平成 24 年度卒業論文

論文題目

異なる戦略の群知能を組み合わせた クラウドファイルシステムの実装

神奈川大学 工学部 電子情報フロンティア学科
学籍番号　200902815
石田　克憲

指導担当者　木下宏揚　教授

目 次

図 目 次

第1章　序論

1.1　背景

インターネットが普及し，近年の社会ではクラウド・コンピューティング [1] が注目されている. 中でも世の中はデータセンターとサービスとしてのクラウドソフトウェア（SaaS）を組み合わせたビジネスモデルである. 我々はクラウドを多様で不確実にインターネット上に存在する情報リソースと人間の「関係性」，そしてその「ふるまい」とみなしている. そこで，情報リソースを PC 内のファイルとみなし，そのファイルのふるまいを群知能によって実現することによって，他者との連携により人間の創発的活動（学術的活動・ビジネス・地域ボランティアなど）を刺激・支援するシステムを研究の目的としている.

従来はファイルの「ふるまい」を記述するシステムはなかった. このようなシステムに対応するのがファイルシステム [4] である. ファイルシステムとは，コンピュータのリソースを操作するためのオペレーティングシステムが持つ機能の一つで，抽象データ型の集まりであり，ストレージ，階層構造，データの操作・アクセス・検索のために実装されたものであるため，ファイルのふるまいを表現できない. 私は，このふるまいの概念を Boid[5] の Separation・Alignment・Cohesion をベースとする群知能とみなす. また，Boid の他に Ant Colony Optimization や Particle Swarm Optimization などいくつかの群知能を使用する. そして，このファイルふるまいを Android を用いて実装することを試みた. Andoid とはスマートフォン・タブレット PC などのモバイルデバイス向けのオープンでフリーなソフトウェアプレットフォームであり，クラウドの端末としてとても重要なシステムである.

キュレーションという言葉がある. キュレーションとは，博物館や図書館の学芸員を「キュレータ（curator）」という. 「情報を収集し，選別し，意味付けを与えて，それをみんなと共有すること」という定義. 情報の洪水の中で，それぞれのユーザにとって必要なものをコンテクスト（意味の文脈）に沿ってピックアップし，整理し，新しい付加価値を与えることである. クラウド内の大量にあるファイルの種類を自分なりにキュレーションし,「ファイルの見える化」を行い，ユーザにとって見やすいファイルシステムを作ることがこの研究の目的である. また，ファイルそのものに書き込まれたソースではなく，より人間に近い「ファイルのふるまい」に着目するため，コンピュータの価値観に依存しすぎない. このため情報漏えい [6] やアクセス制御 [7] の問題の解決につながる手段にもなりうる.

1.2 現在の問題点

1.2.1 情報漏えい [6]

現在のファイルのソート及びファイルの管理は，ファイルに直接書き込まれた情報のみをソースとして行われる．ファイル情報のみに頼ったファイルの管理は，それらの情報がクラッキングで改ざんされることで，正しく行われなくなってしまう．また，物理的なハードの持ち出しでも情報の改ざんが行われ，正しくファイルの管理がされなくなる．ファイルのセキュリティは日々強固なものになっているが，セキュリティを解かれた後に改ざんされたデータを以前のデータのように扱うのは非常に難しい．これは，我々がファイル自体に書き込まれているデータを参照してファイル管理を行っているためである．現在大量のデータを扱うクラウドシステムではこのファイル管理の重要性が増してきている．ファイルに書き込まれた情報に依存しないファイルの管理方法を生み出すことが情報漏えいを防ぐ鍵の一つとしてあげられる．

1.2.2 アクセス制限 [7]

アクセス制限はコンピュータの中にある絶対的な価値観で行われる．そのため，例外などコンピュータは臨機応変に対応することができない．ファイルが有害でなくともコンピュータの条件に当てはまれば制限は行われ，また一度制限されたものは人為的に行わない限り，アクセス制限からは解除されることはない．アクセス制限は人間が決めた一つの価値で白黒をつけるためこのようなことが起こってしまう．情報漏えいの問題と同様に，一つの価値観に頼りすぎない，または価値観が統一されずに常に様々な価値観が同時に存在するような，ファイル管理を行う方法を作ることが重要と言える．

1.3 提案手法

ファイルに書き込まれている情報に依存せず，かつ管理システムが一つの価値観に頼りすぎないシステムを作ることが問題の解決に繋がると考えた．様々な群知能の要素を組み込み，新しいファイル管理やふるまいの群を作れるのではないかと考えた．いくつかの群知能やシステム，概念を組み込みながら，実用できる新しいファイルシステムを作り，このシステムにより似たファイルが近くに集まるため，他者との創発的活動を刺激することが可能である．

1.4 アウトライン

第 2 章では，クラウドや群知能や開発に必要なアンドロイドなどクラウドファイルシステムに必要な基礎知識について解説する．
第 3 章では，今回提案するクラウドファイルシステムの概要や，群知能の要素の組み込み方について提案する．
第 4 章では，java.Applet で行った boid シミュレーションについて述べる．
第 5 章では，本論文の結論を示す．

第2章　基礎知識

2.1　クラウド・コンピューティング

アメリカ国立標準技術研究所（National Institute of Standards and Technology：NIST）によってクラウド・コンピューティングの定義を定めている.
クラウドコンピューティングとは，コンフィグレーションが可能なコンピューティングリソース（ネットワーク／サーバ／ストレージ／アプリケーション／サービス）で構成され，分散されたコンピューティングリソースへのオンデマンドのネットワークアクセスを可能にする. 利便性の高いモデルのことである. そして，それらのリソースは，最小の管理手順もしくは，サービスプロバイダーとのやりとりにより，迅速に供給され，また，解消されるものとなる. このクラウドモデルは，5つの本質的な特徴と，3つのサービスモデル，そして4つのディプロイメントモデルで構成され，可用性を促進するものとなる.

2.1.1　本質的な特徴

- オンデマンドのセルフサービス

それぞれのサービスプロバイダーとの人的な対話に依存することなく，消費者は必要に応じて自動的かつ一方的に，サーバやネットワーク，ストレージの利用時間といった，コンピューティングの能力を供給する.

- 幅広いネットワーク・アクセス

このコンピューティング能力は，ネットワーク上で利用でき，また標準的なメカニズムを介してアクセスできる. それにより，異種のシン／シッククライアントプラットフォーム（携帯電話／ラップトップ／ PDA）からの利用が促進される.

- 資源の置き場所

プロバイダーのコンピューティングリソースは共有され，マルチテナントモデルを利用する多数の消費者に提供される. そこでは，消費者からの需要にしたがって動的に割当／解消される，物理的あるいは仮想的なリソースを用いられる. 一般的に，そこで供給されるリソースの正確な位置を，顧客が制御／知覚することはな

い. そのため，ロケーションから独立した感覚で，より高い抽象レベル（国／州／DC）においてロケーションが特定されてもよい. こうしたリソースの例としては，ストレージ／プロセッサ／メモリ／ネットワーク帯域幅／仮想マシンなどが含まれる.

- 弾力性に富む

コンピューティング能力の配給は，弾力性に富む. すなわち，いくつかのケースでは自動的に，スケールアウトの際に拡大し，また，スケールインの際に縮小する. 消費者にとって，この供給能力は無限に追加できるものになり，また，従量制で購入できるものとなる.

- 最適化サービス

サービスの種類（ストレージ／プロセッサ／バンド幅／アクティブユーザアカウント）に適した抽象レベルにおける測定機能を高めることで，クラウドシステムは自動的にリソース利用を制御し，最適化する. こうしたリソースの使用量については，利用されたサービスのプロバイダーと消費者から，透過的にモニター／コントロール／レポートされる.

2.1.2 サービスモデル

- サービスとしてのクラウドソフトウェア（SaaS）

このコンピューティングの能力は，クラウドインフラストラクチャ上で実行されるプロバイダーのアプリケーションを用いて，消費者に提供される. そのアプリケーションは，Web ブラウザ（Web メールなど）といったシンクライアントインターフェイスを介して，各種のクライアントデバイスからアクセスできる. 消費者はネットワーク，サーバ，オペレーティング・システム，ストレージや，個別のアプリケーション機能さえも含めて，基礎となるクラウドインフラストラクチャの管理／制御は行わないが，個々のユーザに特定されるアプリケーションコンフィグレーションは例外となる.

