示されるようにプロジェクトメンバ2のツリーノードの描画位置を決定
/// </summary>
/// <param name="node">プロジェクトメンバ1のツリーノード</param>
private void DecideDrawPositionOfSimilarNode(TreeNode node)
{
    // nodeの類似属性を近くに描画させるようにする
    if(node!=this.treeViewForMember1.TopNode)
    {
        string item = ((AttributeItem)node.Tag).Resource;
        // 類似属性があるとき
        if(result1[item]!=null)
        {
            // そのノードを探す
            TreeNode tn = SearchNode(((Relation)result1[item]).Node2, this.treeViewForMember2.To
pNode);
            // 互いに重なるように描画位置を決定
            ((AttributeItem)tn.Tag).x = ((AttributeItem)node.Tag).x;

```

```

        ((AttributeItem)tn.Tag).y = ((AttributeItem)node.Tag).y;
    }
}
// node の子ノードも再帰的に計算
foreach(TreeNode tn in node.Nodes)
{
    DecideDrawPositionOfSimilarNode(tn);
}
}

/// <summary>
/// 類似属性検出アルゴリズム
/// </summary>
private void Algorithm()
{
    // 候補を格納するテーブルを初期化し
    candidate.Clear();
    // 候補を生成
    Join(this.treeViewForMember1.TopNode);
    // 解を選択
    Prune();
}

/// <summary>
/// 類似属性の候補生成プロセス
/// </summary>
/// <param name="node">プロジェクトメンバ 1 のツリーノード</param>
private void Join(TreeNode node)
{
    // node に対応する類似属性を探す
    MakeCandidate(node, this.treeViewForMember2.TopNode);
    // node の子ノードに対して再帰的に実行
    foreach(TreeNode tn in node.Nodes)
    {
        Join(tn);
    }
}

/// <summary>
/// node1 に対応する類似属性を探す
/// </summary>
/// <param name="node1">類似属性を求めたいノード</param>
/// <param name="node2">検索対象</param>
private void MakeCandidate(TreeNode node1, TreeNode node2)
{
    double sim;
    // 属性ノードを取得
    AttributeItem item1 = (AttributeItem)node1.Tag;
    AttributeItem item2 = (AttributeItem)node2.Tag;
    // 属性を付加したコンテンツの集合を取得
    Hashtable contents1 = (Hashtable)mem1[item1.Resource];
    Hashtable contents2 = (Hashtable)mem2[item2.Resource];
    // 両方とも中身があれば

```

```

if(contents1!=null && contents2!=null)
{
// 用いる類似度尺度によって場合分け
// Jaccard 係数
if(c.Type==Constant.JACCARD)
{
// 枝刈り
// 類似度の上限が閾値以上の場合のみ検索する意味がある
if(c.Sim*contents1.Count<=contents2.Count)
{
// 実際の類似度を求める
sim = Jaccard(contents1, contents2);
MakeCandidateMain(sim, node1, node2);
}
}
// Simpson 係数
else if(c.Type==Constant.SIMPSON)
{
// 枝刈り
// 信頼性の点で劣るので、コンテンツ数の少ない属性については考えない
if(contents2.Count>Constant.THRESHOLD)
{
sim = Simpson(contents1, contents2);
MakeCandidateMain(sim, node1, node2);
}
}
// Dice 係数
else if(c.Type==Constant.DICE)
{
// 枝刈り
if((c.Sim/(2-c.Sim))*contents1.Count<=contents2.Count)
{
sim = Dice(contents1, contents2);
MakeCandidateMain(sim, node1, node2);
}
}
// Cosine 係数
else if (c.Type==Constant.COSINE)
{
// 枝刈り
if(c.Sim*c.Sim*contents1.Count<=contents2.Count)
{
sim = Cosine(contents1, contents2);
MakeCandidateMain(sim, node1, node2);
}
}
}
else if(contents1!=null)
{
// <プロジェクトメンバ 2>のコンテンツ分類というノードの子ノードを検索
foreach(TreeNode tn in node2.Nodes)
{
MakeCandidate(node1, tn);
}
}
}

```

