平成21年度

筑波大学第三学群情報学類

卒業研究論文

題目
LifeIndicator: 習慣の規則性を視覚的に提示する手法

主専攻　　情報科学主専攻

著者　　鈴木 俊吾

指導教員　志築文太郎 高橋伸 三末和男 田中二郎

要　旨

我々が生活をしていくうえで考慮すべきことに“健康”がある。我々は、規則的な生活を過ごし、体を健康的な状態に保つことで毎日を充実したものにすることができる。

規則的な生活をしているかどうかを判断するためには、過去の行動を振り返る必要がある。しかし、過去の行動を振り返る際に生じる問題として、過去の行動をそもそも覚えていない、どの程度その生活が良くなったあるいは悪くなったかという基準が無い、という点が挙げられる。

そこで、本研究では、行方履歴を円として視覚化し、この円を組み合わせることによって生活の行動パターンを確認するシステムを構築した。また、このシステムでは、生活の規則性を表す指標を独自に定義し、履歴同士の比較によって算出されたその指標をアナログメータ形式で提示することにより、どの程度規則的あるいは不規則になったかを視覚的に提示することを可能にしている。

目 次

図 目 次

第1章　はじめに

1.1　生活と健康

我々が生活をしていくうえで考慮すべき事の 1 つに “健康”がある。この健康というのは、毎日を充実したものにするためには不可欠な要素である。

人間も含めた動物にはサーカディアンリズムという生活リズムが存在している。このサーカディアンリズムは、生物に備わっている様々な生活リズムの内、約 24 時間のサイクルのリズムのことである。例として、「日の出とともに朝起床し、暗くなったら眠る」という生活リズムが代表的であるが、その他にもホルモンの分泌であったり、体温や血圧、神経活動等を含む多くの生命活動がこのリズムに従っている。

現代社会は、我々の生活を昼型から夜型の方向へとずれ込ませる傾向にある。これは、本来昼型の生活リズムであるはずの生物の仕組みから反した変化であり、結果として日常生活を過ごしていくうちに心身の不調を訴えることが多くなってきている。

このような背景から、現代社会では日常生活に問題を抱えている生活者の健康増進、すなわち生活の質（QOL = Quality of Life）の向上が注目されるようになってきている。

1.2　生活が規則的であることの重要性

佐藤らは、大学生のライフスタイルと精神的健康度の関連についての研究を行っており [1]、食事のタイミングが不規則であったり起床時刻が不規則であるなどといった不規則なライフスタイルと、精神的健康との間には関連が強いということを明らかにしている。すなわち、習慣が不規則な場合には精神的な健康に悪影響を及ぼすということを示している。

また、森本はライフスタイルと健康について言及しており [2]、ライフスタイルが悪ければ悪いほど、すなわち不規則な生活を送ることは、消化性潰瘍、循環器疾患などといった疾病になる危険性が増す、というデータを示している。

これらの研究が示すように、生活の規則性と健康の間には密接な関係があり、健康な生活を送るためには規則正しい生活をする必要があるということが分かる。そのためには、自分の習慣がどの程度規則的であるかを振り返ることが大切になってくる。

1.3　習慣を振り返る際の問題点

習慣を振り返る際に生じる問題として以下の 2 点が挙げられる。

- 自分の過去の行動を思い出す必要がある。
- 生活の規則性の基準がない。

まず 1 つ目の問題点として、習慣を振り返るためには、いつ、どこで、何をしていたかといった、自分の過去に行った行動についての内容を思い出さなければならない。しかし、普段から過去の行動について意識することはあまり無いため、思い出そうとしてもなかなか思い出すことが難しいと思われる。直近の行動は比較的記憶していることが多いが、長期間にわたる行動を詳細に把握しておくのは困難である。

これらの問題に対する対処法の 1 つに「日記をつける」という方法がある。しかしながら、毎日行動を日記に記録するという手間がかかるということもあり、長い間継続していくには相当な根気を要する。万人が無理なく続けるためには、より手軽で負担の少ない行動の記録手段が必要になってくる。

また、もう 1 つの問題点としては、生活の規則性そのものの基準が無いという点が挙げられる。過去の行動をうまく記録する手段ができたとしても、過去の行動はどの程度規則的な生活をしているのか、最近の行動は以前よりも規則的になっているのかそれとも不規則になっているのかという事を把握することが難しい。

上記の問題に対処するためには、過去の行動履歴を手軽に記録、蓄積する手段と生活の規則性の基準があれば良いのではないかと考えた。また、生活を振り返ることを考慮すると、過去の行動履歴を可視化して分かりやすく提示することが重要ではないかと考えた。

1.4 本研究の目的

そこで、本研究では、過去の行動履歴を可視化すること、そして、生活の規則性を表す基準を定義し、視覚的に提示することによって、生活の振り返りを支援することを目的とする。過去の行動履歴と生活の規則性を同時に閲覧することで、「あの時は生活が乱れ気味だったから気をつけよう」、「最近は生活が規則的になってきているからこの調子でいこう」などといった、過去の生活に関する評価が可能となる。また、視覚化された情報をメンバー間で共有し、自分以外の履歴を閲覧することによって、他者と比べて自分はどうなのかという自分の行動の行動を振り返るための参考情報にすることができる。

