ATRテクニカルレポート表紙

[公開]

TR-M-0057

仲介システムについて

門林　理恵子
Rieko KADOBAYASHI

間瀬　健二
Kenji MASE

2000.6.20

ATR 知能映像通信研究所

Part I

仲介システムについて

門林 理恵子　　間瀬 健二

平成 12 年 6 月 20 日

1　はじめに

1.1　仲介システムの概要

仲介システムは，博物館の展示の説明文として陰に表現される学芸員の知識と，展示に対して示される見学者の興味を“仲介”して，不特定多数の見学者向けに設計された博物館展示を見学者それぞれに個人化するシステムである．本システムでは，説明文と説明文に含まれるキーワードに対して統計処理を行い，博物館展示に含まれる展示物どうしの関連を 2 次元空間構造として可視化する．展示物は，文字アイコンまたは静止画アイコンとして 2 次元空間に配置される．見学者はこれらのアイコンを操作することで，展示物の数と配置が元の展示とは異なる新しい空間を創出することができる．つまり，本システムは，展示物の意味的関連に基づく構造の個人化を行うものであり，博物館展示の物理的配置そのものを変更して個人化するものではない．しかし，本システムを見学すべき展示物を推奨するシステムとして使用し，VisTA-walk システムや CMC システムと連携させることで，仮想空間における展示の配置を変更し，個人化させることが可能である．

1.2　本稿の目的と構成

仲介システムには，AIDE システムの機能を拡張して構築した第 1 版と，新たに java でプログラミングした第 2 版の 2 つがある．本稿では，java 版の仲介システムについて述べる．本稿の構成は，次の通りである．

パート I　　仲介システムの実装に関する覚え書き (このドキュメント)
パート II　　付録 1: 仲介システムに関する論文
パート III　付録 2: プログラムソースコード

2　java 版仲介システム

2.1　バージョンの違い

仲介システムの第 1 版は，AIDE システム (特に PD 機能) を利用して開発した．第 2 版では，java アプリケーションとして実装した．大きな変更点は次の 2 点である．

- 空間配置方法を，双対尺度法から主成分分析に変更:
  実装のための時間を短縮するためと，数値データを扱えるようにするため．

- 形態素解析機能の未実装:
  従って，仲介システムを利用する前に，対象とする説明文オブジェクトから手作業でキーワードを切り出し，それに重みを付与したファイルを作成しておく必要がある．

第1版はSUN OSまたはIRIXで動作する (現在はSUN OSではなくSolaris)．第2版はjavaアプリケーションなので，javaの動作環境があればOSを問わずに動作する．ただし，メニューなどの日本語を正しく表示させるには，Windowsが望ましい．

2.2 空間構成手順

java版仲介システムにおける空間構成手順は次の通りである．まず，人手で次の作業を行なう．

1. 展示物の説明文からキーワードを抽出し，重みを与える．
2. キーワードど重みからなるテキストファイルを展示物ごとに作成する．さらに，このファイルには展示物のIDや説明文そのものも含める．詳細は後述する．

次に，仲介システムを起動し入力ファイルを指定すると，以下の処理が行われ，展示空間が構成され，表示される．

1. 説明文 (展示物ID) とキーワードの行列を作成する．
2. キーワードの分散共分散行列を作成する．
   分散共分散行列とは分散と共分散を行列形式に並べたもので，ij成分は第i変数と第j変数の共分散である．(対角成分は分散)．キーワードkがn個あるとき，その分散共分散行列Covのi行j列の値は

$$Cov(i,j) = \frac{1}{n-1}\sum_{l=1}^{n}(k_{il} - \overline{k_i})(k_{jl} - \overline{k_j})$$

3. 分散共分散行列から第1主成分，第2主成分を求める．
   固有ベクトルの計算は，CLAPACKのJAVA版を使用する．
4. 第1主成分を縦軸，補正した第2主成分を横軸として展示物アイコンを配置する．
   第2主成分の補正の仕方は次の通り．

$$\text{補正した第2主成分} = \text{元の第2主成分} \times \frac{\text{第1主成分の寄与率}}{\text{第2主成分の寄与率}}$$

2.3 キーワードファイルの作成

PD版では，AIDEの形態素解析機能を使用することができたため，入力はタイトル (最初の9文字を文字列アイコンとして使用する) と本文からなるテキストファイル，もしくはテキストファイルの名前を列挙したリストファイルであったが，java版では形態素解析機能を実装していないため，展示物ごとに作成したキーワードとその重み，説明文そのもの等からなるファイル，もしくはそのファイルの名前を列挙したリストファイルを入力とする．

ただし，PD版の場合も，コンテンツが歴史関係のためうまくキーワードを切り出せないときには，手で修正した．たとえば「正倉院」など．

表 1: キーワードファイルの構成

フィールド名	内容	注意
ID	識別子	必ず必要
TAG	アイコンに使用するテキスト	IMAGE があればなくてもよい
IMAGE	アイコンに使用するイメージ	TAG があればなくてもよい．あれば必ず使用．jpg ファイルもしくは gif ファイルを指定する．
TEXT	説明文の開始指示	
	説明の本文	本文ウィンドウで表示される
TEXT_END	説明文の終了指示	
KEYWORD	キーワードの開始指示	
	キーワード (空白) その重み	PD 版/java 版ともにデフォルトでは 1.0
	⋮	GUI を利用して重みを変更できる．
KEYWORD_END	キーワードの終了指示	

```
ID 201
TAG 東大寺
IMAGE image/todaiji-daibutsuden.gif
TEXT
東大寺の歴史と所在．聖武天皇の詔勅により，建立され 760 に完成しました．当初は東西に七重塔がありました．現在の南大門は鎌倉時代の再建です．奈良県奈良市雑司町 406-1．最寄り駅は近鉄奈良駅，またはＪＲ奈良駅．
TEXT_END
KEYWORD
東大寺 1.00
歴史 1.00
天皇 1.00
七重塔 1.00
(以下略)
KEYWORD_END
```

図 1: キーワードファイルの例

```
OBJECT    data/openhouse/001.txt
OBJECT    data/openhouse/002.txt
OBJECT    data/openhouse/003.txt
⋮         ⋮
```

図 2: リストファイルの例

java 版仲介システムにおける展示物ごとのファイルの構成とサンプルを表 1 と図 1 に示す.

なお, 展示物がたくさんある場合は, 展示物データファイルを列挙したリストファイルを作成し, そのファイルを読み込めばよい. リストファイルの例を図 2 に示す.

2.4 仲介システムのディレクトリ構成

仲介システムではキーワードと重みのデータファイルおよびアイコン用イメージデータを下記のディレクトリにおく必要がある. ただし, データファイル中の IMAGE の項でファイル名を指定していないときは, `mediation/image` にイメージデータを置く必要はない.

- mediation: 仲介機能プログラム本体
- mediation/data: キーワードデータ
- mediation/image: 表示アイコンイメージ

2.5 起動方法

仲介システムの起動は次のようにして行なう.

```
% java MediationFunc (directory)/file hostname portNo
```

`java MediationFunc` の引数は, 以下の通りである.

- file: キーワードファイル.
  既に (形態素) 解析された, キーワードと重みを記述したファイルを指定する. 起動時に指定しなくても, 実行後にメニューからも指定できる.
- hostname: V_Nara 実行ホスト.
  Virtual 奈良ウォークスループログラムを実行しているホスト名を指定する. 指定しなければデフォルト値が使用される.
- portNo: 通信ポート番号.
  ソケットの通信ポート番号を指定する. 指定しなければデフォルト値は 23456 である.

2.6 操作方法

仲介システムを起動すると, まず展示空間ウィンドウが表示される (図 3 参照).

展示空間には展示物オブジェクトが文字アイコンまたは静止画アイコンで表示される. ただし, 起動時の引数にデータファイルを指定しなかった場合は, アイコンが表示されない. この場合は, ファイルメニューを選び, データファイルを開く必要がある.

複数のオブジェクトを選択した後で, 展示空間ウィンドウ下部にある「興味空間」ボタンをクリックすると興味空間ウィンドウが開く. さらに, その後,「個人化空間」ボタンをクリックすると, 個人化空間ウィンドウとパターンウィンドウが開く.

オブジェクトを選択しなおして, 新たに興味空間, 個人化空間を作るときは, 一旦, これらのウィンドウの下部にある「閉じる」ボタンをクリックしてウィンドウを閉じる. その後, 展示空間でオブジェクトを選択しなおし, 上記の手順を繰り返せばよい.

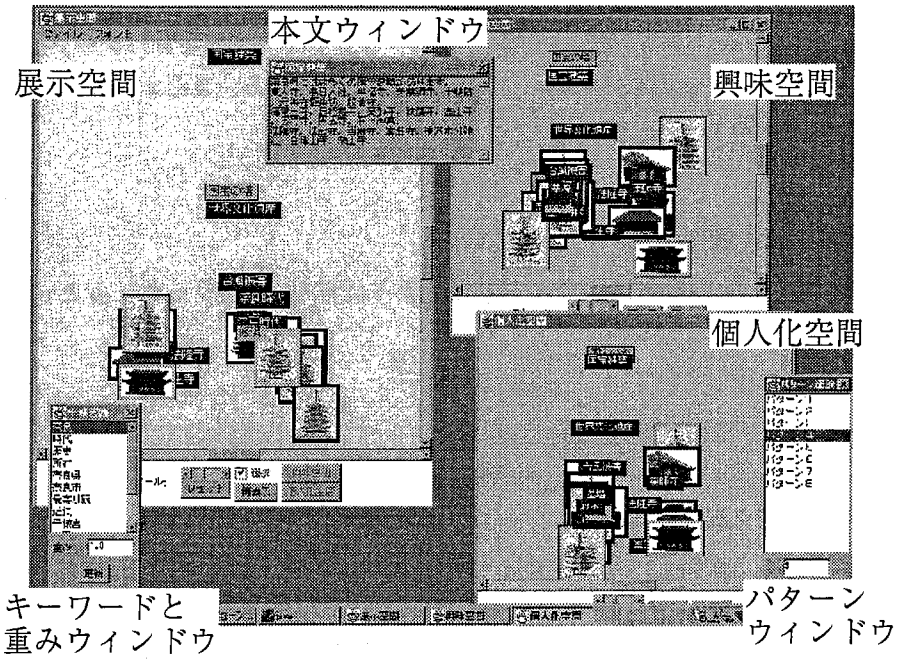


図 3: 仲介システムの画面スナップ

仲介システムを終了するときは，展示空間のファイルメニューの中の終了を選択すればよい．このとき，展示空間の状態を保存して終了することが可能である．こうすれば，後で同じデータファイルを操作するときに，主成分分析 (行列演算) をする必要がなくなるため，起動から表示までの時間が短縮される．保存されるのは，展示空間におけるオブジェクト間の距離である．

展示空間，興味空間，個人化空間の各ウィンドウでは，次の操作が可能である．

- 選択: 左クリック．同じものを 2 度選択すると，選択を解除することになる．
- 本文ウィンドウ表示: 右クリック．
- キーワードとその重みウィンドウ表示: Ctrl + 左クリック．ウィンドウ下部に，選択されたキーワードの重みが表示されるので，数値を入力すると重みの変更が可能である．
- オブジェクトの移動: Shift + 左クリックしながらドラッグ．

キーワードウィンドウでは，選択したキーワードの重みが下部に表示される．変更する場合は，そこに数値を入力する．

パターンウィンドウでは，8 パターンのいずれかをクリックするとオブジェクトの数がウィンドウ下部に表示される．ダブルクリックするとそのパターンに含まれるオブジェクトの ID がソケットを利用して他のプログラムに送信される．

3 パターン分類の方法

パターン分類の方法は，第 1 版も第 2 版も同じであり，以下のようにして行なう．

2つのテキストオブジェクト間の距離の遠近が，展示空間，興味空間，個人化空間の順に生成される空間において，どのように変化したかに基づき，オブジェクト対を8パターンに分類する．従って，1つのオブジェクトが複数のパターンに含まれることもある．またオブジェクトが1つも含まれないパターンが生じることもある．

3.1 距離計算の方法

2つのオブジェクト間の距離の遠近の判定は，次のように行う．オブジェクト o_i と o_j $(i \neq j)$ の正規化距離 d_{ij} を，

$$d_{ij} = \frac{x_{ij} - \overline{x}}{\sigma}$$

によって求める．ただし，x_{ij} は各空間における o_i と o_j の実測距離，$\overline{x}$ および σ は，それぞれ各空間における全オブジェクト間の実測距離の平均と標準偏差である．そして，d_{ij} が一定の閾値より大きい場合を遠い，小さい場合を近いとする．現在は，$d_{ij} > 0.3$ の場合を遠い，$d_{ij} < -0.3$ の場合を近いとしている．閾値を変更するGUIは未実装．

3.2 距離変化とパターンの対応

表 2: 距離変化のパターン

パターン	展示空間	興味空間	個人化空間
1	近い	近い	近い
2	近い	近い	遠い
3	近い	遠い	近い
4	近い	遠い	遠い
5	遠い	近い	近い
6	遠い	近い	遠い
7	遠い	遠い	近い
8	遠い	遠い	遠い

4 テキストオブジェクトとキーワードの例

付録1の論文中では，国立歴史民俗博物館の常設展示のデータを例として使用している．このデータは各展示室ごとにホームページに掲載されていたもので，展示室そのものの説明を除くと全部で25のテキストオブジェクトが得られた．これらのテキストオブジェクトの一覧を示す．それぞれについて，タグ (表示用文字列)，本文，キーワードの順に記している．なお，本文中の操作例で挙げた「日本文化のあけぼの」「王朝文化」「民衆の生活と文化」「沖ノ島」の4つのオブジェクトを最初に記し，残りについては展示室の順番にしたがって記している．

%
% 国立歴史民俗博物館の展示の説明文とキーワード
%
==========

TAG 日本文化のあけぼの

SENTENCE
日本文化のあけぼの

旧石器時代から縄文時代までを展示してあります．日本列島に人が住みついてから，
特色のある文化が出来あがるまでの暮らしのうつりかわりをみてください．

入り口からの展示場風景
手前の像は，縄文時代のビーナス（約 3800 年前，縄文後期）．

縄文中期の土器
日本各地の土器の様相．

青森県三内丸山遺跡の復元家屋
5500 年前から 4000 年前（縄文前・中期）までの約 1500 年間続いた大規模な村の跡．

KEYWORD
日本列島，時代，日本，文化，縄文，土器，あけぼの，旧石器，人，特色，暮
らし，像，ビーナス，後期，中期，各地，様相，青森県，三内丸山，遺跡，復
元，家屋，前期，大規模，村，跡
==========

TAG 王朝文化

SENTENCE
王朝文化（10〜12 世紀）

平安時代には，それまでの唐風文化に代わって日本的特色をもったいわゆる
国風文化（王朝文化）が花開きました．この文化は貴族的・女性的で都市的な
要素を強くもっています．平仮名や片仮名の仮名文字が創出され，漢文に
よらずに，自由に日本語の文章を記すことができるようになったのもこの時期です．
ここでは貴族たちが，どのような生活をおくったのかを中心に展示してあります．

入り口からの展示風景

王朝貴族の服装
男子では束帯が，儀式の際の正装で，ほかに通常服の直衣・狩衣がありました．
女子では女房装束が女官の正装でした．

調度を配した室内
12 世紀末ごろの貴族の邸内の一部を推定復元したもの.

KEYWORD
時代，日本，文化，生活，特色，儀式，都市，平安，貴族，王朝，唐風，貴族的，
女性的，要素，平仮名，片仮名，仮名，文字，創出，漢文，自由，日本語，文章，
服装，男子，束帯，正装，通常，服，直衣，狩衣，女子，女房，装束，女官，
調度，室内，邸内，一部，推定，国風
==========

TAG 民衆の生活と文化

SENTENCE
民衆の生活と文化（14～16 世紀）

下剋上の一語がよく示すように，中世後期は，底辺の民衆が歴史の表舞台に
はなばなしく登場した時代です．その前夜，鎌倉（かまくら）期には
民衆のエネルギーが開花し，農業技術はめざましく発展し，山民・海民の
漂泊から定着への動きがみられました．ここでは様々な生業や芸能，
農業・手工業の技術などについて，展示しています．

芸能と職人～展示場風景
一服一銭・大鋸と大鋸引き

大鋸引き
室町時代になると大鋸とよばれる縦引鋸が中国から導入され，初めて
製材が可能になり，建築界に一大革新をもたらしました．

惣の寄り合いと農村展示風景
水力自転揚水車復元模型（14 世紀）

KEYWORD
登場，時代，文化，生活，後期，中世，様々，鎌倉，建築，技術，大鋸，民衆，
生業，芸能，一大，下剋上，底辺，歴史，表舞台，エネルギー，開花，農業，
発展，山民，海民，漂泊，定着，農業，手工業，職人，一服，一銭，大鋸引き，
室町，縦引，鋸，中国，導入，製材，可能，革新，惣，寄り合い，農村，水力，
自転，揚水車
==========

TAG 沖ノ島

SENTENCE
沖ノ島（4～10 世紀）

沖ノ島は，玄海灘のまっただ中に浮かぶ東西 1.5 キロ，南北 1 キロほどの

小さな孤島です．この島には，福岡の宗像大社の沖津宮があり，古代から
「神の島」として人々の信仰を集め，4 世紀後半から 10 世紀にかけて，
おおがかりな神まつりが行われていました．そしてこの祭祀は，その
驚くべき豪華な奉献品からも，朝鮮半島との航海の安全を祈る
国家的な性格であったと思われます．

沖ノ島磐座分布模型

沖ノ島 17 号地点磐座模型（4 世紀後半）

沖ノ島 8 号地点磐座模型（8 世紀初期）

KEYWORD
国家，祭祀，沖ノ島，品，古代，人々，信仰，神，まつり，玄海灘，孤島，島，
福岡，宗像，大社，沖津宮，豪華，奉献，朝鮮，半島，航海，安全，性格，磐座，
分布

==========

TAG 稲と倭人

SENTENCE
稲と倭人（前 5～後 3 世紀）

稲は日本列島に自然に生えていた植物ではなく，大陸から伝わってきたもので，
弥生時代に本格的に稲の栽培がはじまりました．ここでは米作りが当時の
人々の生活に，どのような影響や変化をもたらしたかを展示してあります．

高床倉庫（1 世紀）
稲の穂を摘んで収穫し，たくわえるための高い床をもつ倉．

大塚遺跡（前 1 世紀）
村のまわりに濠と土塁をめぐらせた，環濠集落の典型的な形．

銅鐸
弥生時代の儀式に関連する青銅器．筒状で裾の広がった身と鈕からなる．

KEYWORD
日本列島，時代，大陸，弥生，器，人々，生活，稲，倭人，遺跡，村，植物，
栽培，米，作り，影響，床，倉庫，穂，倉，儀式，関連，青銅，筒，裾，身，
鈕，高床，収穫，高い，大塚，土塁，濠，環濠，典型，形，銅鐸，集落

==========

TAG 前方後円墳の時代

SENTENCE

前方後円墳の時代（3～7 世紀）

古墳とよばれる小山のように大きな墓があります．3 世紀から 7 世紀にかけて
つくられた，王とそれにつかえた豪族たちの墓です．もっとも大きな
前方後円墳は，長さが 486 メートルに及びます．ここでは巨大古墳をはじめ，
数多くの古墳が 400 年間にわたって，なぜ各地でつくり続けられたかを
展示してあります．

箸墓古墳の墳丘模型（3 世紀）
出現期の前方後円墳の中で最大の大きさをもつ前方後円墳．墳丘長が 276m ある．

特殊器台から円筒埴輪へ（2～3 世紀）
古墳の円筒埴輪は，弥生時代後期の岡山県を中心とする吉備地方の首長墓に
供献されていた特殊な器台から生まれた．

長持型石棺（5 世紀）
石製の棺．これは群馬県お富士山古墳の数個の石材を組み合わせた長持型石棺．

KEYWORD
時代，弥生，古墳，前方後円墳，巨大，後期，各地，墓，王，豪族，長さ，数，
多く，箸墓，墳丘，出現，期，最大，大きさ，長，特殊，器台，円筒，埴輪，
岡山県，吉備，地方，首長，供献，長持，石棺，石製，棺，群馬県，お富士山，
石材
==========
TAG 律令国家

SENTENCE
律令国家（7 世紀～10 世紀初）

8 世紀初め，奈良に都がつくられました．平城京です．平城京は，
東西 5.9 キロメートルの広さをもち，道路をごばんの目のように通した
計画都市です．道幅が 70 メートルもある朱雀大路．青や赤で
はなやかにいろどられた建物とつらなる白い壁．そうした都の美しさに，
当時の人々は目を見張ったことでしょう．ここでは都の人々のくらし，
村のありさま，中央政府と諸国との間の文書行政などを中心に展示してあります．

羅城門の復元模型
平城京の入口の正面で，高さが 20 メートルあまりありました．

隼人の楯の復元
九州南部の人々が，都の行事に参加したときに使った呪ないの文様付の楯．

正倉院の宝庫模型

奈良時代の文書や宝物を現代に伝えた木造の高床式の倉庫.

KEYWORD
律令，国家，時代，正倉院，文書，人々，行政，村，倉庫，高床，奈良，都，
平城京，広さ，道路，ごばん，計画，都市，道幅，朱雀，大路，青，赤，建物，
白い，壁，美しさ，当時，くらし，ありさま，中央，政府，諸国，羅城門，入口，
正面，高さ，隼人，楯，九州，南部，行事，参加，呪ない，文様，宝庫，宝物，
現代，木造

==========

TAG 東国と西国

SENTENCE
東国と西国（12～15 世紀）

東国の武士団を中心に鎌倉幕府が成立して以来，日本は鎌倉を中心とした
東国と，京都を中心とした西国の二元的な国家となり,「西船東馬」とも
言うべき，東と西の地域的な差異も決定づけられました.
ここでは，東国中心の武家政権の拠点としての鎌倉と，荘園領主の拠点である
京都および西国流通圏とを対比して，東と西の差を示しています.

東国の武家・瀬戸内海交通・仏像
鉄造薬師如来坐像（重要文化財・栃木県・薬師堂蔵）・武士の館復元模型・
瀬戸内の港湾と交通体系

鎌倉の都市構造模型

備前国福岡市復元模型（14 世紀）
『一遍上人絵伝』に見える備前国福岡市（現岡山県）の光景をジオラマに
復元したもの.

KEYWORD
国家，成立，日本，岡山県，都市，武士，鎌倉，東国，京都，西国，武家，流通，
団，幕府，二元，西船東馬，東，西，地域，差異，政権，拠点，荘園，領主，圏，
対比，差，瀬戸内海，交通，仏像，鉄，薬師，如来，坐像，重要，文化財，栃木県，
薬師堂蔵，館，瀬戸内，港湾，交通，体系，構造，備前国，福岡市，一遍上人絵伝，
光景，ジオラマ

==========

TAG 大名と一揆

SENTENCE
大名と一揆（15～16 世紀）

この時代，人々は地域や階層ごとに横断的な結びつきを強めていました.

村々には自ら掟を定めたり，土一揆を起こしたりする動きがみられ，また
地域の領主である国人たちも，一揆連合を結びました．これに対して
大名は主従制にもとづく，一元的な権力による支配をめざしました．
ここでは大名の館や，村や都市のあり様を展示しています．

展示場風景
朝倉氏本館復元模型（16 世紀）・京都町並復元模型（戦国時代末ごろの
京都四条室町を復元）

一乗谷朝倉氏本館復元模型（16 世紀）

洛中洛外図屏風（歴博甲本，右隻，16 世紀）
歴博甲本（旧町田本），上杉本，東博模本のレプリカなどを順次
展示しています．
歴博甲本は歴博ギャラリーで詳細にご覧いただけます．

KEYWORD
時代，人々，村，都市，階層，京都，四条室町，大名，一揆，戦国，地域，領主，
館，横断，結びつき，村々，掟，国人，連合，主従制，一元，権力，あり様，
朝倉氏，本館，町並，一乗谷，洛中洛外図，屏風，歴博甲本，右隻，町田本，
上杉本，東博，模本，レプリカ，歴博，ギャラリー
==========

TAG 大航海時代

SENTENCE
大航海時代のなかの日本（15～17 世紀中頃）

東アジアには，中国を中心とする国際秩序がありましたが，ヨーロッパ勢力の
東アジアへの進出はこの体制を崩し，また多くの文物をもたらしました．
特に鉄炮とキリスト教の影響は大きく，戦国時代にあった日本では，
戦いの方法が変化し，軍団編成や兵農分離が促進され，やがて統一政権が
強力な中央集権国家を作りあげました．

展示場風景
手前が南蛮屏風（復元模写・17 世紀）と奥が御朱印船模型

御朱印船模型（17 世紀）
中国のジャンク，ヨーロッパのガレオン船の長所をとりいれ，日本風に
改良した当時としては最新の船舶といえます．

鉄炮伝来
日本の鉄炮は，火ばさみの向きや機関部が西洋のものと異っており，
形式から東南アジアで改良されたものと推測されます．

KEYWORD
国家，時代，日本，統一，影響，多く，航海，戦国，アジア，東，政権，屏風，
中国，国際，秩序，ヨーロッパ，勢力，進出，体制，文物，鉄炮，キリスト教，
戦い，方法，変化，軍団，編成，兵農，分離，促進，強力，中央集権，南蛮，
模写，御朱印船，ジャンク，ガレオン船，長所，日本風，改良，最新，船舶，
伝来，火ばさみ，機関部，西洋，形式，東南

==========

TAG 印刷文化

SENTENCE
印刷文化（8～17 世紀）

中国では，宋代になると印刷業が盛んになり，大量の書物の出版が行われ，
日本にも多くの版本が輸入されました．その影響をうけて，日本でも
平安時代中頃から仏教経典を中心にして，各種の書物の出版が本格的に
行われるようになりました．その場所も，最初は京都・奈良周辺の寺院から，
次第に地方にまで広まっていきました．またその対象も，仏書以外の
儒書・漢詩文集から『源氏物語』等の仮名文学書にまでおよぶようになりました．
なお，展示品は毎月展示替えをしております．

展示場風景

宋版後漢書（国宝・12 世紀）
日本に輸入された宋版の一例．卑弥呼の記述で有名な東夷伝の部分．

版本伊勢物語（古活字版・17 世紀）
木活字を使用し，挿絵版画を挿入した優美な古典文学の版本．

KEYWORD
時代，日本，文化，影響，多く，地方，奈良，平安，京都，印刷，経典，文学
書，仮名，中国，宋，業，盛ん，大量，書物，出版，版本，輸入，中頃，仏教，
各種，書物，本格，場所，周辺，寺院，次第，対象，仏書，儒書，漢詩，文集，
源氏物語，宋版，後漢書，国宝，一例，卑弥呼，記述，有名，東夷伝，部分，
伊勢物語，古活字版，木，活字，使用，挿絵，版画，挿入，優美，古典，文学，
最初

==========

TAG 百姓の世界

SENTENCE
百姓の世界

人びとの大半は，村で暮らしていました．村の運営は，百姓と呼ばれた

農民たちにまかされていました．領主は，支配に支障がなければ，
農民たちの営みに干渉はしませんでした．

入口からの展示場風景
村びとの日常にとって，最大の関心は安定した耕作でした．

遍路道の道標
これらを頼りに，遠方へ出かけることも稀にはありました．

大庄屋豊島家の模型
村にあって大きな力をもった富農の屋敷，主屋だけで約 130 坪もあります．

KEYWORD
村，最大，領主，人びと，百姓，世界，村びと，大半，運営，農民，支配，支障，
営み，干渉，日常，関心，安定，耕作，遍路道，道標，遠方，稀，大庄屋，豊島家，
力，富農，屋敷，主屋
==========
TAG 都市の繁栄

SENTENCE
都市の繁栄

都市には武士と商人・職人が居を構えました．商工業などに携わった人々は，
武士の思惑をこえて，自分たちの文化を創り出すようになっていきました．

江戸橋広小路復元模型
手前の橋は日本橋，その向こうに町人たちが作り上げた街がひろがります．

展示室の一部
越後屋の大看板と出世双六（すごろく）．双六では，理想とする町人像と
誘惑とが交差しています．

長崎の唐商い
オランダとの交易を思い浮かべますが，実際には中国文化が多く入ってきました．

KEYWORD
文化，多く，都市，武士，職人，中国，繁栄，長崎，商人，居，商工業，思惑，
自分，江戸橋，広小路，橋，日本橋，町人，街，越後屋，看板，出世，双六，
理想，町人像，誘惑，交差，唐，商い，オランダ，交易，実際
==========
TAG 道と旅

SENTENCE

この時代には，人と物資が全国規模で動き廻るようになりました．
人々にとって寺社参詣を名目とした旅は楽しみとなり，各地には特産品が生まれ
あちこちへもたらされました．

最上川舟運と紅花
紅花を載せて最上川を下っていった川舟の模型です．紅花は河口の酒田で
積み替えられ，京へ送られていきました．

展示場風景
日本海を行き交った船の模型と，積み荷の数々です．

伊勢国椋本村旅篭角屋
京阪から伊勢詣でに出かけた人たちが泊まった宿屋の店先です．

KEYWORD
人々，人，各地，物資，全国，規模，道，旅，寺社，参詣，名目，楽しみ，
特産品，最上川，舟運，紅花，川舟，河口，酒田，京，日本海，船，積み荷，
数々，伊勢国，椋本村，旅篭，角屋，京阪，伊勢，詣で，宿屋，店先
==========
TAG 躍動する民衆

SENTENCE
躍動する民衆

江戸時代のおわりごろ，各地にいた民衆は，それぞれの生産活動や遊芸などの
場で，新しい時代を切り開く力を着実に培っていました．明日への可能性を
秘めていた民衆が，自ら作り上げた文化の数々を示します．

展示場風景
養蚕を生業とした人びとの技術革新の様子と，商いをとおした視野の広がりを
示しています．

からくり人形
生産手段として開発された歯車は，遊びのなかへも転用されていきました．

阿波人形の頭
人形芝居に用いられた数々，演目に応じてさまざまに使い分けられました．

KEYWORD
時代，文化，各地，技術，民衆，生業，革新，江戸，人びと，可能性，躍動，
商い，数々，生産，活動，遊芸，着実，明日，自ら，養蚕，様子，視野，広がり，
からくり，人形，手段，開発，歯車，遊び，転用，阿波，頭，芝居，演目
==========

TAG 文書と絵図は語る

SENTENCE
文書と絵図は語る

それまでの時代では支配者のための文物であった文書と絵図が，江戸時代
後半になるにしたがい民衆のなかに取り込まれていった様子を
展開しています．

展示場風景
周防（山口県の一部）国絵図の原寸大写真で，村名のすべて，主な道筋などが
描かれています．

村役人の執務
記録として文字に残すことの重要性が人びとに認識された時代です．

伊能図
伊能忠敬は，全国の街道や海岸線を中心とした正確な地図を作成しました．

KEYWORD
時代，文書，村，民衆，文字，重要，文物，江戸，全国，絵図，様子，支配者，
後半，周防，山口県，原寸大，写真，村名，道筋，役人，執務，記録，認識，
伊能，図，忠敬，街道，海岸線，正確，地図，作成
==========

TAG 都市の風景

SENTENCE
都市の風景

もともと生まれも育ちも違うたくさんの人々が集まり，肩寄せあって
暮らしていく場所が「都市」です．そんな都市には，人を引きつける
どんな力があるのでしょうか．今日も都市に集まり続ける人たちは，何を思い，
何を願い，何を夢見て暮らしているのでしょうか．そしてそれは，われわれの
祖先たちの暮らしとどのようにつながり，どのようにかけ離れたものに
なっているのでしょうか．ここでは，寺院や神社のお祭や行事などを
手がかりに，他でもない今のわれわれの多くが日々生きているそのような
「都市の暮らし」の広がりについて考えます．

法善寺水掛不動模型
小説や歌などで全国に広く知られている大阪の法善寺横町の実物大模型とともに，
民俗的な歴史をもった都市の「盛り場」のひとつの姿を示しています．

笠森稲荷大明神模型

何もかも合理的で新しいものに貫かれているかに見える都市の暮らしにも，
さまざまな不安や葛藤はあります．そのような時，一見古臭いものに見える
神仏の加護を人々は求めたりもします．きらびやかな流行と
めまぐるしい変化に駆り立てられる都市の暮らしの中にも，そのような
信仰はひっそりと入り込んでいます．

展示室の一部
パネルや映像なども使いながら「都市の暮らし」のさまざまな側面を
紹介していきます．手前に見えるのは十日戎の縁起物です．

KEYWORD
神社，人々，信仰，人，暮らし，都市，行事，歴史，変化，場所，寺院，
全国，力，広がり，実物大，風景，生まれ，育ち，今日，祖先，祭，法善寺，
水掛不動，小説，歌，大阪，横町，民俗的，盛り場，姿，笠森稲荷，大明神，
合理的，不安，葛藤，神仏，加護，流行，パネル，映像，側面，十日戎，縁起物
==========
TAG 村里の民

SENTENCE
村里の民

弥生時代から近年に至るまで，田を耕し，米をつくることが，日本の農村の
基本でした．人手のいる米づくりをするために，村人たちは協力し，助け合い，
そこに自然ときまりがうまれました．村の人々の生活をみてみましょう．

鹿嶋様
村の外から邪悪な霊や危険な人物が侵入するのを防ぐ呪術的装置．

展示室入口
中央は歳神様と種子俵．

アエノコト
家の永続を願い稲霊を祀る神事．

KEYWORD
時代，日本，弥生，人々，生活，村，米，農村，村里，民，近年，田，基本，
人手，村人，協力，助け合い，きまり，鹿嶋様，外，邪悪，霊，危険，人物，
侵入，呪術，装置，歳神様，種子俵，アエノコト，家，永続，稲霊，神事
==========
TAG 山の人生

SENTENCE
山の人生

山村の人々は，豊富な自然資源を巧みに利用して村里や町に供給し，多彩な
民俗をはぐくみ，独自の生活様式をつくりあげてきました．焼畑耕作に
携わる人々，木地屋・マタギなどの生活と，これら山棲みの人たちの
精神生活を紹介します．

焼畑耕作
ヒエや粟などの雑穀を焼畑で耕作しました．

マタギの道具
マタギは山中に熊や鹿を追い求めた狩猟者で，東北地方を中心に広く活動しました．

山の怪異
天狗，山姥，山の神など，さまざまな形相と風体の神像．

KEYWORD
人々，生活，神，像，地方，耕作，活動，民俗，山村，精神，村里，山，人生，
豊富，自然，資源，巧み，利用，生活様式，町，供給，多彩，独自，焼畑，木
地屋，マタギ，山棲み，ヒエ，粟，雑穀，道具，山中，熊，鹿，狩猟者，東北，
怪異，天狗，山姥，山の神，形相，風体
==========

TAG 海浜の民

SENTENCE
海浜の民

海と日本人との間には海幸彦の昔から，長く深い歴史があります．
海に生きる人々にとっては，海は絶えず恐怖と畏敬の対象でした．海浜に
生きる人たちの多様な生活形態をご覧下さい．

鰹船
高知県土佐清水市の「龍王丸」

エビス神像
漁民に大漁をもたらす神として広く信仰されました．

舟屋模型
中央の船は「トモプト」と呼ばれます．

KEYWORD
人々，生活，信仰，神，像，歴史，対象，船，日本人，多様，民，海浜，海，
海幸彦，昔，恐怖，畏敬，形態，鰹船，高知県，土佐清水市，龍王丸，エビス神，
漁民，大漁，舟屋，トモプト

==========

TAG 南島の世界

SENTENCE
南島の世界

北は屋久島・種子島からトカラ列島・奄美諸島・沖縄諸島を経て，わが国
最南端の八重山諸島に至る，1000 キロメートルにわたる南海の島々を
南島と呼びます．この島々は，ある時はさらに南方からの文化を日本本土に
伝える海上の道であり，一方では日本本土や中国大陸の文化を受容しつつ
独自の民俗文化を形成してきました．

シーサーと石敢当
シーサーは村の守り神．石敢当は悪霊をはらう魔除けの石（ともに沖縄）．

古宇利島のウンジャミアシビ（海神の祭）のビデオ映像
古宇利島には海神を迎える祭が伝わります．

来訪神
ニライカナイという海上遥（はる）かな他界から年に一度訪れてきます．
左からパーント，ミルク，フサマラー．

KEYWORD
日本，文化，形成，大陸，神，村，中国，世界，道，民俗，南島，他界，海，
祭，映像，独自，北，屋久島，種子島，トカラ列島，奄美諸島，沖縄諸島，
わが国，最南端，八重山諸島，南海，島々，南方，本土，海上，受容，シーサー，
石敢当，守り，悪霊，魔除け，石，沖縄，古宇利島，ウンジャミアシビ，海神，
ビデオ，来訪，ニライカナイ，パーント，ミルク，フサマラー

==========

TAG 再生の世界

SENTENCE
再生の世界

死後の世界に関わる「他界」の空間は日本人の民俗世界の基層を占めてきました．
天空他界・山中他界・海上他界の三種の考え方と，こうした他界と現世との
交流を儀礼的に表現したものとしての山車・神輿，民俗芸能などについて
展示を行っています．

立石寺準提堂模型
山中には神霊が鎮まると同時に死者の霊が登ると信じられてきました．

民俗仮面

日本人の心意と信仰を象徴的にあらわすもの．人，神（仏），鬼，動物の
四系統に分類されます．

海上他界
他界におもむく船．手前より鹿嶋船，流し雛，精霊船の模型．

KEYWORD
交流，信仰，神，人，芸能，世界，船，日本人，民俗，空間，象徴，現世，
他界，死後，再生，霊，山中，海上，基層，天空，三種，儀礼的，表現，
山車，神輿，立石寺，準提堂，死者，仮面，心意，仏，鬼，動物，四系統，
分類，鹿嶋船，流し雛，精霊船
==========

TAG 文明開化

SENTENCE
文明開化

文明開化とは，西欧の文明を取り入れることによって，近代社会に
生まれ変わろうとしたひとつの大きな変革です．これは明治政府によって
強行されたものですが，庶民の間にも，自由や権利を勝ち取ろうとする意識を
うながしました．

開花万華鏡
西欧文明の摂取による変化と変わらない庶民生活の姿を 24 台の
テレビモニターで紹介します．

明治 23 年製山葉オルガン（国産）
オルガンの普及で，唱歌は学校教育に定着しました．オルガンの曲が聞けます．

KEYWORD
生活，政府，庶民，社会，西欧，自由，定着，変化，姿，近代，明治，文明，
開化，摂取，変革，大きな，強行，権利，意識，万華鏡，テレビモニター，
オルガン，国産，普及，唱歌，学校，教育，曲
==========

TAG 産業と開拓

SENTENCE
産業と開拓

日本が近代国家として生まれ変わるために，明治政府はさまざまな
近代化政策を行いました．そうした政策のなかから，交通・運輸網の整備，
生糸生産と海外貿易，近代製鉄の開始，北海道の開拓とその「開拓」に苦しんだ
アイヌに焦点をあてて紹介します．

生糸と海外貿易
明治から昭和の戦前期まで，外貨獲得の花形だった生糸の生産から輸出までを
女工の視点から再現しています．

KEYWORD
国家，日本，政府，交通，生産，近代，明治，製鉄，北海道，開拓，アイヌ，
視点，産業，近代化，政策，運輸，網，整備，生糸，海外，貿易，開始，焦点，
昭和，戦前期，外貨，獲得，花形，輸出，女工，再現
==========
TAG 都市の大衆の時代

SENTENCE
都市の大衆の時代

文明開化がもたらした，都市化と消費生活．1920 年代に，東京の人口は
300 万人を突破します．1923 年に襲った関東大震災は，そうした都市化の
かかえる人口・住宅・交通の問題に対応をせまるものでした．
このコーナーでは，そうした震災の問題や，資本主義経済がもたらした
消費生活と大衆の娯楽に焦点をあてます．

文化住宅
関東大震災後に中流向けに建てられた同潤会アパートの台所と居間．

浅草の街並実大復元模型
大正時代の娯楽の王様は活動写真と呼ばれた無声映画．道路の左手の映画館で
４本立映画上映中．

カフェ・うるとら
映画館の向いにある浅草の猥雑な露地裏に大衆娯楽の原点をみる．

KEYWORD
時代，文化，原点，生活，道路，都市，経済，交通，活動，写真，文明，開化，
関東，震災，消費，無声，映画，大衆，資本，主義，焦点，都市化，東京，人口，
突破，住宅，問題，対応，娯楽，震災後，中流，同潤会，アパート，台所，居間，
浅草，街並，実大，大正，王様，映画館，上映，猥雑，露地裏

==========

5 Virtual奈良ウォークスルーシステムについて

5.1 概要

Virtual 奈良ウォークスルーシステムは，VisTA_walk システムを利用して開発した奈良の世界遺産を題材にした仮想展示である．本システムでは，仮想の奈良の町を歩きまわり，世界遺産に指定された寺院を訪問し，情報を引き出すとことができる．本システム単体で利用することも可能であるが，仲介システムを見学コースの推奨システムとして使用することが可能である．この場合，まず，仲介システムを操作し興味のある展示物 (この場合は寺院そのものやその建築の一部など) を選択する．次に 8 種類のパターンのうちの 1 つを選択する．すると，そのパターンに含まれる展示物 ID が Virtual 奈良システムに送られる．Virtual 奈良システムは，これらの展示物を奈良の北東部から順に訪ねる見学コースを自動生成し，自動航行モードで見学者を案内する．

5.2 Virtual奈良システムのディレクトリ構成

Virtual 奈良システムでは，地形や建築物などの iv ファイルと展示物オブジェクト (寺院建築など) の情報をおさめた html 形式のファイルを下記のディレクトリにおく必要がある．

- V_nara: プログラム本体
- V_nara/html: 寺に関する HTML
- V_nara/data: ジョイントされた全体データ

5.3 実行方法

Virtual 奈良システムの起動方法は次の通りである．

```
% setenv SA_NETSCAPE /usr/local/bin/netscape.3.01
% vnara -M mapfile -m minimapfile -s portNo -p hostname  &
```

環境変数 SA_NETSCAPE には，vnara から起動される Netscape を指定する．
vnara の引数のオプションは，以下の通り．

- -M <mapfile>: マップファイルを指定
- -m <minimapfile>: 鳥観図用マップファイルを指定
- -s <portNo>: 仲介プログラムとのソケットのポートを指定
- -p <hostname>: pfinder のホスト名を指定
- -h : ヘルプを表示

マップファイルはいずれも.iv ファイルである．

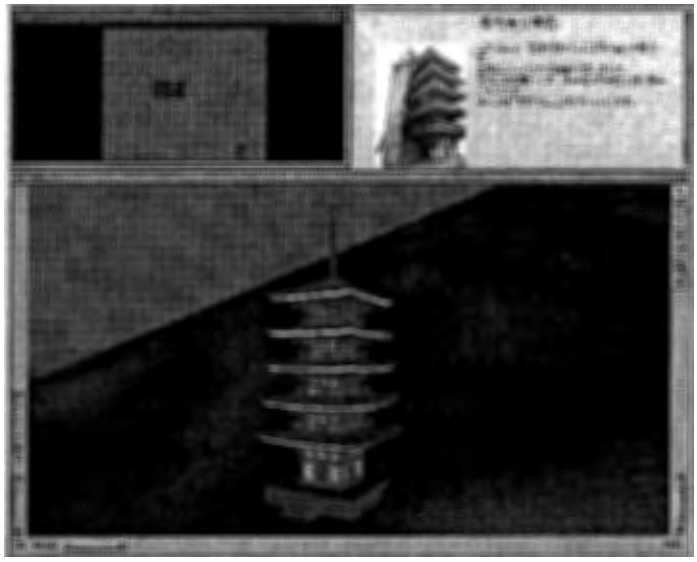


図 4: Virtual 奈良ウォークスルー画面一例

vnara の画面の一例を図 4 に示す．画面下に探検ウインドウ，上左側に鳥観図，上右側に情報ウインドウ (ブラウザ) が表示される．

参考文献

[1] 門林 理恵子, 西本 一志, 角 康之, 間瀬 健二: 学芸員と見学者を仲介して博物館展示の意味構造を個人化する手法の提案, 情報処理学会論文誌, Vol. 40, No. 3, pp. 980-989 (March 1999).

[2] 門林 理恵子 間瀬 健二: 豊かな博物館体験の実現への試み: 仲介システムによる展示の個人化, インタラクション'99 論文集, pp. 69-70 (March 1999).

Part II

Vol. 40 No. 3 情報処理学会論文誌 Mar. 1999

学芸員と見学者を仲介して博物館展示の意味構造を個人化する手法の提案

門林 理恵子† 西本 一志†
角 康之† 間瀬 健二†

博物館の展示は，学芸員の専門知識や関心の体系的表現であるが，ある視点からの構造化の一実現例であり，しかも一方的に見学者に提示される．このため，様々な興味や知識を持つ見学者の知的欲求を常に満たすことは難しい．そこで本稿では，学芸員と見学者を仲介し，博物館展示の意味的関連に基づく構造を見学者ごとに個人化する手法を提案する．本手法では，展示物等に付与される説明文を，キーワードを元に統計処理することで，展示物間の関連に基づく構造を2次元空間に可視化する．まず元の展示の構造を示す展示空間を構成し，次に見学者の興味に基づく興味空間を作る．最後に両者を融合して個人化空間を作成する．個人化空間に可視化された展示の構造は，学芸員の視点からの関連を保持しつつ，同時に見学者の視点を反映したものとなる．こうすることにより，既存の，見学者の立場を中心とした個人化手法における，展示が断片化して関連が失われ，かえって理解が困難になるという問題を回避することができる．これらの空間を用いることで，同じ展示を見学者ごとに個人化することが可能となる．さらに，これらの空間が学芸員へもフィードバックされることにより，学芸員自身が展示についての新たな視点を獲得できるが，これは従来の博物館の展示や既存の個人化手法では困難であったものである．本稿では，本手法の詳細とその実施例，さらに評価についても述べる．

Personalizing Semantic Structure of Museum Exhibitions by Mediating between Curators and Visitors

Rieko Kadobayashi,† Kazushi Nishimoto,† Yasuyuki Sumi† and Kenji Mase†

Museum exhibitions are thought to be well organized representations of the expert knowledge of curators, but they are just one example of structures of knowledge among many possibilities, given to museum visitors in a one-sided way. Therefore, traditional museum exhibitions can hardly meet the vast requirements of general visitors who possess a variety of interests. In this paper, we propose a method for personalizing the semantic structure of museum exhibitions by mediating curators and visitors. The semantic relations of displays are visualized as a two-dimensional spatial structure based on the viewpoints of the curators and visitors separately, and then together. The structures reflect the interests of the visitors, while maintaining the knowledge of the curators. We disucuss the detail of the method and show an example of personalization. Evaluation results through a subjective experiment is also given.