- サービスとしてのクラウドプラットフォーム（PaaS）

プロバイダーがサポートするプログラム言語とツールで作成したクラウドインフラストラクチャに，消費者が作成もしくは取得したアプリケーションを配置することが，消費者に提供される機能となる. 消費者はネットワーク，サーバ，オペレーティング・システム，ストレージなどの，基礎となるクラウドインフラストラクチャの管理／制御は行わないが，配置されたアプリケーションを制御し，また，そのホスティング環境をコンフィグレーションすることがある.

- サービスとしてのクラウド基礎（IaaS）

オペレーティングシステムやアプリケーションを含む任意のソフトウェアを，消費者が配置／実行することができる場所で，プロセッサ／ストレージ／ネットワーク／重要なコンピューティングリソースなどを供給するための機能が，消費者に対して提供される. 消費者は，基礎となるクラウドインフラストラクチャの管理／制御は行わないが，オペレーティングシステム／ストレージ／配置されたアプリケーションを制御し，また，選択された（ホストファイアウォールなどの）ネットワークコンポーネントを限定的に制御する.

2.1.3 ディプロイメントモデル

- プライベートクラウド

このクラウドインフラストラクチャは，特定の組織のために単独で運用される. そして，当該組織あるいは第三者団体により管理され，敷地内あるいは敷地外で運用されるだろう.

- コミュニティクラウド

このクラウドインフラストラクチャは，いくつかの組織により共有され，また，関心事（ミッション／セキュリティ要件／ポリシー／コンプライアンス）を共有する特定のコミュニティをサポートする. そして，当該組織あるいは第三者団体により管理され，敷地内あるいは敷地外で運用されるだろう.

- パブリッククラウド

このクラウドインフラストラクチャは，不特定多数の人々や大規模な業界団体などに提供され，対象となるクラウドサービスを販売する組織により所有される.

- ハイブリットクラウド

このクラウドインフラストラクチャは，複数のクラウド定義（プライベート／コミュニティ／パブリック）から，2 つ以上を組み合わせたものとなる. それぞれに固有の実体は保持するが，標準あるいや個別のテクノロジーによりバインドされ，データとアプリケーションのポータビリティ（クラウド間でのロードバランシングのためのクラウドバーストなど）を実現する.

2.2 群知能

2.2.1 Boid

Boid とは 1987 年にアメリカのアニメーション・プログラマであるクレイグ・レイノルズによって考案・作製された人工生命シミュレーションプログラムである. Boid というモデル名は，鳥もどきという意味の言葉である“ bird - android（バード・アンドロイド）”が短くなったことに由来している.
Boid の群を実現させるふるまいは，３つの要素からなり,「衝突の回避」,「速度を合わせる」,「群の中心に向かう」といった３つのルールを規定するだけで鳥の群をシミュレーションすることができる.

- Separation（衝突の回避）: 近くにいる仲間と衝突しないようにする
- Alignment（速度を合わせる）: 近くの仲間と速度を一致させようとする
- Cohesion（群の中心（重心）に向かう）: 近くにいる仲間に周りを囲まれた状態になろうとする

群の一連の動きは，自分自身と仲間との間の距離を最適に保とうとするルールを重要視している. このような３つの単純な行動規範をそれぞれの個体が持ち，全体として複雑な群の行動が創発する.
衝突回避のために，boid はそれぞれ自分にとっての「最適距離」を持っている. 自分の最も近くにいる仲間との間で，この距離を保とうと振る舞う. また，速度を合わせるために，最も近くにいる仲間と平行に（同じベクトルで）移動する. これによるスピードの変化はない. さらに，群の中心（boid 全体の集合の重心）に向かうようにも速度を常に変更している.

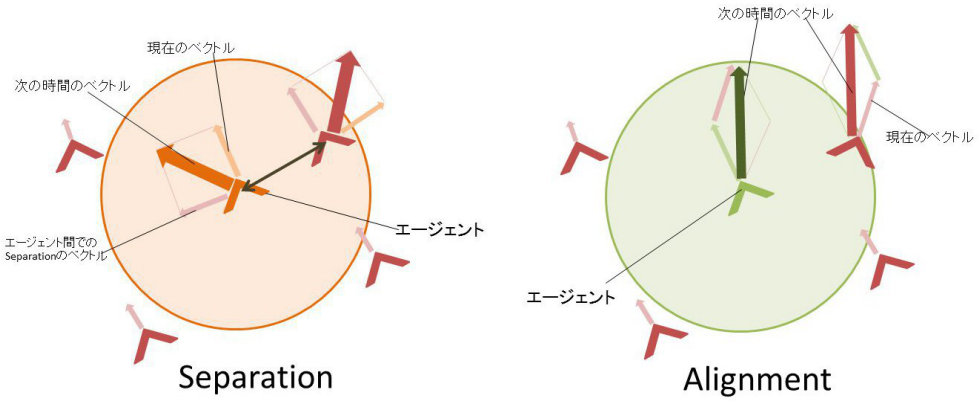

図 2.1: Separation・Alignment

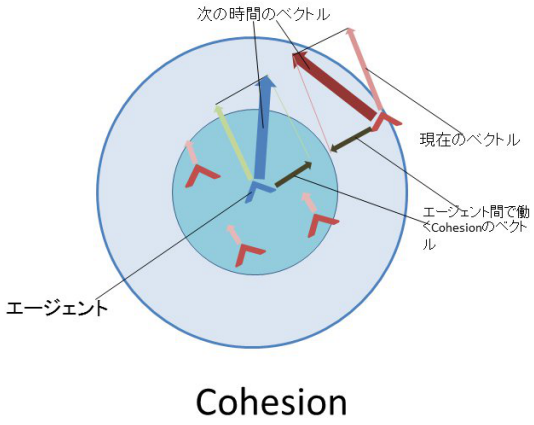

図 2.2: Cohesion

2.2.2 ACO（Ant Colony Optimization）

アリの摂食行動から着想を得たアルゴリズム.
フェロモンという揮発性物質を模したパラメータを最適化する.
組合せ最適化やネットワークルーティングなどに応用.

実世界では，アリは始めランダムにうろつき，食物を見つけるとフェロモンの跡を付けながらコロニーへ戻る．他のアリがその経路を見つけると，アリはランダムな彷徨を止めてその跡を辿り始め，食物を見つけると経路を補強しながら戻る．しかし，時間とともにフェロモンの痕跡は蒸発しはじめ，その吸引力がなくなっていく．その経路が長いほどフェロモンは蒸発しやすい．それに対して，経路が短ければ行進にも時間がかからず，フェロモンが蒸発するよりも早く補強されるため，フェロモン濃度は高いまま保たれる．従って，あるアリがコロニーから食料源までの良い（すなわち短い）経路を見つけると，他のアリもその経路を辿る可能性が高くなり，正のフィードバック効果によって結局すべてのアリが 1 つの経路を辿ることになる．群知能を用いて最適解を求めるアルゴリズムの一つである．

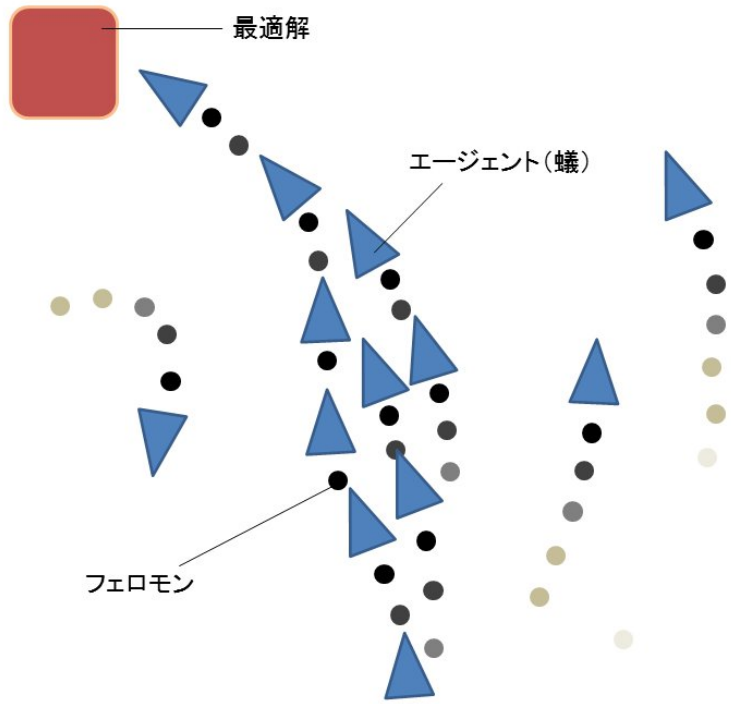

図 2.3: Ant Colony Optimization

2.2.3 PSO（Particle Swarm Optimization）

魚や鳥や昆虫などの生物の検索行動に着目し James Kennedy が考案したアルゴリズムである．最適化を求める過程での群れる性質を使用し，重要なファイルを中心を集合にするのに応用．