```

    }
  }
}

/// <summary>
/// 類似属性の組の候補を作成
/// </summary>
/// <param name="sim">類似度</param>
/// <param name="node1">ある概念</param>
/// <param name="node2">node1 に対応する概念</param>
private void MakeCandidateMain(double sim, TreeNode node1, TreeNode node2)
{
    // 属性ノードを取得
    AttributeItem item1 = (AttributeItem)node1.Tag;
    AttributeItem item2 = (AttributeItem)node2.Tag;
    // 類似度が閾値以上であれば
    if(sim>=c.Sim)
    {
        // 候補オブジェクトを生成
        Relation r = new Relation();
        r.Node1 = item1.Resource;
        r.Node2 = item2.Resource;
        r.Sim = sim;
        // node1 に対応する属性が見つからない場合は無条件に候補となる
        if(candidate[r.Node1]==null)
        {
            candidate.Add(r.Node1, r);
        }
        // すでに見ついている場合は、それより類似といえれば置き換える
        else if(((Relation)candidate[r.Node1]).Sim<=sim)
        {
            candidate[r.Node1] = r;
        }
    }
    // node2 の子ノードに対して再帰的に実行
    foreach(TreeNode tn in node2.Nodes)
    {
        MakeCandidate(node1, tn);
    }
}

/// <summary>
/// 2つのコンテンツ集合間の Jaccard 係数の計算
/// </summary>
/// <param name="contents1">コンテンツ集合 1</param>
/// <param name="contents2">コンテンツ集合 2</param>
/// <returns>Jaccard 係数</returns>
private double Jaccard(Hashtable contents1, Hashtable contents2)
{
    Hashtable temp = UnionAndIntersection(contents1, contents2);
    // 和集合の個数が 0 でなければ類似度を定義できる
    if((int)temp["union"]!=0)
    {

```

```

        return (double)(int)temp["intersection"] / (double)(int)temp["union"];
    }
    // そうでなければ類似度は定義できない(仮に 0 を返す)
    else
    {
        return 0;
    }
}

/// <summary>
/// 2つのコンテンツ集合間の Simpson 係数の計算
/// </summary>
/// <param name="contents1">コンテンツ集合 1</param>
/// <param name="contents2">コンテンツ集合 2</param>
/// <returns>Simpson 係数</returns>
private double Simpson(Hashtable contents1, Hashtable contents2)
{
    Hashtable temp = UnionAndIntersection(contents1, contents2);
    // Simpson 係数の分母に相当
    int min;
    if(contents1.Count<contents2.Count)
    {
        min = contents1.Count;
    }
    else
    {
        min = contents2.Count;
    }
    // 0 でなければ Simpson 係数を定義できる
    if(min!=0)
    {
        return (double)(int)temp["intersection"] / (double)min;
    }
    // そうでなければ類似度は定義できない(仮に 0 を返す)
    else
    {
        return 0;
    }
}

/// <summary>
/// 2つのコンテンツ集合間の Dice 係数の計算
/// </summary>
/// <param name="contents1">コンテンツ集合 1</param>
/// <param name="contents2">コンテンツ集合 2</param>
/// <returns>Dice 係数</returns>
private double Dice(Hashtable contents1, Hashtable contents2)
{
    Hashtable temp = UnionAndIntersection(contents1, contents2);
    // 0 でなければ Dice 係数を定義できる
    if(contents1.Count+contents2.Count!=0)
    {
        return (double)(2*(int)temp["intersection"]) / (double)(contents1.Count+contents2.Coun

```

```

t);
    }
    // そうでなければ類似度は定義できない(仮に 0 を返す)
    else
    {
        return 0;
    }
}

/// <summary>
/// 2つのコンテンツ集合間の Cosine 係数を計算
/// </summary>
/// <param name="contents1">コンテンツ集合 1</param>
/// <param name="contents2">コンテンツ集合 2</param>
/// <returns>Cosine 係数</returns>
private double Cosine(Hashtable contents1, Hashtable contents2)
{
    Hashtable temp = UnionAndIntersection(contents1, contents2);
    // 平方根の中身が 0 でなければ類似度を定義できる
    if(contents1.Count*contents2.Count!=0)
    {
        return (double)(int)temp["intersection"] / Math.Sqrt(contents1.Count*contents2.Count);
    }
    // そうでなければ類似度は定義できない(仮に 0 を返す)
    else
    {
        return 0;
    }
}

/// <summary>
/// 2つのコンテンツ集合の和集合と積集合の要素数を返す
/// </summary>
/// <param name="contents1">コンテンツ集合 1</param>
/// <param name="contents2">コンテンツ集合 2</param>
/// <returns>和集合の要素数と積集合の要素数を格納したハッシュテーブル</returns>
private Hashtable UnionAndIntersection(Hashtable contents1, Hashtable contents2)
{
    Hashtable ret = new Hashtable();
    Hashtable temp = new Hashtable();
    int i = 0;
    // 2つの集合の和集合の個数を計算
    foreach(DictionaryEntry e1 in contents1)
    {
        try
        {
            temp.Add(e1.Key, true);
        }
        catch(System.ArgumentException)
        {
        }
    }
    foreach(DictionaryEntry e2 in contents2)

```