1. はじめに

博物館（本論文では美術館や資料館なども総称して博物館と呼ぶ）は，単にモノを展示して見せる場ではなく，社会全体で知識を共有する場である[2]．しかし従来の博物館は必ずしもそのようには機能してこなかった．その最大の理由は，興味や知識のレベルの異なる不特定多数の見学者に対して，ただ一つの同じ内容の展示しか提供できなかったことにある．つまり，見学者が展示に興味を持てなかったり，内容を十分に理解したりできず，学芸員が展示を通して伝えようとする知識がうまく伝わらずに終ってしまうからである．したがって，博物館の主たる情報提供手段である「展示」を，訪れる見学者の要求に沿って様々な側面で個人化することが必要である[2]．そこで，学芸員すなわち情報の発信側が主導で行ってきた博物館での情報提供を，情報を受信し利用する側の見学者を主体とし，

† 株式会社 エイ・ティ・アール知能映像通信研究所
ATR Media Integration & Communications Research Laboratories

一人一人異なる要求に応じて適切に行うための研究が盛んとなりつつある．

たとえば，数多くの「モノ」を含んだオリジナルの展示から，見学者が指定した作者の作品だけを選択し，どの展示室にあるかを教えてくれるシステムがある[7),8)]．元の展示の中から見学者の興味に基づいて見学すべきモノを抽出するという手法は，「モノの個人化」ということができる．モノの個人化の場合，モノとそれに関する学芸員の知識が構造化されてできた展示の一部を切り出すために，本来の関連が全く失われ，そのためにかえって内容の理解が困難になることが懸念される．

これに対し，携帯型の情報機器を利用して「情報の個人化」を図るシステムも存在する．情報の個人化とは，展示されたモノの説明として，説明札やパネルなどを通して提供される情報の表現方法や伝達方式を，見学者の状況に応じて適切に変更することである．状況とは，見学者の国籍や年齢，現在観賞している展示，これまでの訪問回数など，見学者に関わる様々な情報を指す．これらの状況に応じて，システムが自動的に，あるいは見学者が自ら選択することで，現在見ている展示の情報を端末に表示する，英語で説明をする，文字だけでなく音声で説明する，説明の内容や詳細度を変更するといったことを行い，見学者それぞれのレベルに応じて展示の情報を個人化する[1),6),10)~12)]．

ところで，博物館の展示は，学芸員が展示のテーマに沿ったモノを選び出し，説明札やパネルで提供する情報を選別し，モノどうしを関連づけて展示に意味的構造を持たせ，その構造がうまく伝わるようにモノを展示室へ配置することで成り立つ．換言すれば，展示の構成要素には，モノや情報だけでなく，意味的関連に基づく「構造」や「配置」も含まれる．したがって，個人化はモノや情報のみならず，構造や配置についても行う必要がある．

展示物の意味的関連に注目した取り組みとして，バネモデルで表現された展示物の関連を，Maplet[14)]や，Semantic map[12)]と呼ばれるJAVAアプレットによって表示するものがある．前者は，学芸員があらかじめ準備した展示物の関連をそのままに，表示する部分を制限できるだけであり，展示の構造自体を変化させるものではない．後者は，学芸員があらかじめ準備した展示物の関連のうち，見学者が選択したキーワードによる関連のみを抽出し，表示する．これは，指定されたキーワードのみを含む展示物のリストを提示する，従来の「モノの個人化」手法と基本的に同じであり，表現手法が異なるだけである．したがってこれらも，見学者の興味に応じて展示の構造そのものを個人化するものではない．

本論文では，学芸員と見学者を仲介し，博物館展示の意味的関連性に基づく構造を個人化する手法を提案する．従来の手法では，見学者が与えられた選択肢を選択することで個人化が実現されており，これまでの展示が学芸員から一方的に与えられたものであったのと同様，見学者側が一方的に個人化するものである．これに対し，本手法は，学芸員の知識によって構成された展示を，見学者の興味だけを利用して個人化するのではなく，いったん見学者の興味を学芸員の知識によって膨らませ，それから展示を個人化するものである[4),5)]．このようなアプローチを採用する理由は，見学者の狭くて浅い興味のみを利用するよりも，まずその興味を学芸員の広くて深い知識で膨らませてから利用する方が，見学者にとって有益な個人化が実現できると考えるからである．

さらに本手法は，展示が本来持っていた関連を保持しつつ，見学者の興味というあらたな視点の導入によって，専門家の視点では隠れていた可能性のある関連性が明らかになるという利点を持つため，見学者が自分の視点で展示を見ることができ理解しやすくなるだけでなく，専門家も見学者からの興味のフィードバックにより新たな関連性に気づくという効果がある．これにより，博物館が知識を共有する場として機能するために必須と考えられる双方向コミュニケーション[2)]が非同期ながらも実現できるようになる．

以下，まず2章で展示物の関連を統計的手法により処理して，展示の意味的構造を2次元空間に可視化しつつ，個人化を行う手法について述べる．次に3章で，関連性が可視化された空間の利用方法を議論する．4章では，本手法の主観評価実験の結果を報告し，5章でまとめを述べる．

2. 展示の意味的構造の個人化手法

2.1 統計的手法による関連の可視化

筆者らはこれまでに，情報可視化によって共有情報を個人化し，その結果を利用することで視点の異なる人同士の相互理解を促進する手法を提案している[13)]．そこでは，情報や視点の可視化のために双対尺度法[9)]という統計手法を使用している．

本研究においても，学芸員の知識や見学者の興味を定量的に処理し，展示物の関連を空間構造として可視化するために，双対尺度法を使用する．加えて，見学者の持つ浅い知識に基づく興味をそのまま使用するのではなく，学芸員の持つ深い知識を用いてその興味を

拡張した上で展示の個人化を行う手段を新たに組み込む．これによって，見学者の浅い知識に基づく一方的な個人化を脱却し，学芸員の知識を効果的に活用した個人化が達成されることが期待できる．

ところで双対尺度法とは，複数の数量化属性で構成されたオブジェクト集合が与えられたときに，オブジェクトどうしの属性共有性と属性どうしの共起性を顕在化するように，オブジェクト集合と属性集合にそれぞれ得点数量を与えることによって，構造を可視化する手法である．本研究では，展示されているモノに付与される説明文をオブジェクト，説明文に含まれるキーワードを属性として扱う．キーワードは，形態素解析によって説明文から自動抽出した名詞と未知語からなり，重みの初期値はすべて同じとする．なお，学芸員や見学者が重みを変更することは可能である．

展示物の関連は，双対尺度法によって多次元空間内のオブジェクト間の距離として求まる．これを利用者にとって直観的でなじみやすいものとするために，第1主成分と第2主成分を軸とする2次元空間にマッピングして可視化する．このとき，それぞれの主成分の寄与率に基づいた主成分得点の補正を行ってからオブジェクトを配置する[9]．この2次元空間内の説明文オブジェクトの配置を観察することで，近くにあるものほど関連性が強く，遠くになるほど関連性が弱くなることを利用者は直観的に知ることができる．

展示の個人化は，以下に示す順に，展示に反映されている学芸員の知識，展示に対する見学者の興味，それらを仲介した結果を，上述の手法で空間に構成することによって行われる．各ステップにおいて空間を構成するときに用いられる説明文オブジェクトとキーワードの関連を模式的に示したものが図1である．図中0で示す領域は，その領域に対応するオブジェクトが，その領域に対応するキーワードを含まないことを示す．

(1) **展示空間：学芸員の知識空間の可視化**

展示室や展示物に付与されているすべての説明文オブジェクトからなる集合 O_c と，O_c のすべての構成要素から自動抽出されるすべての重みつきキーワードの集合 K_c とによって学芸員の知識は構成されているものとする．この O_c と K_c に対して双対尺度法を適用することによって，すべての説明文オブジェクトおよびキーワードの関係を2次元空間構造として可視化する．この空間は学芸員が持っている知識の構造を可視化したものとみなせる．この空間を展示空間と呼ぶ．

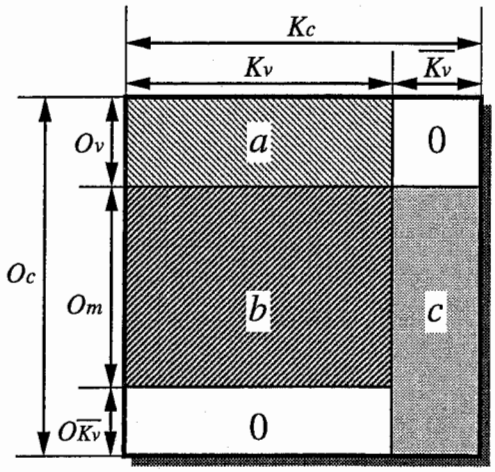


図1 各空間を構成するオブジェクトとキーワードの関係の模式図
Fig. 1 Diagram of relationship between objects and keywords organizing each space

(2) **興味空間：見学者の興味のみに基づく再構成空間**

O_c に含まれる説明文オブジェクトの中から，見学者が興味を持つオブジェクトのみを選択することによって，O_c の部分オブジェクト集合 O_v と，O_v の構成要素である説明文オブジェクトに含まれるすべての重みつきキーワードの集合 K_v を生成する．したがって，K_v は K_c の部分集合となる．この O_v と K_v に対して双対尺度法を適用することによって，O_v に含まれるすべての説明文オブジェクトと K_v に含まれるすべてのキーワードからなる関連構造を2次元空間構造として可視化する．つまり，見学者が選択した説明文オブジェクトとそれに含まれる重みつきキーワードのみで，まず空間の基底を求め，これらを配置する．

次に，見学者が選択しなかった説明文オブジェクトを空間に配置する．O_m に含まれる説明文オブジェクトは，K_v に含まれるキーワードを含んでいるので主成分得点を求めることができ，この空間上に配置することが可能である．一方，図1に $O_{\overline{Kv}}$ で示す，K_v に含まれるキーワードを一切含まない説明文オブジェクト集合に含まれるオブジェクトについてはこの空間上には理論的に配置できないので，廃棄される．

このようにして，見学者が選択した説明文オブジェクトに基づいて展示全体を再構成することができる．この空間を興味空間と呼ぶ．ただし，キーワードは学芸員が付与した説明文から形態素解析によって自動抽出したものであるの

で，興味空間にも学芸員の知識が反映される．

(3) **個人化空間：学芸員の知識と見学者の興味が融合された空間**

説明文オブジェクト集合 $O_v \cup O_m$ とキーワード集合 K_v に対して双対尺度法を適用することによって，$O_v \cup O_m$ に含まれるすべての説明文オブジェクトと，K_v に含まれるすべてのキーワード，すなわち，図1中の a と b の領域に含まれるオブジェクトとキーワードの関係を2次元空間構造として可視化する．この空間は，キーワード集合 K_v によって張られているため見学者の興味による制約を受けている．しかし，一方で O_m に含まれる説明文オブジェクトによって，K_v に含まれるキーワード間に見学者によっては導入されなかった新たな関係が導入される．この関係はそもそも学芸員によって与えられたものであるから，結局この空間は学芸員の知識構造と見学者の興味を融合させることによって再構成した展示空間であるとみなせる．この空間を個人化空間と呼ぶ．

ここで注意しなければならないのは，この個人化空間は学芸員の知識構造に基づく展示空間の単なる部分構造ではないということである．展示空間には，K_v に含まれないキーワード $\overline{Kv}$ による関連が含まれているが，個人化空間からはこれらのキーワードによる寄与（図1中の c の部分）が削除されている．このことは，単に一部の関連が削除されるのみならず，これらのキーワードによって与えられる関連性によって覆い隠されていたような関連性を顕在化させる効果を持つ．この結果，個人化空間には展示空間になかった関連が現れる可能性がある．

このような学芸員と見学者を仲介しながら進める展示の個人化のプロセスは，見学者の持つ狭くて浅い範囲の知識を，学芸員の持つ広くて深い知識によって少し深く掘り下げるものと見ることもできる．

2.2 関連に基づく個人化の例

前述の仲介による個人化手法を適用した例を示す．展示として利用したのは，国立歴史民俗博物館のホームページ[15]にある常設展示の案内ページである．常設展示室は第1展示室から第5展示室まであり，それぞれが3〜6のテーマを含んでいる．展示室ごとに各テーマとそれに関連する展示物について説明したページが用意されている．1テーマの説明を1オブジェクトとして扱い，全部で25のオブジェクトを得た（**表1**）．

これらを実際の展示で使用される説明文と見なし，

表1 例に用いたオブジェクトの一覧
Table 1 List of objects

展示室	オブジェクト
1	日本文化のあけぼの，稲と倭人，前方後円墳，沖ノ島 律令国家
2	王朝文化，東国と西国，大名と一揆，民衆の生活と文化 大航海時代，印刷文化
3	百姓の世界，都市の繁栄，道と旅，躍動する民衆 文書と絵図は語る
4	村里の民，山の人生，海浜の民，南島の世界，再生の世界
5	文明開化，産業と開拓，都市の大衆の時代

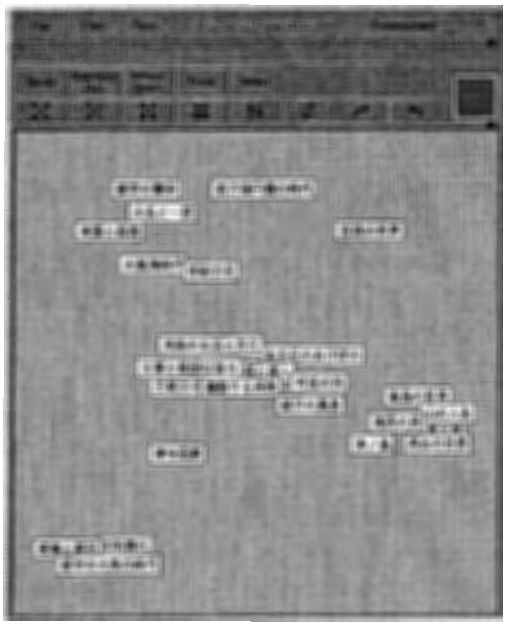


図2 展示空間の例
Fig. 2 Example of exhibition space

提案手法によって関連を可視化した展示空間が**図2**である☆．この空間では，同じ展示室に含まれるオブジェクトはかなり近い位置に配置されている．特に，第4展示室と第5展示室に含まれるオブジェクトはそれぞれ関連度が高かった．これに比べると第1展示室から第3展示室までは展示室ごとのまとまりは，多少緩やかなものではあるが，これら全体では比較的まとまっており，この三つの展示室で通史的な展示を行っていることを考え合わせると，展示空間は学芸員の意図した展示構成を比較的よく表現できていると思われる．

図3は，見学者が興味あるものとして“日本文化のあけぼの”，“王朝文化”，“民衆の生活と文化”の三つのオブジェクトを選択して得られた興味空間を示す☆☆．

☆ キーワードも同一空間上に配置可能であるが，簡単のため説明文オブジェクトのみを表示している．図3，図4も同様である
☆☆ 見学者は展示空間に配置されているオブジェクトをマウスでク

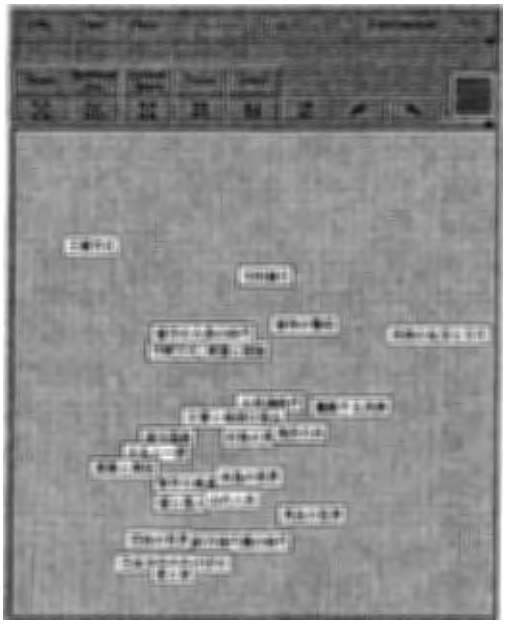


図 3 興味空間の例
Fig. 3 Example of visitor interest space

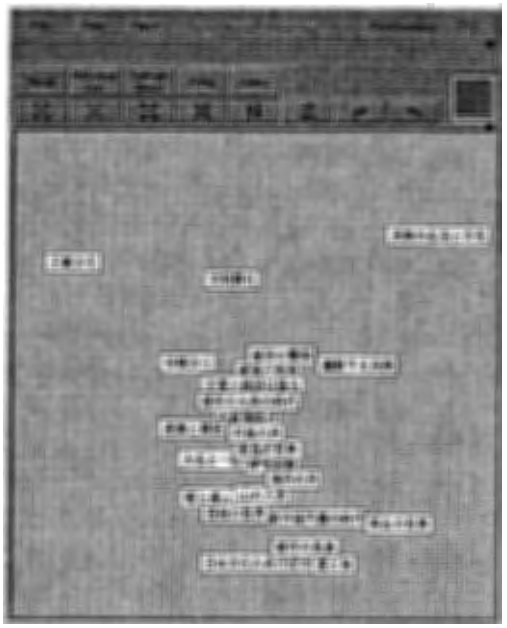


図 4 個人化空間の例
Fig. 4 Example of personalized space

図 3 の興味空間上には，O_m に含まれるオブジェクトも配置されている．一方，展示空間に含まれていた“沖ノ島”というオブジェクトは，上記の三つのオブジェクトが含むキーワードを一つも含んでいなかったため，$O_{\overline{Kv}}$ に含まれるものとして削除されている．

図 4 は，見学者の興味と学芸員の知識を融合させて創り出した個人化空間である．対象としているオブジェクトは興味空間に残った 24 のオブジェクトである．O_m によって導入される関連性の影響で，興味空間とは違った構造が得られているのがわかる．

図 2，図 3，図 4 を比較して全体的な構造の変化についてみると，展示室ごとのまとまりは，興味空間ではかなり緩くなり，学芸員の知識が反映された個人化空間ではまた少し緊密なものとなったが，当初の展示空間に比較すると緩やかなものとなっている．これは，当初の展示の構造が見学者の興味を反映して解体され再構成されたためと考えることができる．

3. 関連が可視化された 2 次元空間の利用方法

学芸員の知識，見学者の興味，そして両者を仲介して得られた結果は，展示空間，興味空間，個人化空間として 2 次元空間に可視化できることを示した．これらの空間は，見学者および学芸員が直接観察して利用することも，システムが自動的に処理し，1 章で述べたモノ，情報あるいは配置の個人化に利用することも可能である．以下，両方の利用方法について述べる．

リックするだけで選択できる．

表 2 距離変化のパターン
Table 2 Patterns of transition of distance

パターン	展示空間	興味空間	個人化空間
1	近い	近い	近い
2	近い	近い	遠い
3	近い	遠い	近い
4	近い	遠い	遠い
5	遠い	近い	近い
6	遠い	近い	遠い
7	遠い	遠い	近い
8	遠い	遠い	遠い

3.1 見学者と学芸員が観察する

2 次元空間上に配置されたオブジェクト間の関係は，距離を基にして，近い，遠いの 2 通りに分類できる．そして，展示空間，興味空間，個人化空間という順に生成される 3 種類の空間のそれぞれにおける距離の変化のパターンは，表 2 に示すように 8 通りある．これらのパターンが見学者や学芸員にとってどのような意味を持つのかをパターンごとに議論する．

パターン 1:

このパターンに属するオブジェクト対の関連は，学芸員の知識の空間においても見学者の興味の空間においても同様に「近い」と意識されていたものであり，両者を仲介して得られた個人化空間においても変化がない．学芸員が展示を通し

て伝えようとした関連性に，当初から見学者も気付いており，納得のいく組み合わせといえる．

パターン 2:

この変化は，学芸員と見学者の双方が同様に関連性が高いと意識していたものが，学芸員の知識に見学者の視点を導入した結果，関連性が低くなるという，見学者にとっても学芸員にとっても意外性を与えるものである．

パターン 3:

この変化は，学芸員が用意した展示空間では関連性が高いとされていたにもかかわらず，見学者の興味空間では関連性が薄れてしまったが，両者を仲介することによって再度近く配置されることになり，その関連性があらためて見学者に伝えられることになると見ることができる．従来の展示方法では，学芸員が伝えようとしたにもかかわらず，見学者が見落としたままになるものに，見学者が気付くことができると解釈できる．また既存の「モノ」や「情報」の個人化手法においては，見学者の立場が優先されるので，このパターンのように興味空間において遠いすなわち関連性が低いモノや情報が提供される可能性は低い．このように，学芸員と見学者を仲介しつつ個人化する本手法は，博物館における伝統的な展示でも，既存の個人化手法でも提供できなかったものを見学者に提供できるという利点がある．

パターン 4:

元の展示空間では関連度が高かったが，見学者の興味空間では関連性が薄れ，最終的に関連性が低くなったままというパターンである．これは，学芸員の広い知識に基づく展示を見学者の狭い興味で見たときに，本来存在した関連が見学者の視野から外れてしまったと解釈できる．

パターン 5:

学芸員が意識していなかった関連が，見学者によって示され，仲介機能が見学者の意見を支持する形で両者を融合したと考えることができる．これは学芸員にとっては，見学者からのフィードバックによるあらたな視点の獲得と言える．このような見学者から学芸員へのフィードバックは，従来の博物館の展示においては困難であり，本手法の有効性を示すものである．

ここで注意すべきことは，このパターンが見学者にとって意味がないパターンであるとは限らない点である．つまり，見学者からは自分の興味に応じた納得のいく結果が得られたことになる．学芸員があまり意識していなかった関連性を明らかにすることができたのであるから，見学者にとっては情報の受信者，消費者側から，反対の立場へと転換でき，より積極的に展示を見るきっかけとなる可能性がある．

パターン 6:

学芸員にとってはもとの展示空間と同じ結果が得られており，見学者にとっては関連があると思われたものが，やはり関連性が低かったとなる変化である．見学者の興味によって顕在化した関連性が，学芸員の豊富な知識に基づく展示全体においては局所的な些末な関連であったと取ることができる．

パターン 7:

この場合は，学芸員も見学者も関連が深いとは意識していなかったものが，仲介によって両者の知識と視点が融合されて初めて顕在化する関連である．これもまた，従来の展示や既存の個人化手法では提供できなかったものの例である．

パターン 8:

いずれの空間においても遠くに配置されており，関連性が低いことにかわりがないというパターンである．他のパターンに比べ，見学者も学芸員も積極的な意味を見いだす可能性が低いと思われる．

このように，見学者や学芸員がそれぞれの空間を観察し，説明文オブジェクト間の距離を比較することで，埋もれていた関連性に気付くなどの効果があり，提案手法での個人化は有用であると考えられる．

3.2 システムが利用する

本節では，システムが利用者による空間観察の際の支援や 1 章で述べたモノ，情報，配置の個人化のために，前述のパターン分類を指標として使用する場合について述べる．展示物間の関連性は多次元空間に表現されるが，ここで 2 次元空間における距離を元にしたパターン分類を使用するのは，利用者が観察するのが多次元ではなく 2 次元の空間であり，その利用者の主観に応じて個人化した情報提供を行うには，システムも同じ 2 次元空間を使用するほうがよいと考えるからである．

まず，利用者の空間観察を支援するには，パターンごとに説明文オブジェクトを色分けして表示する，注目するパターンに属するものだけを強調して表示する，特定のパターンに属するものは全く表示しない，など各空間での説明文オブジェクトの表示方法を制御する

方法がある．

次に，個人化のための指標として利用する場合は，特定のパターンに属するものだけを抽出する，パターンごとに配置を決定するなどの方法により，仮想的に展示室を構成する方法が挙げられる．たとえば，知識の範囲を広げたいと望む見学者に対しては，パターン3またはパターン7に含まれるものを選んで展示を構成する，知識を深めたいと望む見学者に対してはパターン1に属するものを中心に展示を構成するといった利用の仕方がある．

この手法は，いわゆる電子博物館に対して特に有効であると考えられる．なぜなら，電子博物館では展示されるモノもデジタル化されており，モノの配置を自由自在に変更できるからである．しかし，インターネット上に存在する電子博物館は，複数のモノをまとめた仮想的展示室や仮想的フロアといった構造を有する場合でも，その配置は固定的であり，見学者ごとにモノの配置も容易に変更できるという電子博物館の利点が有効に活用されていないのが現状である．これからの電子博物館においては，一方的に情報を提供するのではなく，利用者の状況に応じて動的に情報提供シナリオを変更できる仕組みが必要となると考えられるが，提案手法を用いることで，見学者の個々の要求に応じて，個人化された展示を提供できるようになる．

配置も含めた個人化は，実空間に存在する博物館での実施は非常に困難である．しかし，仮想的に構成した展示室をもとに，見学の順序を工夫することで電子博物館の場合に近いことを演出することができると考える．また，モノや情報に関する個人化は，従来の博物館にも電子博物館にも適用可能である．

さて，このようにパターン分類を利用するためには，利用者の観察による主観的な「近い」「遠い」という評価をシステムが定量的に行うことが必要である．説明文オブジェクト間の距離の遠近の判定は，次のように行う．オブジェクト o_i と o_j $(i \neq j)$ の正規化距離 d_{ij} を，

$$d_{ij} = \frac{x_{ij} - \overline{x}}{\sigma}$$

によって求める．ただし，x_{ij} は各空間における o_i と o_j の直線距離，$\overline{x}$ および σ は，それぞれ各空間における全オブジェクト間の直線距離の平均と標準偏差である．そして，d_{ij} が一定の閾値より大きい場合を遠い，小さい場合を近いとする．

図2，図3，図4に示した例を用いて，$d_{ij} > 0.3$ の場合を遠い，$d_{ij} < -0.3$ の場合を近いとした場合の各パターンに属するオブジェクト対の例を**表3**に示す．

表3 各パターンに属するオブジェクト対の例
Table 3 Example object pairs of each pattern

パターン	オブジェクト1	オブジェクト2
1	日本文化のあけぼの	稲と倭人
2	稲と倭人	再生の世界
3	道と旅	海浜の民
4	前方後円墳の時代	印刷文化
5	東国と西国	都市の大衆の時代
6	道と旅	産業と開拓
7	村里の民	都市の大衆の時代
8	道と旅	印刷文化

4. 評　価

3章での議論は，展示空間と個人化空間が学芸員の視点からの展示物間の関連性を，興味空間と個人化空間が見学者の視点からの展示物間の関連性を表現できているという仮定に基づいている．2.2節では国立歴史民俗博物館の常設展示案内のホームページを使用した個人化例を示し，展示空間が学芸員の視点を反映できているのではないかと述べた．そこで今回はまず見学者の立場から，2次元空間に投影された二つの説明文オブジェクト間の距離が，これらのオブジェクトの関連度についての見学者の主観的な評価とどの程度一致しているかについて，主観評価実験による評価を行った．個人化空間が学芸員の視点をどの程度反映しているかについての評価は今後の課題である．

実験には，前述の国立歴史民俗博物館のホームページを使用した．被験者は，同じ研究所に所属する5人の研究者であり，日本史の専門家ではない．実験手順は次の通りである．

(1) 被験者に25の説明文オブジェクトの中から興味のあるものを任意の数だけ選択させる．このとき，被験者には展示空間をそのまま提示することはせず，一つのテキストオブジェクトを1枚のカードに書いたものを25枚用意し，50音順に見せた．なぜなら，展示空間での説明文オブジェクトの2次元配置が，説明文オブジェクト間の関連度についての被験者の評価に影響を与えることが懸念されたためである．
被験者は平均して8個の説明文オブジェクトを選択した．

(2) 被験者が選択した説明文オブジェクトを用いて，興味空間と個人化空間を作成したのち，前述の8パターンのいずれかに属する説明文オブジェクト対を得た．そのうち，パターン1，3，5，7のいずれかに属するものだけを抽出した．これらは，個人化空間での距離が“近い”になるも

表4 被験者の選択によって得られた評価対象説明文オブジェクト対の個数
Table 4 Number of object pairs obtained from each subject's selection

	パターン				
	no. 1	no. 3	no. 5	no. 7	計
被験者 1	81	3	22	2	108
被験者 2	98	2	1	1	102
被験者 3	69	12	23	1	105
被験者 4	66	9	11	16	102
被験者 5	99	2	4	0	105
合計	413	28	61	20	522

表5 説明文オブジェクト対の関連性の主観評価
Table 5 Subjective rating of relevance between two objects

パターン	no. 1	no. 3	no. 5	no. 7
平均	2.66	1.96	2.23	1.75
分散	1.28	1.15	1.51	1.04

のである．それぞれのパターンに属する説明文オブジェクト対の被験者ごとの個数を**表4**に示す．

(3) 各被験者は，自分の選択によって得られた各説明文オブジェクト対に対して，強い関連があるものを5，関連がないものを1とした5段階の評価を行う．たとえば，被験者1の場合は，計108の説明文オブジェクト対について評価を行い，被験者2は計102の説明文オブジェクト対について評価を行った．このとき説明文オブジェクト対は，それがどのパターンに属するかということを被験者から隠蔽するために，ランダムに提示した．

表5は，関連度の評価の平均と分散を示す．この表より，パターン1，パターン5，パターン3，パターン7の順に関連性の主観的評価値に大小の傾向があることがわかる．

まず，パターン1に属する説明文オブジェクト対，すなわち，三つの空間のいずれにおいても距離が“近い”説明文オブジェクト対の関連性が高く評価されているのに対し，パターン7に属するオブジェクト対，すなわち，展示空間と興味空間では“遠く”，個人化空間においてのみ“近い”説明文オブジェクト対の関連性が最も低く評価されたことから，“近い”分類となる2次元空間の多さが，主観的関連性に影響すると考えられる．これは，“近い”もの同士が多次元空間においてクラスタを形成している可能性が高くなる分，共有するキーワードが多いことになり，それが主観的関連性に反映されるのは自然といえる．

次に，パターン5に属する説明文オブジェクト対の関連性は，パターン3に属する説明文オブジェクト対の関連性よりも高いとの評価を得た．展示空間より興味空間における近さのほうが，主観的関連性に影響することを示すものと思われるが，これは，この評価実験が見学者の立場で行われたため，見学者の視点が専門家の視点よりも強く反映されたと考えられる．これについては，残りの4パターンの評価および学芸員の視点での評価も行った上で検証する必要がある．

以上のことから，2次元空間における説明文オブジェクト対の距離は，関連性についての見学者の主観的評価を反映していると言える．この結果は，3.2節で議論したように，システムが自動的に情報の提供方法を見学者の要求に応じてカスタマイズし，配置も含めた個人化を実現できる可能性を示唆するものである．ただしこの点については今後の詳細な検討と評価を要する．

5. おわりに

本論文ではまず，学芸員と見学者を仲介し，学芸員の専門知識が体系的に表現された博物館の展示を見学者の興味と融合させて，展示物の意味的関連に基づき個人化する手法を提案した．

本手法では，展示室や展示物に付与される説明文の集合を学芸員の知識とみなし，それらの説明文からキーワードを自動抽出し，双対尺度法によって統計的に処理した後，2次元空間にマッピングして展示空間として可視化する．そして，展示空間にあるオブジェクトの中から興味あるものを見学者に選択させ，見学者の興味空間を構成する．最後に見学者の興味空間を展示空間に融合させて，個人化空間を構成する．この手法を，国立歴史民俗博物館がインターネット上で公開している常設展示案内のページを展示とみなして適用したところ，展示室ごとにまとまりのあった構造が見学者の興味と融合することで，新たな構造を創造することが確認された．

次に，これらの空間を利用する方法について述べた．オブジェクト間の関連性の強さが2次元空間における距離によって示されることを用いて，展示空間，興味空間，個人化空間のそれぞれにおけるオブジェクト間の距離の変化を8パターンに分類し，各パターンが，空間を直接観察する見学者や学芸員にとってどのような意味合いを持つのかを議論した．さらに，システムが個人化を支援するための指標としてこれらのパターン分類を使用できることも示した．

最後に，提案した手法について見学者の立場から評価を行った．その結果，展示物の関連を説明文を用いて2次元空間に配置する方法は，それらの関連性に

ついての見学者の主観的評価を反映していることがわかった．また，説明文オブジェクト対の距離変化のパターンの分類が，見学者の要望に沿ってモノや情報を選択したり，仮想的な展示室を構成するための指標となりうることが示唆された．

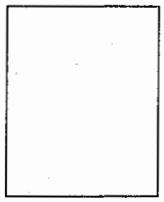
今後の展望として，1章で述べたモノ，情報，関連，配置のすべての側面に渡って個人化された博物館展示を構成する手法の研究が挙げられる．しかも配置変更が容易な電子博物館だけを対象とするのではなく，人々に感銘を与えるすぐれたモノに溢れた従来の博物館も対象としたいと考えている．筆者らはすでに，見学者の状況に応じて適切な展示ガイドを行うC-MAPシステム[12]と，見学者ごとに仮想空間内の案内の仕方を変更するパーソナルガイドエージェントを実現している[3]．このような技術と組み合わて，本論文で提案した手法をもとに仮想的に創出される展示を物理空間中の展示へと対応づけ，適切なガイドを行うことで，あたかも見学者ごとに博物館が存在しているかのような感覚を与えられるようになれば，ようやく博物館という組織が社会全体での知識の交流，共有の場として機能することができるようになると考えている．

謝辞 本研究の機会を与えて下さった（株）ATR知能映像通信研究所の酒井保良会長ならびに中津良平社長に感謝致します．

参考文献

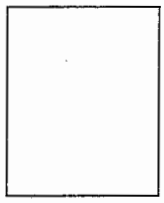
1) Hitzeman, J., Mellish, C. and Oberlander, J.: ILEX: The ingelligent labelling explorer, *Archives and Museum Informatics*, Vol. 11, pp. 107–115 (1997).
2) 門林理恵子, 間瀬健二: 新しいコミュニケーション環境としてのMetaMuseum, マルチメディア通信と分散処理ワークショップ論文集, Vol. 95, No. 2, pp. 71–78 (1995).
3) Kadobayashi, R. and Mase, K.: Seamless Guidance by Personal Agent in Virtual Space Based on User Interaction in Real World, *The Third International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM98)* (Nwana, H. S. and Ndumu, D. T.(eds.)), pp. 191–200 (1998).
4) 門林理恵子, 西本一志, 角康之, 間瀬健二: 学芸員と見学者を仲介するエージェントによる博物館展示の個人化, マルチメディア, 分散, 協調とモーバイルワークショップ論文集, Vol. 97, No. 2, pp. 413–418 (1997).
5) Kadobayashi, R., Nishimoto, K., Sumi, Y. and Mase, K.: Personalizing Museum Exhibition by Mediating Agents, *Tasks and Methods in Applied Artificial Intelligence* (del Pobil, A.P., Mira, J. and Ali, M.(eds.)), Lecture Notes in Artificial Intelligence, Vol. 1416, Springer-Verlag, pp. 648–657 (1998).
6) 栗田靖之, 和田哲也, 松田卓, 松村浩一: 携帯情報端末による新しい展示手法, 人文学と情報処理, No. 17, pp. 42–48 (1998).
7) Cognitive Applications Limited: The Micro Gallery, http://www.cogapp.com/home/microgallery.html.
8) Maio, D. and Rizzi, S.: CICERO: An Assistant for Planning Visits to Museum, *Lecture Notes in Computer Science* (Revell, N. and Tjoa, A. M.(eds.)), Vol. 978, Springer-Verlag, pp. 564–573 (1995).
9) 西里静彦: 質的データの数量化, 朝倉書店 (1982).
10) 坂村健: デジタルミュージアム 電脳博物館—博物館の未来 (1997).
11) 白井博章, 横山和俊, 須藤昌徳, 箱守聰, 井上潮: 携帯端末を用いた個人向け情報提供システムの実現について, 情報処理学会研究報告 98-DBS-115, 98-FI-49, pp. 9–16 (1998).
12) 角康之, 江谷為之, シドニーフェルス・フェルス, ニコラ・シモネ, 小林薫, 間瀬健二: C-MAP: context-aware な展示ガイドシステムの試作, 情報処理学会論文誌, Vol. 39, No. 10, pp. 2866–2878 (1998).
13) 角康之, 西本一志, 間瀬健二: 協同発想と情報共有を促進する対話支援環境における情報の個人化, 電子情報通信学会論文誌, Vol. J80-D-I, No. 7, pp. 542–550 (1997).
14) Smithsonian Institution: Revealing Things, http://www.si.edu/revealingthings/.
15) http://www.rekihaku.ac.jp/.

(平成10年8月24日受付)
(平成11年1月 8 日採録)



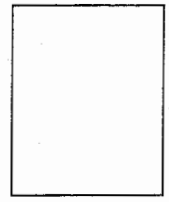
門林理恵子（正会員）

1985年大阪大学文学部史学科卒業．1997年同大学院工学研究科博士後期課程修了．博士（工学）．ソフトウェア会社勤務を経て，1990年シャープ（株）入社，移動体通信プロトコルの研究等に従事．1994年9月より，(株) ATR通信システム研究所，1995年4月より，(株) ATR知能映像通信研究所に出向．現在，(株) ATR知能映像通信研究所客員研究員．コミュニケーション支援技術，情報処理技術の博物館への応用研究に従事．訳書「オープンシステムネットワーキング」（ソフトバンク，共訳）．電子情報通信学会会員．



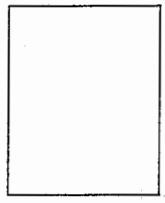
西本　一志（正会員）

1987年京都大学工学部工学研究科機械工学専攻修士課程修了．同年松下電器産業株式会社入社．現在（株）ATR知能映像通信研究所第二研究室客員研究員．エージェントによる人の創造的活動の支援の研究に従事．人工知能学会，言語処理学会，International Computer Music Association各会員．博士（工学）．



角　康之（正会員）

1990年早大・理工・電子通信卒．1995年東京大学大学院（情報工学）修了．同年より，(株) ATR知能映像通信研究所研究員．博士（工学）．発想支援システム，知識処理システムの開発，およびその人間協調系への応用研究に従事．人工知能学会，電子情報通信学会，AAAI各会員．



間瀬　健二（正会員）

1979年名大・工・電気卒．1981年同大大学院修士（情報）課程修了．同年日本電信電話公社（現在NTT）入社．1988～1989年米国MITメディア研究所客員研究員．1995年より（株）ATR知能映像通信研究所第二研究室長．博士（工学）．コンピュータグラフィックス，画像処理とそのヒューマンインタフェース、コミュニケーション支援への応用が主な研究テーマ．IEEE，ACM，電子情報通信学会各会員．

Part III

Index of all Fields and Methods

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

固有値ベクトル
printEvec(). Method in class CovarianceMatrixX
固有値ベクトル
printMat(). Method in class CovarianceMatrix
行列表示

R

readEachObject(String, double, double). Method in class MediationFunc
ファイルから各オブジェクトデータを読み込む
readObjectData(String). Method in class MediationFunc
ファイルからオブジェクトデータを読み込む
resetOffset(). Method in class AppPanel
表示オフセットをリセットする。
resetScale(). Method in class AppPanel
スケールパラメータをリセットする。
reshape(int, int, int, int). Method in class MediationFuncPanel
画面のリサイズ
RIGHT. Static variable in class MultiLineLabel
Alignment constants

S

SAME_ALL. Static variable in class MediationFunc
全てのキーワードと重みが同じ
saveObjectData(String). Method in class MediationFunc
展示空間の主成分値をファイルに保存をファイルに保存する
scale. Variable in class MediationFuncPanel
表示用スケール値
自動調整
scaleSlider. Variable in class AppPanel
表示スケールスライダ
scaleSlider. Variable in class MediationFunc
表示スケールスライダ
selectCheckbox. Variable in class MediationFunc
選択可能状態表示
selectColor. Static variable in class MediationFunc
selected. Variable in class Exhibit
選択されているかどうか(Ov)
selected. Variable in class KeywordList
そのキーワードが選択されているか
selectItem(int). Method in class InfoDialog
リストの項目を選択
selectMode. Variable in class MediationFunc
オブジェクト選択モード
selectStringColor. Static variable in class MediationFunc
setAlignment(int). Method in class MultiLineLabel
setComponent(int, double, double). Method in class Exhibit
主成分の計算
すでに計算された主成分値をファイルから読み込んでセットする
setDrawOrder(). Method in class MediationFuncPanel
描画の順番を決める
setFilename(String). Method in class Exhibit
展示オブジェクトのファイル名をセット
setFont(Font). Method in class MultiLineLabel
setFontAndFM(int). Method in class MediationFuncPanel
フォントサイズの変更
setForeground(Color). Method in class MultiLineLabel
setId(String). Method in class Exhibit
ID をセット
setImage(String). Method in class Exhibit
画像を読み込む
setKeyword(int, double). Method in class Exhibit
キーワードインデックスと重みのセット
setKeywords(). Method in class InfoDialog
キーワードをセット

展示空間は持っているすべてのキーワードを表示
その他の空間は選択されたオブジェクトに含まれるキーワードのみ表示
setLabel(int, String). Method in class Pattern
パターンラベルの設定
setLabel(String). Method in class MultiLineLabel
Methods to set the various attributes of the component
setMarginHeight(int). Method in class MultiLineLabel
setMarginWidth(int). Method in class MultiLineLabel
setTag(String). Method in class Exhibit
TAG をセット
setText(String). Method in class Exhibit
説明文のセット
setTextLines(). Method in class TextAreaDialog
説明文をセット
setTitle(String). Method in class Exhibit
TITLE をセット
sigma. Variable in class Distance
標準偏差
sliderInitVal. Static variable in class AppPanel
スライダの初期値
sliderKnob. Static variable in class AppPanel
スライダのつまみのサイズ
sliderMax. Static variable in class AppPanel
スライダの最大値
sliderMin. Static variable in class AppPanel
スライダの最小値

T

U

V

W

X

Y

Class Hierarchy

- class java.lang.Object
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Canvas
      - class MultiLineLabel
    - class java.awt.Container
      - class java.awt.Panel
        - class AppPanel
          - class MediationFuncTopPanel
          - class TopPanel
        - class MediationFuncPanel
          - class MediationFuncPickablePanel
      - class java.awt.Window
        - class java.awt.Dialog
          - class DDialog
            - class InfoDialog
            - class PatternDialog
            - class TextAreaDialog
          - class WarningDialog
          - class YesNoDialog
            - class ConfirmSaveDialog
        - class java.awt.Frame (implements java.awt.MenuContainer)
          - class DFrame
          - class MainFrame
  - class CovarianceMatrix
    - class CovarianceMatrixX
  - class Distance
  - class Exhibit
  - class Keyword
  - class KeywordList
  - class MediationFunc
  - class Pattern
  - class SuffixFilenameFilter (implements java.io.FilenameFilter)

Class AppPanel

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Panel
                           |
                           +----AppPanel
```

---

public class **AppPanel**
extends Panel

・トップパネルのベースとなるクラス

Field Index

- **BPanel1**
  ボタンパネル１
- **BPanel2**
  ボタンパネル２
- **BPanel3**
  ボタンパネル３
- **BPanel4**
  ボタンパネル４(下位で実装)
- **funcPanel**
  空間表示パネル
- **hscroll**
  横方向スクロールバー
- **index**
  空間種類
- **medFunc**
  アプリケーションメインオブジェクト
- **scaleSlider**
  表示スケールスライダ
- **sliderInitVal**
  スライダの初期値
- **sliderKnob**
  スライダのつまみのサイズ
- **sliderMax**
  スライダの最大値
- **sliderMin**
  スライダの最小値
- **vscroll**
  縦方向スクロールバー

Constructor Index

- **AppPanel**(MediationFunc, int)
  コンストラクタ

Method Index

- **action**(Event, Object)
  ボタンイベントの処理
- **clearOffset**()
  描画オフセット値のクリア
- **handleEvent**(Event)
  スライダイベントを処理
- **resetOffset**()
  表示オフセットをリセットする。
- **resetScale**()
  スケールパラメータをリセットする。

Fields

● **sliderKnob**

```
static final int sliderKnob
```

スライダのつまみのサイズ

● **sliderInitVal**

```
static final int sliderInitVal
```

スライダの初期値

● **sliderMin**

```
static final int sliderMin
```

スライダの最小値

● **sliderMax**

```
static final int sliderMax
```

スライダの最大値

medFunc

```
MediationFunc medFunc
```

アプリケーションメインオブジェクト

index

```
int index
```

空間種類

funcPanel

```
MediationFuncPanel funcPanel
```

空間表示パネル

vscroll

```
Scrollbar vscroll
```

縦方向スクロールバー

hscroll

```
Scrollbar hscroll
```

横方向スクロールバー

scaleSlider

```
Scrollbar scaleSlider
```

表示スケールスライダ

BPanel1

```
Panel BPanel1
```

ボタンパネル1

BPanel2

```
Panel BPanel2
```

ボタンパネル2

BPanel3

```
Panel BPanel3
```

ボタンパネル3

BPanel4

```
Panel BPanel4
```

ボタンパネル4(下位で実装)

Constructors

AppPanel

```
public AppPanel(MediationFunc appMain,
                int idx)
```

コンストラクタ

Parameters:
appMain - アプリケーションメインオブジェクト
idx - 空間種類

Methods

clearOffset