複数のエージェント/citedc を座標内のランダムの位置に配置し，ランダムに動く．個々のエージェントは個々の最適位置を評価関数を参照し，更新する．各エージェントは新しい適性位置を共有しあい，空間内の新しい位置に遷移する．エージェント同士の最適位置を更新と共有は何度も行い，群全体としての最適位置を見つける．ACO と同様に群知能を用いて最適解を求めるアルゴリズムの一つである．

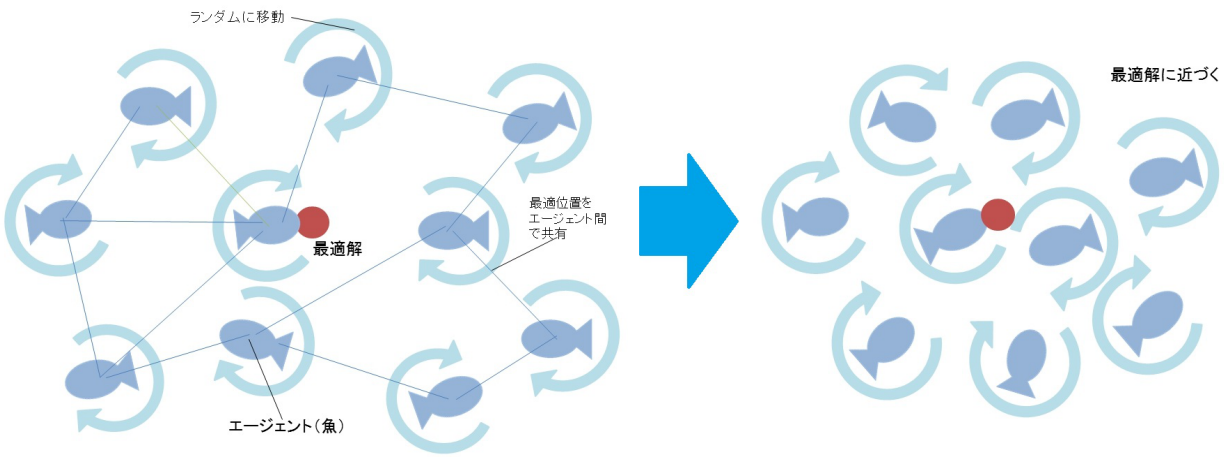

図 2.4: Particle Swarm Optimization

2.3 家族的類似

ルートヴィヒ・ウィトゲンシュタインの言語ゲームで振る舞う particle は，家族的類似性によって群を作ると定義した. 家族的類似性は言語と行為の類似性を表現するものである. 家族的類似性は同値関係でもないし，等価関係でもない. 常に変動しつつ，少しづつ似ているエンティティの集まりである. しかし，それは自己から見れば，同値関係であってもよいし等価関係であってもよい. そのような個人個人の意味論を統合して世界を記述する意味論が無いということである. 家族的類似性は不確実な世界の中の同類の定義である．家族的類似性は公的言語世界の観察視点の同類の定義である.

群知能において群れるふるまいは家族的類似に基づいている.

集まる力の源は,「家族的類似」であり，群知能のパラメータとして表現される. 群れる正の力

- 群の中心に向かう力： Cohesion
- 隣人と家族的類似行為をする力： Alignment
- 行為濃度： Pheromone

Pheromone は行為の軌跡の重要性を表現する．Pheromone は揮発性である．濃度が濃い pheromone は重要な行為を表す.

- 群れる負の力： Separation

群れる負の力は，群から排除する力，群から離れる（解除する）力である.

2.4 Android

Android とは，スマートフォン・タブレット PC などのモバイルデバイス向けのオープンでフリーなソフトウェアプラットフォームである. ここで言うソフトウェアプラットフォームとは，アプリケーションやアプリケーションフレームワークのみではなく OS やミドルウェアなども含んだ広範にわたるソフトウェアの集合体を指す. このことから，Android ではソフトウェア開発は大きく 2 つの視点に分けて考えることができる.

- ある端末の上で動作する Android プラットフォームを作る

Android はソフトウェアの集合体であるため，最終的にはあるハードウェアの上に載せて動かす必要がある．Android とハードウェアを結び付けていく作業のことをポーティング（porting）と呼ぶ. この領域の開発を行うには，特に Linux カーネルやデバイスドライバに関する知識が必要不可欠．アプリケーションの開発者は，Android プラットフォームを直接操作することはなく，アプリケーションフレームワークの機能を通じて利用する形になる．

- Android プラットフォームの上で動作するアプリケーションを作る

開発者は「Android SDK」として提供されているアプリケーションフレームワークの機能や開発ツールを用いてアプリケーションを開発する．Android でアプリケーションを開発するためには，Java やアプリケーションフレームワークに関する知識が必要になる．

2.4.1 Android のアーキテクチャ

アーキテクチャは大きく４つの層，５つの領域に分かれている．

- Linux カーネル層

最下層となる「Linux カーネル」は，バージョン 2.6 を元にモバイルデバイス向けに変更が加えられている. システムサービスをはじめとしたアプリケーションを実行するために必要な基本的な機能を提供する，いわば基礎となる層. 他の３つの層で動作する機能は，この Linux カーネル上で動作する．

- ライブラリ層

「Linux カーネル」層の上位層は,「ライブラリ」層となる.Android は，さまざまなコンポーネントから使用されるライブラリを提供する. ライブラリは，C や C++ 言語で作成されたライブラリを含む. これらの機能，基本的にアプリケーション開発者が直接使用することはなく，上位層であるアプリケーションフレームワーク層の機能を通じて使用されることになる．

- Android ランタイム

「ライブラリ」層と同レベルの機能として「Android ランタイム」がある.Android ランタイムは Java 言語に準拠するコアライブラリと Android アプリケーションを実行する Dalvik 仮想マシンにより構成されている．

- アプリケーションフレームワーク層

「ライブラリ」層の上位層は,「アプリケーションフレームワーク」層になる. アプリケーション開発者は，SDK にあらかじめ含まれているアプリケーションで使用されているものと同じクラスを使用することができる. アプリケーションを構築する際に使用するアーキテクチャは，コンポーネントの再利用が簡単にできるように設計されている.

- アプリケーション層

最上位の層が「アプリケーション」層.SDK には，電話や Web ブラウザなどの実行可能なアプリケーションがあらかじめ含まれている. アプリケーション開発者が作成したさまざまなアプリケーションもこの層に位置づけられる.

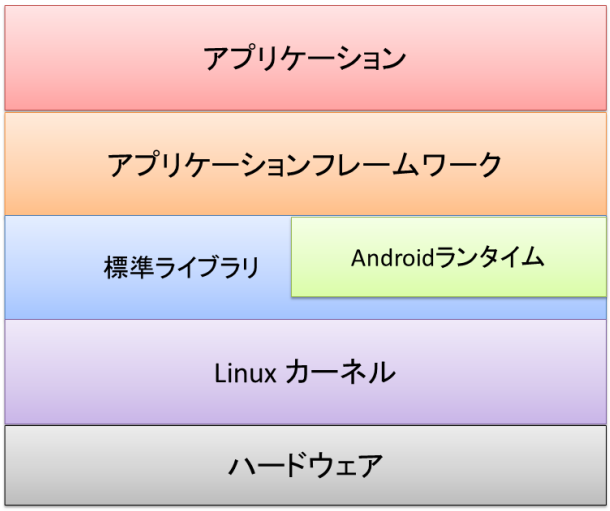

図 2.5: Android のアーキテクチャ

2.4.2 アプリケーション開発者の視点から見た Android の特徴

- Java 言語を使用してアプリケーションを作成する
- サーバアプリケーション開発で使用されているものに近いアプリケーションフレームワークが提供されている.

Web アプリケーションを構築する際によく使用される MVC2 パターンの要素（Model, View，Controllor の 3 つの要素）を Android アプリケーションの構成要素に対応づけることができる.

- Java 言語と他の言語の使い分けをしている.

ライブラリ層には，モバイルデバイスに含まれるハードウェアを制御するためのソフトウェアが含まれている．この部分については C/C++言語などの Java 以外の言語で記述されている.Java 言語は，C/C++言語と比較した場合，実行速度やタイミング制御，メモリの管理などを不得意としている．なので，Android はそうした不得意な箇所には適切な言語で実装を行い，アプリケーション開発者には意識させないクラスのインターフェイスを提供している.