```

    {
        try
        {
            temp.Add(e2.Key, true);
        }
        catch(System.ArgumentException)
        {
            // 2つの集合の積集合の個数に相当
            i++;
        }
    }
    // 要素数をハッシュテーブルに格納し、返す
    ret.Add("union", temp.Count);
    ret.Add("intersection", i);
    return ret;
}

/// <summary>
/// 類似属性候補のうち、実際に類似属性となるものを選択
/// </summary>
private void Prune()
{
    // 解を格納するハッシュテーブルを初期化
    result1.Clear();
    result2.Clear();
    // 全ての候補を検査
    foreach(DictionaryEntry e in candidate)
    {
        Relation cand = (Relation)e.Value;
        // Node2 から見て最も類似度の高い属性のみ解とする
        if(result2[cand.Node2]==null)
        {
            result2.Add(cand.Node2, cand);
        }
        else if(((Relation)result2[cand.Node2]).Sim<cand.Sim)
        {
            result2[cand.Node2] = cand;
        }
    }
    // 検索キーを変えたコピーを用意
    foreach(DictionaryEntry e in result2)
    {
        result1.Add(((Relation)e.Value).Node1, (Relation)e.Value);
    }
}

/// <summary>
/// ツリーリストビューに表示させるアイテムを生成
/// </summary>
/// <param name="content">アイテムに対応するコンテンツデータ</param>
/// <param name="proxy">プロキシ</param>
/// <returns>アイテム</returns>
private TreeListViewItem MakeTreeListViewItem(PrContent content, ABRDFProxy proxy)

```

```

{
    string tid = proxy.BeginTransaction();
    TreeListViewItem item = new TreeListViewItem();
    // コンテンツ名
    item.Text = content.Name;
    // コンテンツ作成者
    if(agentTable[content.Creator]==null)
    {
        item.SubItems.Add("不明");
    }
    else
    {
        item.SubItems.Add(agentTable[content.Creator].ToString());
    }
    // コンテンツ作成日
    if(content.Created=="")
    {
        item.SubItems.Add("不明");
    }
    else
    {
        item.SubItems.Add(DateTime.Parse(content.Created).ToString());
    }
    // コンテンツ URI
    item.SubItems.Add(content.Resource);
    return item;
}

/// <summary>
/// コントロールのオンオフを設定 (TreeView)
/// </summary>
/// <param name="TF">true/false</param>
private void SetControlPermissionForTreeView(bool TF)
{
    this.treeViewForMember1.Enabled = TF;
    this.treeViewForMember2.Enabled = TF;
    // コンボボックスとの共通部分
    SetControlPermissionForComboBox(TF);
}

/// <summary>
/// コントロールのオンオフを設定 (ComboBox)
/// </summary>
/// <param name="TF">true/false</param>
private void SetControlPermissionForComboBox(bool TF)
{
    this.comboBoxForMember1.Enabled = TF;
    this.comboBoxForMember2.Enabled = TF;
    this.DrawTreeButton.Enabled = TF;
    if(TF==false)
    {
        Cursor.Current = Cursors.WaitCursor;
    }
}

```

```

else
{
    Cursor.Current = Cursors.Default;
}
}

/// <summary>
/// プロジェクトメンバが作った属性を取得
/// </summary>
/// <param name="cb1">インデックスが変化したコンボボックス</param>
/// <param name="cb2">もう片方のコンボボックス</param>
/// <param name="tv">更新するツリービュー</param>
/// <param name="type">プロジェクトメンバ番号</param>
private void comboBoxForMember_SelectedIndexChanged(ComboBox cb1, ComboBox cb2, TreeView tv, int type)
{
    // コントロールをロック
    SetControlPermissionForComboBox(false);
    // AB サーバに接続
    ABRDFProxy proxy = getProxy();
    // AB サーバからビューを取得
    AttrAttribute[] attr = proxy.GetAllAttributeByProjectAndAgent
        (projectURI[this.comboBoxForProject.SelectedIndex].ToString(),
        agentURI[cb1.SelectedIndex].ToString());
    // AB サーバから切断
    proxy.Logout();
    // 属性ビューを更新
    InitTreeView(cb1, tv, attr);
    // 属性に含まれるコンテンツの情報をハッシュテーブルに格納
    if(type==1)
    {
        mem1 = new Hashtable();
    }
    else if(type==2)
    {
        mem2 = new Hashtable();
    }
    foreach(TreeNode tn in tv.Nodes[0].Nodes)
    {
        MakeHashtable(tn, type);
    }
    // もう片方のプロジェクトメンバも選ばれていれば、類似属性を計算
    if(cb2.SelectedItem!=null)
    {
        Algorithm();
    }
    // コントロールのロック解除
    SetControlPermissionForComboBox(true);
}