```
public void clearOffset()
```

描画オフセット値のクリア

handleEvent

```
public boolean handleEvent(Event evt)
```

スライダイベントを処理

Parameters:
ev - 発生イベント
Overrides:
handleEvent in class Component

action

```
public boolean action(Event evt,
                      Object arg)
```

ボタンイベントの処理

Parameters:
evt - 発生イベント
arg - ボタンのラベル
Overrides:
action in class Component

resetOffset

```
public void resetOffset()
```

表示オフセットをリセットする。

resetScale

```
public void resetScale()
```

スケールパラメータをリセットする。

Class ConfirmSaveDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----YesNoDialog
                                           |
                                           +----ConfirmSaveDialog
```

---

public class **ConfirmSaveDialog**
extends YesNoDialog

主成分値をセーブする際、確認するためのダイアログ

Constructor Index

- **ConfirmSaveDialog**(Frame)
  コンストラクタ

Method Index

- **no**()
  No が押された場合の処理
- **yes**()
  Yes が押された場合の処理

Constructors

ConfirmSaveDialog

```
public ConfirmSaveDialog(Frame parent)
```

コンストラクタ

Methods

yes

```
public void yes()
```

Yes が押された場合の処理

Overrides:
yes in class YesNoDialog

no

```
public void no()
```

No が押された場合の処理

Overrides:
no in class YesNoDialog

Class CovarianceMatrix

```
java.lang.Object
   |
   +----CovarianceMatrix
```

---

public class **CovarianceMatrix**
extends Object

・分散共分散行列

Field Index

- **ave**
  各キーワードの平均
- **dim**
  分散共分散行列の次元
- **ev**
  固有値
- **mat**
  分散共分散行列 -> 固有ベクトル

Constructor Index

- **CovarianceMatrix**(int, Exhibit[], int)
  コンストラクタ
  すべてのキーワードから分散共分散を作成する。
- **CovarianceMatrix**(int[], Exhibit[], int[], int)
  コンストラクタ
  キーワードのサブセットから分散共分散を作成する。

Method Index

- **getEVal**(int)
  固有値取得
- **getEVec**(int, double[])
  固有ベクトル取得
- **printAve**()
  平均値表示
- **printEv**()
  固有値表示
- **printEvec**()
  固有値ベクトル
- **printMat**()
  行列表示
- **syev**()
  固有値、固有ベクトルの計算 DSYEV

Fields

dim

```
int dim
```

分散共分散行列の次元

mat

```
double mat[]
```

分散共分散行列 -> 固有ベクトル

ave

```
double ave[]
```

各キーワードの平均

ev

```
double ev[]
```

固有値

Constructors

CovarianceMatrix

```
public CovarianceMatrix(int dimension,
                        Exhibit ex[],
                        int numEx)
```

コンストラクタ
すべてのキーワードから分散共分散を作成する。

Parameters:
dimension - 配列の次元
ex - 展示オブジェクト
numEx - 展示オブジェクトの数

CovarianceMatrix

```
public CovarianceMatrix(int subkey[],
                        Exhibit ex[],
                        int subset[],
                        int numSub)
```

コンストラクタ
キーワードのサブセットから分散共分散を作成する。

Parameters:
subkey - キーワードのサブセット
ex - 展示オブジェクト
subset - 展示オブジェクトのサブセット
numSub - 展示オブジェクトのサブセットの数

Methods

syev

```
public void syev()
```

固有値、固有ベクトルの計算 DSYEV

printAve

```
public void printAve()
```

平均値表示

getEVal

```
public double getEVal(int idx)
```

固有値取得

Parameters:
idx - 固有値のインデックス(寄与率が高い方から 1, 2, ...)
Returns:
固有値

getEVec

```
public void getEVec(int idx,
                    double vec[])
```

固有ベクトル取得

Parameters:
idx - 固有ベクトルのインデックス (固有値の寄与率が高い方から 1, 2, ...)
vec - 固有ベクトル

printEv

```
public void printEv()
```

固有値表示

printEvec

```
public void printEvec()
```

固有値ベクトル

printMat

```
public void printMat()
```

行列表示

Class CovarianceMatrixX

```
java.lang.Object
   |
   +----CovarianceMatrix
           |
           +----CovarianceMatrixX
```

---

public class **CovarianceMatrixX**
extends CovarianceMatrix

・分散共分散行列
固有値、固有ベクトルの計算方法を DSYEV から DSYEVX に変更したもの

Field Index

- **zmat**
  固有ベクトル

Constructor Index

- **CovarianceMatrixX**(int, Exhibit[], int)
  コンストラクタ
  すべてのキーワードから分散共分散を作成する。
- **CovarianceMatrixX**(int[], Exhibit[], int[], int)
  コンストラクタ
  キーワードのサブセットから分散共分散を作成する。

Method Index

- **getEVal**(int)
  固有値取得
- **getEVec**(int, double[])
  固有ベクトル取得
- **printEv**()
  固有値表示
- **printEvec**()
  固有値ベクトル
- **syev**()
  固有値、固有ベクトルの計算 DSYEVX
  ここでは CovarianceMatrix の syev() をオーバーライドするため 関数名が syev である
  が、SDYEVX を使っているので注意

Fields

zmat

```
double zmat[]
```

固有ベクトル

Constructors

CovarianceMatrixX

```
public CovarianceMatrixX(int dimension,
                         Exhibit ex[],
                         int numEx)
```

コンストラクタ
すべてのキーワードから分散共分散を作成する。

Parameters:
dimension - 配列の次元
ex - 展示オブジェクト
numEx - 展示オブジェクトの数

CovarianceMatrixX

```
public CovarianceMatrixX(int subkey[],
                         Exhibit ex[],
                         int subset[],
                         int numSub)
```

コンストラクタ
キーワードのサブセットから分散共分散を作成する。

Parameters:
subkey - キーワードのサブセット
ex - 展示オブジェクト
subset - 展示オブジェクトのサブセット
numSub - 展示オブジェクトのサブセットの数

Methods

syev

```
public void syev()
```

固有値、固有ベクトルの計算 DSYEVX
ここでは CovarianceMatrix の syev() をオーバーライドするため 関数名が syev である
が、SDYEVX を使っているので注意

Overrides:
syev in class CovarianceMatrix

getEVal

```
public double getEVal(int idx)
```

固有値取得

Parameters:
idx - 固有値のインデックス(寄与率が高い方から 1, 2)
Returns:
固有値
Overrides:
getEVal in class CovarianceMatrix

getEVec

```
public void getEVec(int idx,
                    double vec[])
```

固有ベクトル取得

Parameters:
idx - 固有ベクトルのインデックス (固有値の寄与率が高い方から 1, 2)
vec - 固有ベクトル
Overrides:
getEVec in class CovarianceMatrix

printEv

```
public void printEv()
```

固有値表示

Overrides:
printEv in class CovarianceMatrix

printEvec

```
public void printEvec()
```

固有値ベクトル

Overrides:
printEvec in class CovarianceMatrix

Class DDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----DDialog
```

---

public class **DDialog**
extends Dialog

・dispose 可能なダイアログ

Constructor Index

- **DDialog**(Frame, boolean)
  コンストラクタ
- **DDialog**(Frame, String, boolean)
  コンストラクタ

Method Index

- **handleEvent**(Event)
  WINDOW_DESTROY イベントによりウインドウを消去

Constructors

DDialog

```
public DDialog(Frame parent,
               boolean mode)
```

コンストラクタ

Parameters:
parent - 親フレーム
mode - ダイアログのモード

DDialog

```
public DDialog(Frame parent,
               String title,
               boolean mode)
```

コンストラクタ

Parameters:
parent - 親フレーム
title - ダイアログのタイトル
mode - ダイアログのモード

Methods

handleEvent

```
public boolean handleEvent(Event ev)
```

WINDOW_DESTROY イベントによりウインドウを消去

Parameters:
ev - 発生イベント
Overrides:
handleEvent in class Component

Class DFrame

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
                                   |
                                   +----DFrame
```

---

public class **DFrame**
extends Frame

WINDOW_DESTROY イベントを取る Frame

Field Index

- **medFunc**
  アプリケーションメインオブジェクト

Constructor Index

- **DFrame**(MediationFunc, String)
  コンストラクタ

Method Index

- **handleEvent**(Event)
  WINDOW_DESTROY によりウインドウを消去

Fields

medFunc

```
MediationFunc medFunc
```

アプリケーションメインオブジェクト

Constructors

DFrame

```
public DFrame(MediationFunc appMain,
              String title)
```

コンストラクタ

Parameters:
appMain - アプリケーションメインオブジェクト
title - フレームのタイトル

Methods

handleEvent

```
public boolean handleEvent(Event ev)
```

WINDOW_DESTROY によりウインドウを消去

Parameters:
ev - 発生イベント
Overrides:
handleEvent in class Component

Class Distance

```
java.lang.Object
   |
   +----Distance
```

---

public class **Distance**
extends Object

・空間距離

Field Index

- **ave**
  平均値
- **dist**
  意味距離
  対角に正規化距離を格納
- **ex**
- **exCnt**
- **FAR**
  0: 遠い
- **idx**
  空間種別
- **NEAR**
  1: 近い
- **OTHER**
  2: それ以外
- **sigma**
  標準偏差

Constructor Index

- **Distance**(Exhibit[], int, int)
  コンストラクタ
  全体の展示オブジェクトに対して意味距離を計算
- **Distance**(Exhibit[], int, int, int[])
  コンストラクタ
  展示オブジェクトのサブセットに対して意味距離を計算

Method Index

- **calcDistance**(int, int)
  2オブジェクト間の意味距離の計算
- **checkDistance**(int, int)
  2オブジェクト間の意味距離の判定
- **print**()
  テスト用表示(全体)
- **print**(int[])
  テスト用表示(一部)

Fields

ex

```
Exhibit ex[]
```

exCnt

```
int exCnt
```

idx

```
int idx
```

空間種別

sigma

```
double sigma
```

標準偏差

ave

```
double ave
```

平均値

dist

```
double dist[][]
```

意味距離
対角に正規化距離を格納

FAR

```
static final int FAR
```

0: 遠い

NEAR

```
static final int NEAR
```

1: 近い

OTHER

```
static final int OTHER
```

2: それ以外

Constructors

Distance

```
public Distance(Exhibit exhibit[],
                int objCnt,
                int index)
```

コンストラクタ
全体の展示オブジェクトに対して意味距離を計算

Parameters:
exhibit - 展示オブジェクト
objCnt - 展示オブジェクトの数
index - 空間種別

Distance

```
public Distance(Exhibit exhibit[],
                int objCnt,
                int index,
                int subset[])
```

コンストラクタ
展示オブジェクトのサブセットに対して意味距離を計算

Parameters:
exhibit - 展示オブジェクト
objCnt - 展示オブジェクトの数
index - 空間種別
subset - 展示オブジェクトのサブセット

Methods

calcDistance

```
public double calcDistance(int i,
                           int j)
```

2オブジェクト間の意味距離の計算

Parameters:
i - オブジェクト1のインデックス
j - オブジェクト2のインデックス
Returns:
意味距離

checkDistance

```
public int checkDistance(int i,
                         int j)
```

2オブジェクト間の意味距離の判定

Parameters:
i - オブジェクト1のインデックス
j - オブジェクト2のインデックス
Returns:
判定結果
- NEAR
- FAR
- OTHER

print

```
public void print()
```

テスト用表示(全体)

print

```
public void print(int subset[])
```

テスト用表示(一部)

Parameters:
subset - サブセット

Class Exhibit

```
java.lang.Object
    |
    +----Exhibit
```

---

public class **Exhibit**
extends Object

・展示物オブジェクト

Field Index

- **comp1**
  第一主成分
- **comp2**
  第二主成分に寄与率の比をかけたもの
- **fname**
  展示オブジェクトファイル名
- **focus**
  マウスのフォーカス
- **frBottom**
  表示枠の大きさ
- **frLeft**
  表示枠の大きさ
- **frRight**
  表示枠の大きさ
- **frTop**
  表示枠の大きさ
- **hasImage**
  画像を持っているか
- **hasTitle**
  タイトルを持っているか
- **id**
  識別子
- **image**
  画像
- **included**
  表示の対象となっているか(Ov U Om)
- **keyword**
  キーワード
- **selected**
  選択されているかどうか(Ov)
- **tag**
  ラベル(タグ)
- **text**
  説明文
- **textLines**
  行数
- **title**
  タイトル
- **xoffset**
  表示のxオフセット
- **yoffset**
  表示のyオフセット

Constructor Index

- **Exhibit**(int, int)
  コンストラクタ

Method Index

- **calcComponent**(double, double, double[], double[])
  主成分の計算
  すべてのキーワードについて計算
- **calcComponent**(double, double, double[], double[], int, int[])
  主成分の計算
  キーワードのサブセットについて計算
- **print**(KeywordList)
  テスト表示
- **setComponent**(int, double, double)
  主成分の計算
  すでに計算された主成分値をファイルから読み込んでセットする
- **setFilename**(String)
  展示オブジェクトのファイル名をセット
- **setId**(String)
  ID をセット
- **setImage**(String)
  画像を読み込む
- **setKeyword**(int, double)
  キーワードインデックスと重みのセット
- **setTag**(String)
  TAG をセット
- **setText**(String)
  説明文のセット
- **setTitle**(String)
  TITLE をセット

Fields

fname

```
String fname
```

展示オブジェクトファイル名

id

```
String id
```

識別子

tag

```
String tag
```

ラベル(タグ)

keyword

```
Keyword keyword[]
```

キーワード

text

```
String text[]
```

説明文

textLines

```
int textLines
```

行数

hasTitle

```
boolean hasTitle
```

タイトルを持っているか

title

```
String title
```

タイトル

hasImage

```
boolean hasImage
```

画像を持っているか

image

```
Image image
```

画像

selected

```
boolean selected
```

選択されているかどうか(Ov)

included

```
boolean included
```

表示の対象となっているか(Ov U Om)

comp1

```
double comp1[]
```

第一主成分

comp2

```
double comp2[]
```

第二主成分に寄与率の比をかけたもの

xoffset

```
int xoffset[]
```

表示のxオフセット

yoffset

```
int yoffset[]
```

表示のyオフセット

focus

```
boolean focus[]
```

マウスのフォーカス

frLeft

```
int frLeft[]
```

表示枠の大きさ

frRight

```
int frRight[]
```

表示枠の大きさ

frTop

```
int frTop[]
```

表示枠の大きさ

frBottom

```
int frBottom[]
```

表示枠の大きさ

Constructors

Exhibit

```
public Exhibit(int keyMax,
               int lineMax)
```

コンストラクタ

Parameters:
keyMax - 持てるキーワードの数の最大値
lineMax - キーワードの長さの最大値

Methods

setFilename

```
public void setFilename(String st)
```

展示オブジェクトのファイル名をセット

Parameters:

st - 展示オブジェクトのファイル名

setId

```
public void setId(String st)
```

ID をセット

Parameters:
st - ID の文字列

setTag

```
public void setTag(String st)
```

TAG をセット

Parameters:
st - TAG の文字列をセット

setTitle

```
public void setTitle(String st)
```

TITLE をセット

Parameters:
st - TITLE の文字列をセット

setKeyword

```
public void setKeyword(int idx,
                       double w)
```

キーワードインデックスと重みのセット

Parameters:
idx - キーワードのインデックス
w - 重み

setImage

```
public void setImage(String imagefile)
```

画像を読み込む

Parameters:
imagefile - 画像ファイル名(.gif or .jpg)

setText

```
public void setText(String string)
```

説明文のセット

Parameters:
string - 展示オブジェクトの説明文

setComponent

```
public void setComponent(int idx,
                         double c1,
                         double c2)
```

主成分の計算
すでに計算された主成分値をファイルから読み込んでセットする

Parameters:
idx - 空間種別
c1 - 第一主成分
c2 - 第二主成分(寄与率の比でスケールされたもの)

calcComponent

```
public void calcComponent(double ev1,
                          double ev2,
                          double evec1[],
                          double evec2[])
```

主成分の計算
すべてのキーワードについて計算

Parameters:
ev1 - 固有値1
ev2 - 固有値2
evec1 - 固有ベクトル1
evec2 - 固有ベクトル2

calcComponent

```
public void calcComponent(double ev1,
                          double ev2,
                          double evec1[],
                          double evec2[],
                          int idx,
                          int subkey[])
```

主成分の計算
キーワードのサブセットについて計算

Parameters:
ev1 - 固有値1
ev2 - 固有値2
evec1 - 固有ベクトル1
evec2 - 固有ベクトル2
idx - 空間の種類
subkey - キーワードのサブセット

print

```
public void print(KeywordList klist)
```

テスト表示

Parameters:
klist - キーワードリスト

Class InfoDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----DDialog
                                           |
                                           +----InfoDialog
```

---

public class **InfoDialog**
extends DDialog

・オブジェクトの情報ダイアログ
展示空間のみ、重み表示と更新ボタンを持つ

Field Index

- **button**
- **ex**
- **index**
  空間種類
- **klist**
- **label**
- **list**
- **textField**

Constructor Index

- **InfoDialog**(Frame, Exhibit, KeywordList, int)
  コンストラクタ

Method Index

- **handleEvent**(Event)
  特定イベント処理
- **listIndex2KeyIndex**()
  リストのインデックスを各オブジェクトのキーワードインデックスに変換
- **listIndex2KeyIndex**(int)
  リストのインデックスを各オブジェクトのキーワードインデックスに変換
- **selectItem**(int)
  リストの項目を選択
- **setKeywords**()
  キーワードをセット

  展示空間は持っているすべてのキーワードを表示
  その他の空間は選択されたオブジェクトに含まれるキーワードのみ表示

Fields

ex

```
Exhibit ex
```

klist

```
KeywordList klist
```

list

```
List list
```

label

```
Label label
```

textField

```
TextField textField
```

button

```
Button button
```

index

```
int index
```

  空間種類

Constructors

InfoDialog

```
public InfoDialog(Frame parent,
```

```
                Exhibit exhibit,
                KeywordList keylist,
                int idx)
```

コンストラクタ

Parameters:
parent - 親フレーム
exhibit - 展示オブジェクト
keylist - キーワードリスト
idx - 空間種類

Methods

setKeywords

```
public void setKeywords()
```

キーワードをセット

展示空間は持っているすべてのキーワードを表示
その他の空間は選択されたオブジェクトに含まれるキーワードのみ表示

selectItem

```
public void selectItem(int idx)
```

リストの項目を選択

Parameters:
idx - キーワードリストのインデックス

listIndex2KeyIndex

```
public int listIndex2KeyIndex()
```

リストのインデックスを各オブジェクトのキーワードインデックスに変換

Returns:
各オブジェクトのキーワードのインデックス

listIndex2KeyIndex

```
public int listIndex2KeyIndex(int idx)
```

リストのインデックスを各オブジェクトのキーワードインデックスに変換

Parameters:
idx - キーワードリストのインデックス
Returns:
各オブジェクトのキーワードのインデックス

handleEvent

```
public boolean handleEvent(Event ev)
```

特定イベント処理

Parameters:
ev - 発生イベント
Overrides:
handleEvent in class DDialog

Class Keyword

```
java.lang.Object
   |
   +----Keyword
```

---

class **Keyword**
extends Object

・キーワード

Field Index

- **listId**
  キーワードリスト内でのインデックス
- **weight**
  重み

Constructor Index

- **Keyword()**
  コンストラクタ

Fields

listId

```
int listId
```

キーワードリスト内でのインデックス

weight

```
double weight
```

重み

Constructors

Keyword

```
public Keyword()
```

コンストラクタ

Class KeywordList

```
java.lang.Object
   |
   +----KeywordList
```

---

public class **KeywordList**
extends Object

・キーワードリスト

Field Index

- **keyword**
  キーワード文字列
- **kwCnt**
  キーワードの数
- **selected**
  そのキーワードが選択されているか

Constructor Index

- **KeywordList**(int)
  コンストラクタ

Method Index

- **addItem**(String)
  キーワードを追加
- **deselectAll**()
  すべてのキーワードを未選択に
- **print**()
  テスト表示

Fields

keyword

```
String keyword[]
```

キーワード文字列

kwCnt

```
int kwCnt
```

キーワードの数

selected

```
boolean selected[]
```

そのキーワードが選択されているか

Constructors

KeywordList

```
public KeywordList(int keyMax)
```

コンストラクタ

Parameters:
keyMax - キーワードの数の最大値

Methods

addItem

```
public int addItem(String kw)
```

キーワードを追加

Parameters:
kw - 追加するキーワード文字列
Returns:
キーワードリスト内でのインデックス

deselectAll

```
public void deselectAll()
```

すべてのキーワードを未選択に

print

```
public void print()
```

テスト表示

Class MainFrame

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
                                   |
                                   +----MainFrame
```

---

public class **MainFrame**
extends Frame

・メインウインドウ

Field Index

- **confirm**
  confirm ダイアログの返し
- **medFunc**
  アプリケーションメインオブジェクト
- **openMenuItem**
  [開く]のメニュー項目
- **subMenu0**
- **subMenu1**
- **subMenu2**
- **subMenu3**

Constructor Index

- **MainFrame**(MediationFunc, String)
  コンストラクタ

Method Index

- **action**(Event, Object)
  メニューイベントの処理

Fields

confirm

```
static int confirm
```

confirm ダイアログの返し

medFunc

```
MediationFunc medFunc
```

アプリケーションメインオブジェクト

openMenuItem

```
MenuItem openMenuItem
```

[開く]のメニュー項目

subMenu0

```
Menu subMenu0
```

subMenu1

```
Menu subMenu1
```

subMenu2

```
Menu subMenu2
```

subMenu3

```
Menu subMenu3
```

Constructors

MainFrame

```
public MainFrame(MediationFunc appMain,
                 String title)
```

コンストラクタ

Parameters:
appMain - アプリケーションメインオブジェクト
title - フレームのタイトル

Methods

action

```
public boolean action(Event evt,
                      Object arg)
```

メニューイベントの処理

Parameters:
evt - 発生イベント
arg - イベントの名前
Overrides:
action in class Component

Class MediationFunc

```
java.lang.Object
   |
   +----MediationFunc
```

---

public class **MediationFunc**
extends Object

アプリケーションメインクラス
main class

Field Index

- **calculated**
  主成分がすでに計算されているか
- **cntKv**
  Kv の数
- **cntOv**
  Ov の数
- **cntOvOm**
  OvOm の数
- **covarianceMat**
  全体の分散共分散行列
- **DIFFERENT**
  同じでない
- **exhibit**
  展示オブジェクト全部(Oc)
- **EXHIBITION**
  0: 展示空間
- **exhibitionTopPanel**
  展示空間トップパネル
- **exhibitionWin**
  メイン親ウインドウ
- **focusColor**
- **frameColor**
- **hostName**
  ソケットのサーバホスト名
- **hscroll**
  横方向スクロールバー
- **INTEREST**
  1: 興味空間
- **interestButton**
  興味空間表示用ボタン
- **interestCMat**
  興味空間の分散共分散行列
- **interestSubkey**
  サブセット Kv
- **interestSubset**
  サブセット Ov
- **interestTopPanel**
  興味空間トップパネル
- **interestWin**
  興味空間ウインドウ
- **keywordList**
  キーワードリスト(Kc)
- **maxExhibits**
- **maxKeywords**
- **maxTextLines**
- **nodeColor**
  定数
- **objCnt**
  展示オブジェクトの数
- **outPort**
  ソケットのポート番号
- **pattern**
  パターン分け
- **PERSONAL**
  2: 個人化空間
- **personalButton**
  個人化空間表示用ボタン
- **personalCMat**
  個人化空間の分散共分散行列
- **personalSubset**
  サブセット OvOm
- **personalTopPanel**
  個人化空間トップパネル
- **personalWin**
  個人化空間ウインドウ
- **SAME_ALL**
  全てのキーワードと重みが同じ
- **scaleSlider**
  表示スケールスライダ
- **selectCheckbox**
  選択可能状態表示
- **selectColor**
- **selectMode**
  オブジェクト選択モード
- **selectStringColor**
- **stringColor**
- **vscroll**
  縦方向スクロールバー

Constructor Index

- **MediationFunc**()
  コンストラクタ

Method Index

- **checkKeyword**(int, int)
  2つのオブジェクトのキーワードとその重みを調べ、それが全て同じであるか否かを判定する
- **checkProperty**()
  選択された全てのオブジェクトのキーワードとその重みを調べ、それが全て同じであるか否かを判定する
- **divideIntoPatterns**()
  正規化距離を計算しパターン分けする
- **main**(String[])
  メインルーチン
- **makeExhibitionSpace**(String)
  展示空間を作成
- **makeInterestSpace**(Event)
  興味空間を作成
- **makePersonalSpace**()
  個人化空間を作成
- **readEachObject**(String, double, double)
  ファイルから各オブジェクトデータを読み込む
- **readObjectData**(String)
  ファイルからオブジェクトデータを読み込む
- **saveObjectData**(String)
  展示空間の主成分値をファイルに保存をファイルに保存する

Fields

nodeColor

```
static final Color nodeColor
```

定数

selectColor

```
static final Color selectColor
```

focusColor

```
static final Color focusColor
```

frameColor

```
static final Color frameColor
```

stringColor

```
static final Color stringColor
```

selectStringColor

```
static final Color selectStringColor
```

maxExhibits

```
static final int maxExhibits
```

maxKeywords

```
static final int maxKeywords
```

maxTextLines

```
static final int maxTextLines
```

EXHIBITION

```
static final int EXHIBITION
```

0: 展示空間

INTEREST

```
static final int INTEREST
```

1: 興味空間

PERSONAL

```
static final int PERSONAL
```

2: 個人化空間

SAME_ALL

```
static final int SAME_ALL
```

全てのキーワードと重みが同じ

DIFFERENT

```
static final int DIFFERENT
```

同じでない

hostName

`String hostName`

ソケットのサーバホスト名

outPort

`int outPort`

ソケットのポート番号

exhibit

`Exhibit exhibit[]`

展示オブジェクト全部(Oc)

objCnt

`int objCnt`

展示オブジェクトの数

keywordList

`KeywordList keywordList`

キーワードリスト(Kc)

calculated

`boolean calculated`

主成分がすでに計算されているか

interestSubset

`int interestSubset[]`

サブセット Ov

cntOv

`int cntOv`

Ov の数

interestSubkey

`int interestSubkey[]`

サブセット Kv

cntKv

`int cntKv`

Kv の数

personalSubset

`int personalSubset[]`

サブセット OvOm

cntOvOm

`int cntOvOm`

OvOm の数

covarianceMat

`CovarianceMatrixX covarianceMat`

全体の分散共分散行列

interestCMat

`CovarianceMatrixX interestCMat`

興味空間の分散共分散行列

personalCMat

`CovarianceMatrixX personalCMat`

個人化空間の分散共分散行列

exhibitionWin

`MainFrame exhibitionWin`

メイン親ウインドウ

interestWin

`DFrame interestWin`

興味空間ウインドウ

personalWin

```
DFrame personalWin
```

個人化空間ウインドウ

exhibitionTopPanel

```
MediationFuncTopPanel exhibitionTopPanel
```

展示空間トップパネル

interestTopPanel

```
TopPanel interestTopPanel
```

興味空間トップパネル

personalTopPanel

```
TopPanel personalTopPanel
```

個人化空間トップパネル

vscroll

```
Scrollbar vscroll
```

縦方向スクロールバー

hscroll

```
Scrollbar hscroll
```

横方向スクロールバー

selectCheckbox

```
Checkbox selectCheckbox
```

選択可能状態表示

scaleSlider

```
Scrollbar scaleSlider
```

表示スケールスライダ

interestButton

```
Button interestButton
```

興味空間表示用ボタン

personalButton

```
Button personalButton
```

個人化空間表示用ボタン

selectMode

```
boolean selectMode
```

オブジェクト選択モード

pattern

```
Pattern pattern
```

パターン分け

Constructors

MediationFunc

```
public MediationFunc()
```

コンストラクタ

Methods

divideIntoPatterns

```
public void divideIntoPatterns()
```

正規化距離を計算しパターン分けする

makeInterestSpace

```
public boolean makeInterestSpace(Event ev)
```

興味空間を作成

Parameters:
ev - イベント

makePersonalSpace

```
public boolean makePersonalSpace()
```

個人化空間を作成

readObjectData

```
public boolean readObjectData(String filename)
```

ファイルからオブジェクトデータを読み込む

Parameters:
filename - ファイル名

readEachObject

```
public boolean readEachObject(String filename,
                              double comp1,
                              double comp2)
```

ファイルから各オブジェクトデータを読み込む

Parameters:
filename - ファイル名
comp1 - 第一主成分値
comp2 - 第二主成分値

saveObjectData

```
public boolean saveObjectData(String filename)
```

展示空間の主成分値をファイルに保存をファイルに保存する

Parameters:
filename - ファイル名

makeExhibitionSpace

```
public boolean makeExhibitionSpace(String filename)
```

展示空間を作成

Parameters:
filename - ファイル名

checkProperty

```
public int checkProperty()
```

選択された全てのオブジェクトのキーワードとその重みを調べ、それが全て同じであるか否かを判定する

Returns:
SAME_ALL 全て同じ

checkKeyword

```
public int checkKeyword(int obj1,
                        int obj2)
```

２つのオブジェクトのキーワードとその重みを調べ、それが全て同じであるか否かを判定する

Parameters:
obj1 - オブジェクト１
obj2 - オブジェクト２
Returns:
SAME_ALL 全て同じ

main

```
public static void main(String args[])
```

メインルーチン

Class MediationFuncPanel

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Panel
                           |
                           +----MediationFuncPanel
```

---

public class **MediationFuncPanel**
extends Panel

・展示空間表示パネル

Field Index

- **cenXOff**
  表示中心のＸオフセット
- **cenYOff**
  表示中心のＹオフセット
- **dimPanel**
  パネルのサイズ
- **drawOrder**
  描画の順番
- **fm**
  フォント情報
- **index**
  空間種類
- **medFunc**
  アプリケーションメイン
- **offg**
  オフスクリーングラフィックス
- **offImg**
  オフスクリーンイメージ
- **scale**
  表示用スケール値
  自動調整
- **trgObj**
  Drag の対象
- **userScale**
  表示用スケール値
  スライダによりユーザが指定した値
- **xorg**
  Drag のための起点
- **yorg**
  Drag のための起点

Constructor Index

- **MediationFuncPanel**(MediationFunc, int)
  コンストラクタ

Method Index

- **checkInOut**(int, int, Exhibit)
  マウスポインタの内外判定
- **clear**()
  画面のクリア
  オフスクリーンバッファをクリア
- **mouseDown**(Event, int, int)
  マウスクリック
- **mouseMove**(Event, int, int)
  マウスのフォーカス
- **mouseUp**(Event, int, int)
  マウスボタンを離した時の処理
- **paint**(Graphics)
  描画 paint
- **paintNode**(Graphics, Exhibit, FontMetrics, double, double)
  各ノードの描画
- **reshape**(int, int, int, int)
  画面のリサイズ
- **setDrawOrder**()
  描画の順番を決める
- **setFontAndFM**(int)
  フォントサイズの変更
- **update**(Graphics)
  描画
  ちらつきを防ぐ為、update() を paint() でオーバーライド

Fields

medFunc

```
MediationFunc medFunc
```

アプリケーションメイン

scale

```
double scale
```

表示用スケール値
自動調整

userScale

```
double userScale
```

表示用スケール値
スライダによりユーザが指定した値

cenXOff

```
int cenXOff
```

表示中心のＸオフセット

cenYOff

```
int cenYOff
```

表示中心のＹオフセット

dimPanel

```
Dimension dimPanel
```

パネルのサイズ

index

```
int index
```

空間種類

xorg

```
int xorg
```

Drag のための起点

yorg

```
int yorg
```

Drag のための起点

trgObj

```
int trgObj
```

Drag の対象

fm

```
FontMetrics fm
```

フォント情報

drawOrder

```
int drawOrder[]
```

描画の順番

offImg

```
Image offImg
```

オフスクリーンイメージ

offg

```
Graphics offg
```

オフスクリーングラフィックス

Constructors

MediationFuncPanel

```
public MediationFuncPanel(MediationFunc appMain,
                          int idx)
```

コンストラクタ

Parameters:
appMain - アプリケーションメイン
idx - 空間種類

Methods

setDrawOrder

```
public void setDrawOrder()
```

描画の順番を決める

mouseDown

```
public boolean mouseDown(Event evt,
                         int x,
                         int y)
```

マウスクリック

Parameters:
evt - マウスイベント
x - クリックのx座標
y - クリックのy座標
Overrides:
mouseDown in class Component

mouseUp

```
public boolean mouseUp(Event evt,
                       int x,
                       int y)
```

マウスボタンを離した時の処理

Parameters:
evt - マウスイベント
x - クリックのx座標
y - クリックのy座標
Overrides:
mouseUp in class Component

mouseMove

```
public boolean mouseMove(Event evt,
                         int x,
                         int y)
```

マウスのフォーカス

Parameters:
evt - マウスイベント
x - クリックのx座標
y - クリックのy座標
Overrides:
mouseMove in class Component

clear

```
public void clear()
```

画面のクリア
オフスクリーンバッファをクリア

reshape

```
public void reshape(int x,
                    int y,
                    int width,
                    int height)
```

画面のリサイズ

Parameters:
x - ウインドウ位置x
y - ウインドウ位置y
width - ウインドウ幅
height - ウインドウ高さ
Overrides:
reshape in class Component

update

```
public void update(Graphics g)
```

描画
ちらつきを防ぐ為、update() を paint() でオーバーライド

Parameters:
g - グラフィックス
Overrides:
update in class Container

paint

```
public void paint(Graphics g)
```

描画 paint

Parameters:
g - グラフィックス
Overrides:
paint in class Container

paintNode

```
public void paintNode(Graphics g,
                      Exhibit ex,
                      FontMetrics fm,
                      double cenX,
                      double cenY)
```

各ノードの描画

Parameters:

g - グラフィックス
ex - 描画対象展示オブジェクト
fm - フォント情報
cenX - 表示の中心
cenY - 表示の中心

checkInOut

```
public boolean checkInOut(int x,
                          int y,
                          Exhibit ex)
```

マウスポインタの内外判定

Parameters:
x - x位置
y - y位置
ex - 展示オブジェクト

setFontAndFM

```
public void setFontAndFM(int size)
```

フォントサイズの変更

Parameters:
size - フォントサイズ

Class MediationFuncPickablePanel

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Panel
                           |
                           +----MediationFuncPanel
                                   |
                                   +----MediationFuncPickablePanel
```

---

public class **MediationFuncPickablePanel**
extends MediationFuncPanel

展示空間表示パネル (Pickable)

Constructor Index

- **MediationFuncPickablePanel**(MediationFunc)
  コンストラクタ

Method Index

- **mouseDown**(Event, int, int)
  マウスクリック

Constructors

MediationFuncPickablePanel

```
public MediationFuncPickablePanel(MediationFunc appMain)
```

コンストラクタ

Parameters:
appMain - アプリケーションメインオブジェクト

Methods

mouseDown

```
public boolean mouseDown(Event evt,
                         int x,
                         int y)
```

マウスクリック

Parameters:
evt - マウスイベント
x - x位置
y - y位置
Overrides:
mouseDown in class MediationFuncPanel

Class MediationFuncTopPanel

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Panel
                           |
                           +----AppPanel
                                   |
                                   +----MediationFuncTopPanel
```

---

public class **MediationFuncTopPanel**
extends AppPanel

展示空間トップパネル

Constructor Index

- **MediationFuncTopPanel**(MediationFunc)
  コンストラクタ

Method Index

- **action**(Event, Object)
  ボタンイベントの処理

Constructors

MediationFuncTopPanel

```
public MediationFuncTopPanel(MediationFunc appMain)
```

コンストラクタ

Methods

action

```
public boolean action(Event evt,
                      Object arg)
```

ボタンイベントの処理

Parameters:
evt - 発生イベント
arg - ボタンのラベル
Overrides:
action in class AppPanel

Class MultiLineLabel

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Canvas
                   |
                   +----MultiLineLabel
```

---

public class **MultiLineLabel**
extends Canvas

Field Index

- **alignment**
  The alignment of the text.
- **CENTER**
  Alignment constants
- **LEFT**
  Alignment constants
- **line_ascent**
  Font height above baseline
- **line_height**
  Total height of the font
- **line_widths**
  How wide each line is
- **lines**
  The lines of text to display
- **margin_height**
  Top and bottom margins
- **margin_width**
  Left and right margins
- **max_width**
  The width of the widest line
- **num_lines**
  The number of lines
- **RIGHT**
  Alignment constants

Constructor Index

- **MultiLineLabel**(String)
- **MultiLineLabel**(String, int)
- **MultiLineLabel**(String, int, int)
- **MultiLineLabel**(String, int, int, int)
  Here are four versions of the cosntrutor.
  Break the label up into separate lines, and save the other info.

Method Index

- **addNotify**()
  This method is invoked after our Canvas is first created but before it can actually be displayed.
- **getAlignment**()
- **getMarginHeight**()
- **getMarginWidth**()
- **measure**()
  This method figures out how the font is, and how wide each line of the label is, and how wide the widest line is.
- **minimumSize**()
  This method is called when the layout manager wants to know the bare minimum amount of space we need to get by.
- **newLabel**(String)
  This method breaks a specified label up into an array of lines.
  It uses the StringTokenizer utility class.
- **paint**(Graphics)
  This method draws the label (applets use the same method).
- **preferredSize**()
  This method is called by a layout manager when it wants to know how big we'd like to be.
- **setAlignment**(int)
- **setFont**(Font)
- **setForeground**(Color)
- **setLabel**(String)
  Methods to set the various attributes of the component
- **setMarginHeight**(int)
- **setMarginWidth**(int)

Fields

LEFT

```
public static final int LEFT
```

Alignment constants

CENTER

```
public static final int CENTER
```

Alignment constants

- **RIGHT**

```
public static final int RIGHT
```

Alignment constants

- **lines**

```
protected String lines[]
```

The lines of text to display

- **num_lines**

```
protected int num_lines
```

The number of lines

- **margin_width**

```
protected int margin_width
```

Left and right margins

- **margin_height**

```
protected int margin_height
```

Top and bottom margins

- **line_height**

```
protected int line_height
```

Total height of the font

- **line_ascent**

```
protected int line_ascent
```

Font height above baseline

- **line_widths**

```
protected int line_widths[]
```

How wide each line is

- **max_width**

```
protected int max_width
```

The width of the widest line

- **alignment**

```
protected int alignment
```

The alignment of the text.

Constructors

- **MultiLineLabel**

```
public MultiLineLabel(String label,
                      int margin_width,
                      int margin_height,
                      int alignment)
```

Here are four versions of the cosntrutor.
Break the label up into separate lines, and save the other info.

- **MultiLineLabel**

```
public MultiLineLabel(String label,
                      int margin_width,
                      int margin_height)
```

- **MultiLineLabel**

```
public MultiLineLabel(String label,
                      int alignment)
```

- **MultiLineLabel**

```
public MultiLineLabel(String label)
```

Methods

- **newLabel**

```
protected void newLabel(String label)
```

This method breaks a specified label up into an array of lines.
It uses the StringTokenizer utility class.

- **measure**

```
protected void measure()
```

This method figures out how the font is, and how wide each line of the label is, and how wide the widest line is.

setLabel

```
public void setLabel(String label)
```

Methods to set the various attributes of the component

setFont

```
public void setFont(Font f)
```

Overrides:
setFont in class Component

setForeground

```
public void setForeground(Color c)
```

Overrides:
setForeground in class Component

setAlignment

```
public void setAlignment(int a)
```

setMarginWidth

```
public void setMarginWidth(int mw)
```

setMarginHeight

```
public void setMarginHeight(int mh)
```

getAlignment

```
public int getAlignment()
```

getMarginWidth

```
public int getMarginWidth()
```

getMarginHeight

```
public int getMarginHeight()
```

addNotify

```
public void addNotify()
```

This method is invoked after our Canvas is first created but before it can actually be displayed. After we've invoked our superclass's addNotify() method, we have font metrics and can successfully call measure() to figure out how big the label is.

Overrides:
addNotify in class Canvas

preferredSize

```
public Dimension preferredSize()
```

This method is called by a layout manager when it wants to know how big we'd like to be.

Overrides:
preferredSize in class Component

minimumSize

```
public Dimension minimumSize()
```

This method is called when the layout manager wants to know the bare minimum amount of space we need to get by.

Overrides:
minimumSize in class Component

paint

```
public void paint(Graphics g)
```

This method draws the label (applets use the same method). Note that it handles the margins and the alignment, but that it doesn't have to worry about the color or font--the superclass takes care of setting those in the Graphics object we're passed.