2.4.3 アプリケーション開発の流れ

開発作業の流れは，“プロジェクトの作成” “ソースコードの作成” “アプリケーションの実行” の 3 つに分けることができる.

・プロジェクトの作成 ••• プロジェクトはアプリケーションの単位で作成する.
・ソースコードの作成
・アプリケーションの実行 ••• アプリケーションが完成したら，Android エミュレータ上でアプリケーションを実行する. その手順は以下の通り.

1．Java プログラムのコンパイル
2．Java バイトコード（.class）を Android バイトコード（.dex）に変換
3．Android アプリケーションをパッケージング（.apk ファイルの作成）
4．Android エミュレータの起動
5．アプリケーションを Android エミュレータにインストール
6．アプリケーションを Android エミュレータ上で起動

上記の作業が正常に完了したら，Android エミュレータ上でアプリケーションが動作する.

2.4.4 アプリケーションを構成する要素

Android では，クラスとして提供されるコンポーネント要素（API）を開発者が組み合わせることで 1 つのアプリケーションを作成する. このクラスには 2 つのタイプがある.

- Android アプリケーションとして動作するために必要なクラス
- Android アプリケーションの機能を提供するクラス

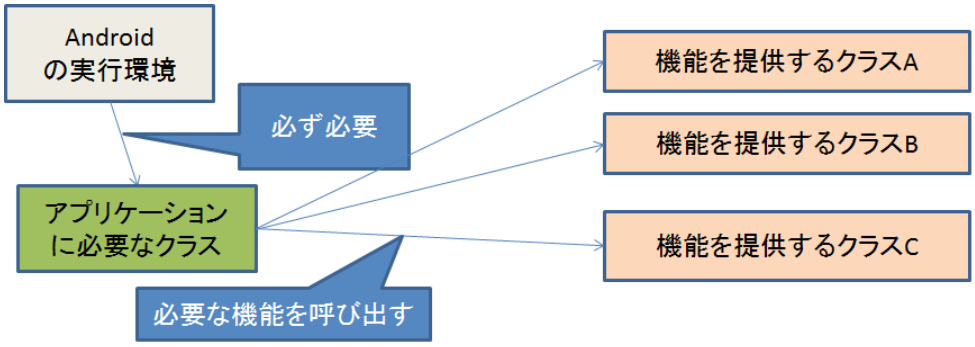

図 2.6: Android アプリケーションを構成するクラス. 2 つのクラスの関係性

Android の実行環境は，システム上にアプリケーションが必要となるクラスに紐づけたプロセスを作り，クラスの呼び出しを通じてアプリケーションの実行を制御する. この特別な役割を持ったクラスには 4 つの要素がある.

- アクティビティ

ユーザとアプリケーションとの間で行われるやり取り全般を仲介する.

- インテント

アプリケーションの実行時に各要素を紐づける

- サービス

ユーザの画面に対する操作に依存することなく処理を実行する.（バックグラウンド）

- コンテントプロバイダ

アプリケーション間で情報を共有する.

アプリケーションの開発において，これらの要素が一度にすべて必要になるわけではない．たとえば，ユーザが画面に対して操作を行うような標準的なモバイルアプリケーションでは，アクティビティを利用してアプリケーションを構築する．また，画面上に行う必要のない処理，いわゆるバックグラウンド処理と呼ばれるようなタイプのアプリケーションでは，サービスのみを使用するといった具合である.

Android アプリケーションとして機能を提供するためのクラスには，モバイルデバイスに装備されたハードウェアを利用するためのクラスを含む．また，アプリケーションを作成するための基本的な機能，たとえば画面制御やデータ制御などの機能も提供させる．さらには，Java 言語のレベルで提供されるクラスも広い意味ではこのタイプに含めてよい.

第3章　ふるまいの群に着目した ファイルシステムの提案

3.1　目的

従来のファイルシステム（木構造）では，探しているファイルがどのフォルダに入っているか忘れてしまうことがある. また，クラウドでは，ファイルが大量なので整理・分類するのが大変である. クラウド内の大量なファイルを自動的に整理・分類できるようにし，重要なファイル・関連するファイルを視覚的に見やすいファイルシステムを開発することが目的である. そこで以下のような特徴を持つ新しいクラウドのファイルシステムの一環として，PC 上のローカルなシステムを対象としたファイルシステムを提案する.

- Boid は、ファイルが群れるシステムの基礎となり，多様で不確実なクラウド内のファイルを一種の群として扱う.
- ACO のフェロモンを使い類似的なファイルを群の中心に集める.
- ACO のフェロモンを使い重要なファイルを群の中心に集める.
- PSO の評価関数を用い，より良い質の良い群を作る.

このクラウドファイルシステムを使用すると，例えば PC のデスクトップに様々のファイルがばらばらに散らばってしても，探しているファイルや関連するファイルを容易に見つけることができる.
また，自分ではあまり重要ではないとファイル・関連性のないファイルだと思っているものが以外と重要である場合に，クラウドファイルシステムによって提示しユーザの支援を行う.

3.2　クラウドファイルシステム

3.2.1　重要度の定義

すべてのファイルに重要度を設定し，群の中でのファイルの位置を決める. 重要度は 10 段階に設定する. どのようなファイルが重要度が高いか，以下のように定める.

- 使用頻度

使用頻度が多いファイルが重要度が高いとする.

- 時系列

仕事や学校での課題，研究等の締切を設定する. 締切が近いファイルは重要度が高いとする. また締切が近づいてくると自動的に重要度が高くなる.

- 依存関係

ファイル同士の重要度の依存関係のこと.A の仕事の前に B の仕事をやらなくてはならない. そのような場合，重要度は A ＜ B となる.

- ノルマ

ノルマが設定されてる仕事などに対して，ノルマが達成されてない場合は重要度が高いとする. 達成されて時点で重要度は低くなる.

- 任意の重要度

今まで述べた重要度の定義に当てはまらなくても重要なファイルはある. そこでユーザが重要度を指定できるようにする.

3.2.2 タグ

ファイルに対してあらかじめタグというものを付けておく. タグの種類を以下のようにする.

- 色

ユーザがそのファイルに対して色を指定する. 色の付け方はそのファイルを使う目的別に分けると良い. 卒業研究に必要なファイルであれば青色，就職活動に必要なファイルであれば赤といったように色を付けると，色ごとに群を作り関係するファイルが集まる.

- 文字列

そのファイルに対するキーワードを付ける. 付ける数は多ければ多いほど良い. 理由としては,「人の創発的活動を刺激・支援」するためである. 群は自分が指定した色同士でしか集まらない. 異なる色でも関係しているファイルはたくさんあるだろう. いろんなキーワードで検索をかけると自分が関係ないと思っていたファイルが以外なところで関係していて良いアイディアが得られることがある. このタグのおかげでインターネットクラウドの中での「人の創発的活動を刺激・支援」につながる. タグ付けは「整理・管理」の為だけでなく,「予想外の物を得る」ためでもある.

- 時系列

締切がある仕事や学校での課題，研究等のファイルに対して設定する.

- ノルマ

ノルマがある仕事や学校での課題，研究等のファイルに対して設定する.

- 任意の重要度

重要度の定義で当てはまらないが重要なファイルに対して，ユーザが重要度を指定できる. 基本的に指定した重要度は固定される.

3.2.3 重要度やタグの設定

クラウドファイルシステムを使用するにあたり，各ファイルに対して重要度やタグを設定し反映させる必要がある. ファイルの重要度に対してはそのファイルが今までに何回開かれているか調べるカウンターをクラウドファイルシステムに設定しておき，開かれた回数によって重要度を定める. 全体のファイルに対して相対的に多く開かれているファイルを重要度が高いファイルとする.

タグに対しては 3.5.3「タグ」で述べた各タグを設定してクラウドファイルシステムで使用する. ファイルにタグの設定画面を設け設定する. ファイルは普段は色を基準として群を作るため，色の設定は必須条件とする. 色はあらかじめ用意されているカラーテーブルから選択する. また文字列の「キーワード」も「人の創発的活動を刺激・支援」のために最低でも 1 つ設定することを必須条件とする. キーワードはできるだけ多く設定するほうが望ましい.「時系列」,「ノルマ」,「任意の重要度」は必要であれば設定する.