1.5 本研究のアプローチ

我々の研究室では、DOCoCa という電子行方表システム [8] が運用されている。DOCoCa では、誰が、どこにいるかを表す情報である行方履歴を記録している。この DOCoCa を利用することで、過去の行方情報を記録、蓄積することができる。

本研究では、DOCoCa で記録している行方履歴に注目している。生活の規則性の基準を独自に定義することで、行方履歴を基に値として算出する。また、算出された値は視覚的に提

示する。同時に、行方履歴も提示することにより、過去の行動 (行方) を確認しながら、どの程度規則的な生活をしているかを視覚的に分かるようにし、習慣の振り返りを支援する。

1.6 本論文の構成

本論文では、まず 2 章で関連する研究について述べる。その後、3 章で本研究で実装したシステムである LifeCircle の概要および使用法について述べる。また、LifeIndicator の利用例を示す。4 章では、LifeCircle の具体的な実装について説明する。この章では LifeIndicator の視覚化アルゴリズムの詳細についても述べる。5 章では、LifeCircle を運用した結果から考察を行い、6 章でまとめと今後の課題について述べる。

第2章　関連研究

本章では、本研究の関連研究について述べる。

2.1　行動パターンの抽出

青木ら [3] は、人物の位置・姿勢に視点をおいた行動パターンの学習、認識に関する研究を行っている。この研究では、学習期間に行った動作を自動的に分類し、個別モデルを作成することにより、人物の動作の認識、動作の順序を考慮した行動パターンを認識する手法を提案している。全方位カメラの画像から人物領域の特徴量を抽出し、隠れマルコフモデルを使用した学習から行動パターンを認識することによって、非日常的動作の検出を行っている。この研究では、高齢者などの介護が必要な人物に対する見守りシステムなどへの応用を目的としている。

山越ら [4] は、所在表を利用した面会支援システムの構築を行っている。この研究では、所在表から得られるワークリズムを抽出し、不在の相手がいつ頃戻ってくるかという予測を訪問者に提示することを目的としている。提示する情報には言葉を使用しており、「ちょっとしたら」「しばらくしたら」などといった、あいまいな表現を用いている。

土持ら [5] も、山越らと同様に所在表に注目し、所在表を拡張したシステムである ExDB を試作している。このシステムは、従来の所在表では手動で行っていた状態の更新を、スケジュール情報と位置情報による活動内容の推測を用いることによって自動化している。また、個人のプライバシーを考慮するために、提示する情報の詳細度を制御する機能を持っている。

2.2　時系列データの視覚化

Webera ら [6] や Carlis ら [7] は、時系列データを螺旋状に視覚化する手法について言及している。螺旋の円周上に時間軸を、データの周期を円周の長さに対応させることによって、周期性を持つデータの把握がしやすい提示手法となっている。

藤原ら [8] は、誰が、いつ、どこにいるのかを表す情報である行方履歴を記録し、視覚化する電子行方表システムである DOCoCa を作成している。行方履歴の視覚化には [6][7] で述べている螺旋状に視覚化する手法を採用している。著者は DOCoCa を開発するメンバーの一員である。

本研究では、この DOCoCa から得られる行方履歴を用いたシステムの開発を行っている。DOCoCa の詳細については 4.3 節で述べる。

2.3　本研究の位置づけ

本研究では、日常の行動履歴として行方履歴に注目し、行方履歴から習慣の規則性を算出し、視覚化して提示するシステムを構築している。本研究は、習慣の規則性を定量化することによって、単純に行方履歴を閲覧するのみでは分かりづらい習慣の規則性の程度を針の振れ具合という尺度で表現することに特徴がある。

第3章　実装システム LifeCircle

この章では、本研究で実装したシステム LifeCircle の概要および利用法について述べる。

3.1　LifeCircle とは

LifeCircle とは、本研究で実装した行方履歴および習慣の規則性を視覚的に提示するシステムである。行方履歴の視覚化には、複数の同心円を並べることによって実現している。行方履歴は同心円を複数並べることで、習慣の規則性は 3.4 節で述べる LifeIndicator によって視覚化する。

このシステムは Java アプレットとして実装しており、Web ブラウザ上から本システムを利用することが可能である。

3.2　LifeCircle の利用

LifeCircle を利用する流れとしては、まずシステムへのログインを行ったあとで、システムの操作及び閲覧ができるようになっている。次節からは、ログインの処理、そしてシステムにログインした後における LifeCircle の利用について説明する。

3.2.1　LifeCircle へのログイン

LifeCircle を利用する際には、まず最初に LifeCircle を設置している Web ページにアクセスする。すると、ユーザ名とパスワードを訪ねる画面が表示される。この画面上で、ユーザは ID とパスワードを入力してログインする。このログイン画面をなくしてしまうと、不特定の人物から LifeCircle にアクセスできてしまう。行方履歴は各個人の行動を表す情報であり、むやみに公開するのは望ましくない。この処理は各個人のプライバシーを守るために必要である。