Overrides:
paint in class Canvas

Class Pattern

```
java.lang.Object
   |
   +----Pattern
```

---

public class **Pattern**
extends Object

・パターン分類して保存

Field Index

- **cnt**
  各パターンに含まれるオブジェクトの数
- **label**
  各パターンのラベル
- **list**
  各パターンに含まれるオブジェクトのリスト

Constructor Index

- **Pattern**(int, int)
  コンストラクタ

Method Index

- **checkPatternList**(int, int, int)
  パターンデータを追加
- **print**()
  テスト用表示
- **setLabel**(int, String)
  パターンラベルの設定

Fields

list

```
boolean list[][]
```

各パターンに含まれるオブジェクトのリスト

cnt

```
int cnt[]
```

各パターンに含まれるオブジェクトの数

label

```
String label[]
```

各パターンのラベル

Constructors

Pattern

```
public Pattern(int kind,
               int max)
```

コンストラクタ

Parameters:
kind - パターンの数
max - 各パターンに含まれるオブジェクトの数の最大値

Methods

setLabel

```
public void setLabel(int p,
                     String string)
```

パターンラベルの設定

Parameters:
p - パターン番号
string - ラベル

checkPatternList

```
public void checkPatternList(int p,
                             int i,
                             int j)
```

パターンデータを追加

Parameters:

p - パターン番号
i - オブジェクトID1
j - オブジェクトID2

● **print**

```
public void print()
```

テスト用表示

Class PatternDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----DDialog
                                           |
                                           +----PatternDialog
```

---

public class **PatternDialog**
extends DDialog

・パターン表示 UI

Field Index

- **list**
- **medFunc**
  アプリケーションメインオブジェクト
- **pattern**
  パターン分類
- **textField**

Constructor Index

- **PatternDialog**(MediationFunc, Frame, Pattern)
  コンストラクタ

Method Index

- **clientSend**()
  ソケットにより送信
- **handleEvent**(Event)
  特定イベント処理

Fields

medFunc

```
MediationFunc medFunc
```

アプリケーションメインオブジェクト

pattern

```
Pattern pattern
```

パターン分類

list

```
List list
```

textField

```
TextField textField
```

Constructors

PatternDialog

```
public PatternDialog(MediationFunc appMain,
                     Frame parent,
                     Pattern pat)
```

コンストラクタ

Parameters:
parent - 親フレーム
pat - パターン分類

Methods

handleEvent

```
public boolean handleEvent(Event ev)
```

特定イベント処理

Parameters:
ev - 発生イベント
Overrides:

handleEvent in class DDialog

clientSend

```
public boolean clientSend()
```

ソケットにより送信

Class SuffixFilenameFilter

```
java.lang.Object
   |
   +----SuffixFilenameFilter
```

---

public class **SuffixFilenameFilter**
extends Object
implements FilenameFilter

・拡張子によるファイル名フィルタ

Field Index

- **suffix**
  ファイルの拡張子

Constructor Index

- **SuffixFilenameFilter**(String)
  コンストラクタ

Method Index

- **accept**(File, String)
  accept のオーバーライト

Fields

suffix

```
String suffix
```

ファイルの拡張子

Constructors

SuffixFilenameFilter

```
public SuffixFilenameFilter(String suffix)
```

コンストラクタ

Parameters:
suffix - ファイルの拡張子

Methods

accept

```
public boolean accept(File dir,
                      String name)
```

accept のオーバーライト

Parameters:
dir - ディレクトリ名
name - ファイル名

Class TextAreaDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----DDialog
                                           |
                                           +----TextAreaDialog
```

---

public class **TextAreaDialog**
extends DDialog

・テキストエリアダイアログ

Field Index

- **ex**
- **textArea**

Constructor Index

- **TextAreaDialog**(Frame, Exhibit)
  コンストラクタ

Method Index

- **setTextLines**()
  説明文をセット

Fields

ex

```
Exhibit ex
```

textArea

```
TextArea textArea
```

Constructors

TextAreaDialog

```
public TextAreaDialog(Frame parent,
                      Exhibit exhibit)
```

コンストラクタ

Parameters:
parent - 親フレーム
exhibit - 展示オブジェクト

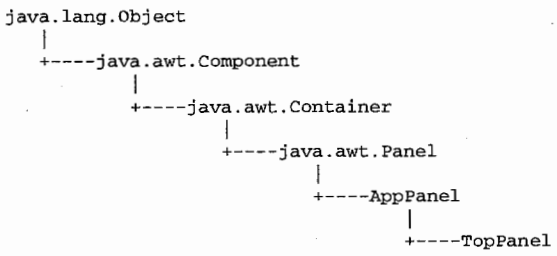
Methods

setTextLines

```
public void setTextLines()
```

説明文をセット

Class TopPanel



---

public class **TopPanel**
extends AppPanel

・空間トップパネル

Constructor Index

Method Index

Constructors

TopPanel

```
public TopPanel(MediationFunc appMain,
                int idx)
```

コンストラクタ

Parameters:
appMain - アプリケーションメインオブジェクト
idx - 空間種類

Methods

action

```
public boolean action(Event evt,
                      Object arg)
```

ボタンイベントの処理

Parameters:
evt - 発生イベント
arg - ボタンのラベル
Overrides:
action in class AppPanel

Class WarningDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----WarningDialog
```

---

public class **WarningDialog**
extends Dialog

・警告ダイアログ

Field Index

- **button**
- **label**

Constructor Index

- **WarningDialog**(Frame, String, String)

Method Index

- **action**(Event, Object)
  Pop down the window when the button is clicked.
- **gotFocus**(Event, Object)
  When the window gets the keyboard focus, give it to the button.
  This allows keyboard shortcuts to pop down the dialog.

Fields

button

```
protected Button button
```

label

```
protected MultiLineLabel label
```

Constructors

WarningDialog

```
public WarningDialog(Frame parent,
                     String title,
                     String message)
```

Methods

action

```
public boolean action(Event e,
                      Object arg)
```

Pop down the window when the button is clicked.

Overrides:
action in class Component

gotFocus

```
public boolean gotFocus(Event e,
                        Object arg)
```

When the window gets the keyboard focus, give it to the button.
This allows keyboard shortcuts to pop down the dialog.

Overrides:
gotFocus in class Component

Class YesNoDialog

```
java.lang.Object
    |
    +----java.awt.Component
            |
            +----java.awt.Container
                    |
                    +----java.awt.Window
                            |
                            +----java.awt.Dialog
                                    |
                                    +----YesNoDialog
```

---

public class **YesNoDialog**
extends Dialog

Yes/No ダイアログ

Field Index

- **CANCEL**
- **cancel**
- **label**
- **no**
- **NO**
- **YES**
- **yes**

Constructor Index

- **YesNoDialog**(Frame, String, String, String, String, String)
  コンストラクタ

Method Index

- **action**(Event, Object)
  Handle button events by calling the answer() method.
  Pass the appropriate constant value, depending on the button.
- **answer**(int)
  Call yes(), no(), and cancel() methods depending on the button the user clicked.
- **cancel**()
- **no**()
- **yes**()

Fields

NO

```
public static final int NO
```

YES

```
public static final int YES
```

CANCEL

```
public static final int CANCEL
```

yes

```
protected Button yes
```

no

```
protected Button no
```

cancel

```
protected Button cancel
```

label

```
protected MultiLineLabel label
```

Constructors

YesNoDialog

```
public YesNoDialog(Frame parent,
                   String title,
                   String message,
                   String yes_label,
                   String no_label,
                   String cancel_label)
```

コンストラクタ

Methods

action

```
public boolean action(Event e,
                      Object arg)
```

Handle button events by calling the answer() method.
Pass the appropriate constant value, depending on the button.

Parameters:
e - イベント
Overrides:
action in class Component

answer

```
protected void answer(int answer)
```

Call yes(), no(), and cancel() methods depending on the button the user clicked. Subclasses define how the answer is processed by overriding this method or the yes(), no(), and cancel() methods.

Parameters:
answer - 押されたボタンの種類

yes

```
protected void yes()
```

no

```
protected void no()
```

cancel

```
protected void cancel()
```

#
MEMO
#

◎実行方法

java MediationFunc data/newOpenhouseAll2.dat miris66 23456

◎バックアップ

- back_openhouse98 オープンハウス時
- back981215 オープンハウス後修正
  + ダブルバッファ
  + 表示(興味、個人化空間)キーワード修正
  + セーブ機能付ける前
- back981221 セーブ機能追加
- back990105 スクロールパネル追加前

(- back990106_scroll スクロールパネル(ScrollPaneにて)追加)
※ しかし動作があまり良くないので保留
現在元に戻している。

- back990111 スクロールパネル追加
※ スライダにて独自に実装。ScrollPane 使わず。

パネルにスライダ等を実装するクラスを作成する前のバックアップ

- back990112 スクロール、スケールを実装
とりあえず 98OpenHouse 終了後の修正完成版

End of File

```
JC = javac
JFLAGS =

TARGETS = MultiLineLabel.class WarningDialog.class InfoDialog.class\
        TextAreaDialog.class PatternDialog.class\
                TextAreaDialog.class MainFrame.class SuffixFilenameFilter.class\
        DFrame.class TopPanel.class\
                AppPanel.class\
        Distance.class Pattern.class\
        Exhibit.class CovarianceMatrix.class CovarianceMatrixX.class\
        MediationFuncPickablePanel.class MediationFuncPanel.class\
                MediationFuncTopPanel.class MediationFunc.class\
        YesNoDialog.class ConfirmSaveDialog.class

###
.SUFFIXES:  .class .java

.java.class:
                $(JC) $(JFLAGS) $<


all: $(TARGETS)
```

```
/**********************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1999.1.11   T.Takumi(CSK)
 **********************************************************************/

import java.awt.*;

/**********************************************************************
 ** ・トップパネルのベースとなるクラス
 **
 **********************************************************************/
public class AppPanel extends Panel {

 // スクロールスライダの設定
 // Windows で負値を取るとおかしくなるのでずらしている
                              /** スライダのつまみのサイズ */
 static final int         sliderKnob   = 100;
                              /** スライダの初期値 */
 static final int         sliderInitVal = 0;
                              /** スライダの最小値 */
 static final int         sliderMin    = -300;
                              /** スライダの最大値 */
 static final int         sliderMax    = 300;

                              /** アプリケーションメインオブジェクト */
 MediationFunc            medFunc;
                              /** 空間種類 */
 int                          index;
                              /** 空間表示パネル */
 MediationFuncPanel funcPanel;

                              /** 縦方向スクロールバー */
 Scrollbar                vscroll;
                              /** 横方向スクロールバー */
 Scrollbar                hscroll;

                              /** 表示スケールスライダ */
 Scrollbar                scaleSlider;

                              /** ボタンパネル１ */
 Panel                        BPanel1;
                              /** ボタンパネル２ */
 Panel                        BPanel2;
                              /** ボタンパネル３ */
 Panel                        BPanel3;
                              /** ボタンパネル４(下位で実装) */
 Panel                        BPanel4;

 /*--------------------------------------------------------------------*/
 /** コンストラクタ
  *
  * @param appMain  アプリケーションメインオブジェクト
  * @param idx          空間種類
  */
 /*--------------------------------------------------------------------*/
 public AppPanel(MediationFunc appMain, int idx) {
```



```
  medFunc = appMain;
  index = idx;

  setLayout(new BorderLayout());

  switch (index) {
  case medFunc.EXHIBITION:
   funcPanel = (MediationFuncPanel)(new MediationFuncPickablePanel(medFunc));
   funcPanel.setBackground(new Color(255, 250, 205));
   break;
  case medFunc.INTEREST:
   funcPanel = new MediationFuncPanel(medFunc, index);
   funcPanel.setBackground(new Color(230, 230, 250));
   break;
  case medFunc.PERSONAL:
   funcPanel = new MediationFuncPanel(medFunc, index);
   funcPanel.setBackground(new Color(255, 228, 225));
   break;
  }
  add("Center", funcPanel);

  // スライダとボタン類
  vscroll = new Scrollbar(Scrollbar.VERTICAL, sliderInitVal-sliderMin, sliderKnob, 0, sliderMax+sliderKnob-sliderMin);
  add("East", vscroll);

  Panel buttonSliderPanel = new Panel();
  buttonSliderPanel.setLayout(new BorderLayout());
  add("South", buttonSliderPanel);

  hscroll = new Scrollbar(Scrollbar.HORIZONTAL, sliderInitVal-sliderMin, sliderKnob, 0, sliderMax+sliderKnob-sliderMin);
  buttonSliderPanel.add("North", hscroll);

  // ボタン等
  Panel buttonPanel = new Panel();

  buttonSliderPanel.add("South", buttonPanel);

  BPanel1 = new Panel();
  BPanel1.setLayout(new BorderLayout());
  BPanel2 = new Panel();
  BPanel2.setLayout(new BorderLayout());
  BPanel3 = new Panel();
  BPanel3.setLayout(new BorderLayout());
  BPanel4 = new Panel();
  BPanel4.setLayout(new BorderLayout());

  buttonPanel.add(BPanel1);
  buttonPanel.add(BPanel2);
  buttonPanel.add(BPanel3);
  buttonPanel.add(BPanel4);

  // BPanel1
  BPanel1.add("North", new Label("スケール:"));

  // BPanel2
  scaleSlider = new Scrollbar(Scrollbar.HORIZONTAL, 10, 1, 1, 30);
  BPanel2.add("North", scaleSlider);
  BPanel2.add("South", new Button("リセット"));
```

```
  // BPanel3
  BPanel3.add("South", new Button("再表示"));

  // BPanel4
  // 下位のクラスにて実装
 }

 /*-----------------------------------------------------------------------*/
 /** 描画オフセット値のクリア
  */
 /*-----------------------------------------------------------------------*/
 public void clearOffset() {
   for (int i = 0; i < medFunc.objCnt; i++) {
     medFunc.exhibit[i].xoffset[index] = 0;
     medFunc.exhibit[i].yoffset[index] = 0;
   }
 }

 /*-----------------------------------------------------------------------*/
 /** スライダイベントを処理
  *
  * @param ev          発生イベント
  */
 /*-----------------------------------------------------------------------*/
 public boolean handleEvent(Event evt) {

   // スライダのイベントを処理
   if (evt.target.equals(scaleSlider)) {
//     System.out.println(">>>> slider!! "+scaleSlider.getValue());
     funcPanel.userScale = scaleSlider.getValue()/10.0;
     funcPanel.clear();
     funcPanel.repaint();
     return(true);
   }
   else if (evt.target.equals(vscroll)) {
//     System.out.println(">>>> vscroll!! "+vscroll.getValue());
     funcPanel.cenYOff = vscroll.getValue()+sliderMin;
     funcPanel.clear();
     funcPanel.repaint();
     return(true);
   }
   else if (evt.target.equals(hscroll)) {
//     System.out.println(">>>> hscroll!! "+hscroll.getValue());
     funcPanel.cenXOff = hscroll.getValue()+sliderMin;
     funcPanel.clear();
     funcPanel.repaint();
     return(true);
   }

   // チェックボックス、ボタンのイベントを処理
   if (evt.target instanceof Checkbox || evt.target instanceof Button) {
     return(action(evt, evt.arg));
   }

   // その他は下位のウインドウに伝播する
   return(false);
 }
```



```
 /*-----------------------------------------------------------------------*/
 /** ボタンイベントの処理
  *
  * @param evt                 発生イベント
  * @param arg                 ボタンのラベル
  */
 /*-----------------------------------------------------------------------*/
 public boolean action(Event evt, Object arg) {

   if ("再表示".equals(arg)) {
     clearOffset();
     funcPanel.clear();
     funcPanel.repaint();
     return(true);
   }
   else if ("リセット".equals(arg)) {
     resetOffset();
     resetScale();
     funcPanel.clear();
     funcPanel.repaint();
     return(true);
   }

   return(false);
 }

 /*-----------------------------------------------------------------------*/
 /** 表示オフセットをリセットする。
  */
 /*-----------------------------------------------------------------------*/
 public void resetOffset() {
   funcPanel.cenXOff = 0;
   funcPanel.cenYOff = 0;
   hscroll.setValue(sliderInitVal-sliderMin);
   vscroll.setValue(sliderInitVal-sliderMin);
 }

 /*-----------------------------------------------------------------------*/
 /** スケールパラメータをリセットする。
  */
 /*-----------------------------------------------------------------------*/
 public void resetScale() {
   funcPanel.userScale = 1.0;
   scaleSlider.setValue(10);
 }

}


// End of File
```

```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.12.21   T.Takumi(CSK)
 **************************************************************************/

import java.awt.*;

/**************************************************************************
 ** 主成分値をセーブする際、確認するためのダイアログ
 **
 **************************************************************************/
public class ConfirmSaveDialog extends YesNoDialog {

 /*-----------------------------------------------------------------*/
 /** コンストラクタ
  */
 /*-----------------------------------------------------------------*/
 public ConfirmSaveDialog(Frame parent) {
  super(parent, "保存確認", "data/saveComponents.dat に保存してよろしいですか？", "Yes", "No", null);
 }

 /*-----------------------------------------------------------------*/
 /** Yes が押された場合の処理
  */
 /*-----------------------------------------------------------------*/
 public void yes() {
  MainFrame.confirm = YES;
 }

 /*-----------------------------------------------------------------*/
 /** No が押された場合の処理
  */
 /*-----------------------------------------------------------------*/
 public void no() {
  MainFrame.confirm = NO;
 }
}

// End of File
```



```
/**************************************************************************
 ** PROGRAM : jlapack を使った固有値、固有ベクトルの計算
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.06   T.Takumi(CSK)
 **************************************************************************/

import org.netlib.util.*;
import org.netlib.lapack.Dsyev;
import org.netlib.lapack.DSYEV;

/**************************************************************************
 ** ・分散共分散行列
 **
 **************************************************************************/
public class CovarianceMatrix {
                        /** 分散共分散行列の次元 */
 int                    dim;
                        /** 分散共分散行列 -> 固有ベクトル */
 double     mat[];
                        /** 各キーワードの平均 */
 double     ave[];
                        /** 固有値 */
 double     ev[];

 /*-----------------------------------------------------------------*/
 /** コンストラクタ<br>
  * すべてのキーワードから分散共分散を作成する。
  *
  * @param dimension            配列の次元
  * @param ex                   展示オブジェクト
  * @param numEx    展示オブジェクトの数
  */
 /*-----------------------------------------------------------------*/
 public CovarianceMatrix(int dimension, Exhibit ex[], int numEx) {
  dim = dimension;

  mat = new double[dim*dim];
  ave = new double[dim];
  ev = new double[dim];

  // 各キーワードの平均を計算
  for (int i = 0; i < dim; i++) {
   ave[i] = 0.0;
   for (int j = 0; j < numEx; j++) {
    ave[i] += ex[j].keyword[i].weight;
   }
   ave[i] /= (double)numEx;
  }

  // 分散共分散行列を計算
  double work;
  for (int x = 0; x < dim; x++) {
   for (int y = 0; y < dim; y++) {

        if (x < y) {
     mat[x*dim+y] = 0.0;
     continue;          // 三角行列にする
```

```
    }

    work = 0.0;
    for (int i = 0; i < numEx; i++) {
      work += (ex[i].keyword[x].weight - ave[x])
            * (ex[i].keyword[y].weight - ave[y]);
    }
    mat[x*dim+y] = work/(double)(numEx-1);
   }
  }
 }

/*-------------------------------------------------------------------------*/
/** コンストラクタ<br>
 * キーワードのサブセットから分散共分散を作成する。
 *
 * @param subkey    キーワードのサブセット
 * @param ex                展示オブジェクト
 * @param subset    展示オブジェクトのサブセット
 * @param numSub   展示オブジェクトのサブセットの数
 */
/*-------------------------------------------------------------------------*/
public CovarianceMatrix(int subkey[], Exhibit ex[], int subset[], int numSub) {
  dim = subkey.length;

  mat = new double[dim*dim];
  ave = new double[dim];
  ev = new double[dim];

  int       ii, jj;

  // 各キーワードの平均を計算
  for (int i = 0; i < dim; i++) {
    ii = subkey[i];
    ave[i] = 0.0;
    for (int j = 0; j < numSub; j++) {
      jj = subset[j];
      ave[i] += ex[jj].keyword[ii].weight;
    }
    ave[i] /= (double)numSub;
  }

  // 分散共分散行列を計算
  double work;
  for (int x = 0; x < dim; x++) {
    for (int y = 0; y < dim; y++) {

          if (x < y) {
      mat[x*dim+y] = 0.0;
      continue;         // 三角行列にする
      }

      work = 0.0;
      for (int i = 0; i < numSub; i++) {
        ii = subset[i];
        work += (ex[ii].keyword[subkey[x]].weight - ave[x])
              * (ex[ii].keyword[subkey[y]].weight - ave[y]);
      }
```



```
      mat[x*dim+y] = work/(double)(numSub-1);
    }
   }
  }

 /*-------------------------------------------------------------------------*/
 /** 固有値、固有ベクトルの計算 DSYEV
  */
 /*-------------------------------------------------------------------------*/
 public void syev() {

/* 一次元配列で使う場合と二次元配列で使う場合では、行と列が逆に
  扱われているので注意

 (例) 上三角行列を示す

   double   a[] =
{470.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,
 363.8, 459.4,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,
 384.9, 407.8, 583.6,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,
 333.5, 383.8, 426.9, 462.1,  0.0,  0.0,  0.0,  0.0,  0.0,
 356.2, 369.9, 421.6, 391.1, 531.9,  0.0,  0.0,  0.0,  0.0,
 259.0, 229.7, 245.9, 224.5, 263.3, 300.1,  0.0,  0.0,  0.0,
 248.2, 113.2, 106.3,  66.2, 186.7, 201.8, 708.3,  0.0,  0.0,
 321.4, 326.6, 320.3, 344.7, 301.7, 209.4,  95.3, 500.7,  0.0,
 493.6, 501.6, 576.5, 485.3, 479.9, 304.7, 232.3, 404.6, 874.7};

 (例) 下三角行列を示す

   double   a2[][] =
{{470.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0},
 {363.8, 459.4,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0},
 {384.9, 407.8, 583.6,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0},
 {333.5, 383.8, 426.9, 462.1,  0.0,  0.0,  0.0,  0.0,  0.0},
 {356.2, 369.9, 421.6, 391.1, 531.9,  0.0,  0.0,  0.0,  0.0},
 {259.0, 229.7, 245.9, 224.5, 263.3, 300.1,  0.0,  0.0,  0.0},
 {248.2, 113.2, 106.3,  66.2, 186.7, 201.8, 708.3,  0.0,  0.0},
 {321.4, 326.6, 320.3, 344.7, 301.7, 209.4,  95.3, 500.7,  0.0},
 {493.6, 501.6, 576.5, 485.3, 479.9, 304.7, 232.3, 404.6, 874.7}};

*/
                        /** 固有値、固有ベクトルの両方 */
   String    jobz = "V";
                        /** 上三角行列 */
   String    uplo = "U";
                        /** 行列 A の次数*/
   int                  n = dim;
                        /** 配列 A の第一次元。LDA >= max(1,N). */
   int                  lda = dim;
                        /** 配列 work の大きさ
                         * LWORK >= max(1,3*N-1).<br>
                         * 最適効率を得るには LWORK >= (NB+2)*N で、
                         * NB は ILAENV が戻す DSYTRD 用の最適ブロック長である。
                         */
   int                  lwork = 3*dim-1;
   double    work[] = new double[lwork];
                        /** 終了状態 */
   intW      info = new intW(-1);
```

```
   Dsyev.dsyev(jobz, uplo, n, mat, 0, lda, ev, 0, work, 0, lwork, info);
//   DSYEV.DSYEV(jobz, uplo, n, a2, w, work, lwork, info);
  }

  /*-------------------------------------------------------------------*/
  /** 平均値表示
   */
  /*-------------------------------------------------------------------*/
  public void printAve() {
    System.out.println("(平均値)----------------------------------------");
    for (int i = 0; i < ave.length; i++)
      System.out.print(ave[i] + " ");
    System.out.println();
    System.out.println();
  }

  /*-------------------------------------------------------------------*/
  /** 固有値取得
   *
   * @param idx         固有値のインデックス(寄与率が高い方から 1, 2, ...)
   * @return 固有値
   */
  /*-------------------------------------------------------------------*/
  public double getEVal(int idx) {
    return(ev[dim-idx]);
  }

  /*-------------------------------------------------------------------*/
  /** 固有ベクトル取得
   *
   * @param idx         固有ベクトルのインデックス
   *         (固有値の寄与率が高い方から 1, 2, ...)
   * @param vec         固有ベクトル
   */
  /*-------------------------------------------------------------------*/
  public void getEVec(int idx, double vec[]) {
    int line = dim - idx;
    for (int col = 0; col < dim; col++) {
      vec[col] = mat[line*dim+col];
    }
  }

  /*-------------------------------------------------------------------*/
  /** 固有値表示
   */
  /*-------------------------------------------------------------------*/
  public void printEv() {
    System.out.println("(固有値)----------------------------------------");
    for (int i = 0; i < ev.length; i++)
      System.out.print(ev[i] + " ");
    System.out.println();
    System.out.println();
  }

  /*-------------------------------------------------------------------*/
  /** 固有値ベクトル
   */
```



```
  /*-------------------------------------------------------------------*/
  public void printEvec() {
    System.out.println("(固有ベクトル)----------------------------------");
    printMat();
  }

  /*-------------------------------------------------------------------*/
  /** 行列表示
   */
  /*-------------------------------------------------------------------*/
  public void printMat() {
    System.out.println("(行列)----------------------------------------");
    for (int i = 0; i < mat.length; i++) {
      System.out.print(mat[i] + " ");
      int j = (i+1) % dim;
      if (j == 0) {
        System.out.println();
      }
    }
    System.out.println();
  }

}


// End of File
```

```
/*************************************************************************
 ** PROGRAM : jlapack を使った固有値、固有ベクトルの計算(DSYEVX)
 **         : 固有値、固有ベクトルを２つだけ計算
 ** FILENAME :
 **         :
 ** DATE    : 1998.12.11   T.Takumi(CSK)
 *************************************************************************/

import org.netlib.util.*;
import org.netlib.lapack.Dsyevx;
import org.netlib.lapack.DSYEVX;

/*************************************************************************
 ** ・分散共分散行列<br>
 **  固有値、固有ベクトルの計算方法を DSYEV から DSYEVX に
 **  変更したもの
 *************************************************************************/
public class CovarianceMatrixX extends CovarianceMatrix{
                        /** 固有ベクトル */
  double     zmat[];

  /*-----------------------------------------------------------------*/
  /** コンストラクタ<br>
   * すべてのキーワードから分散共分散を作成する。
   *
   * @param dimension              配列の次元
   * @param ex                     展示オブジェクト
   * @param numEx    展示オブジェクトの数
   */
  /*-----------------------------------------------------------------*/
  public CovarianceMatrixX(int dimension, Exhibit ex[], int numEx) {
    super(dimension, ex, numEx);
    zmat = new double[dim*2];
  }

  /*-----------------------------------------------------------------*/
  /** コンストラクタ<br>
   * キーワードのサブセットから分散共分散を作成する。
   *
   * @param subkey     キーワードのサブセット
   * @param ex                     展示オブジェクト
   * @param subset     展示オブジェクトのサブセット
   * @param numSub   展示オブジェクトのサブセットの数
   */
  /*-----------------------------------------------------------------*/
  public CovarianceMatrixX(int subkey[], Exhibit ex[], int subset[], int numSub) {
    super(subkey, ex, subset, numSub);
    zmat = new double[dim*2];
  }

  /*-----------------------------------------------------------------*/
  /** 固有値、固有ベクトルの計算 DSYEVX <br>
   * ここでは CovarianceMatrix の syev() をオーバーライドするため
   * 関数名が syev であるが、SDYEVX を使っているので注意
   */
  /*-----------------------------------------------------------------*/
  public void syev() {
```



```
                        /** 固有値、固有ベクトルの両方 */
    String    jobz = "V";
                        /** il 番目から iu 番目までの固有値を見つける */
    String    range = "I";
                        /** 上三角行列 */
    String    uplo = "U";
                        /** 行列 A の次数*/
    int                 n = dim;
                        /** 配列 A の第一次元。LDA >= max(1,N). */
    int                 lda = dim;
    double    vl = 0.0;   // 現在使わない
    double    vu = 0.0;   // 現在使わない
                        /** il 番目 */
    int                 il = n-1;
                        /** iu 番目 */
    int                 iu = n;
                        /** 固有値に対する絶対許容誤差 */
    double    abstol = 0.0;             // 何を設定してよいかわからない
                        /** 見つかった固有値の総数 */
    intW      m = new intW(-1);
                        /** 配列 Z の第一次元。LDZ >= max(1,N). */
    int                 ldz = dim;

                        /** 配列 work の大きさ
                         * LWORK >= max(1,8*N).<br>
                         * 最適効率を得るには LWORK >= (NB+3)*N で、
                         * NB は ILAENV が戻す DSYTRD 用の最適ブロック長である。
                         */
    int                 lwork = 8*dim;
    double    work[] = new double[lwork];
                        /** 作業用<br> 次元 5 * N */
    int                 iwork[] = new int[5*dim];
                        /** 収束しそこなった固有ベクトルの添字<br> 次元 N */
    int                 ifail[] = new int[dim];
                        /** 終了状態 */
    intW      info = new intW(-1);

    Dsyevx.dsyevx(jobz, range, uplo, n, mat, 0, lda, vl, vu, il, iu,
            abstol, m, ev, 0, zmat, 0, ldz,
            work, 0, lwork, iwork, 0, ifail, 0, info);
    // 収束エラー
    if (info.val != 0) {
      System.out.println("ERROR: couldn't get the Eigenvalues.");
      System.out.println("     : ifail --> "+ifail[0]+" "+ifail[1]);
    }
  }

  /*-----------------------------------------------------------------*/
  /** 固有値取得
   *
   * @param idx         固有値のインデックス(寄与率が高い方から 1, 2)
   * @return 固有値
   */
  /*-----------------------------------------------------------------*/
  public double getEVal(int idx) {
    switch (idx) {
    case 1:
      return(ev[1]);
```

```
  case 2:
    return(ev[0]);
  default:
    return(0.0);          // dummy
  }
}

/*-------------------------------------------------------------------*/
/** 固有ベクトル取得
 *
 * @param idx        固有ベクトルのインデックス
 *        (固有値の寄与率が高い方から 1, 2)
 * @param vec        固有ベクトル
 */
/*-------------------------------------------------------------------*/
public void getEVec(int idx, double vec[]) {
  int line = 2 - idx;
  for (int col = 0; col < dim; col++) {
    vec[col] = zmat[line*dim+col];
  }
}

/*-------------------------------------------------------------------*/
/** 固有値表示
 */
/*-------------------------------------------------------------------*/
public void printEv() {
  System.out.println("(固有値)----------------------------------------");
  for (int i = 0; i < 2; i++)
    System.out.print(ev[i] + " ");
  System.out.println();
  System.out.println();
}

/*-------------------------------------------------------------------*/
/** 固有値ベクトル
 */
/*-------------------------------------------------------------------*/
public void printEvec() {
  System.out.println("(固有ベクトル)----------------------------------------");
  for (int i = 0; i < zmat.length; i++) {
    System.out.print(zmat[i] + " ");
    int j = (i+1) % dim;
    if (j == 0) {
      System.out.println();
    }
  }
  System.out.println();
}

}

// End of File
```



```
/**********************************************************************
** PROGRAM : 仲介機能プログラム
** FILENAME :
**       :
** DATE    : 1998.10.23   T.Takumi(CSK)
**********************************************************************/

import java.awt.*;

/**********************************************************************
** ・dispose 可能なダイアログ
**
**********************************************************************/
public class DDialog extends Dialog {

 /*-------------------------------------------------------------------*/
 /** コンストラクタ
  *
  * @param parent     親フレーム
  * @param mode       ダイアログのモード
  */
 /*-------------------------------------------------------------------*/
 public DDialog(Frame parent, boolean mode) {
  super(parent, mode);
 }

 /*-------------------------------------------------------------------*/
 /** コンストラクタ
  *
  * @param parent     親フレーム
  * @param title      ダイアログのタイトル
  * @param mode       ダイアログのモード
  */
 /*-------------------------------------------------------------------*/
 public DDialog(Frame parent, String title, boolean mode) {
  super(parent, title, mode);
 }

 /*-------------------------------------------------------------------*/
 /** WINDOW_DESTROY イベントによりウインドウを消去
  *
  * @param ev         発生イベント
  */
 /*-------------------------------------------------------------------*/
 public boolean handleEvent(Event ev) {
  switch (ev.id) {
  case Event.WINDOW_DESTROY:
    dispose();
    return(true);
  }
  return(super.handleEvent(ev));
 }

}


// End of File
```

```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.19   T.Takumi(CSK)
 **************************************************************************/

import java.awt.*;
import java.io.*;

/**************************************************************************
 ** WINDOW_DESTROY イベントを取る Frame
 **
 **************************************************************************/
public class DFrame extends Frame {

                        /** アプリケーションメインオブジェクト */
 MediationFunc          medFunc;

 /*-------------------------------------------------------------------*/
 /** コンストラクタ
  *
  * @param appMain  アプリケーションメインオブジェクト
  * @param title    フレームのタイトル
  */
 /*-------------------------------------------------------------------*/
 public DFrame(MediationFunc appMain, String title) {
  super(title);
  medFunc = appMain;
 }

 /*-------------------------------------------------------------------*/
 /** WINDOW_DESTROY によりウインドウを消去
  *
  * @param ev          発生イベント
  */
 /*-------------------------------------------------------------------*/
 public boolean handleEvent(Event ev) {
  switch (ev.id) {
  case Event.WINDOW_DESTROY:
   if (this == medFunc.interestWin) {
    System.out.println("interestWin!!");
    medFunc.personalTopPanel.clearOffset();
    medFunc.personalWin.dispose();          // 個人化空間も消去
    medFunc.keywordList.deselectAll();
    medFunc.selectMode = true;
//    medFunc.selectCheckbox.setState(medFunc.selectMode);
    medFunc.selectCheckbox.setEnabled(true);
    medFunc.interestButton.setEnabled(true);
    medFunc.personalButton.setEnabled(false);
    medFunc.exhibitionWin.openMenuItem.setEnabled(true);
    medFunc.interestTopPanel.clearOffset();
   }
   else if (this == medFunc.personalWin) {
    System.out.println("personalWin!!");
    medFunc.personalTopPanel.clearOffset();
    medFunc.personalButton.setEnabled(true);
   }
```

```
    dispose();
    return(true);
   }
  return(false);
 }

}

// End of File
```



```
/*************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.29   T.Takumi(CSK)
 *************************************************************************/

import java.lang.Math;

/*************************************************************************
 ** ・空間距離
 **
 *************************************************************************/
public class Distance {
 Exhibit   ex[];
 int                exCnt;
                    /** 空間種別 */
 int                idx;

                    /** 標準偏差 */
 double    sigma;
                    /** 平均値 */
 double    ave;

                    /** 意味距離<br>対角に正規化距離を格納 */
 double    dist[][];

                              /** 0: 遠い */
 static final int   FAR       = 0;
                              /** 1: 近い */
 static final int   NEAR      = 1;
                              /** 2: それ以外 */
 static final int   OTHER     = 2;

 /*-----------------------------------------------------------------*/
 /** コンストラクタ<br>
  * 全体の展示オブジェクトに対して意味距離を計算
  *
  * @param exhibit    展示オブジェクト
  * @param objCnt     展示オブジェクトの数
  * @param index      空間種別
  */
 /*-----------------------------------------------------------------*/
 public Distance(Exhibit exhibit[], int objCnt, int index) {
  ex = exhibit;
  exCnt = objCnt;
  idx = index;

  // 初期化
  sigma = 0.0;
  ave = 0.0;

  dist = new double[exCnt][exCnt];
  for (int i = 0; i < exCnt; i++) {
   for (int j = 0; j < exCnt; j++) {
    dist[i][j] = 0.0;
   }
  }
```

```
  // 各オブジェクト間の距離をセット
  int cnt = 0;
  for (int i = 0; i < exCnt; i++) {
   for (int j = 0; j < exCnt; j++) {
    if (i < j) {
     dist[i][j] = calcDistance(i, j);
     ave += dist[i][j];
     cnt++;
    }
   }
  }

  // 平均、標準偏差のセット
  ave /= (double)cnt;
  for (int i = 0; i < exCnt; i++) {
   for (int j = 0; j < exCnt; j++) {
    if (i < j) {
     double tmp = dist[i][j] - ave;
     sigma += tmp*tmp;
        }
   }
  }
  sigma /= (double)cnt;
  sigma = Math.sqrt(sigma);

  // 距離の正規化
  for (int i = 0; i < exCnt; i++) {
   for (int j = 0; j < exCnt; j++) {
    if (i < j) {
     dist[j][i] = (dist[i][j] - ave)/sigma;      // 対角に格納
        }
   }
  }
}

/*-------------------------------------------------------------------------*/
/** コンストラクタ<br>
 * 展示オブジェクトのサブセットに対して意味距離を計算
 *
 * @param exhibit    展示オブジェクト
 * @param objCnt     展示オブジェクトの数
 * @param index      空間種別
 * @param subset     展示オブジェクトのサブセット
 */
/*-------------------------------------------------------------------------*/
public Distance(Exhibit exhibit[], int objCnt, int index, int subset[]) {
 ex = exhibit;
 exCnt = objCnt;
 idx = index;

 // 初期化
 sigma = 0.0;
 ave = 0.0;

 // 無駄な部分もあるが全オブジェクトの数の配列を取る
 dist = new double[exCnt][exCnt];
 for (int i = 0; i < exCnt; i++) {
```



```
  for (int j = 0; j < exCnt; j++) {
   dist[i][j] = 0.0;
  }
 }

 // 各オブジェクト間の距離をセット
 int cnt = 0;
 for (int i = 0; i < subset.length; i++) {
  int ii = subset[i];

  for (int j = 0; j < subset.length; j++) {
   int jj = subset[j];

   if (ii < jj) {
    dist[ii][jj] = calcDistance(ii, jj);
    ave += dist[ii][jj];
    cnt++;
   }
  }
 }

 // 平均、標準偏差のセット
 ave /= (double)cnt;
 for (int i = 0; i < subset.length; i++) {
  int ii = subset[i];

  for (int j = 0; j < subset.length; j++) {
   int jj = subset[j];

   if (ii < jj) {
    double tmp = dist[ii][jj] - ave;
    sigma += tmp*tmp;
       }
  }
 }
 sigma /= (double)cnt;
 sigma = Math.sqrt(sigma);

 // 距離の正規化
 for (int i = 0; i < subset.length; i++) {
  int ii = subset[i];

  for (int j = 0; j < subset.length; j++) {
   int jj = subset[j];

   if (ii < jj) {
    dist[jj][ii] = (dist[ii][jj] - ave)/sigma;    // 対角に格納
       }
  }
 }
}

/*-------------------------------------------------------------------------*/
/** 2オブジェクト間の意味距離の計算
 *
 * @param i          オブジェクト1のインデックス
 * @param j          オブジェクト2のインデックス
 * @return 意味距離
```

```
 */
/*-------------------------------------------------------------------------*/
public double calcDistance(int i, int j) {

  double d;
  double x = ex[i].comp1[idx] - ex[j].comp1[idx];
  double y = ex[i].comp2[idx] - ex[j].comp2[idx];

  d = x*x + y*y;
  d = Math.sqrt(d);

  return(d);
 }

/*-------------------------------------------------------------------------*/
/** 2オブジェクト間の意味距離の判定
 *
 * @param i             オブジェクト1のインデックス
 * @param j             オブジェクト2のインデックス
 * @return 判定結果
 * <ul>
 * <li> NEAR
 * <li> FAR
 * <li> OTHER
 * </ul>
 */
/*-------------------------------------------------------------------------*/
public int checkDistance(int i, int j) {
  double   val;

  if (i == j) {
    System.out.println("WARNING: checkDistance(): The same indexes.");
    return(OTHER);
  }
  else if (i < j) {                    // 正規化距離を選択
    val = dist[j][i];
  }
  else {
    val = dist[i][j];
  }

  if (val > 0.3) {
    return(FAR);
  }
  else if (val < -0.3) {
    return(NEAR);
  }
  else {
    return(OTHER);
  }
 }

/*-------------------------------------------------------------------------*/
/** テスト用表示(全体)
 */
/*-------------------------------------------------------------------------*/
public void print() {
  System.out.println("distance ----------------------------------");
```



```
  for (int i = 0; i < exCnt; i++) {
    for (int j = 0; j < exCnt; j++) {
      if (i < j) {
        System.out.println(dist[j][i]);
            }
    }
  }
 }

/*-------------------------------------------------------------------------*/
/** テスト用表示(一部)
 *
 * @param subset      サブセット
 */
/*-------------------------------------------------------------------------*/
public void print(int subset[]) {
  System.out.println("distance ----------------------------------");
  for (int i = 0; i < subset.length; i++) {
    int ii = subset[i];

    for (int j = 0; j < subset.length; j++) {
      int jj = subset[j];

      if (ii < jj) {
        System.out.println(dist[jj][ii]);
            }
    }
  }
 }

}


// End of File
```

```
/**********************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.05   T.Takumi(CSK)
 **********************************************************************/

import java.awt.*;
import java.util.*;

/**********************************************************************
 ** ・展示物オブジェクト
 **
 **********************************************************************/
public class Exhibit {
                        /** 展示オブジェクトファイル名 */
  String    fname;
                        /** 識別子 */
  String    id;
                        /** ラベル(タグ) */
  String    tag;
                        /** キーワード */
  Keyword   keyword[];
                        /** 説明文 */
  String    text[];
                        /** 行数 */
  int                   textLines;

                        /** タイトルを持っているか */
  boolean   hasTitle;
                        /** タイトル */
  String    title;

                        /** 画像を持っているか */
  boolean   hasImage;
                        /** 画像 */
  Image                 image;

                        /** 選択されているかどうか(Ov) */
  boolean   selected;
                        /** 表示の対象となっているか(Ov U Om) */
  boolean   included;

                        /** 第一主成分 */
  double    comp1[];
                        /** 第二主成分に寄与率の比をかけたもの */
  double    comp2[];

                        /** 表示のxオフセット */
  int                   xoffset[];
                        /** 表示のyオフセット */
  int                   yoffset[];

                        /** マウスのフォーカス */
  boolean   focus[];

                        /** 表示枠の大きさ */
  int                   frLeft[], frRight[];
```



```
                        /** 表示枠の大きさ */
  int                   frTop[], frBottom[];

  /*--------------------------------------------------------------------*/
  /** コンストラクタ
   *
   * @param keyMax   持てるキーワードの数の最大値
   * @param lineMax   キーワードの長さの最大値
   */
  /*--------------------------------------------------------------------*/
  public Exhibit(int keyMax, int lineMax) {
    keyword = new Keyword[keyMax];
    for (int i = 0; i < keyMax; i++) {
      keyword[i] = new Keyword();
    }

    text = new String[lineMax];
    textLines = 0;

    hasTitle = false;
    hasImage = false;
    comp1 = new double[3];
    comp2 = new double[3];
    xoffset = new int[3];
    yoffset = new int[3];
    focus = new boolean[3];
    frLeft = new int[3];
    frRight = new int[3];
    frTop = new int[3];
    frBottom = new int[3];
    for (int i = 0; i < 3; i++) {
      comp1[i] = 0.0;
      comp2[i] = 0.0;
      xoffset[i] = 0;
      yoffset[i] = 0;
      focus[i] = false;
    }
    selected = false;
    included = false;
  }

  /*--------------------------------------------------------------------*/
  /** 展示オブジェクトのファイル名をセット
   *
   * @param st          展示オブジェクトのファイル名
   */
  /*--------------------------------------------------------------------*/
  public void setFilename(String st) {
    fname = new String(st);
  ]

  /*--------------------------------------------------------------------*/
  /** ID をセット
   *
   * @param st          ID の文字列
   */
  /*--------------------------------------------------------------------*/
  public void setId(String st) {
```

```
  id = new String(st);
}

/*-----------------------------------------------------------------------*/
/** TAG をセット
 *
 * @param st          TAG の文字列をセット
 */
/*-----------------------------------------------------------------------*/
public void setTag(String st) {
  tag = new String(st);
}

/*-----------------------------------------------------------------------*/
/** TITLE をセット
 *
 * @param st          TITLE  の文字列をセット
 */
/*-----------------------------------------------------------------------*/
public void setTitle(String st) {
  hasTitle = true;
  title = new String(st);
}

/*-----------------------------------------------------------------------*/
/** キーワードインデックスと重みのセット
 *
 * @param idx         キーワードのインデックス
 * @param w           重み
 */
/*-----------------------------------------------------------------------*/
public void setKeyword(int idx, double w) {
  keyword[idx].listId = idx;
  keyword[idx].weight = w;
}

/*-----------------------------------------------------------------------*/
/** 画像を読み込む
 *
 * @param imagefile 画像ファイル名(.gif or .jpg)
 */
/*-----------------------------------------------------------------------*/
public void setImage(String imagefile) {
  hasImage = true;
  image = Toolkit.getDefaultToolkit().getImage(imagefile);
}

/*-----------------------------------------------------------------------*/
/** 説明文のセット
 *
 * @param string      展示オブジェクトの説明文
 */
/*-----------------------------------------------------------------------*/
public void setText(String string) {
  text[textLines] = new String(string);
  textLines++;
}
```



```
/*-----------------------------------------------------------------------*/
/** 主成分の計算<br>
 * すでに計算された主成分値をファイルから読み込んでセットする
 *
 * @param idx                 空間種別
 * @param c1          第一主成分
 * @param c2          第二主成分(寄与率の比でスケールされたもの)
 */
/*-----------------------------------------------------------------------*/
public void setComponent(int idx, double c1, double c2) {

 // idx について
 // 0: 展示空間(EXHIBITION)
 // 1: 興味空間(INTEREST)
 // 2: 個人化空間(PERSONAL)

 // 第 1 主成分
 comp1[idx] = c1;

 // 第 2 主成分
 comp2[idx] = c2;
}

/*-----------------------------------------------------------------------*/
/** 主成分の計算<br>
 * すべてのキーワードについて計算
 *
 * @param ev1                 固有値1
 * @param ev2                 固有値2
 * @param evec1       固有ベクトル1
 * @param evec2       固有ベクトル2
 */
/*-----------------------------------------------------------------------*/
public void calcComponent(double ev1, double ev2, double evec1[], double evec2[]) {

 // idx について
 // 0: 展示空間(EXHIBITION)
 // 1: 興味空間(INTEREST)
 // 2: 個人化空間(PERSONAL)

 // 第 1 主成分
 comp1[MediationFunc.EXHIBITION] = 0.0;
 for (int i = 0; i < evec1.length; i++) {
  comp1[MediationFunc.EXHIBITION] += evec1[i]*keyword[i].weight;
 }

 // 第 2 主成分
 comp2[MediationFunc.EXHIBITION] = 0.0;
 for (int i = 0; i < evec2.length; i++) {
  comp2[MediationFunc.EXHIBITION] += evec2[i]*keyword[i].weight;
 }

 // 第 2 主成分には寄与率の比でスケールしておく
 if (ev2 != 0.0) comp2[MediationFunc.EXHIBITION] *= ev1/ev2;
}

/*-----------------------------------------------------------------------*/
/** 主成分の計算<br>
```

```
 * キーワードのサブセットについて計算
 *
 * @param ev1                   固有値1
 * @param ev2                   固有値2
 * @param evec1     固有ベクトル1
 * @param evec2     固有ベクトル2
 * @param idx                   空間の種類
 * @param subkey    キーワードのサブセット
 */
/*---------------------------------------------------------------------*/
public void calcComponent(double ev1, double ev2, double evec1[], double evec2[], int idx, int subkey[]) {

 // idx について
 // 0: 展示空間(EXHIBITION)
 // 1: 興味空間(INTEREST)
 // 2: 個人化空間(PERSONAL)

 int        ii;

 // 第 1 主成分
 comp1[idx] = 0.0;
 for (int i = 0; i < evec1.length; i++) {
  ii = subkey[i];
  comp1[idx] += evec1[i]*keyword[ii].weight;
 }

 // 第 2 主成分
 comp2[idx] = 0.0;
 for (int i = 0; i < evec2.length; i++) {
  ii = subkey[i];
  comp2[idx] += evec2[i]*keyword[ii].weight;
 }

 // 第 2 主成分には寄与率の比でスケールしておく
 if (ev2 != 0.0) comp2[idx] *= ev1/ev2;
}

/*---------------------------------------------------------------------*/
/** テスト表示
 *
 * @param klist        キーワードリスト
 */
/*---------------------------------------------------------------------*/
public void print(KeywordList klist) {
  System.out.println("TAG ----------------------------------------");
  System.out.println(tag);
  System.out.println();
  for (int i = 0; i < klist.kwCnt; i++) {
    if (keyword[i].listId >= 0) {
      System.out.println(i + ": " + klist.keyword[i]
                    + ": " + keyword[i].weight + "   ");
    }
  }
  System.out.println();
 }
}
```



```
/**********************************************************************
 ** ・キーワード
 **
 **********************************************************************/
class Keyword {
                        /** キーワードリスト内でのインデックス */
 int                    listId;
                        /** 重み */
 double     weight;

 /*---------------------------------------------------------------------*/
 /** コンストラクタ
  */
 /*---------------------------------------------------------------------*/
 public Keyword() {
  listId = -1;
  weight = 0.0;
 }
}
```

```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.23   T.Takumi(CSK)
 **************************************************************************/

import java.awt.*;

/**************************************************************************
 ** ・オブジェクトの情報ダイアログ<br>
 ** 展示空間のみ、重み表示と更新ボタンを持つ
 **
 **************************************************************************/
public class InfoDialog extends DDialog {
  Exhibit    ex;
  KeywordList          klist;
  List                 list;
  Label                label;
  TextField textField;
  Button     button;
                       /** 空間種類 */
  int                  index;

  /*-----------------------------------------------------------------*/
  /** コンストラクタ
   *
   * @param parent     親フレーム
   * @param exhibit    展示オブジェクト
   * @param keylist    キーワードリスト
   * @param idx                空間種類
   */
  /*-----------------------------------------------------------------*/
  public InfoDialog(Frame parent, Exhibit exhibit, KeywordList keylist, int idx) {
    super(parent, false);

    if (exhibit.hasTitle) {
      setTitle(exhibit.title);
    }
    else {
      setTitle(exhibit.tag);
    }

    ex = exhibit;
    klist = keylist;
    index = idx;

    // キーワード表示部
    list = new List();
    list.setMultipleMode(false);
    add("Center", list);

    if (index == MediationFunc.EXHIBITION) {          // 展示空間のみ
      Panel p = new Panel();
      add("South", p);
      p.setLayout(new BorderLayout());

      Panel p1 = new Panel();
```



```
      Panel p2 = new Panel();
      p.add("North", p1);
      p.add("South", p2);

      label = new Label("重み");
      textField = new TextField(5);
      button = new Button("更新");
      p1.add(label);
      p1.add(textField);
      p2.add(button);
    }