3.2.4 見せ方

- 群になっている中から目的のファイルをタップ/クリックするとそのファイルの詳細画面（ファイル名，作成者，サイズ，更新日時など）のウィンドが開く. さらにタップ/クリックするとそのファイルが開く.
- そのファイルに指定した色の濃さで重要度を示す. 重要度が高いと濃く，低いと薄くする. グラデーションを付けることによって視覚的に重要度の高いファイルを見つけやすくする.
- 重要度の高いファイルのアイコンは大きく，低いファイルのアイコンは小さく.

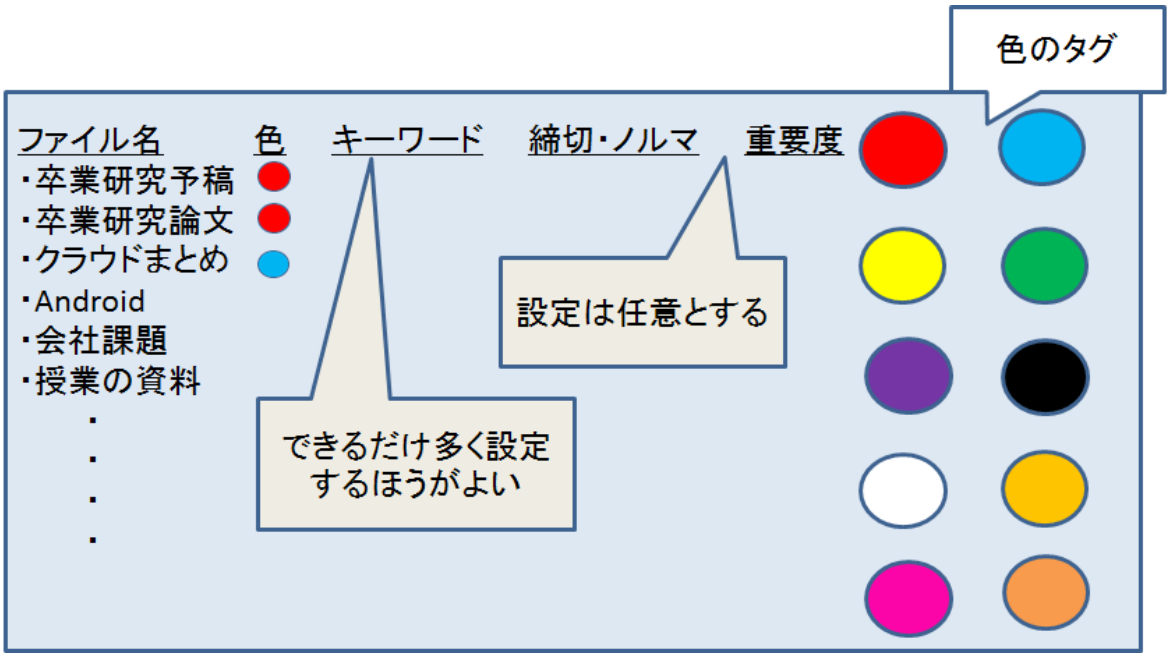

図 3.1: タグの設定画面イメージ

3.3 Boid をベースとする群知能

クラウド内の大量なファイルを自動的に整理・分類したい. また探しているファイル・そのファイルに似たようなファイル・関連するファイルを簡単に見つかるように「ファイルの見える化」を行いたい. そこでこのようなファイルのふるまいを Boid の Alignment，Cohesion，Sparation をベースとする群知能によって実現する. この群知能によって，似たようなファイル・関連するファイル同士が群を作り集まる.

- Alignment,Cohesion(引力)：家族的類似なファイル同士に引力が働く.
- Sparation（斥力)：異なるファイル同士に斥力が働く.

3.4 ACO の概念的特徴の「フェロモン」

群の中心に重要なファイルが集まるようにするとそのファイルを見つけやすくなる. そこで ACO（Ant Colony Optimization）の概念的特徴「フェロモン」を用いる. 群の中心にフェロモンの強いオブジェクトが集まる，なので，ェロモン＝重要度と定義し，群の中心に重要なファイルを集める. フェロモンは揮発性なので，あまり使用されないファイルは重要度が薄れていく. なので，使用頻度の多いファイルは重要度が高いファイルとして群の中心に集まる. 逆に使用頻度の少ないファイルは重要度の低いファイルとして群の外側に移動する.

3.5 Boid とフェロモン

Ant Colony Optimization は最適解を求めるアルゴリズムであり，群を作ることは最適解を求める過程で行われる. そのため群は ACO での目的ではない. その中の ACO のフェロモンの概念をクラウドファイルシステムに組み込む. 各エージェントが独立したフェロモンを出すことで，そのエージェントに似たファイルが集まってくる. これにより，重要で関連が多いファイル程大きな群を作り，大きな群の中心になる.
具体的に，クラウドファイルシステムでのファイルの群のふるまいは Boid の Alignment,Cohesion,Sparation とフェロモンによって創発される.
まず，Boid の Alignment，Cohesion，Sparation で家族的類似しているファイル同士が群を作る. その群の中での位置関係としてフェロモンを使用する. 群の中心にフェロモンの濃度が高いファイルが集まる. フェロモン＝重要度としているので重要度が高いファイルが群の中心に集まる. ファイルの重要度は 10 段階にに設定する. そして，図に示すようにあらかじめファイルの重要度に対して群の中での群れる範囲・位置を定めておく. そうして，各ファイルの群の中でも位置を定めていく.

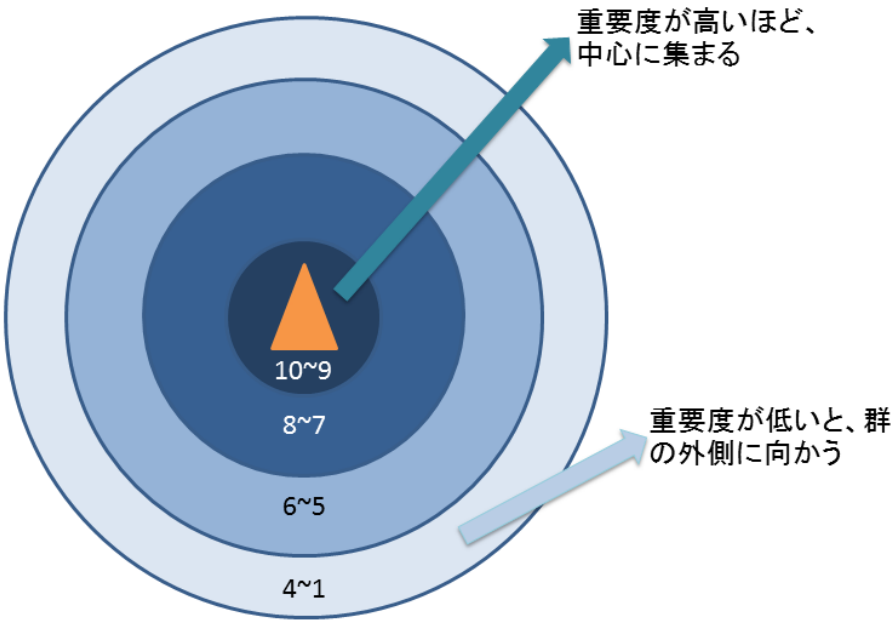

図 3.2: 重要度による位置関係

3.6 Boid と局所的な集まり

PSO の評価関数の概念を取り込むと絶対的にせよ相対的にせよ，“何が最良か”というエージェントの持つ価値が生まれる. これがあると群の目指す方向がしっかりとし，より質の良い群を作ることができるのではないかと考える.
PSO はエージェントが評価関数を参照し，自律分散をしているように群をなす. しかし，外部の絶対的尺度の「評価関数」に依存しており，エージェント自身が自立しているわけではない. 今回提案するモデルはエージェントが自立して集まることを目的としているため，PSO をモデルに組み込むことはできない. エージェントは各エージェントが持つ独自の価値観で局所的類似により局所的に集まり，絶対的な集まりの中心はない. 各エージェントが中心を独自に決める.
しかし，今回提案するモデルには評価関数がないため，群の行きつく（到着する）場所が設定することができない. そのため，一部のエージェントに PSO の外部からの絶対的な価値を持たせ，どのようなことが起こるかをシミュレーションをすることができる. 今後，局所的類似と局所的評価関数を作り出す.