/// <summary>
/// 選んだ属性に属するコンテンツを表示
/// </summary>

```

```

/// <param name="tv">属性ノードを選択したツリービュー</param>
/// <param name="tv2">もう片方のツリービュー</param>
/// <param name="tlv">コンテンツを表示させるツリーリストビュー</param>
/// <param name="cb">もう片方のメンバに対応するコンボボックス</param>
/// <param name="type">プロジェクトメンバ番号</param>
private void treeViewForMember_AfterSelect(TreeView tv, TreeView tv2, TreeListView tlv, Co
mboBox cb, int type)
{
    // コントロールをロック
    SetControlPermissionForTreeView(false);
    // もう片方のツリービューのフォーカスをはずす
    tv2.SelectedNode = null;
    // 選択したものが属性名であれば
    if(tv.SelectedNode.ImageIndex==1)
    {
        // 属性ノードを取得
        AttributeItem aitem = (AttributeItem)tv.SelectedNode.Tag;
        // プロジェクトメンバのコンテンツ一覧を更新開始
        tlv.BeginUpdate();
        // ソート不可にしておく
        tlv.Items.Sortable=false;
        // まずは初期化
        tlv.Items.FastClear();
        ABRDFProxy proxy = getProxy();
        // コンテンツを追加
        foreach(PrContent content in aitem.Contents)
        {
            // 追加するコンテンツ
            TreeListViewItem item = MakeTreeListViewItem(content, proxy);
            // 追加
            tlv.Items.Add(item);
        }
        // AB サーバから切断
        proxy.Logout();
        // ソート可に戻す
        tlv.Items.Sortable=true;
        // プロジェクトメンバのコンテンツ一覧を更新終了
        tlv.EndUpdate();
        // もう片方のプロジェクトメンバが選択されている場合
        if(cb.SelectedItem!=null)
        {
            // ツリーノードの色をリセット
            ResetTreeColor(this.treeViewForMember1.TopNode);
            ResetTreeColor(this.treeViewForMember2.TopNode);
            // 類似属性をハイライト表示
            string sim = ((AttributeItem)tv.SelectedNode.Tag).Resource;
            if(type==1)
            {
                // 類似属性があれば
                if(result1[sim]!=null)
                {
                    // そのノードを探し、ハイライト表示
                    TreeNode tnode = SearchNode(((Relation)result1[sim]).Node2, this.treeViewForMemb

```

```

er2.TopNode);
        tnnode.ForeColor = Color.Red;
    }
}
else if(type==2)
{
    if(result2[sim]!=null)
    {
        TreeNode tnnode = SearchNode(((Relation)result2[sim]).Node1, this.treeViewForMem
er1.TopNode);
        tnnode.ForeColor = Color.Red;
    }
}
}
}
else
{
    // 一覧表示をクリア
    tlv.Items.FastClear();
}
// コントロールのロック解除
SetControlPermissionForTreeView(true);
}

/// <summary>
/// コントロールに表示されているものをクリアする
/// </summary>
private void ClearItems()
{
    this.comboBoxForMember1.Items.Clear();
    this.comboBoxForMember2.Items.Clear();
    this.treeViewForMember1.Nodes.Clear();
    this.treeViewForMember2.Nodes.Clear();
    this.treeListViewForMember1.Items.Clear();
    this.treeListViewForMember2.Items.Clear();
}
}
}
}

```

A.1.8 Pr.cs

```

using System;

namespace SemanticBridge
{
    /// <summary>
    /// pr 空間内のタグ定義
    /// </summary>
    public class Pr
    {
        /// <summary>
        /// pr 空間の URI
        /// </summary>
        public const string pr = "http://ice.kuis.kyoto-u.ac.jp/ont/project/0.1/";
    }
}

```

```

    /// <summary>
    /// fullname プロパティ
    /// </summary>
    public const string fullname = pr + "fullname";
    /// <summary>
    /// format プロパティ
    /// </summary>
    public const string format = pr + "format";
    /// <summary>
    /// name プロパティ
    /// </summary>
    public const string name = pr + "name";
    /// <summary>
    /// created プロパティ
    /// </summary>
    public const string created = pr + "created";
    /// <summary>
    /// creator プロパティ
    /// </summary>
    public const string creator = pr + "creator";
    /// <summary>
    /// modified プロパティ
    /// </summary>
    public const string modified = pr + "modified";
    /// <summary>
    /// text プロパティ
    /// </summary>
    public const string text = pr + "text";

    /// <summary>
    /// インスタンス生成を行わせないための処理
    /// </summary>
    private Pr()
    {
    }
}
}

```

A.1.9 PrAgent.cs

```

using System;

namespace SemanticBridge
{
    /// <summary>
    /// Project Vocabulary のエージェントクラス
    /// </summary>
    public class PrAgent
    {
        /// <summary>
        /// エージェントの URI
        /// </summary>
        private string mResource;