    // デフォルトフォントの設定
/*
    System.out.println("InfoDialog ==> "+getFont());
    Font newFont = new Font("Dialog", Font.PLAIN, 20);
    setFont(newFont);
    System.out.println(getFont());
*/
  }

  /*-----------------------------------------------------------------*/
  /** キーワードをセット<br>
   * <br>
   * 展示空間は持っているすべてのキーワードを表示<br>
   * その他の空間は選択されたオブジェクトに含まれるキーワードのみ表示
   */
  /*-----------------------------------------------------------------*/
  public void setKeywords() {
    for (int i = 0; i < klist.kwCnt; i++) {
      if (ex.keyword[i].listId >= 0) {
        // 展示空間は持っているすべてのキーワードを表示
            if (index == MediationFunc.EXHIBITION) {
              list.addItem(klist.keyword[i]);
            }
        // その他の空間は選択されたオブジェクトに含まれるキーワードのみ表示
            else {
          if (klist.selected[i]) {
              list.addItem(klist.keyword[i]);
            }
          }
      }
    }
  }

  /*-----------------------------------------------------------------*/
  /** リストの項目を選択
   *
   * @param idx                キーワードリストのインデックス
   */
  /*-----------------------------------------------------------------*/
  public void selectItem(int idx) {
    list.select(idx);
    // 展示空間は重みを表示
    if (index == MediationFunc.EXHIBITION) {
      textField.setText(String.valueOf(ex.keyword[listIndex2KeyIndex(idx)].weight));
    }
  }
```

```
/*-----------------------------------------------------------------*/
/** リストのインデックスを各オブジェクトのキーワードインデックスに変換
 *
 * @return 各オブジェクトのキーワードのインデックス
 */
/*-----------------------------------------------------------------*/
public int listIndex2KeyIndex() {
  int idx = list.getSelectedIndex();
  int cnt = -1;
  for (int i = 0; i < klist.kwCnt; i++) {
    if (ex.keyword[i].listId >= 0) cnt++;
    if (idx == cnt) return(i);
  }
  return(-1);
}

/*-----------------------------------------------------------------*/
/** リストのインデックスを各オブジェクトのキーワードインデックスに変換
 *
 * @param idx          キーワードリストのインデックス
 * @return 各オブジェクトのキーワードのインデックス
 */
/*-----------------------------------------------------------------*/
public int listIndex2KeyIndex(int idx) {
  int cnt = -1;
  for (int i = 0; i < klist.kwCnt; i++) {
    if (ex.keyword[i].listId >= 0) cnt++;
    if (idx == cnt) return(i);
  }
  return(-1);
}

/*-----------------------------------------------------------------*/
/** 特定イベント処理
 *
 * @param ev           発生イベント
 */
/*-----------------------------------------------------------------*/
public boolean handleEvent(Event ev) {
  switch (ev.id) {
  case Event.ACTION_EVENT:
    if (ev.target == button) {
      ex.keyword[listIndex2KeyIndex()].weight
              = Double.valueOf(textField.getText()).doubleValue();
    }
    return(true);

  case Event.LIST_SELECT:
    // 展示空間は重みを表示
    if (index == MediationFunc.EXHIBITION) {
            textField.setText(String.valueOf(ex.keyword[listIndex2KeyIndex()].weight));
    }
    return(true);
  }
  return(super.handleEvent(ev));
}
```



```
}

// End of File
```

```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.05   T.Takumi(CSK)
 **************************************************************************/

/**************************************************************************
 ** ・キーワードリスト
 **
 **************************************************************************/
public class KeywordList {
                        /** キーワード文字列 */
 String     keyword[];
                        /** キーワードの数 */
 int                    kwCnt;
                        /** そのキーワードが選択されているか */
 boolean    selected[];

 /*-------------------------------------------------------------------*/
 /** コンストラクタ
  *
  * @param keyMax   キーワードの数の最大値
  */
 /*-------------------------------------------------------------------*/
 public KeywordList(int keyMax) {
  keyword = new String[keyMax];
  kwCnt = 0;
  selected = new boolean[keyMax];
  for (int i = 0; i < selected.length; i++) {
   selected[i] = false;
  }
 }

 /*-------------------------------------------------------------------*/
 /** キーワードを追加
  *
  * @param kw          追加するキーワード文字列
  * @return            キーワードリスト内でのインデックス
  */
 /*-------------------------------------------------------------------*/
 public int addItem(String kw) {
  for (int i = 0; i < kwCnt; i++) {
   if (keyword[i].equals(kw)) {
    return(i);
   }
  }

  keyword[kwCnt] = new String(kw);
  kwCnt++;

  return(kwCnt-1);
 }

 /*-------------------------------------------------------------------*/
 /** すべてのキーワードを未選択に
  */
 /*-------------------------------------------------------------------*/
```



```
 public void deselectAll() {
  for (int i = 0; i < kwCnt; i++) {
   selected[i] = false;
  }
 }

 /*-------------------------------------------------------------------*/
 /** テスト表示
  */
 /*-------------------------------------------------------------------*/
 public void print() {
  System.out.println("KEYWORD ----------------------------------------");
  for (int i = 0; i < kwCnt; i++) {
   System.out.println(keyword[i]);
  }
  System.out.println();
 }
}
```

```
/**********************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.19   T.Takumi(CSK)
 **********************************************************************/

import java.awt.*;
import java.io.*;

/**********************************************************************
 ** ・メインウインドウ
 **
 **********************************************************************/
public class MainFrame extends Frame {
                        /** confirm ダイアログの返し */
  static int   confirm;

                        /** アプリケーションメインオブジェクト */
  MediationFunc         medFunc;

                        /** [開く]のメニュー項目 */
  MenuItem              openMenuItem;

  Menu                  subMenu0;
  Menu                  subMenu1;
  Menu                  subMenu2;
  Menu                  subMenu3;

  /*--------------------------------------------------------------------*/
  /** コンストラクタ
   *
   * @param appMain  アプリケーションメインオブジェクト
   * @param title    フレームのタイトル
   */
  /*--------------------------------------------------------------------*/
  public MainFrame(MediationFunc appMain, String title) {
    super(title);
    medFunc = appMain;

    // メニューを作成
    MenuBar menuBar = new MenuBar();

    Menu fileMenu = new Menu("ファイル");
    openMenuItem = new MenuItem("開く");
    fileMenu.add(openMenuItem);
    fileMenu.add(new MenuItem("主成分値を保存"));
    fileMenu.addSeparator();
    fileMenu.add(new MenuItem("終了"));

    Menu fileMenu2 = new Menu("フォント");

    subMenu0 = new Menu("すべての空間");
    subMenu1 = new Menu("展示空間");
    subMenu2 = new Menu("興味空間");
    subMenu3 = new Menu("個人化空間");

    fileMenu2.add(subMenu1);
```



```
    fileMenu2.add(subMenu2);
    fileMenu2.add(subMenu3);
    fileMenu2.addSeparator();
    fileMenu2.add(subMenu0);

    subMenu0.add(new MenuItem("8"));
    subMenu0.add(new MenuItem("10"));
    subMenu0.add(new MenuItem("12"));
    subMenu0.add(new MenuItem("14"));
    subMenu0.add(new MenuItem("16"));
    subMenu0.add(new MenuItem("18"));
    subMenu0.add(new MenuItem("20"));

    subMenu1.add(new MenuItem("8"));
    subMenu1.add(new MenuItem("10"));
    subMenu1.add(new MenuItem("12"));
    subMenu1.add(new MenuItem("14"));
    subMenu1.add(new MenuItem("16"));
    subMenu1.add(new MenuItem("18"));
    subMenu1.add(new MenuItem("20"));

    subMenu2.add(new MenuItem("8"));
    subMenu2.add(new MenuItem("10"));
    subMenu2.add(new MenuItem("12"));
    subMenu2.add(new MenuItem("14"));
    subMenu2.add(new MenuItem("16"));
    subMenu2.add(new MenuItem("18"));
    subMenu2.add(new MenuItem("20"));

    subMenu3.add(new MenuItem("8"));
    subMenu3.add(new MenuItem("10"));
    subMenu3.add(new MenuItem("12"));
    subMenu3.add(new MenuItem("14"));
    subMenu3.add(new MenuItem("16"));
    subMenu3.add(new MenuItem("18"));
    subMenu3.add(new MenuItem("20"));

    menuBar.add(fileMenu);
    menuBar.add(fileMenu2);
    setMenuBar(menuBar);
  }

  /*--------------------------------------------------------------------*/
  /** メニューイベントの処理
   *
   * @param evt                 発生イベント
   * @param arg                 イベントの名前
   */
  /*--------------------------------------------------------------------*/
  public boolean action(Event evt, Object arg) {
    if ("終了".equals(arg)) {
      System.exit(0);
    }
    else if ("開く".equals(arg)) {
      FileDialog fileDialog = new FileDialog(this, "キーワードファイルを開く", FileDialog.LOAD);
      fileDialog.setDirectory("./data");
      fileDialog.setFile("*.dat");
```

```
    //現在うまく働かない(JDK のバグらしい)
    fileDialog.setFilenameFilter(new SuffixFilenameFilter("dat"));

    fileDialog.show();
    String filename = fileDialog.getFile();
    String dir = fileDialog.getDirectory();
    System.out.println(">>> File ==> "+filename);
    System.out.println(">>> Dir  ==> "+dir);
    if (filename != null) {
     if (!dir.endsWith(File.separator)) {
      dir += File.separator;
     }
     dir += filename;

     if (medFunc.makeExhibitionSpace(dir)) {
      medFunc.exhibitionTopPanel.resetOffset();
      medFunc.exhibitionTopPanel.resetScale();
      medFunc.exhibitionTopPanel.funcPanel.clear();
      medFunc.exhibitionTopPanel.funcPanel.repaint();
     }
    }
    return(true);
   }
   else if ("主成分値を保存".equals(arg)) {

/* 簡単に上書きできてしまうので、現在は FileDialog による指定は保留
    FileDialog fileDialog = new FileDialog(this, "展示空間の主成分値をファイルに保存", FileDialog.SAVE);
    fileDialog.setDirectory("./data");
    fileDialog.setFile("default.dat");

    fileDialog.show();
    String filename = fileDialog.getFile();
    String dir = fileDialog.getDirectory();
    System.out.println(">>> File ==> "+filename);
    System.out.println(">>> Dir  ==> "+dir);
*/

    ConfirmSaveDialog cs = new ConfirmSaveDialog(this);
    Point point = getLocation();
    cs.setLocation(point.x+50, point.y+50);
    cs.show();
    if (confirm == YesNoDialog.YES) {

            String filename = new String("saveComponents.dat");
            String dir = new String("./data");
            System.out.println(">>> File ==> "+filename);
            System.out.println(">>> Dir  ==> "+dir);

            if (filename != null) {
             if (!dir.endsWith(File.separator)) {
              dir += File.separator;
             }
             dir += filename;

             //展示空間の主成分値をファイルに保存
             medFunc.saveObjectData(dir);
            }
    }
```



```
     return(true);
    }
    else {
     try {
      int i = Integer.valueOf((String)evt.arg).intValue();
      Menu tmpMenu = (Menu)((MenuItem)evt.target).getParent();

      //各パネルのフォントを設定
      if (tmpMenu == subMenu1) {
       medFunc.exhibitionTopPanel.funcPanel.setFontAndFM(i);
      }
      else if (tmpMenu == subMenu2) {
       if (medFunc.interestWin.isShowing()) {
        medFunc.interestTopPanel.funcPanel.setFontAndFM(i);
       }
      }
      else if (tmpMenu == subMenu3) {
       if (medFunc.personalWin.isShowing()) {
        medFunc.personalTopPanel.funcPanel.setFontAndFM(i);
       }
      }
      else {
       medFunc.exhibitionTopPanel.funcPanel.setFontAndFM(i);
       if (medFunc.interestWin.isShowing()) {
        medFunc.interestTopPanel.funcPanel.setFontAndFM(i);
       }
       if (medFunc.personalWin.isShowing()) {
        medFunc.personalTopPanel.funcPanel.setFontAndFM(i);
       }
      }

      return(true);
     } catch (NumberFormatException e) {
     }
    }

    return(false);
   }
}


// End of File
```

```
/**********************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.01.12   T.Takumi(CSK)
 **********************************************************************/

import java.applet.Applet;
import java.awt.*;
import java.io.*;
import java.util.*;

/**********************************************************************
 ** アプリケーションメインクラス<br>
 ** main class
 **********************************************************************/
public class MediationFunc {

  /*-------------------------------------------------------------------*/
  /** 定数
   */
  /*-------------------------------------------------------------------*/
  static final Color nodeColor = Color.blue;
  static final Color selectColor = new Color(250, 220, 100);
//  static final Color selectColor = Color.white;
  static final Color focusColor = Color.red;
  static final Color frameColor = Color.black;
  static final Color stringColor = Color.white;
  static final Color selectStringColor = Color.blue;

  static final int          maxExhibits = 100;
  static final int          maxKeywords = 850;
  static final int          maxTextLines = 100;

                            /** 0: 展示空間 */
  static final int          EXHIBITION = 0;
                            /** 1: 興味空間 */
  static final int          INTEREST  = 1;
                            /** 2: 個人化空間 */
  static final int          PERSONAL  = 2;

                            /** 全てのキーワードと重みが同じ */
  static final int          SAME_ALL  = 0;
                            /** 同じでない */
  static final int          DIFFERENT = 1;

  /*-------------------------------------------------------------------*/
  /** グローバルオブジェクト
   */
  /*-------------------------------------------------------------------*/
                            /** ソケットのサーバホスト名 */
  String      hostName = null;
                            /** ソケットのポート番号 */
  int                       outPort = -1;

                            /** 展示オブジェクト全部(Oc) */
  Exhibit     exhibit[];
                            /** 展示オブジェクトの数 */
```



```
  int                       objCnt;
                            /** キーワードリスト(Kc) */
  KeywordList               keywordList;
                            /** 主成分がすでに計算されているか */
  boolean     calculated;

                            /** サブセット Ov */
  int                       interestSubset[];
                            /** Ov の数 */
  int                       cntOv = 0;
                            /** サブセット Kv */
  int                       interestSubkey[];
                            /** Kv の数 */
  int                       cntKv = 0;
                            /** サブセット OvOm */
  int                       personalSubset[];
                            /** OvOm の数 */
  int                       cntOvOm = 0;
/**** **** ****
  int                       personalSubkey[];       // km
  int                       cntKm = 0;
**** **** ****/
                                    /** 全体の分散共分散行列 */
  CovarianceMatrixX         covarianceMat;
                                    /** 興味空間の分散共分散行列 */
  CovarianceMatrixX         interestCMat;
                                    /** 個人化空間の分散共分散行列 */
  CovarianceMatrixX         personalCMat;

                            /** メイン親ウインドウ */
  MainFrame                 exhibitionWin;
                            /** 興味空間ウインドウ */
  DFrame      interestWin;
                            /** 個人化空間ウインドウ */
  DFrame      personalWin;

                            /** 展示空間トップパネル */
  MediationFuncTopPanel                 exhibitionTopPanel;
                            /** 興味空間トップパネル */
  TopPanel interestTopPanel;
                            /** 個人化空間トップパネル */
  TopPanel personalTopPanel;

                            /** 縦方向スクロールバー */
  Scrollbar  vscroll;
                            /** 横方向スクロールバー */
  Scrollbar  hscroll;

                            /** 選択可能状態表示 */
  Checkbox selectCheckbox;
                            /** 表示スケールスライダ */
  Scrollbar  scaleSlider;

                            /** 興味空間表示用ボタン */
  Button      interestButton;
                            /** 個人化空間表示用ボタン */
  Button      personalButton;
```

```
                    /** オブジェクト選択モード */
boolean    selectMode;

                    /** パターン分け */
Pattern    pattern;

/*-------------------------------------------------------------------*/
/** コンストラクタ
 */
/*-------------------------------------------------------------------*/
public MediationFunc() {

 // 展示空間
 exhibitionWin = new MainFrame(this, "展示空間");
 exhibitionTopPanel = new MediationFuncTopPanel(this);
 exhibitionWin.add("Center", exhibitionTopPanel);

 // 興味空間
 interestWin = new DFrame(this, "興味空間");
 interestTopPanel = new TopPanel(this, INTEREST);

 // 個人化空間
 personalWin = new DFrame(this, "個人化空間");
 personalTopPanel = new TopPanel(this, PERSONAL);
}


/*-------------------------------------------------------------------*/
/** 正規化距離を計算しパターン分けする
 */
/*-------------------------------------------------------------------*/
public void divideIntoPatterns() {
 // 正規化距離を計算
 Distance distExhibition = new Distance(exhibit, objCnt, EXHIBITION);
 Distance distInterest   = new Distance(exhibit, objCnt, INTEREST, personalSubset);
 Distance distPersonal   = new Distance(exhibit, objCnt, PERSONAL, personalSubset);

 // パターンデータ
 pattern = new Pattern(8, maxExhibits);
 pattern.setLabel(1, "パターン１");
 pattern.setLabel(2, "パターン２");
 pattern.setLabel(3, "パターン３");
 pattern.setLabel(4, "パターン４");
 pattern.setLabel(5, "パターン５");
 pattern.setLabel(6, "パターン６");
 pattern.setLabel(7, "パターン７");
 pattern.setLabel(8, "パターン８");

 // パターン分け
 for (int i = 0; i < personalSubset.length; i++) {
  int ii = personalSubset[i];

  for (int j = 0; j < personalSubset.length; j++) {
   int jj = personalSubset[j];

   if (ii < jj) {
    int distEx = distExhibition.checkDistance(ii, jj);
    int distIn = distInterest.checkDistance(ii, jj);
```



```
    int distPe = distPersonal.checkDistance(ii, jj);

    if (distEx == Distance.NEAR) {
     if (distIn == Distance.NEAR) {
      if (distPe == Distance.NEAR) {
       // パターン１
       pattern.checkPatternList(1, ii, jj);
      }
      else if (distPe == Distance.FAR) {
       // パターン２
       pattern.checkPatternList(2, ii, jj);
      }
     }
     else if (distIn == Distance.FAR) {
      if (distPe == Distance.NEAR) {
       // パターン３
       pattern.checkPatternList(3, ii, jj);
      }
      else if (distPe == Distance.FAR) {
       // パターン４
       pattern.checkPatternList(4, ii, jj);
      }
     }
    }
    else if (distEx == Distance.FAR) {
     if (distIn == Distance.NEAR) {
      if (distPe == Distance.NEAR) {
       // パターン５
       pattern.checkPatternList(5, ii, jj);
      }
      else if (distPe == Distance.FAR) {
       // パターン６
       pattern.checkPatternList(6, ii, jj);
      }
     }
     else if (distIn == Distance.FAR) {
      if (distPe == Distance.NEAR) {
       // パターン７
       pattern.checkPatternList(7, ii, jj);
      }
      else if (distPe == Distance.FAR) {
       // パターン８
       pattern.checkPatternList(8, ii, jj);
      }
     }
          }

         }

  } // for(j)
 }  // for(i)

 // テスト用表示
 pattern.print();

 // テスト用表示
//  distExhibition.print();
}
```

```
/*-----------------------------------------------------------------*/
/** 興味空間を作成
 *
 * @param ev          イベント
 */
/*-----------------------------------------------------------------*/
public boolean makeInterestSpace(Event ev) {
 int        cnt;

 // 選択されているオブジェクトに含まれるキーワードをチェック
 cntOv = 0;
 cntKv = 0;
 keywordList.deselectAll();
 for (int i = 0; i < objCnt; i++) {
  if (exhibit[i].selected) {
   cntOv++;
   for (int j = 0; j < keywordList.kwCnt; j++) {
    if (exhibit[i].keyword[j].listId >= 0) {
     if (!keywordList.selected[j]) {
      cntKv++;
      keywordList.selected[j] = true;
           }
    }
         }
  }
 }

 // 選択されたオブジェクトが1つ以下の場合
 if (cntOv < 2) {
  System.out.println("WARNING: Select more than 2 objects.");
  WarningDialog d = new WarningDialog(exhibitionWin, "Warning",
                "２つ以上のオブジェクトを選択してください");
  Point point = exhibitionTopPanel.getParent().getLocation();
  d.setLocation(point.x+ev.x, point.y+ev.y);
  d.show();
  keywordList.deselectAll();
  return(false);
 }

 // 選択されたオブジェクトがすべて同じキーワードと重みを持っている場合
 if (checkProperty() == SAME_ALL) {
  System.out.println("WARNING: Selected objects have the same keywords & weight.");
  WarningDialog d = new WarningDialog(exhibitionWin, "Warning",
                "選択されたオブジェクトはすべて同じキーワードと同じ重みを持っています。\n"+"他のオブジェクトも
  Point point = exhibitionTopPanel.getParent().getLocation();
  d.setLocation(point.x+ev.x, point.y+ev.y);
  d.show();
  keywordList.deselectAll();
  return(false);
 }

 // 選択された展示オブジェクト(Ov)のインデックスリストを作成
 interestSubset = new int[cntOv];
 cnt = 0;
 for (int i = 0; i < objCnt; i++) {
  if (exhibit[i].selected) {
   interestSubset[cnt] = i;
```



```
    cnt++;
   }
  }
  if (cnt != cntOv) {
   System.out.println("ERROR: cntOv is wrong.");
   return(false);
  }

  // 選択されたキーワード(Kv)のインデックスリストを作成
  System.out.println("cntKv ==> "+cntKv);
  interestSubkey = new int[cntKv];
  cnt = 0;
  for (int i = 0; i < keywordList.kwCnt; i++) {
   if (keywordList.selected[i]) {
//     System.out.println("cnt ==> "+cnt+"  "+keywordList.keyword[i]);
    interestSubkey[cnt] = i;
    cnt++;
   }
  }
  if (cnt != cntKv) {
   System.out.println("ERROR: cntKv is wrong.");
   return(false);
  }

  // 分散共分散行列
  System.out.println("CovarianceMatrixX(interest) --- start");
  interestCMat
   = new CovarianceMatrixX(interestSubkey, exhibit, interestSubset, cntOv);
//  interestCMat.printMat();
//  interestCMat.printAve();
  System.out.println("CovarianceMatrixX(interest) --- end");

  // 固有値、固有ベクトルの計算
  System.out.println("syev --- start");
  interestCMat.syev();
  System.out.println("syev --- end");
//  interestCMat.printEvec();
//  interestCMat.printEv();

  // 固有値、固有ベクトルの取得
  double ev1, ev2;
  double evec1[] = new double[cntKv];
  double evec2[] = new double[cntKv];

  ev1 = interestCMat.getEVal(1);
  ev2 = interestCMat.getEVal(2);
  interestCMat.getEVec(1, evec1);
  interestCMat.getEVec(2, evec2);
  System.out.println("Eigenvalues ==> "+ev1+"  "+ev2);

/*
  System.out.println("Eigenvector1 --->");
  for (int i = 0; i < cntKv; i++) {
   System.out.print(evec1[i]+" ");
  }
  System.out.println();

  System.out.println("Eigenvector2 --->");
```

```
  for (int i = 0; i < cntKv; i++) {
   System.out.print(evec2[i]+" ");
  }
  System.out.println();
*/

  // 選択されているキーワードを含むオブジェクトをチェック(Ov U Om)
  cntOvOm = 0;
  for (int i = 0; i < objCnt; i++) {
   for (int j = 0; j < cntKv; j++) {
    if (exhibit[i].keyword[interestSubkey[j]].listId >= 0) {
     exhibit[i].included = true;
     cntOvOm++;
     break;
        }
    else {
     exhibit[i].included = false;
    }
   }
  }

  // 選択された展示オブジェクト(Ov U Om)のインデックスリストを作成
  personalSubset = new int[cntOvOm];
  cnt = 0;
  for (int i = 0; i < objCnt; i++) {
   if (exhibit[i].included) {
    personalSubset[cnt] = i;
    cnt++;
   }
  }
  if (cnt != cntOvOm) {
   System.out.println("ERROR: cntOvOm is wrong.");
   return(false);
  }

  // 主成分の計算
  for (int i = 0; i < cntOvOm; i++) {
   // 興味空間の表示は (Ov U Om)を使う
   // キーワードは Kv
   exhibit[personalSubset[i]].calcComponent(ev1, ev2, evec1, evec2, INTEREST, interestSubkey);
  }

  // 描画の順番
  interestTopPanel.funcPanel.setDrawOrder();

  System.out.println("Ov: "+cntOv+"  Kv: "+cntKv+"  OvOm: "+cntOvOm);
  return(true);
 }

 /*-------------------------------------------------------------------------*/
 /** 個人化空間を作成
  */
 /*-------------------------------------------------------------------------*/
 public boolean makePersonalSpace() {
  int        cnt;

/**** **** ****
  // 選択されているオブジェクトに含まれるキーワードをチェック
```



```
  cntKm = 0;
  keywordList.deselectAll();
  for (int i = 0; i < objCnt; i++) {
   if (exhibit[i].included) {
    for (int j = 0; j < keywordList.kwCnt; j++) {
     if (exhibit[i].keyword[j].listId >= 0) {
      if (!keywordList.selected[j]) {
       cntKm++;
       keywordList.selected[j] = true;
           }
      }
          }
   }
  }

  // 選択されたキーワード(Km)のインデックスリストを作成
  System.out.println("cntKm ==> "+cntKm);
  personalSubkey = new int[cntKm];
  cnt = 0;
  for (int i = 0; i < keywordList.kwCnt; i++) {
   if (keywordList.selected[i]) {
//     System.out.println("cnt ==> "+cnt+"  "+keywordList.keyword[i]);
    personalSubkey[cnt] = i;
    cnt++;
   }
  }
  if (cnt != cntKm) {
   System.out.println("ERROR: cntKm is wrong.");
   return(false);
  }
 **** **** ****/

  // 分散共分散行列
  System.out.println("CovarianceMatrixX(personal) --- start");
  personalCMat
//    = new CovarianceMatrixX(personalSubkey, exhibit, personalSubset, cntOvOm);
    = new CovarianceMatrixX(interestSubkey, exhibit, personalSubset, cntOvOm);
//  personalCMat.printMat();
//  personalCMat.printAve();
  System.out.println("CovarianceMatrixX(personal) --- end");

  // 固有値、固有ベクトルの計算
  System.out.println("syev --- start");
  personalCMat.syev();
  System.out.println("syev --- end");
//  personalCMat.printEvec();
//  personalCMat.printEv();

  // 固有値、固有ベクトルの取得
  double ev1, ev2;
//  double evec1[] = new double[cntKm];
//  double evec2[] = new double[cntKm];
  double evec1[] = new double[cntKv];
  double evec2[] = new double[cntKv];

  ev1 = personalCMat.getEVal(1);
  ev2 = personalCMat.getEVal(2);
  personalCMat.getEVec(1, evec1);
```

```
    personalCMat.getEVec(2, evec2);
    System.out.println("Eigenvalues ==> "+ev1+"  "+ev2);

/*
    System.out.println("Eigenvector1 --->");
    for (int i = 0; i < cntKm; i++) {
      System.out.print(evec1[i]+" ");
    }
    System.out.println();

    System.out.println("Eigenvector2 --->");
    for (int i = 0; i < cntKm; i++) {
      System.out.print(evec2[i]+" ");
    }
    System.out.println();
*/

    // 主成分の計算
    for (int i = 0; i < cntOvOm; i++) {
      // 個人化空間の表示は (Ov U Om)を使う
      // キーワードは Kv
//      exhibit[personalSubset[i]].calcComponent(ev1, ev2, evec1, evec2, PERSONAL, personalSubkey);
      exhibit[personalSubset[i]].calcComponent(ev1, ev2, evec1, evec2, PERSONAL, interestSubkey);
    }

    // 描画の順番
    personalTopPanel.funcPanel.setDrawOrder();

//    System.out.println("OvOm: "+cntOvOm+"  Km: "+cntKm);
    System.out.println("OvOm: "+cntOvOm+"  Kv: "+cntKv);
    return(true);
  }

  /*-------------------------------------------------------------------------*/
  /** ファイルからオブジェクトデータを読み込む
   *
   * @param filename  ファイル名
   */
  /*-------------------------------------------------------------------------*/
  public boolean readObjectData(String filename) {
    String              buf;
    FileInputStream     fs = null;
    BufferedReader      br = null;

    // 各オブジェクトファイル名を読み込む
    try {
      fs = new FileInputStream(filename);
                                          // SJIS JIS EUCJIS JISAutoDetect
      br = new BufferedReader(new InputStreamReader(fs, "JISAutoDetect"));
    } catch (Exception e) {
      System.out.println(e);
//      System.exit(1);
      return(false);
    }

    // 主成分計算
    calculated = false;
```



```
    String    token;
    double    comp1, comp2;

    while (true) {
      try {
            buf = br.readLine();    // read 1 line
            if (buf == null) break;
      } catch (IOException e) {
            System.out.println(e);  // no newlines are in buf
            break;
      }

      StringTokenizer st = new StringTokenizer(buf);
      try {
        token = st.nextToken();
        if (token.equals("CALCULATED")) {
          calculated = true;
        }
        else if (token.equals("OBJECT")) {
          if (st.hasMoreTokens()) {
            if (calculated) {// 主成分値が既に計算されている
              token = st.nextToken();
              comp1 = Double.valueOf(st.nextToken()).doubleValue();
              comp2 = Double.valueOf(st.nextToken()).doubleValue();
                                                          // 各オブジェクトのデータ
              readEachObject(token, comp1, comp2);
            }
            else {
              token = st.nextToken();
              readEachObject(token, 0.0, 0.0);      // 各オブジェクトのデータ
            }
          }
          else {
            System.out.println("WARNING: OBJECT file is not specified.");
          }
        }
      } catch (Exception e) {
//        System.out.println(e);
      }
    }

    // file stream の close
    try {
      fs.close ();
    } catch (IOException e) {
      System.out.println (e);
    }

    return(true);
  }

  /*-------------------------------------------------------------------------*/
  /** ファイルから各オブジェクトデータを読み込む
   *
   * @param filename  ファイル名
   * @param comp1     第一主成分値
   * @param comp2     第二主成分値
   */
```

```
/*-------------------------------------------------------------------*/
public boolean readEachObject(String filename, double comp1, double comp2) {
  String              buf;
  FileInputStream     fs = null;
  BufferedReader      br = null;

  // ID, TAG, TITLE, KEYWORD, TEST をファイルから読み込む
  try {
   fs = new FileInputStream(filename);
                                        // SJIS JIS EUCJIS JISAutoDetect
   br = new BufferedReader(new InputStreamReader(fs, "JISAutoDetect"));
  } catch (Exception e) {
   System.out.println(e);
//    System.exit(1);
   return(false);
  }

  String token;

  while (true) {
   try {
          buf = br.readLine();    // read 1 line
          if (buf == null) break;
   } catch (IOException e) {
          System.out.println(e);  // no newlines are in buf
          break;
   }

   StringTokenizer st = new StringTokenizer(buf);
   try {
    token = st.nextToken();
          if (token.equals("ID")) {                    // 識別子の取得
            System.out.println("ID: "+buf);

     if (st.hasMoreTokens()) {
      token = st.nextToken();
      exhibit[objCnt] = new Exhibit(maxKeywords, maxTextLines);
      exhibit[objCnt].setFilename(filename);
      exhibit[objCnt].setId(token);
      exhibit[objCnt].setComponent(EXHIBITION, comp1, comp2);
      objCnt++;
     }
     else {
      System.out.println("WARNING: ID is not specified.");
      exhibit[objCnt] = new Exhibit(maxKeywords, maxTextLines);
      exhibit[objCnt].setFilename(filename);
      exhibit[objCnt].setId("");                   // ID なし
      exhibit[objCnt].setComponent(EXHIBITION, comp1, comp2);
      objCnt++;
     }
          }
          else if (token.equals("TAG")) {              // ラベルの取得
            System.out.println("TAG: "+buf);

     if (st.hasMoreTokens()) {
      token = st.nextToken();
      exhibit[objCnt-1].setTag(token);
     }
```



```
     else {
      System.out.println("WARNING: TAG is not specified.");
      exhibit[objCnt-1].setTag("");            // TAG なし
     }
          }
    else if (token.equals("IMAGE")) {        // 画像ファイル名の取得
     System.out.println("IMAGE: "+buf);

     if (st.hasMoreTokens()) {
      token = st.nextToken();
      exhibit[objCnt-1].setImage(token);
     }
     else {
      System.out.println("WARNING: IMAGE is not specified.");
          }
         }
          else if (token.equals("TITLE")) {              // タイトルの取得
           System.out.println("TITLE: "+buf);

     if (st.hasMoreTokens()) {
      token = st.nextToken();
      exhibit[objCnt-1].setTitle(token);
     }
     else {
      System.out.println("WARNING: TAG is not specified.");
      exhibit[objCnt-1].setTag("");            // TAG なし
     }
          }
    else if (token.equals("TEXT")) {           // 説明文の取得
     while (true) {
      buf = br.readLine();
      if (buf == null) {
       System.out.println("ERROR: TEXT_END is not found.");
       break;
      }
      else if (buf.equals("TEXT_END")) {
       break;
      }
      else {                                                // 説明文
       exhibit[objCnt-1].setText(buf);
      }
          }
         }
    else if (token.equals("KEYWORD")) {   // キーワードの取得
     while (true) {
      buf = br.readLine();
      if (buf == null) {
       System.out.println("ERROR: KEYWORD_END is not found.");
       break;
      }
      else if (buf.equals("KEYWORD_END")) {
       break;
      }
      else {
       int idx;
       StringTokenizer st2 = new StringTokenizer(buf);
       if (st2.hasMoreTokens()) {
        token = st2.nextToken();
```

```
          idx = keywordList.addItem(token);  // キーワード
                }
          else {
           System.out.println("WARNING: KEYWORD is not found.");
           continue;
                }
          if (st2.hasMoreTokens()) {
           token = st2.nextToken();
           double d = Double.valueOf(token).doubleValue(); // 重み
           exhibit[objCnt-1].setKeyword(idx, d);
          }
          else {
           System.out.println("WARNING: WEIGHT is not found.(set to 1.0)");
           exhibit[objCnt-1].setKeyword(idx, 1.0);
                }
         }
              }
             }
      else {
// それ以外の行は無視
//              System.out.println ("IGNORE: "+buf);
             }
    } catch (Exception e) {
//      System.out.println(e);
    }
   }

   // file stream の close
   try {
    fs.close ();
   } catch (IOException e) {
    System.out.println (e);
   }

   return(true);
  }

  /*-------------------------------------------------------------------*/
  /** 展示空間の主成分値をファイルに保存をファイルに保存する
   *
   * @param filename  ファイル名
   */
  /*-------------------------------------------------------------------*/
  public boolean saveObjectData(String filename) {

   FileOutputStream    fos = null;
   BufferedWriter      bw = null;

   try {
    fos = new FileOutputStream(filename);
                                        // SJIS JIS EUCJIS JISAutoDetect
    bw = new BufferedWriter(new OutputStreamWriter(fos, "EUCJIS"));
   } catch (Exception e) {
    System.out.println(e);
    System.exit(1);
   }

   try {
```



```
    String buf = new String("CALCULATED");
    bw.write(buf, 0, buf.length());
    bw.newLine();
    bw.newLine();

    for (int i = 0; i < objCnt; i++) {
     buf = new String("OBJECT "+exhibit[i].fname+" ");
     buf += String.valueOf(exhibit[i].comp1[EXHIBITION])+" ";
     buf += String.valueOf(exhibit[i].comp2[EXHIBITION]);
     bw.write(buf, 0, buf.length());
     bw.newLine();
    }

    bw.newLine();
    buf = new String("# End of File");
    bw.write(buf, 0, buf.length());
    bw.newLine();

    bw.close();

   } catch (Exception e) {
    System.out.println(e);
    System.exit(1);
   }

   return(true);
  }

  /*-------------------------------------------------------------------*/
  /** 展示空間を作成
   *
   * @param filename  ファイル名
   */
  /*-------------------------------------------------------------------*/
  public boolean makeExhibitionSpace(String filename) {

   // 初期化
   exhibit = new Exhibit[maxExhibits];
   objCnt = 0;
   keywordList = new KeywordList(maxKeywords);

   // ファイルからオブジェクトデータを読み込む
   readObjectData(filename);

   // テスト用表示
//   keywordList.print();                    // キーワードリスト
   for (int i = 0; i < objCnt; i++) {
    exhibit[i].print(keywordList);  // 展示オブジェクト
   }
   System.out.println();
   System.out.println("----------------------------------------");

   // 分散共分散行列
   System.out.println("CovarianceMatrixX --- start");
   covarianceMat
     = new CovarianceMatrixX(keywordList.kwCnt, exhibit, objCnt);
//   covarianceMat.printMat();
//   covarianceMat.printAve();
```

```
    System.out.println("CovarianceMatrixX --- end");

    // 主成分が計算されていない場合はこの場で計算
    if (!calculated) {

      // 固有値、固有ベクトルの計算
      System.out.println("syev --- start");
      covarianceMat.syev();
      System.out.println("syev --- end");
//    covarianceMat.printEvec();
//    covarianceMat.printEv();

      // 固有値、固有ベクトルの取得
      double ev1, ev2;
      double evec1[] = new double[keywordList.kwCnt];
      double evec2[] = new double[keywordList.kwCnt];

      ev1 = covarianceMat.getEVal(1);
      ev2 = covarianceMat.getEVal(2);
      covarianceMat.getEVec(1, evec1);
      covarianceMat.getEVec(2, evec2);
      System.out.println("Eigenvalues ==> "+ev1+"  "+ev2);

      // 主成分の計算
      for (int i = 0; i < objCnt; i++) {
              exhibit[i].calcComponent(ev1, ev2, evec1, evec2);
      }
    }

    interestButton.setEnabled(true);

    // 描画の順番
    exhibitionTopPanel.funcPanel.setDrawOrder();

    System.out.println("Oc: "+objCnt+"  Kc: "+keywordList.kwCnt);
    return(true);
  }

  /*-------------------------------------------------------------------*/
  /** 選択された全てのオブジェクトのキーワードとその重みを調べ、
   * それが全て同じであるか否かを判定する
   *
   * @return  SAME_ALL                    全て同じ
   * @return  DIFFERENT                   一つでも違う
   */
  /*-------------------------------------------------------------------*/
  public int checkProperty() {
    int  obj1, obj2;

    // obj1 を決定
    for (int i = 0; i < objCnt-1; i++) {
      if (exhibit[i].selected) obj1 = i;
      else continue;

      // obj2 を決定
      for (int j = i+1; j < objCnt; j++) {
        if (exhibit[j].selected) obj2 = j;
        else continue;
```



```
        if (checkKeyword(obj1, obj2) == DIFFERENT) {
          return(DIFFERENT);
        }
      }
    }

    return(SAME_ALL);
  }

  /*-------------------------------------------------------------------*/
  /** 2つのオブジェクトのキーワードとその重みを調べ、
   * それが全て同じであるか否かを判定する
   *
   * @param   obj1                  オブジェクト1
   * @param   obj2                  オブジェクト2
   * @return  SAME_ALL                      全て同じ
   * @return  DIFFERENT                     一つでも違う
   */
  /*-------------------------------------------------------------------*/
  public int checkKeyword(int obj1, int obj2) {
    Exhibit ex1 = exhibit[obj1];
    Exhibit ex2 = exhibit[obj2];

    for (int i = 0; i < keywordList.kwCnt; i++) {
      if (ex1.keyword[i].listId != ex2.keyword[i].listId) return(DIFFERENT);
      if (ex1.keyword[i].weight != ex2.keyword[i].weight) return(DIFFERENT);
    }

    return(SAME_ALL);
  }


  /*-------------------------------------------------------------------*/
  /** メインルーチン
   */
  /*-------------------------------------------------------------------*/
  public static void main(String args[]) {

    if (args.length > 3) {
      System.out.println("usage: java MediationFunc <file> <hostname> <port>");
      System.exit(1);
    }

    // メインオブジェクトを作成
    MediationFunc mf = new MediationFunc();
//  mf.init();

    if (args.length >= 1) {
      if (!(mf.makeExhibitionSpace(args[0]))) {
        System.exit(1);
      }
    }
    if (args.length >= 2) {
      mf.hostName = new String(args[1]);
    }
    if (args.length == 3) {
      mf.outPort = Integer.valueOf(args[2]).intValue();
```

```
    }

    // メインウインドウの表示
    mf.exhibitionWin.resize(500, 620);
    mf.exhibitionWin.setLocation(10, 10);
    mf.exhibitionWin.show();

  } // End of main()

}

// End of File
```



```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE   : 1998.10.05   T.Takumi(CSK)
 **************************************************************************/

import java.applet.Applet;
import java.awt.*;
import java.io.*;
import java.util.*;

/**************************************************************************
 ** ・展示空間表示パネル
 **
 **************************************************************************/
public class MediationFuncPanel extends Panel {
                        /** アプリケーションメイン */
  MediationFunc         medFunc;

                        /** 表示用スケール値<BR> 自動調整 */
  double    scale;
                        /** 表示用スケール値<BR> スライダによりユーザが指定した値 */
  double    userScale;

                        /** 表示中心の X オフセット */
  int                   cenXOff;
                        /** 表示中心の Y オフセット */
  int                   cenYOff;

                        /** パネルのサイズ */
  Dimension             dimPanel;
                        /** 空間種類 */
  int                   index;
                        /** Drag のための起点 */
  int                   xorg;
                        /** Drag のための起点 */
  int                   yorg;
                        /** Drag の対象 */
  int                   trgObj;

                        /** フォント情報 */
  FontMetrics           fm;

                        /** 描画の順番 */
  int                   drawOrder[];

                        /** オフスクリーンイメージ */
  Image                 offImg;
                        /** オフスクリーングラフィックス */
  Graphics  offg;

  /*-----------------------------------------------------------------*/
  /** コンストラクタ
   *
   * @param appMain  アプリケーションメイン
   * @param idx                  空間種類
   */
```

```
/*-----------------------------------------------------------------------*/
public MediationFuncPanel(MediationFunc appMain, int idx) {
  medFunc = appMain;
  scale = 1.0;
  userScale = 1.0;
  cenXOff = 0;
  cenYOff = 0;
  index = idx;
  trgObj = -1;

  // デフォルトフォントの設定
//   System.out.println("panel ==> "+getFont());
  Font newFont = new Font("Dialog", Font.PLAIN, 12);
  setFont(newFont);
  System.out.println(getFont());

  fm = getFontMetrics(newFont);
 }

 /*-----------------------------------------------------------------------*/
 /** 描画の順番を決める
  */
 /*-----------------------------------------------------------------------*/
 public void setDrawOrder() {
  switch (index) {
  case medFunc.EXHIBITION:
   drawOrder = new int[medFunc.objCnt];
   for (int i = 0 ; i < medFunc.objCnt; i++) {
    drawOrder[i] = i;
   }
   break;

  case medFunc.INTEREST:
  case medFunc.PERSONAL:
   drawOrder = new int[medFunc.cntOvOm];
   for (int i = 0; i < medFunc.cntOvOm; i++) {
    drawOrder[i] = medFunc.personalSubset[i];
   }
   break;
  }
 }

 /*-----------------------------------------------------------------------*/
 /** マウスクリック
  *
  * @param evt        マウスイベント
  * @param x          クリックのx座標
  * @param y          クリックのy座標
  */
 /*-----------------------------------------------------------------------*/
 public boolean mouseDown(Event evt, int x, int y) {
//   System.out.println("Mouse Down2(x, y) --> "+x+" "+y);

  trgObj = -1;
  // オブジェクトを移動する
  if ((evt.modifiers & Event.SHIFT_MASK) > 0) {
   int i;
   if (index == medFunc.EXHIBITION) {
```



```
    i = medFunc.objCnt-1;
   }
   else {
    i = medFunc.cntOvOm-1;
   }
   for (; i >= 0; i--) {
    int ii = drawOrder[i];
          if (checkInOut(x, y, medFunc.exhibit[ii])) {
     xorg = x;
     yorg = y;
     trgObj = ii;
           break;
          }
   }
  }
  // オブジェクトの説明文を出す
  else if ((evt.modifiers & Event.META_MASK) > 0) {
   int i;
   if (index == medFunc.EXHIBITION) {
    i = medFunc.objCnt-1;
   }
   else {
    i = medFunc.cntOvOm-1;
   }
   for (; i >= 0; i--) {
    int ii = drawOrder[i];
          if (checkInOut(x, y, medFunc.exhibit[ii])) {
     trgObj = ii;
     break;
          }
   }
   if (trgObj >= 0) {
    System.out.println("TAK>> "+medFunc.exhibit[trgObj].tag);
          Frame parent = (Frame)(getParent().getParent());
          TextAreaDialog t = new TextAreaDialog(parent, medFunc.exhibit[trgObj]);
    t.setTextLines();
          t.resize(300, 200);
          Point point = parent.getLocation();
          t.setLocation(point.x+medFunc.exhibit[trgObj].frRight[index]+20,
            point.y+medFunc.exhibit[trgObj].frBottom[index]);
          t.show();
          trgObj = -1;
   }
  }
  // オブジェクトの情報(キーワード、重み)を出す
  else if ((evt.modifiers & Event.CTRL_MASK) > 0) {
   int i;
   if (index == medFunc.EXHIBITION) {
    i = medFunc.objCnt-1;
   }
   else {
    i = medFunc.cntOvOm-1;
   }
   for (; i >= 0; i--) {
    int ii = drawOrder[i];
          if (checkInOut(x, y, medFunc.exhibit[ii])) {
     trgObj = ii;
     break;
```

```
            }
        }
        if (trgObj >= 0) {
          System.out.println("TAK>> "+medFunc.exhibit[trgObj].tag);
            Frame parent = (Frame)(getParent().getParent());

          medFunc.exhibit[trgObj].print(medFunc.keywordList);