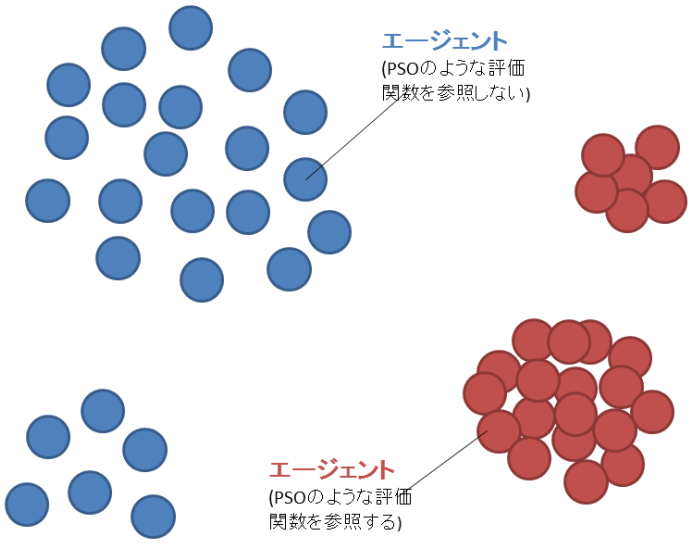

図 3.3: 評価関数を持つエージェントの群と持たないエージェントの群

PSO のような評価関数を参照するエージェントは，集まっても精度が低い. それに対し，評価関数を持つエージェントは局所的に集中して集まる. これにより，群が集中して集まり，より良い質の良い群を作ることが可能である. 各エージェントが離れにくく，群が群らしく見え，その群の概要を把握することができる。

第4章　javaを用いた boidシミュレーションの開発

4.1　開発概要

android 端末に組み込む前に，java.applet を使って boid をシミュレートする．各エージェントの加速度，速度，座標を他のエージェントと照らし合わせながら，boid の 3 要素（Separation，Alignment，Cohesion）を計算し続け，その度に座標上に描写する．
速度制限が無い限りエージェント及び群の速度は増加し続けるので，速度のリミッターも用いて速度制限も行う。

4.1.1　開発環境

- Java
- JDK
- Eclipse
- Java Applet
- ADT Plugin for Eclipse

4.2　開発手順

4.2.1　開発環境のセットアップ

開発に必要な JDK，Eclipse をインストールする. すべて無料でインストールできる．

- JDK（Java SE Development Kit）とは Java のソフトフェア開発キット
- Eclipse とはオープンソースの統合ソフトフェア開発環境
- JavaApplet は JDK をインストールした際，デフォルトで付属されている．

4.2.2 Boid の構想

Boid は，Separation，Alignment，Cohesion の三つで構成される.Separation を最優先にし，次に Aligment，Cohesion と続く.

基本構造

任意の数のエージェントを座標上にランダムに配置する. エージェントには「座標（Position)」,「速さ（Velocity)」,「加速度（Acceleration)」の三つのステータスを x と y それぞれを持たせる.「座標」は更新の度「速さ」を参照し，現在の座標に加算する.「速度」も更新の度,「加速度」を参照し,「速さ」に加算する. 加速度は更新の度，規則に応じて数値を決定し，その度に 0 にリセットする.

$$P_{t+1}(x,y) = P_t(x,y) + V_{t+1}(x,y)$$

$$V_{t+1}(x,y) = V_t(x,y) + A_{t+1}(x,y)$$

P… 座標 V… 速度 A… 加速度

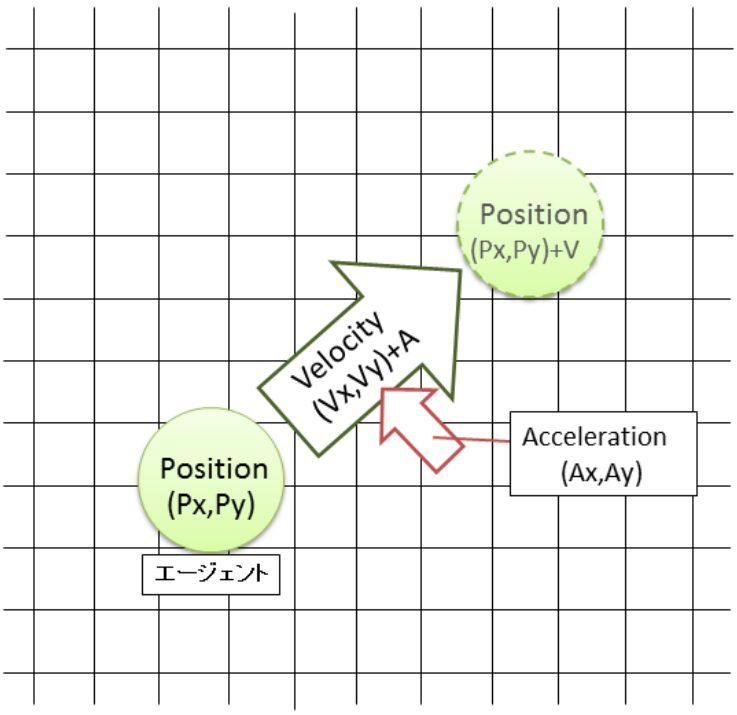

図 4.1: Potision の移動推移

Separation

Separation は，エージェント同士が近ければ近いほど，お互いの斥力が強く働く.

$$\vec{Ac_i} = -F_s \frac{\vec{Ps_i} - \vec{Ps_j}}{|\vec{Ps_i} - \vec{Ps_j}|}$$

$$\vec{Ac_j} = F_s \frac{\vec{Ps_i} - \vec{Ps_j}}{|\vec{Ps_i} - \vec{Ps_j}|}$$

$$F_s = 1 - \frac{Percent}{Threshold(1)}$$

Alignment

Alignment は，一定間隔以内の程良い距離間のエージェント同士が同じ方向に動くための要素である.

$$\vec{Ac_i} = F_a \frac{\vec{Vl_j}}{|\vec{Vl_j}|}$$

$$\vec{Ac_j} = F_a \frac{\vec{Vl_i}}{|\vec{Vl_i}|}$$

$$F_a = -\mathrm{T}\ (Percent - I)^2 + 1$$

Cohesion

Cohesion は，一定間隔の以内の通りエージェントに近づく,「仲間の検索」の働きをする要素である. 遠い程，加速する.

$$\vec{Ac_i} = F_c \frac{\vec{Ps_i} - \vec{Ps_j}}{|\vec{Ps_i} - \vec{Ps_j}|}$$

$$\vec{Ac_j} = -F_c \frac{\vec{Ps_i} - \vec{Ps_j}}{|\vec{Ps_i} - \vec{Ps_j}|}$$

$$F_c = \frac{Percent - Threshold(2)}{1 - Threshold(2)}$$

$\vec{Ac}_i$…i 番目のエージェントの加速度
$\vec{Ps}_i$…i 番目のエージェントの現座標
$\vec{Vl}_i$…i 番目のエージェントの現速度
F_s…Separation のスカラーの関数
F_a…Alignment のスカラーの関数
F_c…Cohesion のスカラーの関数
Percent…i 番目のエージェントの持つ j 番目のエージェントに対するローカルスペースの侵入具合. 遠いほど数値が高い.
Thresold(1)…Percent の閾値 1
Thresold(2)…Percent の閾値 2
I…エージェント間の理想の距離感
T…定数