        /// <summary>
        /// 名前
        /// </summary>
        private string mFullname;

        /// <summary>
        /// エージェントの URI
        /// </summary>
        public string Resource
        {
            get
            {
                return mResource;
            }
        }
    }
}

```

```

    }
    set
    {
        mResource = value;
    }
}

/// <summary>
/// 名前
/// </summary>
public string Fullname
{
    get
    {
        return mFullname;
    }
}

set
{
    mFullname = value;
}
}

/// <summary>
/// エージェントクラスのコンストラクタ
/// </summary>
public PrAgent()
{
    mResource = "";
    mFullname = "";
}
}
}

```

A.1.10 PrContent.cs

```

using System;

namespace SemanticBridge
{
    /// <summary>
    /// Project Vocabulary のコンテンツクラス
    /// </summary>
    public class PrContent
    {
        /// <summary>
        /// コンテンツの URI
        /// </summary>
        private string mResource;
        /// <summary>
        /// フォーマット、MIME のメディアタイプ参照
        /// </summary>
        private string mFormat;
        /// <summary>
        /// 作成日
        /// </summary>
        private string mCreated;
        /// <summary>
        /// 作成者
        /// </summary>
        private string mCreator;
        /// <summary>
        /// 更新日
        /// </summary>
        private string mModified;
        /// <summary>
        /// 名前
        /// </summary>
        private string mName;

        /// <summary>
        /// 返信やり取りなどの参照関係
        /// </summary>
        private string mReference;

        /// <summary>
        /// コンテンツの URI
        /// </summary>
        public string Resource
        {
            get
            {
                return mResource;
            }
            set
            {
                mResource = value;
            }
        }

        /// <summary>
        /// フォーマット、MIME のメディアタイプ参照
        /// </summary>
        public string Format
        {
            get
            {
                return mFormat;
            }
            set
            {
                mFormat = value;
            }
        }
    }
}

```

```

}

/// <summary>
/// 作成日
/// </summary>
public string Created
{
    get
    {
        return mCreated;
    }
    set
    {
        mCreated = value;
    }
}

/// <summary>
/// 作成者
/// </summary>
public string Creator
{
    get
    {
        return mCreator;
    }
    set
    {
        mCreator = value;
    }
}

/// <summary>
/// 更新日
/// </summary>
public string Modified
{
    get
    {
        return mModified;
    }
    set
    {
        mModified = value;
    }
}

}

/// <summary>
/// 名前
/// </summary>
public string Name
{
    get
    {
        return mName;
    }
    set
    {
        mName = value;
    }
}

/// <summary>
/// 返信やリプライなどの参照関係
/// </summary>
public string Reference
{
    get
    {
        return mReference;
    }
    set
    {
        mReference = value;
    }
}

/// <summary>
/// コンテンツクラスのコンストラクタ
/// </summary>
public PrContent()
{
    mResource = "";
    mFormat = "";
    mCreated = "";
    mCreator = "";
    mModified = "";
    mName = "";
    mReference = "";
}
}
}

```

A.1.11 PrProject.cs

```

using System;

namespace SemanticBridge
{
    /// <summary>
    /// Project Vocabulary のプロジェクトクラス

```

```

/// </summary>
public class PrProject
{
    /// <summary>
    /// プロジェクトの URI
    /// </summary>
    private string mURI;
    /// <summary>
    /// プロジェクト名
    /// </summary>
    private string mName;

    /// <summary>
    /// プロジェクトの URI
    /// </summary>
    public string URI
    {
        get
        {
            return mURI;
        }
        set
        {
            mURI = value;
        }
    }
}

/// <summary>
/// プロジェクト名
/// </summary>
public string Name
{
    get
    {
        return mName;
    }
    set
    {
        mName = value;
    }
}

/// <summary>
/// プロジェクトクラスのコンストラクタ
/// </summary>
public PrProject()
{
    mURI = "";
    mName = "";
}
}
}

```

A.1.12 PrTextDocument.cs

```

using System;
using System.Reflection;

namespace SemanticBridge
{
    /// <summary>
    /// Project Vocabulary の TextDocument クラス
    /// </summary>
    public class PrTextDocument : PrContent
    {
        /// <summary>
        /// テキストの言語
        /// </summary>
        private string mLanguage;
        /// <summary>
        /// ドキュメントの内容の raw テキスト
        /// </summary>
        private string mText;
        /// <summary>
        /// 翻訳されたドキュメントの作成日
        /// </summary>
        private string mTranslatedDate;
        /// <summary>
        /// 翻訳文のオリジナルドキュメント

```

```

/// </summary>
private PrTextDocument mTranslatedFrom;