            InfoDialog id = new InfoDialog(parent, medFunc.exhibit[trgObj], medFunc.keywordList, index);
          id.setKeywords();
          if (index == MediationFunc.EXHIBITION) {
              id.resize(180, 340);
            }
            else {
              id.resize(180, 210);
            }
            Point point = parent.getLocation();
            id.setLocation(point.x+medFunc.exhibit[trgObj].frRight[index],
              point.y+medFunc.exhibit[trgObj].frBottom[index]+10);
            id.show();
          id.selectItem(0);                    // 先頭を選択
            trgObj = -1;
        }
      }
      else {
        int i, trg;
        if (index == medFunc.EXHIBITION) {
          i = medFunc.objCnt-1;
          trg = medFunc.objCnt-1;
        }
        else {
          i = medFunc.cntOvOm-1;
          trg = medFunc.cntOvOm-1;
        }
        for (; i >= 0; i--) {
          int ii = drawOrder[i];
            if (checkInOut(x, y, medFunc.exhibit[ii])) {

          // 対象オブジェクトの描画順を最後(最上)に移動
          if (i != trg) {
            int tmpEx = ii;
            for (int j = i+1; j <= trg; j++) {
              drawOrder[j-1] = drawOrder[j];
            }
            drawOrder[trg] = tmpEx;
                }
          break;
              }
        }
        repaint();
      }

      return(true);
    }

    /*-----------------------------------------------------------------*/
    /** マウスボタンを離した時の処理
     *
```



```
     * @param evt       マウスイベント
     * @param x         クリックのx座標
     * @param y         クリックのy座標
     */
    /*-----------------------------------------------------------------*/
    public boolean mouseUp(Event evt, int x, int y) {
//    System.out.println("Mouse Up (x, y) --> "+x+" "+y);
      if (trgObj >= 0) {
/*
        // 中心からスケール
        medFunc.exhibit[trgObj].xoffset[index] += x - xorg;
        medFunc.exhibit[trgObj].yoffset[index] += y - yorg;
*/
        // 左上を起点としてスケール
        medFunc.exhibit[trgObj].xoffset[index] += (int)((x - xorg)/userScale);
        medFunc.exhibit[trgObj].yoffset[index] += (int)((y - yorg)/userScale);
        trgObj = -1;
        clear();
        repaint();
      }
      return(true);
    }

    /*-----------------------------------------------------------------*/
    /** マウスのフォーカス
     *
     * @param evt       マウスイベント
     * @param x         クリックのx座標
     * @param y         クリックのy座標
     */
    /*-----------------------------------------------------------------*/
    public boolean mouseMove(Event evt, int x, int y) {
//    System.out.println("Mouse Move (x, y) --> "+x+" "+y+"  INDEX: "+index);
      boolean  focused = false;         // 一つ選択されたか
      boolean  repaint = false;

      int i;
      if (index == medFunc.EXHIBITION) {
        i = medFunc.objCnt-1;
      }
      else {
        i = medFunc.cntOvOm-1;
      }
      for (; i >= 0; i--) {
        int ii = drawOrder[i];
        if (checkInOut(x, y, medFunc.exhibit[ii])) {
            if (focused == false && medFunc.exhibit[ii].focus[index] == false) {
              medFunc.exhibit[ii].focus[index] = true;
              repaint = true;
            }
            else if (focused && medFunc.exhibit[ii].focus[index]) {
              medFunc.exhibit[ii].focus[index] = false;
              repaint = true;
            }
            focused = true;
        }
        else {
            if (medFunc.exhibit[ii].focus[index]) {
```

```
            medFunc.exhibit[ii].focus[index] = false;
            repaint = true;
           }
    }
   }

   if (repaint) repaint();
   return(true);
  ]

  /*-----------------------------------------------------------------------*/
  /** 画面のクリア<BR>
   * オフスクリーンバッファをクリア
   */
  /*-----------------------------------------------------------------------*/
  public void clear() {
    offg.clearRect(0, 0, size().width, size().height);
//    repaint();
  }

  /*-----------------------------------------------------------------------*/
  /** 画面のリサイズ
   *
   * @param x                    ウインドウ位置x
   * @param y                    ウインドウ位置y
   * @param width      ウインドウ幅
   * @param height     ウインドウ高さ
   */
  /*-----------------------------------------------------------------------*/
  public void reshape(int x, int y, int width, int height) {
    super.reshape(x, y, width, height);

    // パネルサイズの取得
    dimPanel = size();

    // オフスクリーンバッファ
    offImg = createImage(dimPanel.width, dimPanel.height);

    // オフスクリーン Graphics の取得
    if (offImg != null) {
      offg = offImg.getGraphics();
    }
  }

  /*-----------------------------------------------------------------------*/
  /** 描画<br>
   * ちらつきを防ぐ為、update() を paint() でオーバーライド
   *
   * @param g            グラフィックス
   */
  /*-----------------------------------------------------------------------*/
  public void update(Graphics g) {
    paint(g);
  }

  /*-----------------------------------------------------------------------*/
  /** 描画 paint
   *
```



```
   * @param g            グラフィックス
   */
  /*-----------------------------------------------------------------------*/
  public void paint(Graphics g) {
    // パネルのサイズ
    int width, height;
    width = dimPanel.width;
    height = dimPanel.height;

    // オブジェクトには大きさがあるので実際にオブジェクトを
    // 配置できるパネルサイズ
    double vw, vh;
    vw = width * 0.75;
    vh = height * 0.8;

    // 座標軸を表示
/*
    offg.drawLine(0, height/2, width, height/2);
    offg.drawLine(width/2, 0, width/2, height);
*/
    // 中心を座標原点に
//    offg.translate(width/2, height/2);

    // 表示中心値
    double cenX = 0.0;
    double cenY = 0.0;

    double maxX = 0.0;
    double maxY = 0.0;

    double minX = 0.0;
    double minY = 0.0;

    double diffX = 0.0;
    double diffY = 0.0;

    double tmpX = 0.0;
    double tmpY = 0.0;

    // オブジェクトの数
    int end;
    if (index == medFunc.EXHIBITION) {
      end = medFunc.objCnt;
    }
    else {
      end = medFunc.cntOvOm;
    }

    // MAX MIN 値を設定
    for (int i = 0 ; i < end; i++) {
      // 展示空間以外はサブセットを参照のため注意
      int ii = drawOrder[i];

      tmpX = medFunc.exhibit[ii].comp1[index];
      tmpY = medFunc.exhibit[ii].comp2[index];
      if (i == 0) {
        maxX = tmpX;
        maxY = tmpY;
```

```
    minX = tmpX;
    minY = tmpY;
   }
   else {
    if (tmpX > maxX) maxX = tmpX;
    else if (tmpX < minX) minX = tmpX;
    if (tmpY > maxY) maxY = tmpY;
    else if (tmpY < minY) minY = tmpY;
   }
  }

  // 分布距離、中心値を設定
  diffX = maxX - minX;
  diffY = maxY - minY;
  cenX = (maxX+minX)/2.0;
  cenY = (maxY+minY)/2.0;

  // 表示の拡大率を計算
  if (diffY == 0.0) {
   if (diffX == 0.0) {
    scale = 1.0;
   }
   else {
    scale = vw/diffX;
   }
  }
  else {
   if (diffX/diffY > vw/(double)vh) {
          scale = vw/diffX;
   }
   else {
          scale = vh/diffY;
   }
  }

  // Node の表示
  for (int i = 0 ; i < end; i++) {
   int ii = drawOrder[i];
   paintNode(offg, medFunc.exhibit[ii], fm, cenX, cenY);
  }

  // オフスクリーンバッファの描画
  g.drawImage(offImg, 0, 0, this);
 }

 /*-----------------------------------------------------------------------*/
 /** 各ノードの描画
  *
  * @param g          グラフィックス
  * @param ex         描画対象展示オブジェクト
  * @param fm         フォント情報
  * @param cenX       表示の中心
  * @param cenY       表示の中心
  */
 /*-----------------------------------------------------------------------*/
 public void paintNode(Graphics g, Exhibit ex, FontMetrics fm, double cenX, double cenY) {

  // 中心からスケール
```



```
   int x = (int)(((ex.comp1[index]-cenX)*scale
           + ex.xoffset[index]) * userScale)
         + dimPanel.width/2 - cenXOff;
   int y = (int)(((ex.comp2[index]-cenY)*scale
           + ex.yoffset[index]) * userScale)
         + dimPanel.height/2 - cenYOff;
 /*
   // 左上を起点としてスケール
   int x = (int)(((ex.comp1[index]-cenX)*scale
           + ex.xoffset[index] + dimPanel.width/2) * userScale);
   int y = (int)(((ex.comp2[index]-cenY)*scale
           + ex.yoffset[index] + dimPanel.height/2) * userScale);
 */

 //   System.out.println("(x, y) --> "+x+" "+y);

   // 幅を決定
   int w, h;
   if (ex.hasImage) {
    w = ex.image.getWidth(this) + 8;
    h = ex.image.getHeight(this) + 8;
   }
   else {
    w = fm.stringWidth(ex.tag) + 10;
    h = fm.getHeight() + 4;
   }

   // フィールドの大きさ
   int left  = x - w/2;
   int right  = x + w/2;
   int top   = y - h/2;
   int bottom = y + h/2;
   ex.frLeft[index]  = left;
   ex.frRight[index]  = right;
   ex.frTop[index]   = top;
   ex.frBottom[index] = bottom;

   // フィールド
   if (ex.selected) {
    g.setColor(medFunc.selectColor);
   }
   else {
    g.setColor(medFunc.nodeColor);
   }
   g.fillRect(left, top, w, h);

   // フレーム
   if (ex.focus[index]) {
    g.setColor(medFunc.focusColor);
   }
   else {
    g.setColor(medFunc.frameColor);
   }
   g.drawRect(left, top, w, h);

   if (ex.hasImage) {
    // 画像
    g.drawImage(ex.image, left+4, top+4, this);
```

```
  }
  else {
   // 文字
   if (ex.focus[index]) {
            g.setColor(medFunc.focusColor);
   }
   else {
            if (ex.selected) {
              g.setColor(medFunc.selectStringColor);
            }
            else {
              g.setColor(medFunc.stringColor);
            }
   }
   g.drawString(ex.tag, left+5, top+2 + fm.getAscent());
  }
}

/*-----------------------------------------------------------------------*/
/** マウスポインタの内外判定
 *
 * @param x          x位置
 * @param y          y位置
 * @param ex         展示オブジェクト
 */
/*-----------------------------------------------------------------------*/
public boolean checkInOut(int x, int y, Exhibit ex) {
  if (x > ex.frLeft[index] && x < ex.frRight[index]
     && y > ex.frTop[index] && y < ex.frBottom[index]) {
    return(true);                    // Inside
  }
  else {
    return(false);                   // Outside
  }
}

/*-----------------------------------------------------------------------*/
/** フォントサイズの変更
 *
 * @param size       フォントサイズ
 */
/*-----------------------------------------------------------------------*/
public void setFontAndFM(int size) {
  Font curFont = getFont();
  Font newFont = new Font(curFont.getName(), curFont.getStyle(), size);
  setFont(newFont);
  System.out.println("current Font ==> "+getFont());

  // 何故かこれを呼ぶ必要あり(呼ばないとフォントが変らない)
  offg = offImg.getGraphics();

  fm = getFontMetrics(newFont);

  clear();
  repaint();
 }
}
```



```
// End of File
```

```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.05   T.Takumi(CSK)
 **************************************************************************/

import java.applet.Applet;
import java.awt.*;
import java.io.*;
import java.util.*;

/**************************************************************************
 ** 展示空間表示パネル
 ** (Pickable)
 **************************************************************************/
public class MediationFuncPickablePanel extends MediationFuncPanel {

 /*-----------------------------------------------------------------*/
 /** コンストラクタ
  *
  * @param appMain  アプリケーションメインオブジェクト
  */
 /*-----------------------------------------------------------------*/
 public MediationFuncPickablePanel(MediationFunc appMain) {
  super(appMain, appMain.EXHIBITION);
 }

 /*-----------------------------------------------------------------*/
 /** マウスクリック
  *
  * @param evt        マウスイベント
  * @param x          x位置
  * @param y          y位置
  */
 /*-----------------------------------------------------------------*/
 public boolean mouseDown(Event evt, int x, int y) {
//   System.out.println("Mouse Down(x, y) --> "+x+" "+y);

   if ((evt.modifiers & Event.SHIFT_MASK) > 0) {
    System.out.println("SHIFT_MASK !!!");
    return(super.mouseDown(evt, x, y));
   }
   else if ((evt.modifiers & Event.CTRL_MASK) > 0) {
    System.out.println("CTRL_MASK !!!");
    return(super.mouseDown(evt, x, y));
   }
   else if ((evt.modifiers & Event.ALT_MASK) > 0) {
    System.out.println("ALT_MASK !!!");
    return(super.mouseDown(evt, x, y));
   }
   else if ((evt.modifiers & Event.META_MASK) > 0) {
    System.out.println("META_MASK !!!");
    return(super.mouseDown(evt, x, y));
   }
   else {
    for (int i = medFunc.objCnt-1; i >= 0; i--) {
     int ii = drawOrder[i];
```



```
      if (checkInOut(x, y, medFunc.exhibit[ii])) {

       // 興味空間が表示されていないときはオブジェクトを選択する
       if (medFunc.selectMode & !medFunc.interestWin.isShowing()) {
        medFunc.exhibit[ii].selected = !medFunc.exhibit[ii].selected;
       }
      // オブジェクトの中を pick の場合はさらに上位クラスでの処理
      return(super.mouseDown(evt, x, y));
      }
    }
    // オブジェクトの外を pick の場合はこれで終了
    return(true);
   }
  }
}


// End of File
```

```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.05   T.Takumi(CSK)
 **************************************************************************/

import java.awt.*;
import java.io.*;
import java.util.*;

/**************************************************************************
 ** 展示空間トップパネル
 **
 **************************************************************************/
public class MediationFuncTopPanel extends AppPanel {

 /*-----------------------------------------------------------------*/
 /** コンストラクタ
  */
 /*-----------------------------------------------------------------*/
 public MediationFuncTopPanel(MediationFunc appMain) {
  super(appMain, MediationFunc.EXHIBITION);

  // BPanel3
  medFunc.selectMode = true;
  medFunc.selectCheckbox = new Checkbox("選択", medFunc.selectMode);
  BPanel3.add("North", medFunc.selectCheckbox);

  // BPanel4
  medFunc.interestButton = new Button("興味空間");
  BPanel4.add("North", medFunc.interestButton);
  medFunc.interestButton.setEnabled(false);

  medFunc.personalButton = new Button("個人化空間");
  BPanel4.add("South", medFunc.personalButton);
  medFunc.personalButton.setEnabled(false);
 }


 /*-----------------------------------------------------------------*/
 /** ボタンイベントの処理
  *
  * @param evt                  発生イベント
  * @param arg                  ボタンのラベル
  */
 /*-----------------------------------------------------------------*/
 public boolean action(Event evt, Object arg) {

//   System.out.println(">>>> action()!!  "+arg);

   if (evt.target instanceof Checkbox) {
//     System.out.println("taktak>> checkbox!!");
     medFunc.selectMode = ((Checkbox)evt.target).getState();
     return(true);
   }
   else if ("興味空間".equals(arg)) {
    System.out.println("興味空間");
```



```
    // 興味空間を作成
    if (medFunc.makeInterestSpace(evt)) {
     medFunc.selectMode = false;
//     medFunc.selectCheckbox.setState(medFunc.selectMode);
     medFunc.selectCheckbox.setEnabled(false);
     medFunc.interestButton.setEnabled(false);

     medFunc.interestWin.setSize(400, 400);
     medFunc.interestWin.add("Center", medFunc.interestTopPanel);
     medFunc.interestWin.setLocation(520, 10);
     medFunc.interestWin.show();
     medFunc.personalButton.setEnabled(true);
     medFunc.exhibitionWin.openMenuItem.setEnabled(false);
    }
    return(true);
   }
   else if ("個人化空間".equals(arg)) {
    medFunc.personalButton.setEnabled(false);
    System.out.println("個人化空間");
    // 個人化空間を作成
    if (medFunc.makePersonalSpace()) {
     medFunc.personalWin.setSize(400, 400);
     medFunc.personalWin.add("Center", medFunc.personalTopPanel);
     medFunc.personalWin.setLocation(520, 420);
     medFunc.personalWin.show();

     // 意味距離を計算しパターン化
     System.out.println("distance -- start");
     medFunc.divideIntoPatterns();
     System.out.println("distance -- end");

     // パターン選択 UI を表示
     PatternDialog p = new PatternDialog(medFunc, medFunc.personalWin, medFunc.pattern);
     p.resize(140, 260);
     p.setLocation(920, 480);
     p.show();

    }
    else {
     medFunc.personalButton.setEnabled(true);
    }
    return(true);
   }
   else {
    // AppPanel の action 処理
    return(super.action(evt, arg));
   }
  }

 }


 // End of File
```

```
// This example is from the book _Java in a Nutshell_ by David Flanagan.
// Written by David Flanagan. Copyright (c) 1996 O'Reilly & Associates.
// You may study, use, modify, and distribute this example for any purpose.
// This example is provided WITHOUT WARRANTY either expressed or implied.

import java.awt.*;
import java.util.*;

public class MultiLineLabel extends Canvas {
 /** Alignment constants */
 public static final int LEFT = 0;
 /** Alignment constants */
 public static final int CENTER = 1;
 /** Alignment constants */
 public static final int RIGHT = 2;
 /** The lines of text to display */
 protected String[] lines;
 /** The number of lines */
 protected int num_lines;
 /** Left and right margins */
 protected int margin_width;
 /** Top and bottom margins */
 protected int margin_height;
 /** Total height of the font */
 protected int line_height;
 /** Font height above baseline */
 protected int line_ascent;
 /** How wide each line is */
 protected int[] line_widths;
 /** The width of the widest line */
 protected int max_widtb;
 /** The alignment of the text. */
 protected int alignment = LEFT;

 /*------------------------------------------------------------------*/
 /**
  * This method breaks a specified label up into an array of lines.<br>
  * It uses the StringTokenizer utility class.
  */
 /*------------------------------------------------------------------*/
 protected void newLabel(String label) {
  StringTokenizer t = new StringTokenizer(label, "\n");
  num_lines = t.countTokens();
  lines = new String[num_lines];
  line_widths = new int[num_lines];
  for(int i = 0; i < num_lines; i++) lines[i] = t.nextToken();
 }

 /*------------------------------------------------------------------*/
 /**
  * This method figures out how the font is, and how wide each
  * line of the label is, and bow wide the widest line is.
  */
 /*------------------------------------------------------------------*/
 protected void measure() {
  FontMetrics fm = this.getFontMetrics(this.getFont());
  // If we don't have font metrics yet, just return.
  if (fm == null) return;
```



```
  line_height = fm.getHeight();
  line_ascent = fm.getAscent();
  max_width = 0;
  for(int i = 0; i < num_lines; i++) {
   line_widths[i] = fm.stringWidth(lines[i]);
   if (line_widths[i] > max_width) max_width = line_widths[i];
  }
 }

 /*------------------------------------------------------------------*/
 /**
  * Here are four versions of the cosntrutor.<br>
  * Break the label up into separate lines, and save the other info.
  */
 /*------------------------------------------------------------------*/
 public MultiLineLabel(String label, int margin_width, int margin_height,
                       int alignment) {
  newLabel(label);
  this.margin_width = margin_width;
  this.margin_height = margin_height;
  this.alignment = alignment;
 }
 public MultiLineLabel(String label, int margin_width, int margin_height) {
  this(label, margin_width, margin_height, LEFT);
 }
 public MultiLineLabel(String label, int alignment) {
  this(label, 10, 10, alignment);
 }
 public MultiLineLabel(String label) {
  this(label, 10, 10, LEFT);
 }

 /*------------------------------------------------------------------*/
 /** Methods to set the various attributes of the component */
 /*------------------------------------------------------------------*/
 public void setLabel(String label) {
  newLabel(label);
  measure();
  repaint();
 }
 public void setFont(Font f) {
  super.setFont(f);
  measure();
  repaint();
 }
 public void setForeground(Color c) {
  super.setForeground(c);
  repaint();
 }
 public void setAlignment(int a) { alignment = a; repaint(); }
 public void setMarginWidth(int mw) { margin_width = mw; repaint(); }
 public void setMarginHeight(int mh) { margin_height = mh; repaint(); }
 public int getAlignment() { return alignment; }
 public int getMarginWidth() { return margin_width; }
 public int getMarginHeight() { return margin_height; }

 /*------------------------------------------------------------------*/
```

```
/**
 * This method is invoked after our Canvas is first created
 * but before it can actually be displayed.  After we've
 * invoked our superclass's addNotify() method, we have font
 * metrics and can successfully call measure() to figure out
 * how big the label is.
 */
/*-------------------------------------------------------------------*/
public void addNotify() { super.addNotify(); measure(); }

/*-------------------------------------------------------------------*/
/**
 * This method is called by a layout manager when it wants to
 * know how big we'd like to be.
 */
/*-------------------------------------------------------------------*/
public Dimension preferredSize() {
 return new Dimension(max_width + 2*margin_width,
                      num_lines * line_height + 2*margin_height);
}

/*-------------------------------------------------------------------*/
/**
 * This method is called when the layout manager wants to know
 * the bare minimum amount of space we need to get by.
 */
/*-------------------------------------------------------------------*/
public Dimension minimumSize() {
 return new Dimension(max_width, num_lines * line_height);
}

/*-------------------------------------------------------------------*/
/**
 * This method draws the label (applets use the same method).
 * Note that it handles the margins and the alignment, but that
 * it doesn't have to worry about the color or font--the superclass
 * takes care of setting those in the Graphics object we're passed.
 */
/*-------------------------------------------------------------------*/
public void paint(Graphics g) {
 int x, y;
 Dimension d = this.size();
 y = line_ascent + (d.height - num_lines * line_height)/2;
 for(int i = 0; i < num_lines; i++, y += line_height) {
  switch(alignment) {
  case LEFT:
       x = margin_width; break;
  case CENTER:
  default:
       x = (d.width - line_widths[i])/2; break;
  case RIGHT:
       x = d.width - margin_width - line_widths[i]; break;
  }
  g.drawString(lines[i], x, y);
 }
}
}
```



```
/**************************************************************************
** PROGRAM : 仲介機能プログラム
** FILENAME :
**       :
** DATE    : 1998.10.30   T.Takumi(CSK)
**************************************************************************/

/**************************************************************************
** ・パターン分類して保存
**
**************************************************************************/
public class Pattern {
                    /** 各パターンに含まれるオブジェクトのリスト */
 boolean   list[][];
                    /** 各パターンに含まれるオブジェクトの数 */
 int                cnt[];
                    /** 各パターンのラベル */
 String     label[];

 /*-------------------------------------------------------------------*/
 /** コンストラクタ
  *
  * @param kind       パターンの数
  * @param max                  各パターンに含まれるオブジェクトの数の最大値
  */
 /*-------------------------------------------------------------------*/
 public Pattern(int kind, int max) {
  list = new boolean[kind][max];
  cnt = new int[kind];

  // 初期化
  for (int i = 0; i < kind; i++) {
   cnt[i] = 0;
   for (int j = 0; j < max; j++) {
    list[i][j] = false;
   }
  }

  label = new String[kind];
 }

 /*-------------------------------------------------------------------*/
 /** パターンラベルの設定
  *
  * @param p                    パターン番号
  * @param string       ラベル
  */
 /*-------------------------------------------------------------------*/
 public void setLabel(int p, String string) {
  p--;       // パターン n は [n-1] に格納
  label[p] = new String(string);
 }

 /*-------------------------------------------------------------------*/
 /** パターンデータを追加
  *
  * @param p                    パターン番号
  * @param i                    オブジェクトID1
```

```
 * @param j                    オブジェクトID2
 */
/*-------------------------------------------------------------------*/
public void checkPatternList(int p, int i, int j) {
  p--;      // パターン n は [n-1] に格納
  if (!list[p][i]) {
    list[p][i] = true;
    cnt[p]++;
  }
  if (!list[p][j]) {
    list[p][j] = true;
    cnt[p]++;
  }
}

/*-------------------------------------------------------------------*/
/** テスト用表示
 */
/*-------------------------------------------------------------------*/
public void print() {
  for (int i = 0; i < list.length; i++) {
    System.out.println("Pattern "+(i+1)+"---------- "+cnt[i]+" items");
    for (int j = 0; j < list[0].length; j++) {
      if (list[i][j]) {
        System.out.println("  "+j);
          }
    }
  }
}

}


// End of File
```



```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.30   T.Takumi(CSK)
 **************************************************************************/

import java.awt.*;
import java.io.*;
import java.net.*;

/**************************************************************************
 ** ・パターン表示 UI
 **
 **************************************************************************/
public class PatternDialog extends DDialog {

                    /** アプリケーションメインオブジェクト */
 MediationFunc          medFunc;
                    /** パターン分類 */
 Pattern    pattern;
 List               list;
 TextField textField;

 /*-------------------------------------------------------------------*/
 /** コンストラクタ
  *
  * @param parent    親フレーム
  * @param pat                  パターン分類
  */
 /*-------------------------------------------------------------------*/
 public PatternDialog(MediationFunc appMain, Frame parent, Pattern pat) {
  super(parent, "パターン選択", false);

  medFunc = appMain;
  pattern = pat;

  list = new List();
  list.setMultipleMode(false);
  add("Center", list);

  for (int i = 0; i < pat.list.length; i++) {
   list.addItem(pat.label[i]);
  }
  list.select(0);

  Label l = new Label("対象数");
  textField = new TextField(5);
  textField.setText(String.valueOf(pat.cnt[0]));

  Panel p = new Panel();
  add("South", p);
//   p.add(l);
  p.add(textField);

 }

 /*-------------------------------------------------------------------*/
```

```
/** 特定イベント処理
 *
 * @param ev          発生イベント
 */
/*-----------------------------------------------------------------------*/
public boolean handleEvent(Event ev) {
  switch (ev.id) {
  case Event.ACTION_EVENT:
    if (ev.target == list) {                    // ダブルクリック
      clientSend();
      return(true);
    }
    break;

  case Event.LIST_SELECT:
    textField.setText(String.valueOf(pattern.cnt[list.getSelectedIndex()]));
    return(true);
  }
  return(super.handleEvent(ev));
}

/*-----------------------------------------------------------------------*/
/** ソケットにより送信
 */
/*-----------------------------------------------------------------------*/
public boolean clientSend() {

  //テスト用表示
  int i = list.getSelectedIndex();
  for (int j = 0; j < pattern.list[0].length; j++) {
    if (pattern.list[i][j]) {
      System.out.print("  "+medFunc.exhibit[j].id);
    }
  }
  System.out.println();

  // ソケットの設定
  Socket fs = null;

  String hostName;
  if (medFunc.hostName != null) {
    hostName = medFunc.hostName;
  }
  else {
    hostName = new String("miris47");
  }
  int outPort;
  if (medFunc.outPort > 0) {
    outPort = medFunc.outPort;
  }
  else {
    outPort = 23456;
  }

  // ソケットをオープン
  try {
    fs = new Socket(hostName, outPort);
  } catch (Exception e) {
```



```
    System.out.println("WARNING: Unable to open socket");
    return(false);
  }
  PrintStream ps = null;
  try {
    ps = new PrintStream(fs.getOutputStream());
  } catch (Exception e) {
    System.out.println(e);
    return(false);
  }

  // 選択されたパターンに属する ID を送る
  i = list.getSelectedIndex();
  for (int j = 0; j < pattern.list[0].length; j++) {
    if (pattern.list[i][j]) {
      // コマンドは " " で区切る
          ps.print(medFunc.exhibit[j].id +" ");
    }
  }
  ps.print(" ");          // 無い場合でもスペースを送る
  ps.flush();

  try {
    fs.close();
  } catch (IOException e) {
    System.out.println(e);
  }

  return(true);

 }


}


// End of File
```

```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.28   T.Takumi(CSK)
 **************************************************************************/

import java.io.*;

/**************************************************************************
 ** ・拡張子によるファイル名フィルタ
 **
 **************************************************************************/
public class SuffixFilenameFilter implements FilenameFilter {
                        /** ファイルの拡張子 */
  String      suffix;

  /*------------------------------------------------------------------*/
  /** コンストラクタ
   *
   * @param suffix      ファイルの拡張子
   */
  /*------------------------------------------------------------------*/
  public SuffixFilenameFilter(String suffix) {
    this.suffix = new String("."+suffix);
  }

  /*------------------------------------------------------------------*/
  /** accept のオーバーライト
   *
   * @param dir                      ディレクトリ名
   * @param name       ファイル名
   */
  /*------------------------------------------------------------------*/
  public boolean accept(File dir, String name) {

 System.out.println("File: "+dir);
 System.out.println("String: "+name);
 System.out.println("suffix: "+suffix);
    if (name.endsWith(suffix)) {
      return(true);
    }
    return(false);
  }
}


 // End of File
```



```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.23   T.Takumi(CSK)
 **************************************************************************/

import java.awt.*;

/**************************************************************************
 ** ・テキストエリアダイアログ
 **
 **************************************************************************/
public class TextAreaDialog extends DDialog {
  Exhibit    ex;
  TextArea textArea;

  /*------------------------------------------------------------------*/
  /** コンストラクタ
   *
   * @param parent      親フレーム
   * @param exhibit     展示オブジェクト
   */
  /*------------------------------------------------------------------*/
  public TextAreaDialog(Frame parent, Exhibit exhibit) {
    super(parent, false);

    if (exhibit.hasTitle) {
      setTitle(exhibit.title);
    }
    else {
      setTitle(exhibit.tag);
    }

    ex = exhibit;

    textArea = new TextArea("", 3, 5, TextArea.SCROLLBARS_VERTICAL_ONLY);
    textArea.setEditable(false);
    add(textArea);
//    System.out.println(textArea.getFont());

    // デフォルトフォントの設定
/*
    System.out.println("TextAreaDialog ==> "+getFont());
    Font newFont = new Font("Dialog", Font.PLAIN, 20);
    setFont(newFont);
    System.out.println(getFont());
*/
  }

  /*------------------------------------------------------------------*/
  /** 説明文をセット
   */
  /*------------------------------------------------------------------*/
  public void setTextLines() {
    for (int i = 0; i < ex.textLines; i++) {
//      System.out.println("TAKTAK>> "+ex.text[i]);
      textArea.append(ex.text[i]+"\n");
```

```
  }
 }

}


// End of File
```



```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.21   T.Takumi(CSK)
 **************************************************************************/

import java.awt.*;

/**************************************************************************
 ** ・空間トップパネル
 **
 **************************************************************************/
public class TopPanel extends AppPanel {

 /*-----------------------------------------------------------------*/
 /** コンストラクタ
  *
  * @param appMain  アプリケーションメインオブジェクト
  * @param idx                    空間種類
  */
 /*-----------------------------------------------------------------*/
 public TopPanel(MediationFunc appMain, int idx) {
  super(appMain, idx);

  // BPanel4
  BPanel4.add("North", new Button("閉じる"));
 }

 /*-----------------------------------------------------------------*/
 /** ボタンイベントの処理
  *
  * @param evt                    発生イベント
  * @param arg                    ボタンのラベル
  */
 /*-----------------------------------------------------------------*/
 public boolean action(Event evt, Object arg) {

//   System.out.println(">>>> action()!!  "+arg);

   if ("閉じる".equals(arg)) {
    switch (index) {
    case medFunc.INTEREST:

      medFunc.personalTopPanel.clearOffset();
      medFunc.personalWin.dispose();          // 個人化空間も消去
      medFunc.keywordList.deselectAll();
      medFunc.selectMode = true;
//      medFunc.selectCheckbox.setState(medFunc.selectMode);
      medFunc.selectCheckbox.setEnabled(true);
      medFunc.interestButton.setEnabled(true);
      medFunc.personalButton.setEnabled(false);
      medFunc.exhibitionWin.openMenuItem.setEnabled(true);
      medFunc.interestTopPanel.clearOffset();
      medFunc.interestWin.dispose();
      break;
    case medFunc.PERSONAL:
      medFunc.personalButton.setEnabled(true);
```

```
      medFunc.personalTopPanel.clearOffset();
      medFunc.personalWin.dispose();
      break;
    }
    return(true);
   }
   else {
    // AppPanel の action 処理
    return(super.action(evt, arg));
   }
  }

}


// End of File
```



```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **       :
 ** DATE    : 1998.10.16   T.Takumi(CSK)
 **************************************************************************/

import java.awt.*;

/**************************************************************************
 ** ・警告ダイアログ
 **
 **************************************************************************/
public class WarningDialog extends Dialog {
  protected Button button;
  protected MultiLineLabel label;

  public WarningDialog(Frame parent, String title, String message)
   {
    // Create a dialog with the specified title
    super(parent, title, true);

    // Create and use a BorderLayout manager with specified margins
    this.setLayout(new BorderLayout(15, 15));

    // Create the message component and add it to the window
    label = new MultiLineLabel(message, 20, 20);
    this.add("Center", label);

    // Create an Okay button in a Panel; add the Panel to the window
    // Use a FlowLayout to center the button and give it margins.
    button = new Button("OK");
    Panel p = new Panel();
    p.setLayout(new FlowLayout(FlowLayout.CENTER, 15, 15));
    p.add(button);
    this.add("South", p);

    // Resize the window to the preferred size of its components
    this.pack();
   }

  /*-----------------------------------------------------------------------*/
  /** Pop down the window when the button is clicked. */
  /*-----------------------------------------------------------------------*/
  public boolean action(Event e, Object arg)
   {
    if (e.target == button) {
            this.hide();
            this.dispose();
            return true;
     }
    else return false;
   }

  /*-----------------------------------------------------------------------*/
  /**
   * When the window gets the keyboard focus, give it to the button.<br>
   * This allows keyboard shortcuts to pop down the dialog.
```

```
 */
 /*-------------------------------------------------------------------*/
 public boolean gotFocus(Event e, Object arg) {
  button.requestFocus();
  return true;
 }

}


// End of File
```



```
/**************************************************************************
 ** PROGRAM : 仲介機能プログラム
 ** FILENAME :
 **        :
 ** DATE    : from Nutshell Book
 **        : 1998.12.21   T.Takumi(CSK)
 **************************************************************************/

// This example is from the book _Java in a Nutshell_ by David Flanagan.
// Written by David Flanagan.  Copyright (c) 1996 O'Reilly & Associates.
// You may study, use, modify, and distribute this example for any purpose.
// This example is provided WITHOUT WARRANTY either expressed or implied.

import java.awt.*;

/**************************************************************************
 ** Yes/No ダイアログ
 **
 **************************************************************************/
public class YesNoDialog extends Dialog {
 public static final int NO = 0;
 public static final int YES = 1;
 public static final int CANCEL = -1;

 protected Button yes = null, no = null, cancel = null;
 protected MultiLineLabel label;

 /*-------------------------------------------------------------------*/
 /** コンストラクタ
  */
 /*-------------------------------------------------------------------*/
 public YesNoDialog(Frame parent, String title, String message,
                    String yes_label, String no_label, String cancel_label)
  {
   // Create the window.
   super(parent, title, true);

   // Specify a LayoutManager for it
   this.setLayout(new BorderLayout(15, 15));

   // Put the message label in the middle of the window.
   label = new MultiLineLabel(message, 20, 20);
   this.add("Center", label);

   // Create a panel of buttons, center the row of buttons in
   // the panel, and put the pane at the bottom of the window.
   Panel p = new Panel();
   p.setLayout(new FlowLayout(FlowLayout.CENTER, 15, 15));
   if (yes_label != null) p.add(yes = new Button(yes_label));
   if (no_label != null)  p.add(no = new Button(no_label));
   if (cancel_label != null) p.add(cancel = new Button(cancel_label));
   this.add("South", p);

   // Set the window to its preferred size.
   this.pack();
  }

 /*-------------------------------------------------------------------*/
```

```
/** Handle button events by calling the answer() method.<BR>
 * Pass the appropriate constant value, depending on the button.
 *
 * @param           e           イベント
 * @param           arg
 */
/*-------------------------------------------------------------------------*/
public boolean action(Event e, Object arg)
  {
    if (e.target instanceof Button) {
            this.hide();
            this.dispose();
            if (e.target == yes) answer(YES);
            else if (e.target == no) answer(NO);
            else answer(CANCEL);
            return true;
    }
    else return false;
  }

/*-------------------------------------------------------------------------*/
/** Call yes(), no(), and cancel() methods depending on the button the
 * user clicked.  Subclasses define how the answer is processed by
 * overriding this method or the  yes(), no(), and cancel() methods.
 *
 * @param answer    押されたボタンの種類
 */
/*-------------------------------------------------------------------------*/
protected void answer(int answer) {
  switch(answer) {
  case YES: yes(); break;
        case NO:  no(); break;
        case CANCEL: cancel(); break;
                }
}
protected void yes() {}
protected void no() {}
protected void cancel() {}
}
```

#
V奈良 ウォークスルー メモ
#
##

・起動方法

このディレクトリで run を実行するのみ！

run の内容

```
#!/bin/csh

setenv SA_NETSCAPE /usr/local/bin/netscape.3.01

vnara -M data/nara_all.tak.iv -m data/mini_all.iv -s 23456 -p miris76 &
```

ここで SA_NETSCAPE は使用する Netscape を記述

vnara の引数は順に、

奈良データ、奈良鳥観図用データ、socket port 番号である。

※ 現在(1998.11.12) プログラムはデータに依存した書き方になっているので、データを修正した場合は注意が必要である。

・終了方法

メインウインドウ左上 - をダブルクリックしてください。

Enf of File

```
# pfinder
PFINDER = /home/takumi/2KEN/MIT/pfinder_ver2_O2

### Compiler
CCC     = /usr/bin/CC
CC      = cc

CC_DEF  = -DSOCKET
CFLAGS  = -I/usr/include/CC -I/usr/include -I. -I$(PFINDER)
CC_OPT  = -32

### Library
LDFLAGS = -32 -L/usr/lib
#LIB    = -lGLU -lGL -lX11 -lm -lInventorXt
LIB     = -lInventorXt -lInventor -lGLU -lGL -lXt -lX11 -lm -lpfinder

### Files
HEADERS= global.h define.h prototype.h sock_define.h sock_global.h

CPPOBJS  = vnara.o navigation.o camera.o callpf.o pf.o
COBJS    = server.o
OBJS     = $(CPPOBJS) $(COBJS)

EXEC     = vnara

###
.SUFFIXES:  .o .c .c++

.c++.o:
        $(CCC) $(CC_DEF) $(CFLAGS) $(CC_OPT) -c $<

.c.o:
        $(CC) $(CC_DEF) $(CFLAGS) $(CC_OPT) -c $<

###
$(EXEC): $(OBJS)
        $(CCC) -o $(EXEC) $(OBJS) $(LDFLAGS) -L$(PFINDER) $(LIB)

$(OBJS): $(HEADERS)

###
clean:;
        rm -f $(OBJS) $(EXEC)
```

```
/*****************************************************************************
 ** PROGRAM : V_Nara
 ** FILENAME : callpf.c++
 **        :
 ** DATE    : 1997.9.30   T.Takumi(CSK)
 **        : 1998.11.26
 *****************************************************************************/

#include <stream.h>
#include <rpc/rpc.h>
#include <string.h>
#include <bstring.h>
#include <unistd.h>

#include <pfinder_clnt.h>

#include <sys/types.h>
#include <bstring.h>
#include <sys/time.h>

#include "vnara.h"

#include "define.h"     // local definition
#include "global.h"
#include "prototype.h"

static CLIENT           *client;
static pfperson *person;
static pfopts           opt;
static pfv2f*scr;


/*****************************************************************************
 ** FUNCTION : ・msleep()
 **        : 指定時間(ms) sleep する
 *****************************************************************************/
void msleep(long msec)
{
  struct timeval timeout;
  long usec = msec * 1000;
  timeout.tv_sec = usec / 1000000;
  timeout.tv_usec = usec % 1000000;
  select (0, NULL, NULL, NULL, &timeout);
 }


/*****************************************************************************
 ** FUNCTION : ・initializePfinder()
 **        : pfinder と通信するための初期設定
 *****************************************************************************/
int initializePfinder(char *hostname) {

  pfRec.pfExist = PF_NOT_EXIST;             // pfinder status の初期化
  pfRec.pfHandR = PF_NOTHING;
  pfRec.pfHandL = PF_NOTHING;
  pfRec.pfPose  = PF_STANDING;
  pfRec.pfPosition = 0.0;
  pfRec.pfRotation = 0.0;
```



```
  pfRec.sensPos = DEF_SENS_POS;
  pfRec.sensRot = DEF_SENS_ROT;
  pfRec.calib = FALSE;

  bzero(&opt,sizeof(pfopts));
  if (!( client = clnt_create( hostname, PFINDERPROG, PFINDERVERS,"tcp"))) {
    clnt_pcreateerror( hostname );
    return(-1);
  }

  opt.kalman = 0;
  opt.alive_coords = 1;
  opt.f2 = opt.g = opt.c2 = opt.c3 = NO;

  opt.f3   = YES;         /* 3D 位置 */
  opt.g    = YES;         /* ジェスチャ */
  opt.cc2d = NO;
  opt.cc3d = NO;

  scr = pfscrsize_1( NULL, client );
  if (!scr) {
    clnt_perror(client, hostname);
    return(-1);
  }

  return(1);
}


/*****************************************************************************
 ** FUNCTION : ・call_pfinder()
 **        : pfinder からデータを受け取る
 **        :  g mode と f3 mode しか使っていない
 *****************************************************************************/
int  call_pfinder(void)
{
  static int   usrExist = FALSE;

  /* 前回の領域解放 */
  xdr_free((xdrproc_t)xdr_pfperson, person);

  person = pf_1(&opt, client);
  if (!person) {
    printf("WARNING: can't get data from pfinder.\n");
    return(FALSE);
  }

  // ジェスチャの取得 ----------------------------------------
  if (person->g.opt) {
    pfgest_s *gesture = person->g.pfgest_u.bits;

    if (gesture->user_exists) {
      usrExist = TRUE;
      pfRec.pfExist = PF_EXIST;

      // 姿勢(pose)
      if (gesture->standing) {
//      printf("STANDING    ");
```

```
      pfRec.pfPose = PF_STANDING;
    }
    if (gesture->sitting) {
//      printf("SITTING    ");
      pfRec.pfPose = PF_SITTING;
    }
#if 0
    if (gesture->bending_over) {
//      printf("BENDING OVER    ");
      pfRec.pfPose = PF_BENDING_OVER;
    }
#endif

    // 手の状態
    pfRec.pfHandR = PF_NOTHING;
    pfRec.pfHandL = PF_NOTHING;

    if (gesture->point_left) {
//      printf("POINT RIGHT    ");
      pfRec.pfHandR = PF_POINT;                    // 逆
    }
    if (gesture->point_right) {
//      printf("POINT LEFT    ");
      pfRec.pfHandL = PF_POINT;                    // 逆
    }
    if (gesture->kick_left) {
//      printf("KICK RIGHT    ");
      pfRec.pfHandR = PF_POINT;                    // 逆
    }
    if (gesture->kick_right) {
//      printf("KICK LEFT    ");
      pfRec.pfHandL = PF_POINT;                    // 逆
    }
#if 0
    if (gesture->extend_left) {
//      printf("EXTEND RIGHT    ");
      pfRec.pfHandR = PF_POINT;                    // 逆
    }
    if (gesture->extend_right) {
//      printf("EXTEND LEFT    ");
      pfRec.pfHandL = PF_POINT;                    // 逆
    }
#endif

#if 0
    if (gesture->hand_moving_left)      printf("HAND MOVING LEFT    ");
    if (gesture->hand_moving_right)     printf("HAND MOVING RIGHT    ");
    if (gesture->throw_left)            printf("THROW LEFT    ");
    if (gesture->throw_right)           printf("THROW RIGHT    ");
#endif
//    printf("\n");
   }
   else {
    usrExist = FALSE;
    pfRec.pfExist = PF_NOT_EXIST;
   }
  }
```



```
 // 位置 ------------------------------------------------
 if (person->f3.opt) {

  if (usrExist == TRUE) {
   pffeat3d_s *f3 = person->f3.pffeat3d_u.feats;

#if 0
   // raw data
   printf("centroid (%7.2f %7.2f,%7.2f)  ",
       f3->centroid.x, f3->centroid.y, f3->centroid.z, );
#endif

   // default の調整値(立ち位置)
   static float  pfHomeX = PFHOME_X;
   static float  pfHomeY = PFHOME_Y;
   static float  pfHomeZ = PFHOME_Z;

   // calibration (人間の立ち位置の中心を初期化)
   if (pfRec.calib == TRUE) {
    printf("HOME >> (%7.2f %7.2f,%7.2f) --> ",
              pfHomeX, pfHomeY, pfHomeZ);
    pfHomeX = f3->centroid.x;
    pfHomeY = f3->centroid.y;
    pfHomeZ = f3->centroid.z;
    printf("(%7.2f %7.2f,%7.2f)\n",
              pfHomeX, pfHomeY, pfHomeZ);
    pfRec.calib = FALSE;
   }

#if 0
   // 調整された値
   printf("centroid (%7.2f %7.2f,%7.2f)  \n",
             f3->centroid.x - pfHomeX, f3->centroid.y - pfHomeY,
             f3->centroid.z - pfHomeZ, );
#endif

   float xx, zz;
   xx = f3->centroid.x - pfHomeX;
   zz = f3->centroid.z - pfHomeZ;

#if 0       /* 限界 */
   if (xx > MAX_RADIUS || xx < -MAX_RADIUS
     || zz > MAX_RADIUS || zz < -MAX_RADIUS) {
    pfRec.pfRotation = 0.0;
    pfRec.pfPosition = 0.0;
   }
   else {
#endif      /* 限界 */
    if (xx > NEUTRAL_RADIUS)
                pfRec.pfRotation = xx - NEUTRAL_RADIUS;
    else if (xx < -NEUTRAL_RADIUS)
                pfRec.pfRotation = xx + NEUTRAL_RADIUS;
    else   pfRec.pfRotation = 0.0;

    if (zz > NEUTRAL_RADIUS)
                pfRec.pfPosition = NEUTRAL_RADIUS - zz;
    else if (zz < -NEUTRAL_RADIUS)
                pfRec.pfPosition = -NEUTRAL_RADIUS - zz;
```