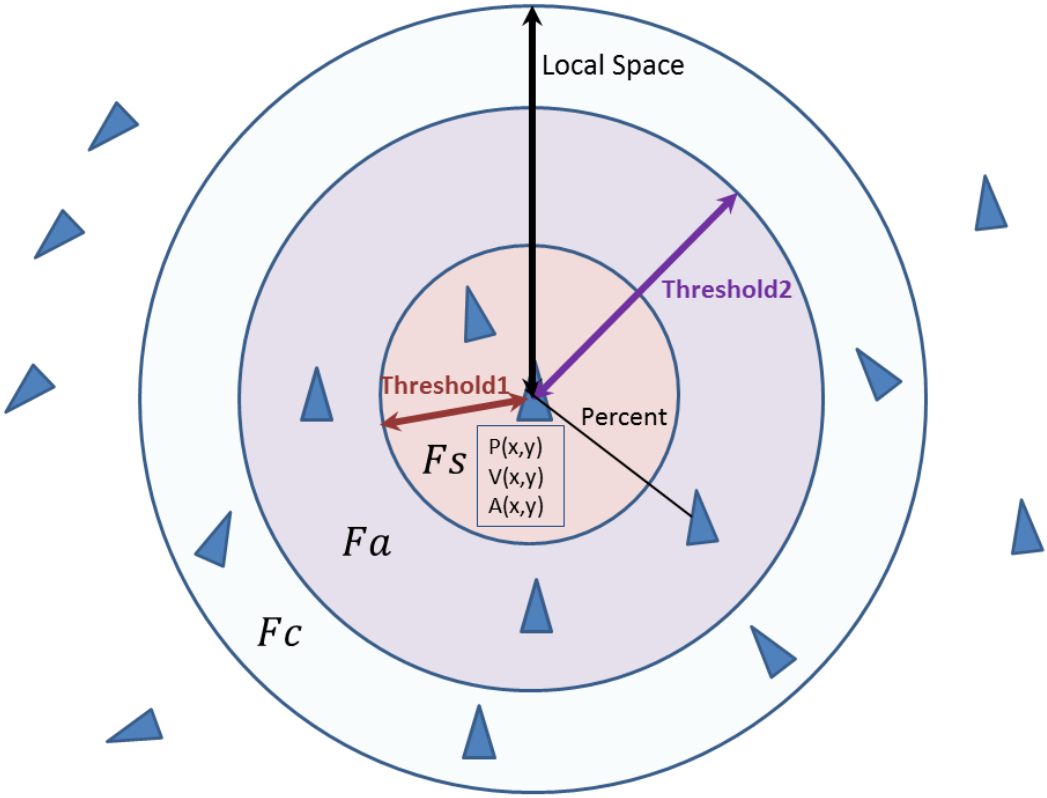

図 4.2: boid 図

4.3 シミュレーションの実装

4.3.1 使用クラス

- acc.java 加速度（acceleration）について決めているクラス
- vel.java 速度（velocity）について決めているクラス
- pos.java 座標（position）について決めているクラス
- particle.java

一つのエージェントの pos.java，vel.java，acc.java それぞれを参照しながら，座標に描画するクラス

- boidMain.java

各エージェント間の距離の計測，boid の 3 要素の計算を行うメインのクラス

4.3.2 シミュレーションの実行

java.applet で実行を行った．上下左右でループするよう設定しているが，座標の数値はループしていない．

4.4 結果

複数の群が存在し，等間隔の距離感を取る Separation，同方向に動いている Alignment，離れたところにいるエージェントの取り込む Cohesion を確認でき，boid のシミュレーションは成功した．

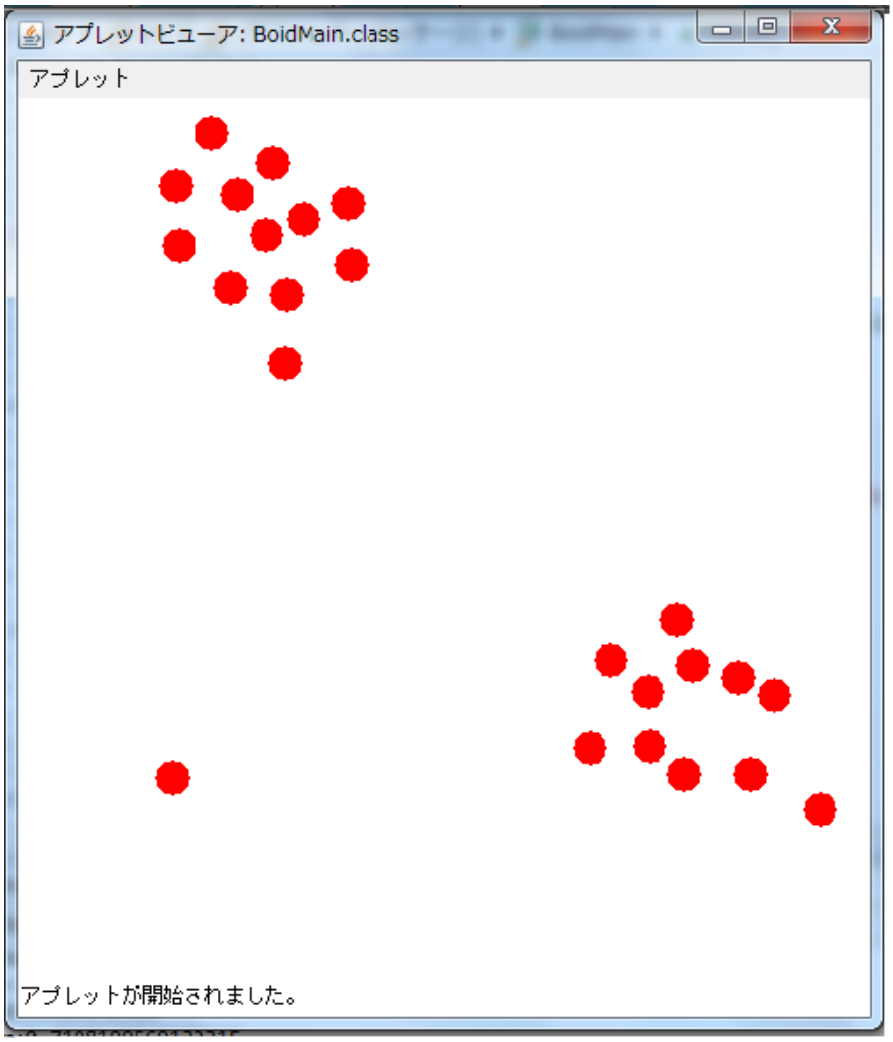

図 4.3: java.applet での実行画面

第5章　結論

今回の研究は「人の創発的活動・使用」するような新しいシステムを提案した. ファイル（エージェント）が自立して集まることを目標として，群知能のシステムを組み込んで開発を行った. 実際に java.applet で boid のシミュレーションを行い，エージェントの群れる基本構造や boid の Separation，Alignment，Cohesion を組み込むことができた.

今後の課題としては，今回作った boid に，ACO のフェロモンの概念を取り込み，複数の種類の群を作り出して重要なファイルを中心に置くことである. また，PSO の評価関数の概念を取り込み群そのものの質をあげることを加えることが今後の課題である. 今回上げた群知能は boid，ACO，PSO の三点があるが，その他にも人工蜂コロニーアルゴリズム（Artificial Bee Colony algorithm），クラスメートモデル， 人工免疫システム (Artificial Immune System) など沢山の群知能が存在している. 現段階ではこれらがどう活用できるかは不明だが，これらの持つ要素を組み合わせながら，提案モデルをより良いふるまいの群を作りたい. 今回行うことが出来なかった群知能のシミュレーションだけでなく，android 端末に組み込むことを目標とする.

第6章　謝辞

本研究を行なうにあたり，終始熱心に御指導していただいた木下宏揚教授に心から感謝致します．また，様々な面で数多くの有益な御助言をしていただいた東洋ネットワークシステムズ株式会社の森住哲也氏に深く感謝致します．さらに，研究活動一般に様々な助言をいただきました南出氏をはじめ公私にわたり良き研究生活を送らせていただいた木下研究室の方々に感謝致します．

2013年　2月
石田　克憲

参考文献

[1] “クラウド・コンピュータの基礎”
http://www.ibm.com/developerworks/jp/cloud/library/cl-cloudintro/

[2] 岩谷　宏：“JAVA の哲学”ソフトバンクパブリッシング (2001)

[3] 伊庭　斉志：“複雑系のシミュレーション　Swarm によるマルチエージェント・システム”コロナ社　(2007)

[4] “ファイルシステムとは” http://www.atmarkit.co.jp/ait/articles/1208/24/news135.html

[5] “Boid とは”
http://members.jcom.home.ne.jp/ibot/boid.html

[6] “情報漏えい発生時の対応ポイント集”
http://www.ipa.go.jp/security/awareness/johorouei/

[7] “情報フィルタリング”
http://www.referencenet.jp/column/column025.html

[8] 新屋真二：“人工知能概論”
共立出版 (2003)

[9] “National Institute of Standards and Technology”
http://www.nist.gov/index.html

[10] “Java 初心者入門講座” http://sunjava.seesaa.net/

[11] “鰯の群れ　ライブ壁紙”QSDN http://www.qsdn.co.jp/service/android+apps/

[12] “Boids もどき” http://hp.vector.co.jp/authors/VA009508/Java/Boids/boids.html

[13] “Processing boid　金魚” http://www.tiu.ac.jp/ zohzemi/processing/index.html

[14] “mdellavo / boids” https://github.com/mdellavo/boids

[15] 「IDEA-MOO ＜知の体験（家族的類似性）＞」http://idea-moo.net/chitaiken/ono08100301.html

[16] “Google USB Driver”Android developers
http://developer.android.com/intl/ja/sdk/win-usb.html