/// <summary>
/// テキストの言語
/// </summary>
public string Language
{
    get
    {
        return mLanguage;
    }
    set
    {
        mLanguage = value;
    }
}

/// <summary>
/// ドキュメントの内容の raw テキスト
/// </summary>
public string Text
{
    get
    {
        return mText;
    }
    set
    {
        mText = value;
    }
}

/// <summary>
/// 翻訳されたドキュメントの作成日
/// </summary>
public string TranslatedDate
{
    get
    {
        return mTranslatedDate;
    }
    set
    {
        mTranslatedDate = value;
    }
}

/// <summary>
/// 翻訳文のオリジナルドキュメント
/// </summary>
public PrTextDocument TranslatedFrom
{

```

```

    get
    {
        return mTranslatedFrom;
    }
    set
    {
        mTranslatedFrom = value;
    }
}

/// <summary>
/// PrContent クラスのオブジェクトからダウンキャストしたオブジェクトを返します
/// </summary>
/// <param name="content">PrContent クラスのオブジェクト</param>
public PrTextDocument(PrContent content) : base()
{
    // プロパティ値をコピー
    CopyProperties(content, this);
    mLanguage = "";
    mText = "";
    mTranslatedDate = "";
    mTranslatedFrom = null;
}

/// <summary>
/// src の全てのパブリックプロパティを dest にコピー
/// </summary>
/// <param name="src">元オブジェクト</param>
/// <param name="dest">先オブジェクト</param>
/// <returns>先オブジェクト</returns>
private static object CopyProperties(object src, object dest)
{
    Type srcType = src.GetType();
    Type destType = dest.GetType();
    PropertyInfo[] srcProperties = srcType.GetProperties(BindingFlags.Instance | BindingFlags.Public);
    PropertyInfo destProperty;
    foreach(PropertyInfo p in srcProperties)
    {
        destProperty = destType.GetProperty(p.Name);
        destProperty.SetValue(dest, p.GetValue(src, null), null);
    }
    return dest;
}
}
}

```

A.1.13 Relation.cs

```

using System;
namespace SemanticBridge
{
    /// <summary>
    /// 類似概念の組を格納するためのクラス
    /// </summary>
    public class Relation

```

```

{
    return mNode2;
}
set
{
    mNode2 = value;
}
}

/// <summary>
/// 2 ノード間の類似度
/// </summary>
public double Sim
{
    get
    {
        return mSim;
    }
    set
    {
        mSim = value;
    }
}

/// <summary>
/// プロジェクトメンバ 1 のノード
/// </summary>
public string Node1
{
    get
    {
        return mNode1;
    }
    set
    {
        mNode1 = value;
    }
}

/// <summary>
/// プロジェクトメンバ 2 のノード
/// </summary>
public string Node2
{
    get
    {

```

A.1.14 TreeForm.cs

```

using System;
using System.Drawing;
using System.Windows.Forms;

namespace SemanticBridge
{
    /// <summary>
    /// ツリー表示を行うクラス
    /// </summary>
    public class TreeForm : System.Windows.Forms.Form
    {
        /// <summary>
        /// プロジェクトメンバ 1 のビュー
        /// </summary>

```

```

private TreeView tree1;
/// <summary>
/// プロジェクトメンバ 2 のビュー
/// </summary>
private TreeView tree2;
/// <summary>
/// 描画領域の幅
/// </summary>
private int wid;
/// <summary>
/// 描画領域の高さ
/// </summary>
private int hei;
/// <summary>
/// 定数を格納
/// </summary>
private Constant c;
/// <summary>
/// ドラッグ開始時のマウスポインタの位置を格納
/// </summary>
private Point p;
/// <summary>
/// 描画領域をどれくらい移動させるかを示す
/// </summary>
private Point diff = new Point(0, 0);
/// <summary>
/// ドラッグできるか否かを定める定数
/// </summary>
private bool canMove = false;

private System.Windows.Forms.Panel BasePanel;
private System.Windows.Forms.Panel DrawPanel;

/// <summary>
/// 必要なデザイナ変数です。
/// </summary>
private System.ComponentModel.Container components = null;

/// <summary>
/// コンストラクタ
/// </summary>
/// <param name="t1">プロジェクトメンバ 1 のビュー</param>
/// <param name="t2">プロジェクトメンバ 2 のビュー</param>
/// <param name="w">描画領域の幅</param>
/// <param name="h">描画領域の高さ</param>
public TreeForm(TreeView t1, TreeView t2, int w, int h)
{
    //
    // Windows フォーム デザイナ サポートに必要です。
    //
    InitializeComponent();
    ConfigForm configForm = new ConfigForm();
    tree1 = t1;

```