```
      else   pfRec.pfPosition = 0.0;
#if 0        /* 限界 */
    }
#endif       /* 限界 */

#if 0
    // 手の座標(今は関係ない)
    printf("left hand (%7.2f %7.2f,%7.2f)  ",
                f3->left_hand.x,
                f3->left_hand.y,
                f3->left_hand.z, );
    printf("right hand (%7.2f %7.2f,%7.2f)\n",
                f3->right_hand.x,
                f3->right_hand.y,
                f3->right_hand.z, );
#endif
   }
  }

////// 今は使わない ////////////////////
#ifdef OTHER_MODES
 // cc3d
 if (person->cc3d.opt) {
  pfcc3d_s *cc = person->cc3d.pfcc3d_u.cc.cc_val;
  int num = person->cc3d.pfcc3d_u.cc.cc_len;

  for (int j=0; j<num; j++) {
   printf(" %1d ", j);
   printf("gm %7.2f %7.2f %7.2f ",
                cc[j].gmean[0], cc[j].gmean[1], cc[j].gmean[2]);
   printf("ge0 %6.2f %6.2f %6.2f %6.2f ",
                cc[j].gL[0],
                cc[j].gS[0].x,
                cc[j].gS[0].y,
                cc[j].gS[0].z);
   printf("ge1 %6.2f %6.2f %6.2f %6.2f ",
                cc[j].gL[1],
                cc[j].gS[1].x,
                cc[j].gS[1].y,
                cc[j].gS[1].z);
   printf("ge2 %6.2f %6.2f %6.2f %6.2f\n",
                cc[j].gL[2],
                cc[j].gS[2].x,
                cc[j].gS[2].y,
                cc[j].gS[2].z);
  }
 }

 // cc2d
 if (person->cc2d.opt) {
  pfcc2d_s *cc = person->cc2d.pfcc2d_u.cc.cc_val;
  int num = person->cc2d.pfcc2d_u.cc.cc_len;

  for (int j=0; j<num; j++) {
   printf(" %1d cm %6.2f %6.2f %6.2f ", j,
                cc[j].cmean[0], cc[j].cmean[1], cc[j].cmean[2]);
   printf("gm %6.2f %6.2f ",
                cc[j].gmean[0], cc[j].gmean[1]);
```



```
   printf("ge0 %6.2f %6.2f %6.2f ",
                cc[j].gL[0], cc[j].gS[0].x, cc[j].gS[0].y);
   printf("ge1 %6.2f %6.2f %6.2f\n",
                cc[j].gL[1], cc[j].gS[1].x, cc[j].gS[1].y);

  }
 }
#endif

 return(TRUE);
}
```

```
/*********************************************************************
 ** PROGRAM : Virtual NARA
 ** FILENAME : camera.c++
 **       :
 ** DATE    : 1998.11.01   T.Takumi(CSK)
 *********************************************************************/

#include "vnara.h"

#include "define.h"     // local definition
#include "global.h"
#include "prototype.h"


/*********************************************************************
 ** FUNCTION : ・setFixedCamera()
 **       : 固定カメラをセット
 *********************************************************************/
void setFixedCamera(void)
{
  static float cameraData[NUM_CAMERA+1][2][7] = {
    // dummy
    {{36882.359375,    980.983215,          35001.003906,      0.0,      1.0,      0.0,      5.885316},
    {36882.359375,     980.983215,          35001.003906,      0.0,      1.0,      0.0,      5.885316}},

    {{15944.867188,    87.537460, 16302.751953,      0.0,      1.0,      0.0,      2.851932},
    {16141.261719,     87.537598, 15622.858398,      0.0,      1.0,      0.0,      -3.419317}},

    {{19846.248047,    154.706970,          16092.491211,      0.0,      1.0,      0.0,      -2.930424},
    {19482.214844,     154.706970,          14395.090820,      0.0,      1.0,      0.0,      -2.930272}},

    {{20613.521484,    181.977539,          17144.830078,      0.0,      1.0,      0.0,      -5.470325},
    {21474.347656,     268.020569,          17956.712891,      0.0,      1.0,      0.0,      -5.472561}},

    {{18925.085938,    138.000671,          16857.804688,      0.0,      1.0,      0.0,      5.073163},
    {17553.714844,     138.000671,          17376.833984,      0.0,      1.0,      0.0,      -1.208783}},

    {{18856.861328,    104.996956,          17403.568359,      0.0,      1.0,      0.0,      6.091743},
    {18757.583984,     104.996956,          17915.837891,      0.0,      1.0,      0.0,      -0.191423}},

    {{14810.626953,    87.537460, 17635.582031,      0.0,      1.0,      0.0,      3.102667},
    {14847.541016,     87.537460, 16688.632812,      0.0,      1.0,      0.0,      -3.180604}},

    {{14705.114258,    92.528961, 18445.683594,      0.0,      1.0,      0.0,      6.071247},
    {14564.794922,     92.528961, 19116.666016,      0.0,      1.0,      0.0,      -0.204658}},

    {{10155.125000,    87.537460, 24421.769531,      0.0,      1.0,      0.0,      5.441260},
    {9576.870117,      87.537460, 24938.074219,      0.0,      1.0,      0.0,      -0.841787}},


    {{95571.742188,  454.771362,    87686.250000,  0.0, 1.0,      0.0,      0.036107},
    {95602.101562,   522.949158,    88538.312500,  0.000000,    1.000000,      0.000000,      0.035407}},
//  {{37937.199219,  434.364655,          32454.671875,      0.0,      1.0,      0.0,      0.097695},
//  {36882.359375,   980.983215,          35001.003906,      0.0,      1.0,      0.0,      5.885316}},

    // ATR のカメラ位置は設定していない
    {{37937.199219,    434.364655,          32454.671875,      0.0,      1.0,      0.0,      0.097695},
    {36882.359375,     980.983215,          35001.003906,      0.0,      1.0,      0.0,      5.885316}}
```



```
  };

  // 地形データの位置が変わったので、xz位置をずらす (1 から 8 のカメラだけ！)
  // (9 のカメラはあとから修正した。dummy(0), 10 のカメラについては適当な値)
  for (int i = 1; i <= 8; i++) {
    for (int j = 0; j < 2; j++) {
      cameraData[i][j][0] += 57286.0;                    // x
      cameraData[i][j][2] += 54995.0;                    // z
    }
  }

  // 回転角は -PI から PI の範囲にする
  for (i = 0; i < NUM_CAMERA+1; i++) {
    for (int j = 0; j < 2; j++) {

      while (fabs(cameraData[i][j][6]) > M_PI) {
              if (cameraData[i][j][6] > M_PI) {
                cameraData[i][j][6] -= 2.0 * M_PI;
              }
              else if (cameraData[i][j][6] < -M_PI) {
                cameraData[i][j][6] += 2.0 * M_PI;
              }
      }


    }
  }

  for (i = 0; i < NUM_CAMERA+1; i++) {
    for (int j = 0; j < 2; j++) {
      fixedCamera[i][j].position.setValue(&cameraData[i][j][0]);
      fixedCamera[i][j].axis.setValue(&cameraData[i][j][3]);
      fixedCamera[i][j].radians = cameraData[i][j][6];
    }
  }

  for (i = 0; i < NUM_CAMERA+1; i++) {
    for (int j = 0; j < 2; j++) {
      for (int k = 0; k < 7; k++) {
        printf("  %f", cameraData[i][j][k]);
      }
      printf("\n");
    }
  }


}


/*********************************************************************
 ** FUNCTION : ・cameraPositionChangedCB()
 **       : camera の位置が変った時に呼ばれる
 *********************************************************************/
void cameraPositionChangedCB(void *data, SoSensor *)
{
//  printf("cameraPositionChangedCB() is called.\n");

  // 寺アイコンをカメラの方向に向ける
  changeTempleDirection(camera->position.getValue());
```

```
 // Top Viewer のカメラCone の位置を変える
 cameraConeTransform->translation = camera->position;
}


/*****************************************************************
 ** FUNCTION : ・cameraOrientationChangedCB
 **       : camera の方向が変った時に呼ばれる
 *****************************************************************/
void cameraOrientationChangedCB(void *data, SoSensor *)
{
//  printf("cameraOrientationChangedCB() is called.\n");

 // Top Viewer のカメラCone の方向を変える
 cameraConeTransform->rotation = camera->orientation;
}


/*****************************************************************
 ** FUNCTION : ・changeTempleDirection()
 **       : 寺アイコンがカメラの方向を向くようにする
 *****************************************************************/
void  changeTempleDirection(SbVec3f camPos)
{
 int                    i;
 SbVec3f          t;
 SoSFVec3f              s;

 /* temples */
 for (i = 0; i < NUM_TEMPLE; i++) {
  s = templeTransform[i]->scaleFactor;
  t = templeTransform[i]->translation.getValue();

  camPos[1] = t[1];
  templeTransform[i]->pointAt(t, camPos);
  templeTransform[i]->scaleFactor = s;
 }

 /* ATR */
 s = atrTransform->scaleFactor;
 t = atrTransform->translation.getValue();

 camPos[1] = t[1];
 atrTransform->pointAt(t, camPos);
 atrTransform->scaleFactor = s;
}
```



```
/*****************************************************************
 ** PROGRAM : Virtual NARA
 ** FILENAME : navigation.c++
 **       :
 ** DATE   : 1998.10.31   T.Takumi(CSK)
 *****************************************************************/

#include "vnara.h"

#include "define.h"     // local definition
#include "global.h"
#include "prototype.h"


/*****************************************************************
 ** FUNCTION : ・pickIcon()
 **       : 寺アイコンを pick
 **       : ※ 室生寺の場合は情報表示
 *****************************************************************/
void  pickIcon(char com[]) {

 unselectNode();

 if (com[0] == '9') {
  callNetscape(com);
 }
 else {
  pickLowerAnyway();
 }
}


/*****************************************************************
 ** FUNCTION : ・naviCameraCB()
 **       : AUTO Navigation モードの callback
 *****************************************************************/
void  naviCameraCB(void *, SoSensor *)
{
 if (naviCnt >= numTrackNavi) {
  // Navigation 終了
  printf("TAKTAK>> Navigation [END] !!!\n");
  naviCameraSensor->unschedule();

  // 情報の提示
//  callNetscape(command[curComIdx]);
  pickIcon(command[curComIdx]);

  // まだコマンドがあれば実行
  checkNextCommand();

  return;
 }

 CameraParamRec  *trackPnt;
 trackPnt = &trackNavi[naviCnt];

#if 0
 printf("TAK (%d) >> (%f, %f, %f)  (%f, %f, %f)  %f\n", naviCnt,
```

```
      trackPnt->position[0],
      trackPnt->position[1],
      trackPnt->position[2],
      trackPnt->axis[0],
      trackPnt->axis[1],
      trackPnt->axis[2],
      trackPnt->radians);
#endif

 camera->position.setValue(trackPnt->position);
 camera->orientation.setValue(trackPnt->axis, trackPnt->radians);

 naviCnt++;
}


/*****************************************************************************
 ** FUNCTION : ・setAnimationPath()
 **       : アニメーションの経路を計算
 *****************************************************************************/
void  setAnimationPath(int idx0, int idx)
{
 // 現在のカメラ位置 (1)
 CameraParamRec     curCam;

 curCam.position = camera->position.getValue();
 SbRotation cameraOrientation = camera->orientation.getValue();
 cameraOrientation.getValue(curCam.axis, curCam.radians);

 // 途中の通過点 (2)
 CameraParamPtr      cam2;
 if (idx0 < 0) {
  cam2 = &curCam;
 }
 else {
  cam2 = &fixedCamera[idx0][1];
 }

 // 途中の通過点 (3)
 CameraParamPtr      cam1 = &fixedCamera[idx][1];

 // ターゲットのカメラ位置 (4)
 CameraParamPtr      cam0 = &fixedCamera[idx][0];

 // 回転軸は常に y軸の正方向とする
 if (curCam.axis[1] < 0.0) {
  printf("curCam.axis[1] < 0.0 !!\n");
  curCam.axis *= -1.0;
  curCam.radians *= -1.0;
 }
 // 回転角は -PI から PI の範囲にする
 while (fabs(curCam.radians) > M_PI) {
  if (curCam.radians > M_PI) {
   curCam.radians -= 2.0 * M_PI;
  }
  else if (curCam.radians < -M_PI) {
   curCam.radians += 2.0 * M_PI;
  }
```



```
 }
 numTrackNavi = 0;

#if 1
//   setAnimationPathBezier(&curCam, cam2, cam1, cam0);

  // いったん視点を上げながら移動する
  // cam2 の高さを大きくする
  if (idx0 == 9 || idx == 9) {
   setAnimationPathBezier2(&curCam, cam2, cam1, cam0, 2000.0);
  }
  else {
   setAnimationPathBezier2(&curCam, cam2, cam1, cam0, 0.0);
  }
#else
 if (idx == 9) {
  setAnimationPath2(&curCam, &fixedCamera[0][0]);
  setAnimationPath2(&fixedCamera[0][0], cam);
 }
 else {
  setAnimationPath2(&curCam, cam);
 }
#endif
}


/*****************************************************************************
 ** FUNCTION : ・setAnimationPath2()
 **       : アニメーションの経路を計算
 *****************************************************************************/
void  setAnimationPath2(CameraParamPtr cam1, CameraParamPtr cam2)
{
 int         i;

 CameraParamRec diff;

 // 差分を計算
 diff.position = cam2->position - cam1->position;
 diff.axis    = cam2->axis - cam1->axis;
 diff.radians  = cam2->radians - cam1->radians;

 // 回転が 180度より大きい場合は逆回転させる
 if (diff.radians > M_PI) {
  diff.radians -= 2.0 * M_PI;
 }
 else if (diff.radians < -M_PI) {
  diff.radians += 2.0 * M_PI;
 }

 diff.position /= (float)NAVI_DIV;
 diff.axis    /= (float)NAVI_DIV;
 diff.radians  /= (float)NAVI_DIV;

 for (i = 1; i <= NAVI_DIV; i++) {
  if (numTrackNavi >= NAVI_TRK_MAX) {
   printf("WARNING: trackNavi[] is overflowing.");
  }
  trackNavi[numTrackNavi].position = cam1->position + i * diff.position;
```

```
    trackNavi[numTrackNavi].axis     = cam1->axis + i * diff.axis;
    trackNavi[numTrackNavi].radians  = cam1->radians + i * diff.radians;
    numTrackNavi++;
  }

}


/*****************************************************************
 ** FUNCTION : ・setAnimationPathBezier()
 **        : アニメーションの経路を計算
 *****************************************************************/
void  setAnimationPathBezier(CameraParamPtr cam0, CameraParamPtr cam1, CameraParamPtr cam2, CameraParamPtr cam3)
{
#if 1
  // 回転角の差分を計算
  float radians  = cam3->radians - cam0->radians;

  // 回転が 180度より大きい場合は逆回転させる
  if (radians > M_PI) {
    radians -= 2.0 * M_PI;
  }
  else if (radians < -M_PI) {
    radians += 2.0 * M_PI;
  }

  float div = 1.0/(float)NAVI_DIV;

  float b0, b1, b2, b3;
  float ff;


  for (float f = 0.0; f < 1.0; f += div) {

    ff = 1.0 - f;
    b0 = ff * ff * ff;
    b1 = 3.0 * f * ff * ff;
    b2 = 3.0 * f * f * ff;
    b3 = f * f * f;

    trackNavi[numTrackNavi].position = cam0->position * b0 + cam1->position * b1
                                     + cam2->position * b2 + cam3->position * b3;
    trackNavi[numTrackNavi].axis     = cam3->axis;
    trackNavi[numTrackNavi].radians  = cam0->radians + radians * f;
    numTrackNavi++;
  }
  trackNavi[numTrackNavi].position = cam3->position;
  trackNavi[numTrackNavi].axis     = cam3->axis;
  trackNavi[numTrackNavi].radians  = cam3->radians;
  numTrackNavi++;

#else
    trackNavi[numTrackNavi].position = cam0->position;
    trackNavi[numTrackNavi].axis     = cam0->axis;
    trackNavi[numTrackNavi].radians  = cam0->radians;
    numTrackNavi++;
    trackNavi[numTrackNavi].position = cam3->position;
    trackNavi[numTrackNavi].axis     = cam3->axis;
```



```
    trackNavi[numTrackNavi].radians  = cam3->radians;
    numTrackNavi++;
#endif

#if 0
  for (int i = 0; i < numTrackNavi; i++) {
    printf("TAK (%d) >> (%f, %f, %f)  (%f, %f, %f)  %f\n", i,
        trackNavi[i].position[0],
        trackNavi[i].position[1],
        trackNavi[i].position[2],
        trackNavi[i].axis[0],
        trackNavi[i].axis[1],
        trackNavi[i].axis[2],
        trackNavi[i].radians);
  }
#endif

}


/*****************************************************************
 ** FUNCTION : ・setAnimationPathBezier2()
 **        : アニメーションの経路を計算
 *****************************************************************/
void  setAnimationPathBezier2(CameraParamPtr cam0, CameraParamPtr cam1, CameraParamPtr cam2, CameraParamPtr cam3, float
{
  // cam2 に加える高さの増分
  SbVec3f  offsetY(0.0, offset, 0.0);

  // 回転角の差分を計算
  float radians  = cam3->radians - cam0->radians;

  // 回転が 180度より大きい場合は逆回転させる
  if (radians > M_PI) {
    radians -= 2.0 * M_PI;
  }
  else if (radians < -M_PI) {
    radians += 2.0 * M_PI;
  }

  float div = 1.0/(float)NAVI_DIV;

  float b0, b1, b2, b3;
  float ff;


  for (float f = 0.0; f < 1.0; f += div) {

    ff = 1.0 - f;
    b0 = ff * ff * ff;
    b1 = 3.0 * f * ff * ff;
    b2 = 3.0 * f * f * ff;
    b3 = f * f * f;

    trackNavi[numTrackNavi].position
      = cam0->position * b0          + cam1->position * b1
      + (cam2->position + offsetY) * b2 + cam3->position * b3;
    trackNavi[numTrackNavi].axis     = cam3->axis;
```

```
    trackNavi[numTrackNavi].radians  = cam0->radians + radians * f;
    numTrackNavi++;
  }
  trackNavi[numTrackNavi].position = cam3->position;
  trackNavi[numTrackNavi].axis     = cam3->axis;
  trackNavi[numTrackNavi].radians  = cam3->radians;
  numTrackNavi++;
}


/*****************************************************************
 ** FUNCTION : ・execCommandCB
 **       : コマンド列の実行
 *****************************************************************/
void  execCommandCB(void *, SoSensor *)
{
  // コマンドの処理
  printf("execCommandCB>> %s\n", command[curComIdx]);
  int idx = command[curComIdx][0]-48;        // 一文字目で判断

  if (idx != curObjIdx) {              // オブジェクトが違う場合、移動
                                       // アニメーションの最後で情報表示
    setAnimationPath(curObjIdx, idx);

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();

    curObjIdx = idx;     // カレントのオブジェクトを保存
  }
  else {                   // オブジェクトが同じ場合、情報の表示のみ
    // 情報の提示
    callNetscape(command[curComIdx]);

    // 次のコマンド
    checkNextCommand();
  }
}


/*****************************************************************
 ** FUNCTION : ・checkNextCommand()
 **       : 次のコマンドをチェック
 *****************************************************************/
void checkNextCommand() {
  curComIdx++;
  if (curComIdx >= comCnt) {
    processingCommand = FALSE;
  }
  else {
    commandSensor->setTimeFromNow(COMMAND_INTERVAL);
    commandSensor->schedule();
  }
}


/*****************************************************************
 ** FUNCTION : ・callNetscape()
```



```
 **       : ネットスケープを call
 *****************************************************************/
void  callNetscape(char *name)
{
  char combuf[256];

//printf("curDir --> %s\n", curDir);

  if (strcmp("atr", name) == 0) {
    sprintf(combuf, CALL_NS_ATR, getenv("SA_NETSCAPE"));
    if (system(combuf)) {
      sprintf(combuf, CALL_NS_ATR2, getenv("SA_NETSCAPE"));
      system(combuf);
    }
  }
  else {
    sprintf(combuf, CALL_NS, getenv("SA_NETSCAPE"), curDir, name);
    if (system(combuf)) {
      sprintf(combuf, CALL_NS2, getenv("SA_NETSCAPE"), curDir, name);
      system(combuf);
    }
  }

printf("callNetscape>> %s\n", combuf);
}


/*****************************************************************
 ** FUNCTION : ・unselectNode()
 **       : 物体の選択を解除する
 *****************************************************************/
void unselectNode(){

  selectionRoot->deselectAll();
  selectionNode = NULL;
  selectionPath = NULL;
  myWalkViewer->render();
  auxWalkViewer->render();

}



/* End of File */
```

```
/*************************************************************************
 ** PROGRAM  : V_Nara
 ** FILENAME : pf.c++
 **        :
 ** DATE     : 1997.9.30   T.Takumi(CSK)
 **        : 1998.11.26
 *************************************************************************/

#include "vnara.h"

#include "define.h"     // local definition
#include "global.h"
#include "prototype.h"


/*************************************************************************
 ** FUNCTION : ・checkPfinderEvent()
 **        : pfinder の状態により処理を行なう
 *************************************************************************/
void checkPfinderEvent()
{
  static int  pfExist0 = PF_NOT_EXIST;
  static int  pfPose0  = PF_STANDING;

  static int  pfLookingUp = 0;      /* 連続してイベントは受けない */
  static int  pfClickR    = 0;
  static int  pfClickL    = 0;

  static struct timeval time0;

  struct timeval time;
  int delta_t;

  if (call_pfinder() != TRUE) return;

  /*== 人の存在チェック ==*/
  if (pfRec.pfExist == PF_NOT_EXIST) {
    if (pfExist0 != PF_NOT_EXIST) {
/*      goHomePosition();*/
      unselectNode();
/*      callNetscape("home");*/
      pfExist0 = PF_NOT_EXIST;
    }

    /*== 時間の計測(time0 クリア) ==*/
    gettimeofday(&time);
    time0.tv_sec = time.tv_sec;
    time0.tv_usec = time.tv_usec;
    return;   // 以下は実行せず抜ける
  }
  else{
    if (pfExist0 == PF_NOT_EXIST) {
      XResetScreenSaver(XtDisplay(XtParent(myWalkViewer->getWidget())));
    }
  }
  pfExist0 = pfRec.pfExist;

#if 0
```



```
  /*== LookUp 動作の時の時間はカウントしない ==*/
  if (clearPfTimer) {
    gettimeofday(&time);
    time0.tv_sec = time.tv_sec;
    time0.tv_usec = time.tv_usec;
    clearPfTimer = 0;
  }
#endif

  /*== 時間の計測(速度の計算に使う) ==*/
  gettimeofday(&time);

  /* Delta_t in microseconds */
  delta_t = (time.tv_sec*1000 + time.tv_usec/1000) -
          (time0.tv_sec*1000 + time0.tv_usec/1000);

#if 0
  if (delta_t)
    printf("taktak>> ---------------------- %d ms  (%d Hz)\n", delta_t, 1000/delta_t);
  else
    printf("taktak>> ---------------------- %d ms  (?? Hz)\n", delta_t);
  printf("taktak>> Pos, Rot ==> %f, %f   --> ",
          pfRec.pfPosition, pfRec.pfRotation);
#endif

  // 得られる数値には意味合いがないので適当に調整している
  pfRec.pfPosition *= (float)delta_t/1000.0 * pfRec.sensPos * 0.15;
  pfRec.pfRotation *= (float)delta_t/1000.0 * pfRec.sensRot * 0.15;
//  printf("taktak>> %f, %f\n",
//            pfRec.pfPosition, pfRec.pfRotation);


#if 0
  /*== 姿勢の状態チェック ==*/
  switch (pfRec.pfPose) {

  case PF_STANDING:
    if (pfPose0 == PF_SITTING) {
#ifdef EYE_HEIGHT
      sitting = FALSE;
#else
      sitting = FALSE;
      changeCameraYPosition(cameraHome->position[1]);
#endif
    }
    pfPose0 = PF_STANDING;
    break;

  case PF_SITTING:
    if (pfPose0 == PF_STANDING) {
#ifdef EYE_HEIGHT
      sitting = TRUE;
#else
      sitting = TRUE;
      changeCameraYPosition(0.5);
#endif
    }
    pfPose0 = PF_SITTING;
```

```
    break;
  }
#endif

  /*== 人の位置(向き)の更新 ==*/
/*  if (pointingAt == FALSE) {*/
    if (pfRec.pfPosition != 0.0) {
      changeCameraXZPosition(0.0, pfRec.pfPosition); // カメラの Z方向
    }

    if (pfRec.pfRotation != 0.0) {
      changeCameraOrientation(pfRec.pfRotation);
    }
/*  }*/

  /*== 手の状態チェック ==*/
  if (pfRec.pfHandR == PF_POINT) {
    if (pfRec.pfHandL == PF_POINT) {
#if 0
      //視線を上に向ける
      if (pfLookingUp == 0) {
        if (exetype & TYPE_LASER_P)  popMenu();
        else                         lookingUp();
        pfLookingUp = 1;
      }
#endif
    }
    else {
      //右側クリック
      if (pfClickR == 0) {
        unselectNode();
        pickRightSide();
        pfClickR = 1;
      }

      pfClickL = 0;
      pfLookingUp = 0;
    }
  }
  else {
    if (pfRec.pfHandL == PF_POINT) {
      //左側クリック
      if (pfClickL == 0) {
        unselectNode();
        pickLeftSide();
        pfClickL = 1;
      }
    }
    else {
      pfClickL = 0;
      pfLookingUp = 0;
    }

    pfClickR = 0;
    pfLookingUp = 0;
  }

  /*== 時間の計測(time0 クリア) ==*/
```



```
  gettimeofday(&time);
  time0.tv_sec = time.tv_sec;
  time0.tv_usec = time.tv_usec;
}


/*****************************************************************
 ** FUNCTION : ・pickPoint()
 **        : 与えたウインドウ座標位置を pick する
 *****************************************************************/
SbBool  pickPoint
(
  const SbViewportRegion &viewport,
  const SbVec2s &cursorPosition
)
{
  SoRayPickAction myPickAction(viewport);

  // Set an 8-pixel wide region around the pixel
  myPickAction.setPoint(cursorPosition);
  myPickAction.setRadius(80.0);   // これは線や点にのみ有効

  // Start a pick traversal
  myPickAction.apply(scene);
  const SoPickedPoint *myPickedPoint =
        myPickAction.getPickedPoint();
  if (myPickedPoint == NULL){
    return FALSE;
  }

  selectionRoot->select(myPickFilter(myPickedPoint));

  return TRUE;
}


/*****************************************************************
 ** FUNCTION : ・pickRightSide
 **        : 中央右側 6点を pick
 *****************************************************************/
void pickRightSide()
{
  int pickedObject = FALSE;

  SbVec2s winSize;
  winSize = myWalkViewer->getSize();

  for (float f = 0.5; f < 0.8; f += 0.05) {
    pickPoint(myWalkViewer->getViewportRegion(),
              SbVec2s(winSize[0]*f, winSize[1]/2));
    if (pickedObject) return;
  }
}


/*****************************************************************
 ** FUNCTION : ・pickLeftSide
 **        : 中央左側 6点を pick
```

```
*************************************************************************/
void pickLeftSide()
{
 int pickedObject = FALSE;

 SbVec2s winSize;
 winSize = myWalkViewer->getSize();

 for (float f = 0.5; f > 0.2; f -= 0.05) {
  pickPoint(myWalkViewer->getViewportRegion(),
            SbVec2s(winSize[0]*f, winSize[1]/2));
  if (pickedObject) return;
 }
}


/*************************************************************************
 ** FUNCTION : ・pickLower
 **       : 壷を pick するための処理
 **       : 同じ住居が連続して pick された時に呼ばれる
 *************************************************************************/
void pickLower()
{
printf("taktak>> pickLower() in.\n");

 int pickedObject = FALSE;                  /* 何も選ばれていない状態に */

 SbVec2s winSize;
 winSize = myWalkViewer->getSize();

 if (pfRec.pfHandR == PF_POINT) {
  for (float f = 0.5; f < 0.9; f += 0.05) {
   pickPoint(myWalkViewer->getViewportRegion(),
             SbVec2s(winSize[0]*f, winSize[1]*0.125));
   if (pickedObject) return;

   pickPoint(myWalkViewer->getViewportRegion(),
             SbVec2s(winSize[0]*f, winSize[1]*0.25));
   if (pickedObject) return;

   pickPoint(myWalkViewer->getViewportRegion(),
             SbVec2s(winSize[0]*f, winSize[1]*0.375));
   if (pickedObject) return;
  }
 }
 else {
  for (float f = 0.5; f > 0.1; f -= 0.05) {
   pickPoint(myWalkViewer->getViewportRegion(),
             SbVec2s(winSize[0]*f, winSize[1]*0.125));
   if (pickedObject) return;

   pickPoint(myWalkViewer->getViewportRegion(),
             SbVec2s(winSize[0]*f, winSize[1]*0.25));
   if (pickedObject) return;

   pickPoint(myWalkViewer->getViewportRegion(),
             SbVec2s(winSize[0]*f, winSize[1]*0.375));
   if (pickedObject) return;
```



```
  }
 }
}


/*************************************************************************
 ** FUNCTION : ・pickLowerAnyway
 **       : 壷を pick するための処理
 **       : 単独で呼ばれる
 **       : とにかく下の方を pick する
 *************************************************************************/
void pickLowerAnyway()
{
 printf("taktak>> pickLowerAnyway() in.\n");

 int pickedObject = FALSE;                  /* 何も選ばれていない状態に */

 SbVec2s winSize;
 winSize = myWalkViewer->getSize();

 // 右側
 for (float f = 0.5; f < 0.9; f += 0.05) {
  pickPoint(myWalkViewer->getViewportRegion(),
            SbVec2s(winSize[0]*f, winSize[1]*0.125));
  if (pickedObject) return;

  pickPoint(myWalkViewer->getViewportRegion(),
            SbVec2s(winSize[0]*f, winSize[1]*0.25));
  if (pickedObject) return;

  pickPoint(myWalkViewer->getViewportRegion(),
            SbVec2s(winSize[0]*f, winSize[1]*0.375));
  if (pickedObject) return;
 }

 // 左側
 for (f = 0.5; f > 0.1; f -= 0.05) {
  pickPoint(myWalkViewer->getViewportRegion(),
            SbVec2s(winSize[0]*f, winSize[1]*0.125));
  if (pickedObject) return;

  pickPoint(myWalkViewer->getViewportRegion(),
            SbVec2s(winSize[0]*f, winSize[1]*0.25));
  if (pickedObject) return;

  pickPoint(myWalkViewer->getViewportRegion(),
            SbVec2s(winSize[0]*f, winSize[1]*0.375));
  if (pickedObject) return;
 }
}


/*************************************************************************
 ** FUNCTION : ・myPickFilter()
 **       : pickPoint で選択された時のフィルタ
 *************************************************************************/
SoPath *
myPickFilter(const SoPickedPoint *pick)
```

```
{
 int selected = 0;

 // See which child of selection got picked
 SoPath *p = pick->getPath();
 int i;
 for (i = 0; i < p->getLength() - 2; i++) {
  SoNode *n = p->getNode(i);
  if (n->isOfType(SoSelection::getClassTypeId())) {
   selected = 1;
   break;
  }
 }

 if (selected) {
  // Copy 5 nodes from the path:
  // selection and the picked child
  return p->copy(i, 5);
 }
 else {
  return((SoPath *)NULL);
 }
}


/******************************************************************************
 ** FUNCTION : ・changeCameraXZPosition()
 **       : pfinder のイベントによりカメラの x, z 位置を変える
 ******************************************************************************/
void changeCameraXZPosition(float x, float z)
{
 // カメラの位置と方向を取得
 SbVec3f cameraPosition = camera->position.getValue();
 SbRotation rot0 = camera->orientation.getValue();

 // マトリクス演算
 SbMatrix mat0;
 SbVec3f  directionX(1.0, 0.0, 0.0), newDirectionX;
 SbVec3f  directionZ(0.0, 0.0, 1.0), newDirectionZ;

 rot0.getValue(mat0);              // 現在の方向
 mat0.multVecMatrix(directionX, newDirectionX);
 mat0.multVecMatrix(directionZ, newDirectionZ);

#if 0
 printf("New Direction: (%f,%f,%f)\n",
       -newDirectionZ[0], -newDirectionZ[1],
       -newDirectionZ[2]);
#endif

 camera->position.setValue(cameraPosition[0] + newDirectionX[0] * x * 1.0
                           + newDirectionZ[0] * z * 1.0,
                 cameraPosition[1],
                 cameraPosition[2] + newDirectionX[2] * x * 1.0
                           + newDirectionZ[2] * z * 1.0);
}
```



```
/******************************************************************************
 ** FUNCTION : ・changeCameraYPosition()
 **       : pfinder のイベントによりカメラの y 位置を変える
 ******************************************************************************/
void changeCameraYPosition(float y)
{
 SbVec3f cameraPosition = camera->position.getValue();

#if 1
 printf("Camera position: (%g,%g,%g)\n",
       cameraPosition[0], cameraPosition[1],
       cameraPosition[2]);
#endif

 camera->position.setValue(cameraPosition[0],
                 y,
                 cameraPosition[2]);
}


/******************************************************************************
 ** FUNCTION : ・changeCameraOrientation()
 **       : pfinder のイベントによりカメラの向きを変える
 ******************************************************************************/
void changeCameraOrientation(float y)
{
 // カメラの方向を取得
 SbRotation rot0 = camera->orientation.getValue();
 SbVec3f   axis(0.0, 1.0, 0.0);    // y 軸
 float     radians = 0.01745 * y;  // (PI/180)
 SbRotation rot1(axis, radians);

 // マトリクス演算
 SbMatrix mat0, mat1, matTrg;
 rot0.getValue(mat0);              // 現在の方向
 rot1.getValue(mat1);              // y 軸回転
 matTrg = mat1 * mat0;
 SbRotation rotTrg(matTrg);

 camera->orientation.setValue(rotTrg);
}
```

```
/*************************************************************************
 ** PROGRAM : Virtual NARA
 ** FILENAME : vnara.c++
 **        :
 ** DATE    : 1998.10.31   T.Takumi(CSK)
 *************************************************************************/

#define TAKMAIN

#include <string.h>
#include <unistd.h>

#include "vnara.h"

extern "C" {
#include "sock_define.h"
#include "sock_global.h"
#include "server.h"
}

#include "define.h"     // local definition
#include "global.h"
#include "prototype.h"

/*************************************************************************
 ** FUNCTION : ・グローバル変数の初期化
 **        : (主に Node Pointer)
 *************************************************************************/
void initializeGlobal(void)
{
 // 各ノード
 root = NULL;
 scene = NULL;

 selectionRoot = NULL;
 dataRoot = NULL;

 // Viewer
 myWalkViewer = NULL;
 camera = NULL;
 auxWalkViewer = NULL;
 auxCamera = NULL;

 // Callback
 sceneEventCB = NULL;

 // 各種センサ
 naviCameraSensor = NULL;
 commandSensor = NULL;
 processingCommand = FALSE;

 // メインループ用
 contLoop = TRUE;

 // 現在の(アニメーションの始点となる)位置
 curObjIdx = -1;        // Home Position
}
```



```
/*************************************************************************
 ** FUNCTION : ・メインルーチン
 **        :
 *************************************************************************/
void main(int argc, char **argv)
{
 //========== コマンドラインオプションの指定 ==========
 strcpy(map_file, DEFAULT_MAP_FILE);
 strcpy(map_file2, DEFAULT_MAP_FILE2);
 port_num = DEFAULT_PORT_NUMBER;
 strcpy(pf_hostname, DEFAULT_PF_HOSTNAME);
 exe_opt = 0;

 getcmdargs(argc, argv);

 printf("Map File   >> %s\n", map_file);
 printf("Map File2  >> %s\n", map_file2);
 printf("Port Number >> %d\n", port_num);
 if (exe_opt & OPT_PFINDER) {
  printf("Pfinder Host Name >> %s\n", pf_hostname);
 }
 printf("\n");

#ifdef SOCKET
 //========== socket(server) の初期化 ==========
 med_sock = init_server(port_num);
#endif /* SOCKET */

 //========== カレントディレクトリの保存 ==========
 curDir = getcwd(NULL, 128);
//printf("curDir --> %s\n", curDir);

 //========== pfinder の初期化 ==========
 if (exe_opt & OPT_PFINDER) {
  if (initializePfinder(pf_hostname) < 0) {
   exit(1);
  }
 }
 //========== グローバル変数の初期化 ==========
 initializeGlobal();

 //========== Inventor と Xt の初期化 ==========
 Widget walkWindow = SoXt::init(argv[0]);
 if (walkWindow == NULL) exit(1);

 //========== 固定カメラ位置のセット ==========
 setFixedCamera();

 //========== main データの読み込み ==========
 root = new SoSeparator();
 root->ref();

 scene = new SoSeparator();
 root->addChild(scene);

 camera = new SoPerspectiveCamera();       // カメラの設定
```

```
 // カメラ初期位置の変更
//  camera->position.setValue(5442.1, 3937.0, 18694.7);
  camera->position.setValue(62728.5, 3937.0, 73689.6);
  SbVec3f tmpAxis(0.0, 1.0, 0.0);
  camera->orientation.setValue(tmpAxis, -1.6);
  camera->nearDistance.setValue(0.1);
  camera->farDistance.setValue(90000.0);

  // selection Node
  selectionRoot = new SoSelection();
  selectionRoot->setPickFilterCallback(pickFilterCB);
  selectionRoot->addSelectionCallback(pickSelectionCB, NULL);

  scene->addChild(camera);
  scene->addChild(selectionRoot);

  selectionRoot->addChild(readFile(map_file));

  //========== Top View 用データの読み込み ==========
  auxScene = new SoSeparator();
  root->addChild(auxScene);

  auxCamera = new SoPerspectiveCamera();  // カメラの設定

  // カメラ初期位置の変更
//  auxCamera->position.setValue(22800.0, 48330.945312, 18418.666016);
  auxCamera->position.setValue(78575.359375, 46698.000000, 72791.289062);

  SbVec3f tmpAxis2(-1.0, 0.0, 0.0);
//  auxCamera->orientation.setValue(tmpAxis2, 1.483529);
  auxCamera->orientation.setValue(tmpAxis2, 1.570796);

//  auxCamera->nearDistance.setValue(0.1);
//  auxCamera->farDistance.setValue(90000.0);

  auxScene->addChild(auxCamera);
  auxScene->addChild(readFile(map_file2));

  //========== Node へのポインタをセット ==========
  /*** データ構造に依存 ***/
  dataRoot = (SoSeparator *)selectionRoot->getChild(0);
  printf("dataRoot has %d children.\n", dataRoot->getNumChildren());
  int i;
  for (i = 0; i < NUM_MAP; i++) {
    mapSeparator[i] = (SoSeparator *)dataRoot->getChild(i);
  }
  for (i = NUM_MAP; i < NUM_MAP+NUM_TEMPLE; i++) {
    int j = i-NUM_MAP;
    templeSeparator[j] = (SoSeparator *)dataRoot->getChild(i);
    templeTransform[j] = (SoTransform *)templeSeparator[j]->getChild(0);
  }
  murouSeparator = (SoSeparator *)dataRoot->getChild(i);
  atrSeparator = (SoSeparator *)dataRoot->getChild(i+1);
  atrTransform = (SoTransform *)atrSeparator->getChild(0);

  //========== Top View でのカメラの位置 ==========
  /*** データ構造に依存 ***/
  SoSeparator           *tmpSep;
```



```
  tmpSep = (SoSeparator *)auxScene->getChild(1);
  tmpSep = (SoSeparator *)tmpSep->getChild(0);
  cameraConeTransform = (SoTransform *)(tmpSep->getChild(0));

  //========== 地形データは UNPICKABLE にする ==========
  for (i = 0; i < NUM_MAP; i++) {
    SoPickStyle *style = new SoPickStyle;
    style->style = SoPickStyle::UNPICKABLE;
    mapSeparator[i]->insertChild(style, 0);
  }

  //========== 各種タイマセンサの初期化 ==========
  naviCameraSensor = new SoTimerSensor();
  naviCameraSensor->setFunction(naviCameraCB);

  commandSensor = new SoAlarmSensor();
  commandSensor->setFunction(execCommandCB);

  cameraPositionSensor = new SoFieldSensor(cameraPositionChangedCB, camera);
  cameraPositionSensor->attach(&camera->position);
  cameraOrientationSensor = new SoFieldSensor(cameraOrientationChangedCB, camera);
  cameraOrientationSensor->attach(&camera->orientation);


  //========== イベントコールバックノードの生成 ==========
  // for key press events
  sceneEventCB = new SoEventCallback;
  if (sceneEventCB == NULL){
    fprintf(stderr, "ERROR: Free memory exhausted! main...\n");
    return;
  }
  sceneEventCB->addEventCallback(SoKeyboardEvent::getClassTypeId(),
                                 myKeyPressCB);
  sceneEventCB->ref();
  scene->addChild(sceneEventCB);

  //========== 各 Viewer(ウインドウ)の設定 ==========
  // 探検(メイン)ウインドウ
  myWalkViewer = new SoXtWalkViewer(walkWindow);
  if (myWalkViewer == NULL){
    fprintf(stderr, "ERROR: Free memory exhausted! main 2...\n");
    return;
  }
  myWalkViewer->setTitle("Nara Map");

  // setSceneGraph はカメラの設定の後で実行する
  // (デフォルトのカメラが採用されてしまう)
  myWalkViewer->setSceneGraph(scene);
  myWalkViewer->setHeadlight(TRUE);
//  myWalkViewer->setDecoration(FALSE);
//  myWalkViewer->setPopupMenuEnabled(TRUE);
  myWalkViewer->setViewing(FALSE);
  myWalkViewer->setAutoClipping(FALSE);

  // 選択された場合の Action
  myWalkViewer->setGLRenderAction(new SoBoxHighlightRenderAction());
//  myWalkViewer->setGLRenderAction(new SoLineHighlightRenderAction());
```

```
// myWalkViewer->redrawOnSelectionChange(landRoot);
// myWalkViewer->redrawOverlayOnSelectionChange(landRoot);
// myWalkViewer->changeMaxStraightSpeed(FALSE);

 SbColor  backColor(0.0, 0.6, 1.0);
 myWalkViewer->setBackgroundColor(backColor);

 // BLEND の設定
// myWalkViewer->setTransparencyType(SoGLRenderAction::SORTED_OBJECT_BLEND);
// myWalkViewer->setTransparencyType(SoGLRenderAction::DELAYED_BLEND);
 myWalkViewer->setTransparencyType(SoGLRenderAction::BLEND);

 auxWalkViewer = new SoXtWalkViewer();
 auxWalkViewer->setTitle("Nara Map");

 // setSceneGraph はカメラの設定の後で実行する
 // (デフォルトのカメラが採用されてしまう)
 auxWalkViewer->setSceneGraph(auxScene);
 auxWalkViewer->setHeadlight(TRUE);
 auxWalkViewer->setDecoration(FALSE);
 auxWalkViewer->setPopupMenuEnabled(FALSE);
 auxWalkViewer->setViewing(FALSE);
// auxWalkViewer->setAutoClipping(FALSE);

 // 選択された場合の Action
 auxWalkViewer->setGLRenderAction(new SoBoxHighlightRenderAction());
// auxWalkViewer->setGLRenderAction(new SoLineHighlightRenderAction());


 SoXt::show(walkWindow);
 myWalkViewer->show();
 auxWalkViewer->show();

 //========== Window 位置の設定 ==========
 XMoveResizeWindow(XtDisplay(XtParent(myWalkViewer->getWidget())),
                   XtWindow(XtParent(myWalkViewer->getWidget())),
                   MAINW_POS_X, MAINW_POS_Y, MAINW_WIDTH, MAINW_HEIGHT);
 XMoveResizeWindow(XtDisplay(XtParent(auxWalkViewer->getWidget())),
                   XtWindow(XtParent(auxWalkViewer->getWidget())),
                   AUXW_POS_X, AUXW_POS_Y, AUXW_WIDTH, AUXW_HEIGHT);

 //========== Main Loop ==========
 myWalkViewer->resetToHomePosition();
 myLoop();
}


/*****************************************************************
 ** FUNCTION : ・メインループ
 **       :
 *****************************************************************/
void myLoop(void)
{
 XtInputMask m;

 while (contLoop){

#ifdef SOCKET
```



```
  // socket のポーリング
  int ret = socket_check(med_sock);
  if (ret) {
   read_command(med_sock, ret);
  }
#endif /* SOCKET */

  // pfinder イベントの処理
  if (!(commandSensor->isScheduled())) {
   if (exe_opt & OPT_PFINDER) {
          checkPfinderEvent();
   }
  }

  // Xt, Inventor イベントの処理
  while (m = XtAppPending(SoXt::getAppContext())) {
   XtAppProcessEvent(SoXt::getAppContext(), m);
  }

 } /* while */

#ifdef SOCKET
 // socket のクローズ
 close_server(med_sock);
#endif /* SOCKET */
}


/*****************************************************************
 ** FUNCTION : ・Key Press コールバック関数
 **       :
 *****************************************************************/
void myKeyPressCB(void *, SoEventCallback *eventCB)
{
 const SoEvent *event = eventCB->getEvent();

 /** アルファベットキー **/
 // ナビゲーションを中断
 if (SO_KEY_PRESS_EVENT(event, Q)) {
  naviCameraSensor->unschedule();
  commandSensor->unschedule();
  processingCommand = FALSE;
 }