[17] 由梨かおる：“Java 入門編” softbankCreative(2008)

[18] “オブジェクト思考
http://think-on-object.blogspot.com/2011/11/is-ahas-is-ahas-top-is-a-is-b.html

[19] “インスタンス化”
http://sunjava.up.seesaa.net/image/java-195.gif

[20] “ボリューム-意味・説明・解説”ASCII.jp
http://yougo.ascii.jp/caltar/

[21] “Flockers”MASON
http://cs.gmu.edu/ eclab/projects/mason/

[22] “Boid”TM’s Workspace
http://termat.sakura.ne.jp/actionscript/boid/extended

[23] “Akito Nakano” http://web.sfc.keio.ac.jp/ akito/

第7章　付録

7.1　boidシミュレーション

7.1.1　Acc.java

```
public class Acc{

        private double x;
        private double y;

        //コンストラクタ//
        public Acc(double x,double y){
                this.x=x;
                this.y=y;
        }

        //取得//
        public double getAX() {
                return x;
                }
        public double getAY() {
                return y;
        }

        //xとyを設定//
        public void set(double x, double y){
                this.x=x;
                this.y=y;
        }
}
```

7.1.2 Vel.java

```
public class Vel{
        private double Vx;
        private double Vy;

        //コンストラクタ//
        public Vel(double Vx,double Vy){
                this.Vx=Vx;
                this.Vy=Vy;
        }

        //取得//
        public double getVX() {
                return Vx;
        }
        public double getVY() {
                return Vy;
        }

        //xとyを設定//
        public void set(double Vx, double Vy){
                this.Vx=Vx;
                this.Vy=Vy;
        }
}
```

7.1.3 Pos.java

```
public class Pos{
        private double Px;
        private double Py;

        //コンストラクタ//
        public Pos(double Px,double Py){
                set(0,0);
        }

        //取得//
        public double getPX() {
                return Px;
        }
        public double getPY() {
                return Py;
        }

        //xとyを設定//
        public void set(double Px, double Py){
                this.Px=Px;
                this.Py=Py;
        }
}
```

7.1.4 particle.java

```
import java.awt.Color;
import java.awt.Graphics;
import java.util.Random;

public class particle extends java.applet.Applet{
        public Pos position=new Pos(0,0);
        public Vel velocity=new Vel(0.1,0.1);
        public Acc acceleration= new Acc(0,0);
        public double velocity_MAX=3;
        public int size=20;
        public int rdx, rdy;
        public static int number=25;
        Random rnd = new Random();

        particle(){
                rdx = rnd.nextInt(500);
                rdy = rnd.nextInt(500);
                position.set(rdx,rdy);
        }


        public void paint(Graphics g, Color color) {
                int PX =(int)position.getPX();
                int PY =(int)position.getPY();
        g.fillOval(PX,PY, size,size);
                g.setColor(color);
                repaint();
        }

        public void update(){
        velocity.set(velocity.getVX()+acceleration.getAX(),
        velocity.getVY()+acceleration.getAY());
                limitter();
                position.set(position.getPX()+velocity.getVX(),
                position.getPY()+velocity.getVY());
                if(position.getPX() >= 500 ){
                position.set(1,position.getPY());
                }
                if(position.getPY() >= 500 ){
                position.set(position.getPX(),1);
                }
                if(position.getPX() <= 0 )      {
                position.set(499,position.getPY());
                }
                if(position.getPY() <= 0 ){
                position.set(position.getPX(),499);
                }
                acceleration.set(0,0);
        }

        private void limitter(){
```

```
                    double length =
Math.sqrt(Math.pow(velocity.getVX(),2)
+Math.pow(velocity.getVY(),2));
                    System.out.println(length);
                    if (length> velocity_MAX){
                            velocity.set(velocity.getVX()*0.1,velocity.getVY()*0.1);
                            length =
                              Math.sqrt(Math.pow(velocity.getVX(),2)+Math.pow(velocity
                            System.out.println("after:" + length);
                            }
                    return;
            }


}
```

7.1.5 boidMain.java

```
import java.awt.*;
import java.util.ArrayList;

public class BoidMain extends java.applet.Applet{
        boolean init_flag=false;
        ArrayList<particle> pnts = new ArrayList<particle>();

        public void init(){
                if(init_flag) return;
                init_flag = true;
                for(int i = 0; i < particle.number; i++){       pnts.add(new particle());}
                 }

        public void paint(Graphics g)   {
                        init();
                        for(int i=0; i < particle.number; i++ ) move();
                        for(int i=0; i < particle.number; i++ ) pnts.get(i).paint(g, Color.red);
                        pause(30);
                        repaint();
        }

        void pause(int time){
        try { Thread.sleep(time);}
        catch (InterruptedException e) {}
        }

        //move()クラス boidの三つはこのクラス
        public void move(){
                float localSpace = 80;
                double thresh0 = 0.45;
                double thresh1 = 0.90;
                   for(int i=0; i<particle.number; i++) {
                       for(int j=0; j<particle.number; j++) {
                           if (i!=j) {
                                   double dist =
                                   Math.sqrt(Math.pow((pnts.get(i).position.getPX()
                                   - pnts.get(j).position.getPX()),2) +
                                    Math.pow((pnts.get(i).position.getPY()
                                   - pnts.get(j).position.getPY()),2));
                               if(dist < localSpace) {
                                   double percent = dist/localSpace;

                                        //Separation
                                   if (percent < thresh0) {
                                           double F = 0.1*((-(1/thresh0)*percent)+1);
                                           double dirX =F* (pnts.get(i).position.getPX()
                                           - pnts.get(j).position.getPX());
                                           double dirY =F* (pnts.get(i).position.getPY()
                                           - pnts.get(j).position.getPY());
                                           pnts.get(i).acceleration.set(
                                            pnts.get(i).acceleration.getAX()
                                            + dirX,pnts.get(i).acceleration.getAY() + dirY);
                                           pnts.get(j).acceleration.set(
                                            pnts.get(j).acceleration.getAX()
                                            - dirX,pnts.get(j).acceleration.getAY() - dirY);
                                   }

                                   //Alignment
                                   else if (percent < thresh1){
                                           double F =
                                            (-16)*Math.pow(percent-0.55,2)+1;
                                           double AliPeri =
                                            Math.sqrt(Math.pow(pnts.get(i).velocity.getVX(),2)
                                            + Math.pow(pnts.get(i).velocity.getVY(),2));
                                           double AliPerj =
                                            Math.sqrt(Math.pow(pnts.get(j).velocity.getVX(),2)
                                            + Math.pow(pnts.get(j).velocity.getVY(),2));
                                           double AlXi =
                                            (pnts.get(i).velocity.getVX() / AliPeri);
                                           double AlYi =
                                            (pnts.get(i).velocity.getVY() / AliPeri);
                                           double AlXj =
                                            (pnts.get(j).velocity.getVX() / AliPerj);
                                           double AlYj =
                                            (pnts.get(j).velocity.getVY() / AliPerj);
                                           double addiX = F*AlXi;
                                           double addiY = F*AlYi;
                                           double addjX = F*AlXj;
                                           double addjY = F*AlYj;
                                           pnts.get(i).acceleration.set(
                                           pnts.get(i).acceleration.getAX()
                                           + addjX,pnts.get(i).acceleration.getAY() + addjY);
                                           pnts.get(j).acceleration.set(
                                           pnts.get(j).acceleration.getAX()
                                           + addiX,pnts.get(j).acceleration.getAY() + addiY);
                                   }

                                   //Cohesion
```

```
                                        else {
                                                double F =
                                                 (1/(1-thresh1))*(percent-thresh1);
                                                double CoX =
                                                 ((pnts.get(i).position.getPX()
                                                 - pnts.get(j).position.getPX())/dist);
                                                double CoY =
                                                 ((pnts.get(i).position.getPY()
                                                 - pnts.get(j).position.getPY())/dist);
                                                double dirX =F* CoX;
                                                double dirY =F* CoY;

                                                pnts.get(i).acceleration.set(
                                                 pnts.get(i).acceleration.getAX()- dirX,
                                                 pnts.get(i).acceleration.getAY() - dirY);
                                                pnts.get(j).acceleration.set(
                                                 pnts.get(j).acceleration.getAX() + dirX,
                                                 pnts.get(j).acceleration.getAY() + dirY);

                                        }
                                    }
                                }
                            }
                        }
                        for(int i=0; i<particle.number; i++) {
                                pnts.get(i).update();

                    }
                }
        }
```