```

tree2 = t2;
wid = w;
hei = h;
// 設定の読み込み
c = configForm.Config;
// 描画したものが全て表示されるだけのサイズを確保
this.DrawPanel.Size = new Size(wid, hei);
// マウスホイールの操作で画像が拡大縮小できるための措置
this.MouseWheel += new MouseEventHandler(DrawPanel_MouseWheel);
}

/// <summary>
/// 使用されているリソースに後処理を実行します。
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if(components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

#region Windows フォーム デザイナで生成されたコード
/// <summary>
/// デザイナ サポートに必要なメソッドです。このメソッドの内容を
/// コード エディタで変更しないでください。
/// </summary>
private void InitializeComponent()
{
    this.BasePanel = new System.Windows.Forms.Panel();
    this.DrawPanel = new System.Windows.Forms.Panel();
    this.BasePanel.SuspendLayout();
    this.SuspendLayout();
    //
    // BasePanel
    //
    this.BasePanel.AutoScroll = true;
    this.BasePanel.Controls.Add(this.DrawPanel);
    this.BasePanel.Dock = System.Windows.Forms.DockStyle.Fill;
    this.BasePanel.Location = new System.Drawing.Point(0, 0);
    this.BasePanel.Name = "BasePanel";
    this.BasePanel.Size = new System.Drawing.Size(792, 566);
    this.BasePanel.TabIndex = 0;
    //
    // DrawPanel
    //
    this.DrawPanel.Location = new System.Drawing.Point(0, 0);
    this.DrawPanel.Name = "DrawPanel";
    this.DrawPanel.Size = new System.Drawing.Size(800, 600);
}

```

```

        this.DrawPanel.TabIndex = 1;
        this.DrawPanel.MouseUp += new System.Windows.Forms.MouseEventHandler(this.DrawPanel_MouseUp);
        this.DrawPanel.Paint += new System.Windows.Forms.PaintEventHandler(this.DrawPanel_Paint);
    };
    this.DrawPanel.MouseMove += new System.Windows.Forms.MouseEventHandler(this.DrawPanel_MouseMove);
    this.DrawPanel.MouseDown += new System.Windows.Forms.MouseEventHandler(this.DrawPanel_MouseDown);
    //
    // TreeForm
    //
    this.AutoScaleBaseSize = new System.Drawing.Size(5, 12);
    this.ClientSize = new System.Drawing.Size(792, 566);
    this.Controls.Add(this.BasePanel);
    this.Name = "TreeForm";
    this.Text = "ツリー表示";
    this.BasePanel.ResumeLayout(false);
    this.ResumeLayout(false);

}
#endregion

/// <summary>
/// 属性ノードを再帰的に描画
/// </summary>
/// <param name="node">属性ノード</param>
/// <param name="g">グラフィックオブジェクト</param>
/// <param name="pen">ペン</param>
/// <param name="member">プロジェクトメンバ番号</param>
private void DrawTree(TreeNode node, Graphics g, Pen pen, int member)
{
    // width、height は c.Distance の 0.5 倍以下であるべき
    int width = c.Distance / 2;
    int height = c.Distance / 2;
    int x = ((AttributeItem)node.Tag).x - width/2;
    int y = ((AttributeItem)node.Tag).y - height/2;
    // プロジェクトメンバ 2 のツリーは少しだけずらして表示
    if(member==2)
    {
        x += 10;
        y += 10;
    }
    // 属性ノードに外接する長方形の大きさ
    Rectangle rect = new Rectangle(x, y, width, height);
    // 上で用意した長方形に内接する楕円を描く
    g.DrawEllipse(pen, rect);
    // 属性名は中央揃え
    StringFormat format = new StringFormat();
    format.Alignment = StringAlignment.Center;
    // 属性名を描画
    g.DrawString(node.Text, this.Font, new SolidBrush(pen.Color), (RectangleF)rect, format);
    // 親ノードがあれば

```

```

if(node.Parent!=null)
{
    // 直線の始点
    int x1 = ((AttributeItem)node.Parent.Tag).x;
    int y1 = ((AttributeItem)node.Parent.Tag).y + height/2;
    // 直線の終点
    int x2 = ((AttributeItem)node.Tag).x;
    int y2 = ((AttributeItem)node.Tag).y - height/2;
    // プロジェクトメンバ 2 のツリーは少しだけずらして表示
    if(member==2)
    {
        x1 += 10;
        x2 += 10;
        y1 += 10;
        y2 += 10;
    }
    // 子ノードから親ノードへ向かう直線を引く
    g.DrawLine(pen, x1, y1, x2, y2);
}
// 子ノードがあれば
if(node.Nodes.Count!=0)
{
    foreach(TreeNode tn in node.Nodes)
    {
        // 子ノードを再帰的に描画する
        DrawTree(tn, g, pen, member);
    }
}
}