 /** Function キー **/
 // カメラ位置出力
 else if (SO_KEY_PRESS_EVENT(event, F1)) {
  SbVec3f cameraPosition = camera->position.getValue();
  SbRotation cameraOrientation = camera->orientation.getValue();

  SbVec3f axis;
  float  radians;

  cameraOrientation.getValue(axis, radians);
  printf("%f,      %f,      %f,      %f,      %f,      %f,      %f\n",
         cameraPosition[0], cameraPosition[1], cameraPosition[2],
         axis[0], axis[1], axis[2], radians);
 }
```

```
// aux カメラ位置出力
else if (SO_KEY_PRESS_EVENT(event, F2)) {
 SbVec3f cameraPosition = auxCamera->position.getValue();
 SbRotation cameraOrientation = auxCamera->orientation.getValue();

 SbVec3f axis;
 float  radians;

 cameraOrientation.getValue(axis, radians);
 printf("%f,        %f,        %f,        %f,        %f,        %f,        %f\n",
        cameraPosition[0], cameraPosition[1], cameraPosition[2],
        axis[0], axis[1], axis[2], radians);
}
// 左側 pick
else if (SO_KEY_PRESS_EVENT(event, F5)) {
 unselectNode();
 pickLeftSide();
}
// 右側 pick
else if (SO_KEY_PRESS_EVENT(event, F6)) {
 unselectNode();
 pickRightSide();
}
// 下側 pick
else if (SO_KEY_PRESS_EVENT(event, F7)) {
 unselectNode();
 pickLowerAnyway();
}
// オブジェクトの選択解除
else if (SO_KEY_PRESS_EVENT(event, F8)) {
 unselectNode();
}

/** 数字キー-- **/
// アニメーションテスト(1 ～ 9 の数字キー)
else if (SO_KEY_PRESS_EVENT(event, NUMBER_1)) {
 if (!(naviCameraSensor->isScheduled())) {

  curComIdx = 0;
  comCnt = 1;
  strcpy(command[curComIdx], "109");
  if (curObjIdx > 0) {
   setAnimationPath(curObjIdx, 1);
  }
  else {
   setAnimationPath(1, 1);
  }
  curObjIdx = 1;

  naviCnt = 0;
  naviCameraSensor->setInterval((double)TRACK_INTERVAL);
  naviCameraSensor->schedule();
 }
}

else if (SO_KEY_PRESS_EVENT(event, NUMBER_2)) {
 if (!(naviCameraSensor->isScheduled())) {
```



```
  curComIdx = 0;
  comCnt = 1;
  strcpy(command[curComIdx], "201");
  if (curObjIdx > 0) {
   setAnimationPath(curObjIdx, 2);
  }
  else {
   setAnimationPath(2, 2);
  }
  curObjIdx = 2;

  naviCnt = 0;
  naviCameraSensor->setInterval((double)TRACK_INTERVAL);
  naviCameraSensor->schedule();
 }
}

else if (SO_KEY_PRESS_EVENT(event, NUMBER_3)) {
 if (!(naviCameraSensor->isScheduled())) {

  curComIdx = 0;
  comCnt = 1;
  strcpy(command[curComIdx], "302");
  if (curObjIdx > 0) {
   setAnimationPath(curObjIdx, 3);
  }
  else {
   setAnimationPath(3, 3);
  }
  curObjIdx = 3;

  naviCnt = 0;
  naviCameraSensor->setInterval((double)TRACK_INTERVAL);
  naviCameraSensor->schedule();
 }
}

else if (SO_KEY_PRESS_EVENT(event, NUMBER_4)) {
 if (!(naviCameraSensor->isScheduled())) {

  curComIdx = 0;
  comCnt = 1;
  strcpy(command[curComIdx], "403");
  if (curObjIdx > 0) {
   setAnimationPath(curObjIdx, 4);
  }
  else {
   setAnimationPath(4, 4);
  }
  curObjIdx = 4;

  naviCnt = 0;
  naviCameraSensor->setInterval((double)TRACK_INTERVAL);
  naviCameraSensor->schedule();
 }
}

else if (SO_KEY_PRESS_EVENT(event, NUMBER_5)) {
```

```
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "504");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 5);
    }
    else {
      setAnimationPath(5, 5);
    }
    curObjIdx = 5;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
 }

 else if (SO_KEY_PRESS_EVENT(event, NUMBER_6)) {
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "605");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 6);
    }
    else {
      setAnimationPath(6, 6);
    }
    curObjIdx = 6;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
 }

 else if (SO_KEY_PRESS_EVENT(event, NUMBER_7)) {
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "711");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 7);
    }
    else {
      setAnimationPath(7, 7);
    }
    curObjIdx = 7;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
 }
```



```
 else if (SO_KEY_PRESS_EVENT(event, NUMBER_8)) {
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "807");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 8);
    }
    else {
      setAnimationPath(8, 8);
    }
    curObjIdx = 8;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
 }

 else if (SO_KEY_PRESS_EVENT(event, NUMBER_9)) {
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "913");
    if (curObjIdx > 0) {                    // 室生寺へ
      setAnimationPath(curObjIdx, 9);
    }
    else {
      setAnimationPath(9, 9);
    }
    curObjIdx = 9;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
 }

 eventCB->setHandled();
}


/*************************************************************************
 ** FUNCTION : ・read_command();
 **          : socket を通じてコマンドを読み込む
 *************************************************************************/
void read_command(int snum, int ret)
{
 static int state = CLIENT_CLOSE;

// printf("[COMMAND: %d]\n", ret);
 switch (ret) {
 case CLIENT_READDATA:
  if (state == CLIENT_CLOSE) {
    /* バッファの初期化 */
```

```
    bzero(tmpBuf, BUF_SIZE);
   }
   strcat(tmpBuf, sock_buf[snum]);
   state = CLIENT_READDATA;
   break;
  case CLIENT_CLOSE:
//   printf("[COMMAND: %s]\n", tmpBuf);
   state = CLIENT_CLOSE;
   if (!(processingCommand)) {
    set_sort_command(tmpBuf);
   }
   else {
    printf("WARNING: Now processing previous commands.");
    printf("      : The commands send now are ignored.");
   }
   break;
  }
}


/*************************************************************************
 ** FUNCTION : ・send_command();
 **       : socket を通じてコマンドを送る
 *************************************************************************/
void send_command(int snum, char *cmd, char *arg)
{
 char       mes[BUF_SIZE];
 int        i;

 sprintf(mes, "%s %s", cmd, arg);
 for (i = 0; i < MAX_CLIENT; i++) {
  if (client_sock[snum][i] > 2) {
   if (write(client_sock[snum][i], mes, strlen(mes)+1) == -1) {
           perror("write");
    }
   }
  }
}


/*************************************************************************
 ** FUNCTION : ・set_sort_command()
 **       : 各コマンドに分け、昇順にソートする。
 **       : 各コマンドは 3 桁の数値 + ' '
 *************************************************************************/
void set_sort_command(char *buf)
{
 char       work[COMMAND_LENGTH];
 char       *p0 = buf, *p1;

 // コマンドのセット
 comCnt = 0;
 while (p1 = strchr(p0, ' ')) {
  *p1 = '\0';
  strcpy(command[comCnt], p0);
  comCnt++;
  p0 = p1 + 1;
 }
```



```
 // 昇順にソート
 for (int i = 0; i < comCnt-1; i++) {
  for(int j = i+1; j < comCnt; j++) {
   if (strcmp(command[i], command[j]) > 0){
    strcpy(work, command[i]);
    strcpy(command[i], command[j]);
    strcpy(command[j], work);
   }
  }
 }

 // テスト表示
 for (int k = 0; k < comCnt; k++) {
  printf("taktest>> %s\n", command[k]);
 }

 // コマンドを実行
 if (comCnt > 0) {
  // 現在位置を Home Position へ
  myWalkViewer->resetToHomePosition();
  curObjIdx = -1;

  processingCommand = TRUE;
  curComIdx = 0;
  commandSensor->setTimeFromNow((double)0.0);
  commandSensor->schedule();
 }
}


/*************************************************************************
 ** FUNCTION : ・readFile()
 **       : .iv 形式のファイルを読み込み Node へのポインタを返す
 *************************************************************************/
SoNode * readFile(const char *filename)
{
 // Open the input file
 SoInput mySceneInput;
 if (!mySceneInput.openFile(filename)) {
  fprintf(stderr,"WARNING: Cannot open file %s\n", filename);
  SoSeparator           *emptyNode = new SoSeparator();
  return emptyNode;
 }

 // Read the whole file into the database
 SoSeparator *myGraph = SoDB::readAll(&mySceneInput);
 if (myGraph == NULL) {
  fprintf(stderr,"WARNING: Wrong file format\n");
  SoSeparator           *emptyNode = new SoSeparator();
  return emptyNode;
 }

 mySceneInput.closeFile();
 return myGraph;
}
```

```
/*****************************************************************
 ** FUNCTION : ・pickFilterCB()
 **        : シーンがピックされた時のフィルタ
 **        : SoSelection から 4 node 分を pickSelectionCB 以降に渡す
 *****************************************************************/
SoPath *
pickFilterCB(void *, const SoPickedPoint *pick)
{
 // ピックされたノードへのパスを取得
 SoPath *p = pick->getPath();

 printf("picked !! ---> %d\n", p->getLength());

 int i;
 for (i = 0; i < p->getLength() - 2; i++) {
  SoNode *n = p->getNode(i);
  if (n->isOfType(SoSelection::getClassTypeId()))
   break;
 }

 // SoSelection ノードをトップとし、4ノード分をコピー
 return p->copy(i, 5);
}


/*****************************************************************
 ** FUNCTION : ・pickSelectionCB()
 **        : マウスのピックイベントにより呼び出される
 *****************************************************************/
void pickSelectionCB(void *, SoPath *selectionPath)
{
 char        name[256];

#if 0
  for (int i = 0; i < 5; i++) {
   SoSeparator *selectedNode = (SoSeparator *)selectionPath->getNode(i);
   printf("%d: The name of selected node is [%s].\n",
                        i, selectedNode->getName().getString());
  }
#endif

  myWalkViewer->render();

  SoSeparator *selectedNode = (SoSeparator *)selectionPath->getNode(4);
  strcpy(name, selectedNode->getName().getString());
  printf("The name of selected node is [%s].\n", name);

  if (name[0] == '_') {
   callNetscape(&name[3]);
  }
  else {
   callNetscape(name);
  }

  auxWalkViewer->render();
 }
```



```
/*****************************************************************
 ** FUNCTION : ・Parse the command line args
 **        :
 *****************************************************************/
void  getcmdargs(int argc, char **argv)
{
 char *cmdargs = "M:m:s:p:h";
 char *usage_m =
  "usage: %s %s\n"
  "          -M <map file>        : マップファイルを指定\n"
  "          -m <map file2>       : 鳥観図用マップファイルを指定\n"
  "          -s <soket port number> : 仲介プログラムとのソケットのポートを指定\n"
  "          -p <RPC host name>   : pfinder のホスト名を指定\n"
  "          -h                   : ヘルプを表示\n";

 int          c;

 while ((c = getopt(argc, argv, cmdargs)) != EOF) {
  switch (c) {

  case 'M':
   strcpy(map_file, optarg);
   break;

  case 'm':
   strcpy(map_file2, optarg);
   break;

  case 's':
   port_num = atoi(optarg);
   break;

  case 'p':
   exe_opt |= OPT_PFINDER;
   strcpy(pf_hostname, optarg);
   break;

  case 'h':
  default:
   printf(usage_m, argv[0], cmdargs);
   exit(0);
   break;
  }
 }
}



// End of File
```

```
/*************************************************************************
 ** PROGRAM  : Virtual NARA
 ** FILENAME : vnara.c++
 **          :
 ** DATE     : 1998.10.31   T.Takumi(CSK)
 *************************************************************************/

#define TAKMAIN

#include <string.h>
#include <unistd.h>

#include "vnara.h"

extern "C" {
#include "sock_define.h"
#include "sock_global.h"
#include "server.h"
}

#include "define.h"     // local definition
#include "global.h"
#include "prototype.h"

/*************************************************************************
 ** FUNCTION : ・グローバル変数の初期化
 **          : (主に Node Pointer)
 *************************************************************************/
void initializeGlobal(void)
{
  // 各ノード
  root = NULL;
  scene = NULL;

  selectionRoot = NULL;
  dataRoot = NULL;

  // Viewer
  myWalkViewer = NULL;
  camera = NULL;
  auxWalkViewer = NULL;
  auxCamera = NULL;

  // Callback
  sceneEventCB = NULL;

  // 各種センサ
  naviCameraSensor = NULL;
  commandSensor = NULL;
  processingCommand = FALSE;

  // メインループ用
  contLoop = TRUE;

  // 現在の(アニメーションの始点となる)位置
  curObjIdx = -1;         // Home Position
}
```



```
/*************************************************************************
 ** FUNCTION : ・メインルーチン
 **          :
 *************************************************************************/
void main(int argc, char **argv)
{
  //========== コマンドラインオプションの指定 ==========
  strcpy(map_file, DEFAULT_MAP_FILE);
  strcpy(map_file2, DEFAULT_MAP_FILE2);
  port_num = DEFAULT_PORT_NUMBER;
  strcpy(pf_hostname, DEFAULT_PF_HOSTNAME);
  exe_opt = 0;

  getcmdargs(argc, argv);

  printf("Map File   >> %s\n", map_file);
  printf("Map File2  >> %s\n", map_file2);
  printf("Port Number >> %d\n", port_num);
  if (exe_opt & OPT_PFINDER) {
    printf("Pfinder Host Name >> %s\n", pf_hostname);
  }
  printf("\n");

#ifdef SOCKET
  //========== socket(server) の初期化 ==========
  med_sock = init_server(port_num);
#endif /* SOCKET */

  //========== カレントディレクトリの保存 ==========
  curDir = getcwd(NULL, 128);
//printf("curDir --> %s\n", curDir);

  //========== pfinder の初期化 ==========
  if (exe_opt & OPT_PFINDER) {
    if (initializePfinder(pf_hostname) < 0) {
      exit(1);
    }
  }
  //========== グローバル変数の初期化 ==========
  initializeGlobal();

  //========== Inventor と Xt の初期化 ==========
  Widget walkWindow = SoXt::init(argv[0]);
  if (walkWindow == NULL) exit(1);

  //========== 固定カメラ位置のセット ==========
  setFixedCamera();

  //========== main データの読み込み ==========
  root = new SoSeparator();
  root->ref();

  scene = new SoSeparator();
  root->addChild(scene);

  camera = new SoPerspectiveCamera();       // カメラの設定
```

```
 // カメラ初期位置の変更
//  camera->position.setValue(5442.1, 3937.0, 18694.7);
  camera->position.setValue(62728.5, 3937.0, 73689.6);
  SbVec3f tmpAxis(0.0, 1.0, 0.0);
  camera->orientation.setValue(tmpAxis, -1.6);
  camera->nearDistance.setValue(0.1);
  camera->farDistance.setValue(90000.0);

  // selection Node
  selectionRoot = new SoSelection();
  selectionRoot->setPickFilterCallback(pickFilterCB);
  selectionRoot->addSelectionCallback(pickSelectionCB, NULL);

  scene->addChild(camera);
  scene->addChild(selectionRoot);

  selectionRoot->addChild(readFile(map_file));

  //========== Top View 用データの読み込み ==========
  auxScene = new SoSeparator();
  root->addChild(auxScene);

  auxCamera = new SoPerspectiveCamera();  // カメラの設定

  // カメラ初期位置の変更
//  auxCamera->position.setValue(22800.0, 48330.945312, 18418.666016);
  auxCamera->position.setValue(78575.359375, 46698.000000, 72791.289062);

  SbVec3f tmpAxis2(-1.0, 0.0, 0.0);
//  auxCamera->orientation.setValue(tmpAxis2, 1.483529);
  auxCamera->orientation.setValue(tmpAxis2, 1.570796);

//  auxCamera->nearDistance.setValue(0.1);
//  auxCamera->farDistance.setValue(90000.0);

  auxScene->addChild(auxCamera);
  auxScene->addChild(readFile(map_file2));

  //========== Node へのポインタをセット ==========
  /*** データ構造に依存 ***/
  dataRoot = (SoSeparator *)selectionRoot->getChild(0);
  printf("dataRoot has %d children.\n", dataRoot->getNumChildren());
  int i;
  for (i = 0; i < NUM_MAP; i++) {
    mapSeparator[i] = (SoSeparator *)dataRoot->getChild(i);
  }
  for (i = NUM_MAP; i < NUM_MAP+NUM_TEMPLE; i++) {
    int j = i-NUM_MAP;
    templeSeparator[j] = (SoSeparator *)dataRoot->getChild(i);
    templeTransform[j] = (SoTransform *)templeSeparator[j]->getChild(0);
  }
  murouSeparator = (SoSeparator *)dataRoot->getChild(i);
  atrSeparator = (SoSeparator *)dataRoot->getChild(i+1);
  atrTransform = (SoTransform *)atrSeparator->getChild(0);

  //========== Top View でのカメラの位置 ==========
  /*** データ構造に依存 ***/
  SoSeparator             *tmpSep;
```



```
  tmpSep = (SoSeparator *)auxScene->getChild(1);
  tmpSep = (SoSeparator *)tmpSep->getChild(0);
  cameraConeTransform = (SoTransform *)(tmpSep->getChild(0));

  //========== 地形データは UNPICKABLE にする ==========
  for (i = 0; i < NUM_MAP; i++) {
    SoPickStyle *style = new SoPickStyle;
    style->style = SoPickStyle::UNPICKABLE;
    mapSeparator[i]->insertChild(style, 0);
  }

  //========== 各種タイマセンサの初期化 ==========
  naviCameraSensor = new SoTimerSensor();
  naviCameraSensor->setFunction(naviCameraCB);

  commandSensor = new SoAlarmSensor();
  commandSensor->setFunction(execCommandCB);

  cameraPositionSensor = new SoFieldSensor(cameraPositionChangedCB, camera);
  cameraPositionSensor->attach(&camera->position);
  cameraOrientationSensor = new SoFieldSensor(cameraOrientationChangedCB, camera);
  cameraOrientationSensor->attach(&camera->orientation);


  //========== イベントコールバックノードの生成 ==========
  // for key press events
  sceneEventCB = new SoEventCallback;
  if (sceneEventCB == NULL){
    fprintf(stderr, "ERROR: Free memory exhausted! main...\n");
    return;
  }
  sceneEventCB->addEventCallback(SoKeyboardEvent::getClassTypeId(),
                                 myKeyPressCB);
  sceneEventCB->ref();
  scene->addChild(sceneEventCB);

  //========== 各 Viewer(ウインドウ)の設定 ==========
  // 探検(メイン)ウインドウ
  myWalkViewer = new SoXtWalkViewer(walkWindow);
  if (myWalkViewer == NULL){
    fprintf(stderr, "ERROR: Free memory exhausted! main 2...\n");
    return;
  }
  myWalkViewer->setTitle("Nara Map");

  // setSceneGraph はカメラの設定の後で実行する
  // (デフォルトのカメラが採用されてしまう)
  myWalkViewer->setSceneGraph(scene);
  myWalkViewer->setHeadlight(TRUE);
//  myWalkViewer->setDecoration(FALSE);
//  myWalkViewer->setPopupMenuEnabled(TRUE);
  myWalkViewer->setViewing(FALSE);
  myWalkViewer->setAutoClipping(FALSE);

  // 選択された場合の Action
  myWalkViewer->setGLRenderAction(new SoBoxHighlightRenderAction());
//  myWalkViewer->setGLRenderAction(new SoLineHighlightRenderAction());
```

```
// myWalkViewer->redrawOnSelectionChange(landRoot);
// myWalkViewer->redrawOverlayOnSelectionChange(landRoot);
// myWalkViewer->changeMaxStraightSpeed(FALSE);

 SbColor  backColor(0.0, 0.6, 1.0);
 myWalkViewer->setBackgroundColor(backColor);

 // BLEND の設定
// myWalkViewer->setTransparencyType(SoGLRenderAction::SORTED_OBJECT_BLEND);
// myWalkViewer->setTransparencyType(SoGLRenderAction::DELAYED_BLEND);
 myWalkViewer->setTransparencyType(SoGLRenderAction::BLEND);

 auxWalkViewer = new SoXtWalkViewer();
 auxWalkViewer->setTitle("Nara Map");

 // setSceneGraph はカメラの設定の後で実行する
 // (デフォルトのカメラが採用されてしまう)
 auxWalkViewer->setSceneGraph(auxScene);
 auxWalkViewer->setHeadlight(TRUE);
 auxWalkViewer->setDecoration(FALSE);
 auxWalkViewer->setPopupMenuEnabled(FALSE);
 auxWalkViewer->setViewing(FALSE);
// auxWalkViewer->setAutoClipping(FALSE);

 // 選択された場合の Action
 auxWalkViewer->setGLRenderAction(new SoBoxHighlightRenderAction());
// auxWalkViewer->setGLRenderAction(new SoLineHighlightRenderAction());


 SoXt::show(walkWindow);
 myWalkViewer->show();
 auxWalkViewer->show();

 //========== Window 位置の設定 ==========
 XMoveResizeWindow(XtDisplay(XtParent(myWalkViewer->getWidget())),
                   XtWindow(XtParent(myWalkViewer->getWidget())),
                   MAINW_POS_X, MAINW_POS_Y, MAINW_WIDTH, MAINW_HEIGHT);
 XMoveResizeWindow(XtDisplay(XtParent(auxWalkViewer->getWidget())),
                   XtWindow(XtParent(auxWalkViewer->getWidget())),
                   AUXW_POS_X, AUXW_POS_Y, AUXW_WIDTH, AUXW_HEIGHT);

 //========== Main Loop ==========
 myWalkViewer->resetToHomePosition();
 myLoop();
}


/*********************************************************************
 ** FUNCTION : ・メインループ
 **       :
 *********************************************************************/
void myLoop(void)
{
 XtInputMask m;

 while (contLoop){

#ifdef SOCKET
```



```
  // socket のポーリング
  int ret = socket_check(med_sock);
  if (ret) {
   read_command(med_sock, ret);
  }
#endif /* SOCKET */

  // pfinder イベントの処理
  if (!(commandSensor->isScheduled())) {
   if (exe_opt & OPT_PFINDER) {
          checkPfinderEvent();
   }
  }

  // Xt, Inventor イベントの処理
  while (m = XtAppPending(SoXt::getAppContext())) {
   XtAppProcessEvent(SoXt::getAppContext(), m);
  }

 } /* while */

#ifdef SOCKET
 // socket のクローズ
 close_server(med_sock);
#endif /* SOCKET */
}


/*********************************************************************
 ** FUNCTION : ・Key Press コールバック関数
 **       :
 *********************************************************************/
void myKeyPressCB(void *, SoEventCallback *eventCB)
{
 const SoEvent *event = eventCB->getEvent();

 /** アルファベットキー **/
 // ナビゲーションを中断
 if (SO_KEY_PRESS_EVENT(event, Q)) {
  naviCameraSensor->unschedule();
  commandSensor->unschedule();
  processingCommand = FALSE;
 }

 /** Function キー **/
 // カメラ位置出力
 else if (SO_KEY_PRESS_EVENT(event, F1)) {
  SbVec3f cameraPosition = camera->position.getValue();
  SbRotation cameraOrientation = camera->orientation.getValue();

  SbVec3f axis;
  float  radians;

  cameraOrientation.getValue(axis, radians);
  printf("%f,      %f,      %f,      %f,      %f,      %f,      %f\n",
         cameraPosition[0], cameraPosition[1], cameraPosition[2],
         axis[0], axis[1], axis[2], radians);
 }
```

```
// aux カメラ位置出力
else if (SO_KEY_PRESS_EVENT(event, F2)) {
  SbVec3f cameraPosition = auxCamera->position.getValue();
  SbRotation cameraOrientation = auxCamera->orientation.getValue();

  SbVec3f axis;
  float  radians;

  cameraOrientation.getValue(axis, radians);
  printf("%f,        %f,        %f,        %f,        %f,        %f\n",
          cameraPosition[0], cameraPosition[1], cameraPosition[2],
          axis[0], axis[1], axis[2], radians);
}
// 左側 pick
else if (SO_KEY_PRESS_EVENT(event, F5)) {
  unselectNode();
  pickLeftSide();
}
// 右側 pick
else if (SO_KEY_PRESS_EVENT(event, F6)) {
  unselectNode();
  pickRightSide();
}
// 下側 pick
else if (SO_KEY_PRESS_EVENT(event, F7)) {
  unselectNode();
  pickLowerAnyway();
}
// オブジェクトの選択解除
else if (SO_KEY_PRESS_EVENT(event, F8)) {
  unselectNode();
}

/** 数字キー **/
// アニメーションテスト(１～９の数字キー)
else if (SO_KEY_PRESS_EVENT(event, NUMBER_1)) {
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "109");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 1);
    }
    else {
      setAnimationPath(1, 1);
    }
    curObjIdx = 1;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
}

else if (SO_KEY_PRESS_EVENT(event, NUMBER_2)) {
  if (!(naviCameraSensor->isScheduled())) {
```



```
    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "201");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 2);
    }
    else {
      setAnimationPath(2, 2);
    }
    curObjIdx = 2;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
}

else if (SO_KEY_PRESS_EVENT(event, NUMBER_3)) {
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "302");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 3);
    }
    else {
      setAnimationPath(3, 3);
    }
    curObjIdx = 3;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
}

else if (SO_KEY_PRESS_EVENT(event, NUMBER_4)) {
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "403");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 4);
    }
    else {
      setAnimationPath(4, 4);
    }
    curObjIdx = 4;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
}

else if (SO_KEY_PRESS_EVENT(event, NUMBER_5)) {
```

```
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "504");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 5);
    }
    else {
      setAnimationPath(5, 5);
    }
    curObjIdx = 5;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
 }

 else if (SO_KEY_PRESS_EVENT(event, NUMBER_6)) {
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "605");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 6);
    }
    else {
      setAnimationPath(6, 6);
    }
    curObjIdx = 6;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
 }

 else if (SO_KEY_PRESS_EVENT(event, NUMBER_7)) {
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "711");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 7);
    }
    else {
      setAnimationPath(7, 7);
    }
    curObjIdx = 7;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
 }
```



```
 else if (SO_KEY_PRESS_EVENT(event, NUMBER_8)) {
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "807");
    if (curObjIdx > 0) {
      setAnimationPath(curObjIdx, 8);
    }
    else {
      setAnimationPath(8, 8);
    }
    curObjIdx = 8;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
 }

 else if (SO_KEY_PRESS_EVENT(event, NUMBER_9)) {
  if (!(naviCameraSensor->isScheduled())) {

    curComIdx = 0;
    comCnt = 1;
    strcpy(command[curComIdx], "913");
    if (curObjIdx > 0) {                    // 室生寺へ
      setAnimationPath(curObjIdx, 9);
    }
    else {
      setAnimationPath(9, 9);
    }
    curObjIdx = 9;

    naviCnt = 0;
    naviCameraSensor->setInterval((double)TRACK_INTERVAL);
    naviCameraSensor->schedule();
  }
 }

 eventCB->setHandled();
}


/*************************************************************************
 ** FUNCTION : ・read_command();
 **          : socket を通じてコマンドを読み込む
 *************************************************************************/
void read_command(int snum, int ret)
{
 static int state = CLIENT_CLOSE;

// printf("[COMMAND: %d]\n", ret);
 switch (ret) {
 case CLIENT_READDATA:
  if (state == CLIENT_CLOSE) {
   /* バッファの初期化 */
```

```
    bzero(tmpBuf, BUF_SIZE);
   }
   strcat(tmpBuf, sock_buf[snum]);
   state = CLIENT_READDATA;
   break;
  case CLIENT_CLOSE:
   printf("[COMMAND: %s]\n", tmpBuf);
   state = CLIENT_CLOSE;
   if (!(processingCommand)) {
    set_sort_command(tmpBuf);
   }
   else {
    printf("WARNING: Now processing previous commands.");
    printf("      : The commands send now are ignored.");
   }
   break;
  }
}


/*************************************************************************
 ** FUNCTION : ・send_command();
 **       : socket を通じてコマンドを送る
 *************************************************************************/
void send_command(int snum, char *cmd, char *arg)
{
 char       mes[BUF_SIZE];
 int        i;

 sprintf(mes, "%s %s", cmd, arg);
 for (i = 0; i < MAX_CLIENT; i++) {
  if (client_sock[snum][i] > 2) {
   if (write(client_sock[snum][i], mes, strlen(mes)+1) == -1) {
            perror("write");
    }
   }
  }
}


/*************************************************************************
 ** FUNCTION : ・set_sort_command()
 **       : 各コマンドに分け、昇順にソートする。
 **       : 各コマンドは 3 桁の数値 + ' '
 *************************************************************************/
void set_sort_command(char *buf)
{
 char       work[COMMAND_LENGTH];
 char       *p0 = buf, *p1;

 // コマンドのセット
 comCnt = 0;
 while (p1 = strchr(p0, ' ')) {
  *p1 = '\0';
  if (strlen(p0) == 3) {
   // 3桁の数字なら
   strcpy(command[comCnt], p0);
   comCnt++;
```



```
  }
  p0 = p1 + 1;
 }

 // 昇順にソート
 for (int i = 0; i < comCnt-1; i++) {
  for(int j = i+1; j < comCnt; j++) {
   if (strcmp(command[i], command[j]) > 0){
    strcpy(work, command[i]);
    strcpy(command[i], command[j]);
    strcpy(command[j], work);
   }
  }
 }

 // テスト表示
 for (int k = 0; k < comCnt; k++) {
  printf("taktest>> %s\n", command[k]);
 }

 // コマンドを実行
 if (comCnt > 0) {
  // 現在位置を Home Position へ
  myWalkViewer->resetToHomePosition();
  curObjIdx = -1;

  processingCommand = TRUE;
  curComIdx = 0;
  commandSensor->setTimeFromNow((double)0.0);
  commandSensor->schedule();
 }
}


/*************************************************************************
 ** FUNCTION : ・readFile()
 **       : .iv 形式のファイルを読み込み Node へのポインタを返す
 *************************************************************************/
SoNode * readFile(const char *filename)
{
 // Open the input file
 SoInput mySceneInput;
 if (!mySceneInput.openFile(filename)) {
  fprintf(stderr,"WARNING: Cannot open file %s\n", filename);
  SoSeparator           *emptyNode = new SoSeparator();
  return emptyNode;
 }

 // Read the whole file into the database
 SoSeparator *myGraph = SoDB::readAll(&mySceneInput);
 if (myGraph == NULL) {
  fprintf(stderr,"WARNING: Wrong file format\n");
  SoSeparator           *emptyNode = new SoSeparator();
  return emptyNode;
 }

 mySceneInput.closeFile();
 return myGraph;
```

```
}

/*****************************************************************
 ** FUNCTION : ・pickFilterCB()
 **       : シーンがピックされた時のフィルタ
 **       : SoSelection から 4 node 分を pickSelectionCB 以降に渡す
 *****************************************************************/
SoPath *
pickFilterCB(void *, const SoPickedPoint *pick)
{
 // ピックされたノードへのパスを取得
 SoPath *p = pick->getPath();

 printf("picked !! ---> %d\n", p->getLength());

 int i;
 for (i = 0; i < p->getLength() - 2; i++) {
  SoNode *n = p->getNode(i);
  if (n->isOfType(SoSelection::getClassTypeId()))
   break;
 }

 // SoSelection ノードをトップとし、4ノード分をコピー
 return p->copy(i, 5);
}


/*****************************************************************
 ** FUNCTION : ・pickSelectionCB()
 **       : マウスのピックイベントにより呼び出される
 *****************************************************************/
void pickSelectionCB(void *, SoPath *selectionPath)
{
 char       name[256];

#if 0
 for (int i = 0; i < 5; i++) {
  SoSeparator *selectedNode = (SoSeparator *)selectionPath->getNode(i);
  printf("%d: The name of selected node is [%s].\n",
                  i, selectedNode->getName().getString());
 }
#endif

 myWalkViewer->render();

 SoSeparator *selectedNode = (SoSeparator *)selectionPath->getNode(4);
 strcpy(name, selectedNode->getName().getString());
 printf("The name of selected node is [%s].\n", name);

 if (name[0] == '_') {
  callNetscape(&name[3]);
 }
 else {
  callNetscape(name);
 }

 auxWalkViewer->render();
```



```
}

/*****************************************************************
 ** FUNCTION : ・Parse the command line args
 **       :
 *****************************************************************/
void  getcmdargs(int argc, char **argv)
{
 char *cmdargs = "M:m:s:p:h";
 char *usage_m =
  "usage: %s %s\n"
  "          -M <map file>        : マップファイルを指定\n"
  "          -m <map file2>       : 鳥観図用マップファイルを指定\n"
  "          -s <soket port number> : 仲介プログラムとのソケットのポートを指定\n"
  "          -p <RPC host name>   : pfinder のホスト名を指定\n"
  "          -h                   : ヘルプを表示\n";

 int         c;

 while ((c = getopt(argc, argv, cmdargs)) != EOF) {
  switch (c) {

  case 'M':
   strcpy(map_file, optarg);
   break;

  case 'm':
   strcpy(map_file2, optarg);
   break;

  case 's':
   port_num = atoi(optarg);
   break;

  case 'p':
   exe_opt |= OPT_PFINDER;
   strcpy(pf_hostname, optarg);
   break;

  case 'h':
  default:
   printf(usage_m, argv[0], cmdargs);
   exit(0);
   break;
  }
 }
}



// End of File
```

```
/************************************************************************
 ** PROGRAM  : Virtual NARA
 ** FILENAME : define.h
 **        :
 ** DATE     : 1998.10.31   T.Takumi(CSK)
 ************************************************************************/

/************************************************************************/
/* command line option */
#define DEFAULT_PORT_NUMBER     23456
#define DEFAULT_MAP_FILE   "data/nara_all.iv"
#define DEFAULT_MAP_FILE2  "data/mini_all.iv"
#define DEFAULT_PF_HOSTNAME     "miris75"

#define OPT_PFINDER         (1<<0)

/************************************************************************/
/* Node */
#define NUM_MAP             16
#define NUM_TEMPLE          8

/* Walk Viewer */
#define MAINW_POS_X         7
#define MAINW_POS_Y         331
#define MAINW_WIDTH         1266
#define MAINW_HEIGHT        686

/* Aux Viewer */
#define AUXW_POS_X          7
#define AUXW_POS_Y          31
#define AUXW_WIDTH          626
#define AUXW_HEIGHT         262

/* Camera */ /* temple + murou + atr */
#define NUM_CAMERA                  (NUM_TEMPLE+2)


/* Navigation 用 */
#define NAVI_TRK_MAX                700

#define TRACK_INTERVAL              0.1
#define NAVI_DIV            50

#define COMMAND_INTERVAL            3.0

/************************************************************************/
/* Socket 用 */
#define TMPBUF_SIZE                 2048
#define COMMAND_LENGTH              8
#define COMMAND_MAX                 128

/************************************************************************/
/* Netscape */
#define CALL_NS             "%s -raise -remote 'openURL\(file:%s/html/%s.html\)'"
#define CALL_NS2    "%s -geometry =640x300-0+0 file:%s/html/%s.html &"
#define CALL_NS_ATR         "%s -raise -remote 'openURL\(http://www.atr.co.jp/\)'"
#define CALL_NS_ATR2        "%s -geometry =640x300-0+0 http://www.atr.co.jp/ &"
```



```
/************************************************************************/
/* pfinder */
#define NEUTRAL_RADIUS      7.0             /* 5.0 */
#define MAX_RADIUS          30.0

#define PFHOME_X  0.0
#define PFHOME_Y  27.61
#define PFHOME_Z  62.21

#define DEF_SENS_POS        5
#define DEF_SENS_ROT        5

/* pfinder status */
#define PF_NOTHING              0

#define PF_EXIST            1
#define PF_NOT_EXIST            2

#define PF_STANDING             3
#define PF_SITTING          4
#define PF_BENDING_OVER         5

#define PF_EXTEND           6
#define PF_POINT            7
#define PF_KICK                 8

#define PF_HAND_MOVING          9
#define PF_THROW            10

#define PF_HOME                 11
#define PF_LOOKINGUP            12



/* End of File */
```

```
/************************************************************************
** PROGRAM : Virtual NARA
** FILENAME : global.h
**       :
** DATE    : 1998.10.31   T.Takumi(CSK)
************************************************************************/

#ifdef TAKMAIN
#define GLOBAL
#else
#define GLOBAL extern
#endif

/************************************************************************/
/* command line options */
GLOBAL  char                    map_file[NAME_BUF];
GLOBAL  char                    map_file2[NAME_BUF];
GLOBAL  int                     port_num;
GLOBAL  char                    pf_hostname[NAME_BUF];
GLOBAL  int                     exe_opt;


/************************************************************************/
/* scene data */
GLOBAL  SoSeparator             *root;
GLOBAL  SoSeparator             *scene;
GLOBAL  SoSeparator             *auxScene;

GLOBAL  SoSelection             *selectionRoot;
GLOBAL  SoSeparator             *dataRoot;

GLOBAL  SoSeparator             *mapSeparator[NUM_MAP];
GLOBAL  SoSeparator             *templeSeparator[NUM_TEMPLE];
GLOBAL  SoTransform             *templeTransform[NUM_TEMPLE];
GLOBAL  SoSeparator             *murouSeparator;
GLOBAL  SoSeparator             *atrSeparator;
GLOBAL  SoTransform             *atrTransform;

/* viewer, camera */
GLOBAL  SoXtWalkViewer          *myWalkViewer;
GLOBAL  SoPerspectiveCamera  *camera;

GLOBAL  SoXtWalkViewer          *auxWalkViewer;
GLOBAL  SoPerspectiveCamera  *auxCamera;
GLOBAL  SoTransform             *cameraConeTransform;

GLOBAL  CameraParamRec          fixedCamera[NUM_CAMERA+1][2];

GLOBAL  SoFieldSensor           *cameraPositionSensor;
GLOBAL  SoFieldSensor           *cameraOrientationSensor;

/* callback */
GLOBAL  SoEventCallback      *sceneEventCB;

/* navigation */
GLOBAL  SoTimerSensor           *naviCameraSensor;
GLOBAL  SoAlarmSensor           *commandSensor;
```



```
GLOBAL  SoSeparator             *selectionNode;
GLOBAL  SoPath                  *selectionPath;

/* */
GLOBAL  char                    *curDir;


/************************************************************************/
GLOBAL  CameraParamRec          trackNavi[NAVI_TRK_MAX];
GLOBAL  int                     numTrackNavi;
GLOBAL  int                     naviCnt;


/************************************************************************/
GLOBAL int                      contLoop;

GLOBAL int                      med_sock;

GLOBAL char                     tmpBuf[TMPBUF_SIZE];
GLOBAL char                     command[COMMAND_MAX][COMMAND_LENGTH];
GLOBAL int                      comCnt;
GLOBAL int                      curComIdx;
GLOBAL int                      processingCommand;

GLOBAL int                      curObjIdx;

/************************************************************************/
/* pfinder */
GLOBAL PfinderRec           pfRec;


/* End of File */
```

```
/*************************************************************************
 ** PROGRAM  : Virtual NARA
 ** FILENAME : prototype.h
 **        :
 ** DATE     : 1998.10.31   T.Takumi(CSK)
 *************************************************************************/

/* vnara.c++ */
void        myLoop(void);

SoNode      *readFile(const char *filename);
void        myKeyPressCB(void *, SoEventCallback *);

void        read_command(int, int);
void        set_sort_command(char *);

SoPath      *pickFilterCB(void *, const SoPickedPoint *);
void        pickSelectionCB(void *, SoPath *);

void        getcmdargs(int, char **);


/* navigation.c++ */
void        naviCameraCB(void *, SoSensor *);
void        setAnimationPath(int, int);
void        setAnimationPath2(CameraParamPtr, CameraParamPtr);
void        setAnimationPathBezier(CameraParamPtr, CameraParamPtr, CameraParamPtr, CameraParamPtr);
void        setAnimationPathBezier2(CameraParamPtr, CameraParamPtr, CameraParamPtr, CameraParamPtr, float);

void        execCommandCB(void *, SoSensor *);
void        checkNextCommand(void);
void        callNetscape(char *);

void        unselectNode(void);

void        pickIcon(char []);


/* camera.c++ */
void        setFixedCamera(void);
void        cameraPositionChangedCB(void *, SoSensor *);
void        cameraOrientationChangedCB(void *, SoSensor *);
void        changeTempleDirection(SbVec3f);

/* callpf.c++ */
int         initializePfinder(char *);
int         call_pfinder(void);

/* pf.c++ */
void        checkPfinderEvent(void);

SbBool      pickPoint(const SbViewportRegion &, const SbVec2s &);
void        pickRightSide(void);
void        pickLeftSide(void);
void        pickLower(void);
void        pickLowerAnyway(void);

SoPath      *myPickFilter(const SoPickedPoint *);
```



```
void        changeCameraXZPosition(float, float);
void        changeCameraYPosition(float);
void        changeCameraOrientation(float);

/* End of File */
```

```
/*********************************************************************
 ** PROGRAM  : socket server
 ** FILENAME : server.h
 **        :
 ** DATE     : 1997.10.23   T.Takumi(CSK)
 *********************************************************************/

int         init_server(int);
int         socket_check(int);
void        close_server(int);

/* End of File */
```



```
/*********************************************************************
 ** PROGRAM  :
 ** FILENAME : define.h
 **        :
 ** DATE     : 1997.10.22   T.Takumi(CSK)
 *********************************************************************/

#define     BUF_SIZE 1024
#define     MAX_CLIENT      5

#define CLR_FD              -1

#define MAX_SERVER          2

/*********************************************************************/
#define CLIENT_NOACTION     0
#define CLIENT_CONNECT      1
#define CLIENT_READDATA     2
#define CLIENT_CLOSE        3

/* End of File */
```

```
/*********************************************************************
 ** PROGRAM :
 ** FILENAME : global.h
 **        :
 ** DATE    : 1997.10.22   T.Takumi(CSK)
 *********************************************************************/

#ifndef GLOBAL

#ifdef TAKMAIN
#define GLOBAL
#else
#define GLOBAL      extern
#endif

#endif


GLOBAL int                  listen_sock[MAX_SERVER];
GLOBAL int                  client_sock[MAX_SERVER][MAX_CLIENT];

GLOBAL char                 sock_buf[MAX_SERVER][BUF_SIZE];


GLOBAL int                  initialize_chatr;
GLOBAL int                  chatr_not_ready;


/* End of File */
```



```
/*********************************************************************
 ** PROGRAM : Virtual NARA
 ** FILENAME : vnara.h
 **        :
 ** DATE    : 1998.10.31   T.Takumi(CSK)
 *********************************************************************/

/*
 * Includes needed fot OpenInventor
 */
#include <Inventor/Sb.h>
#include <Inventor/SbTime.h>
#include <Inventor/SbViewportRegion.h>
#include <Inventor/SoPickedPoint.h>
#include <Inventor/SoPath.h>
#include <Inventor/SoDB.h>
#include <Inventor/SoInput.h>
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/Xt/viewers/SoXtPlaneViewer.h>
#include <Inventor/Xt/viewers/SoXtWalkViewer.h>
#include <Inventor/Xt/SoXtRenderArea.h>
#include <Inventor/actions/SoBoxHighlightRenderAction.h>
#include <Inventor/actions/SoLineHighlightRenderAction.h>
#include <Inventor/actions/SoSearchAction.h>
#include <Inventor/actions/SoRayPickAction.h>
#include <Inventor/actions/SoWriteAction.h>
#include <Inventor/actions/SoSearchAction.h>
#include <Inventor/actions/SoGetMatrixAction.h>
#include <Inventor/events/SoMouseButtonEvent.h>
#include <Inventor/events/SoKeyboardEvent.h>
#include <Inventor/fields/SoSFVec2f.h>
#include <Inventor/fields/SoSFVec3f.h>
#include <Inventor/fields/SoSFVec4f.h>
#include <Inventor/misc/SoBase.h>
#include <Inventor/nodes/SoCone.h>
#include <Inventor/nodes/SoCube.h>
#include <Inventor/nodes/SoCylinder.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoEventCallback.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoPickStyle.h>
#include <Inventor/nodes/SoSelection.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoShape.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/nodes/SoTranslation.h>
#include <Inventor/nodes/SoRotationXYZ.h>
#include <Inventor/nodes/SoScale.h>
#include <Inventor/nodes/SoLevelOfDetail.h>
#include <Inventor/nodes/SoSwitch.h>
#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/sensors/SoTimerSensor.h>
#include <Inventor/sensors/SoAlarmSensor.h>
#include <Inventor/sensors/SoFieldSensor.h>
#include <Inventor/sensors/SoSensor.h>
```

```
/*****************************************************************************/
#define NAME_BUF  512

/*****************************************************************************/
typedef struct _CameraParam
{
 SbVec3f  position;
 SbVec3f  axis;
 float                radians;
} CameraParamRec, *CameraParamPtr;


typedef struct _Pfinder {

/*
 int                  state;
 int                  stateOld;
 float                periodBegin;
 float                periodEnd;
 float                animationBegin;
 float                animationEnd;
 float                duration;
 float                highest;
 float                lowest;
 float                newActualTime;
 int                  accepted;
 int                  trackHuman;
 int                  timeLimit;
 char                 databaseName[NAME_BUF];
 char                 fileName[NAME_BUF];
 char                 fileDir[NAME_BUF];
 int                  runmode;

 char                 trackFileName[NAME_BUF];
 char                 trackFileDir[NAME_BUF];
*/
 int                  pfExist;
 int                  pfPose;
 int                  pfHandL;
 int                  pfHandR;
 float                pfPosition;
 float                pfRotation;
 float                sensPos;
 float                sensRot;
 int                  calib;

} PfinderRec, *PfinderPtr;


/* End of File */
```