/// <summary>
/// 属性ノードの描画処理
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void DrawPanel_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{
    // 描画のための準備
    Graphics g = e.Graphics;
    // 描画
    PaintPanel(g);
    // 使った後は Dispose
    g.Dispose();
}

/// <summary>
/// マウスホイールイベントが発生すると画像を拡大縮小
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void DrawPanel_MouseWheel(object sender, MouseEventArgs e)
{
    // マウスホイールの移動量に応じて拡大縮小

```

```

int offset = e.Delta / 10;
int newdist = c.Distance + offset;
// サイズが大きすぎず小さすぎずという範囲であれば実際に拡大縮小
if(newdist>=1 && newdist<=1000)
{
    // 描画領域の大きさを変更
    wid = wid * newdist / c.Distance;
    hei = hei * newdist / c.Distance;
    this.DrawPanel.Size = new Size(wid, hei);
    // 属性ノードの描画位置を変更
    ChangePosition(tree1.TopNode, newdist);
    ChangePosition(tree2.TopNode, newdist);
    // 画像サイズの値を変更
    c.Distance += offset;
    // 描画準備
    Graphics g = this.DrawPanel.CreateGraphics();
    this.DrawPanel.Invalidate();
    // 描画
    PaintPanel(g);
    // 画面に反映し、Dispose
    this.DrawPanel.Update();
    g.Dispose();
}
}

/// <summary>
/// 属性ノードの描画位置を再帰的に変更
/// </summary>
/// <param name="node">描画位置を変更する属性ノード</param>
/// <param name="newdist">拡大縮小のための値</param>
private void ChangePosition(TreeNode node, int newdist)
{
    // 値の変更
    ((AttributeItem)node.Tag).x = ((AttributeItem)node.Tag).x * newdist / c.Distance;
    ((AttributeItem)node.Tag).y = ((AttributeItem)node.Tag).y * newdist / c.Distance;
    // 子ノードに対しても再帰的に実行
    foreach(TreeNode tn in node.Nodes)
    {
        ChangePosition(tn, newdist);
    }
}

/// <summary>
/// 実際に描画する部分
/// </summary>
/// <param name="g">グラフィックオブジェクト</param>
private void PaintPanel(Graphics g)
{
    // Pen を用意
    Pen penBlue = new Pen(Color.Blue);
    Pen penRed = new Pen(Color.Red);
    // 属性ノードを再帰的に描画
    DrawTree(tree1.TopNode, g, penBlue, 1);
}

```

```

    DrawTree(tree2.TopNode, g, penRed, 2);
    // Pen を使った後は Dispose
    penBlue.Dispose();
    penRed.Dispose();
}

/// <summary>
/// ドラッグアンドドロップが開始された時点で行う処理
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void DrawPanel_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
{
    // 左クリックのとき
    if(e.Button==MouseButtons.Left)
    {
        // 描画領域をずらせるようにする
        canMove = true;
        // マウスカーソルを変更
        Cursor.Current = Cursors.Hand;
        // ドラッグ開始位置を格納
        p = new Point(e.X, e.Y);
    }
}

/// <summary>
/// ドラッグ中の処理
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void DrawPanel_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e)
{
    // 描画領域が移動可能であれば
    if(canMove)
    {
        // 移動の限界を示す点
        Point max = new Point(this.DrawPanel.Size.Width-this.BasePanel.ClientSize.Width, this.
DrawPanel.Size.Height-this.BasePanel.ClientSize.Height);
        // 移動量を仮に決定
        diff = new Point(diff.X-(e.X-p.X), diff.Y-(e.Y-p.Y));
        // ただし移動領域は限られているので、移動量を補正する
        if(diff.X<0)
        {
            diff = new Point(0, diff.Y);
        }
        if(diff.X>max.X)
        {
            diff = new Point(max.X, diff.Y);
        }
        if(diff.Y<0)
        {
            diff = new Point(diff.X, 0);
        }
    }
}

```

```

        if(diff.Y>max.Y)
        {
            diff = new Point(diff.X, max.Y);
        }
        // 実際に描画領域をずらす
        this.BasePanel.AutoScrollPosition = new Point(diff.X, diff.Y);
        // 描画領域をずらすと、マウスを止めていても相対的にマウスポインタが移動していることにな
なる
        // これに対処するための措置
        p = new Point(e.X-(e.X-p.X), e.Y-(e.Y-p.Y));
    }
}

/// <summary>
/// ドラッグアンドドロップが終了した時点で行う処理
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void DrawPanel_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)
{
    // 描画領域を移動不能にする
    canMove = false;
    // マウスカーソルを元の形に戻す
    Cursor.Current = Cursors.Default;
}
}
}

```