3.2.2　LifeCircle の概観

LifeCircle へのログイン処理が完了すると、図 3.1 に示すような Java アプレットが Web ブラウザ上に表示される。ユーザは、このアプレットを操作することで行方履歴や LifeIndicator の閲覧及び操作を行うことができる。

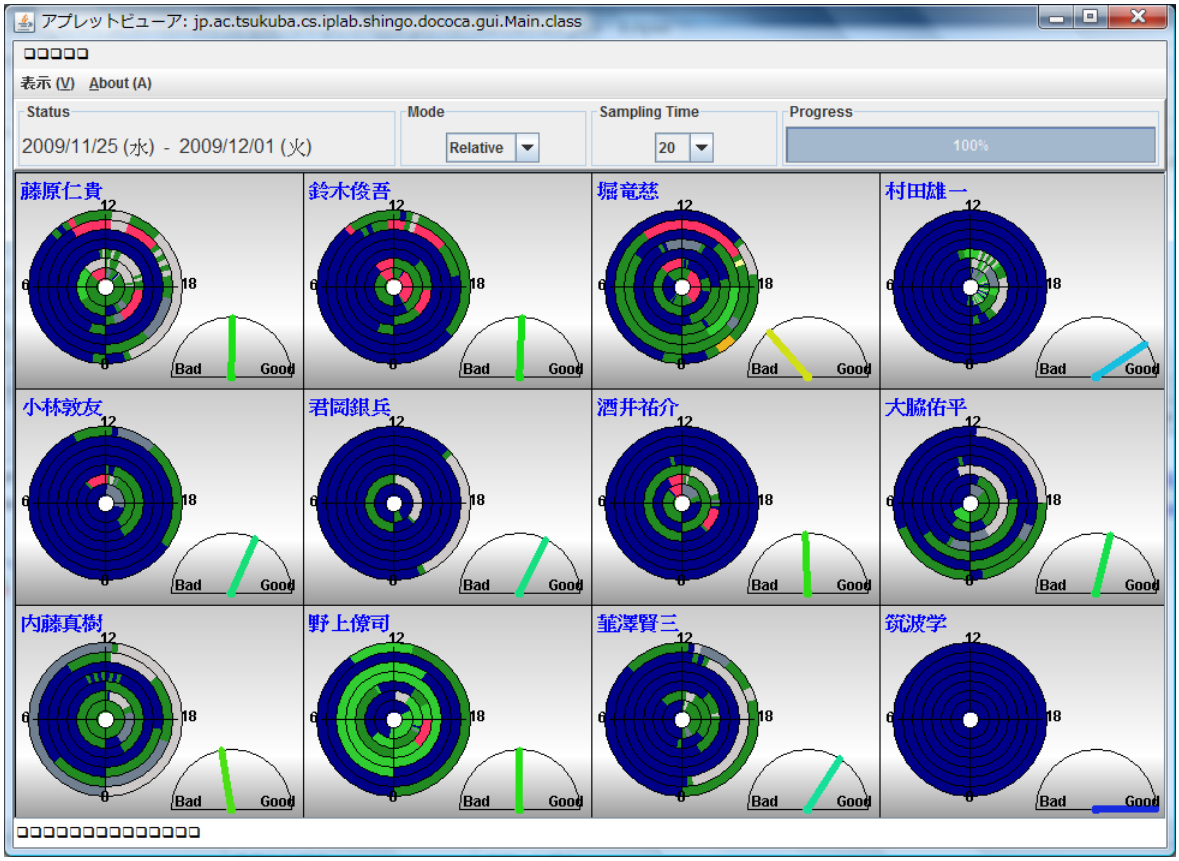

図 3.1: LifeCircle の概観

3.2.3　実装機能

現在 LifeCircle に実装している機能としては、「提示期間の変更」「比較方法の変更」「サンプリング時間の変更」がある。以下で、それぞれの機能について紹介する。

提示期間の変更

ユーザは、LifeCircle 上でキーボードの上下左右の矢印キーを押すことで、行方履歴と LifeIndicator の視覚化する期間を変更することができる。これらのキーが押されると、押されたキーに応じて提示すべき期間を計算し、設定されている比較方法およびサンプリング時間で、行方履歴と LifeIndicator の視覚化を行う。処理の進行状況は、LifeCircle 上部の右側にあるプログレスバーによって把握することができる。

図 3.2、3.3、3.4 に提示期間を変更させて表示させた例を示す。図 3.2 は、矢印キーを押して期間の変更を指示した直後の状態である。このとき、プログレスバーは 0% になっている。次に図 3.3 では、バックグラウンドで描画処理を行っている。プログレスバーを見ると、ある程度処理が進んでいることが分かる。その後、描画処理が終了すると図 3.4 のようになる。プログレスバーが 100% になると画面全体を更新する。この図を見ると、表示期間が変わり、行方履歴と LifeIndicator の再描画が行われているのが分かる。

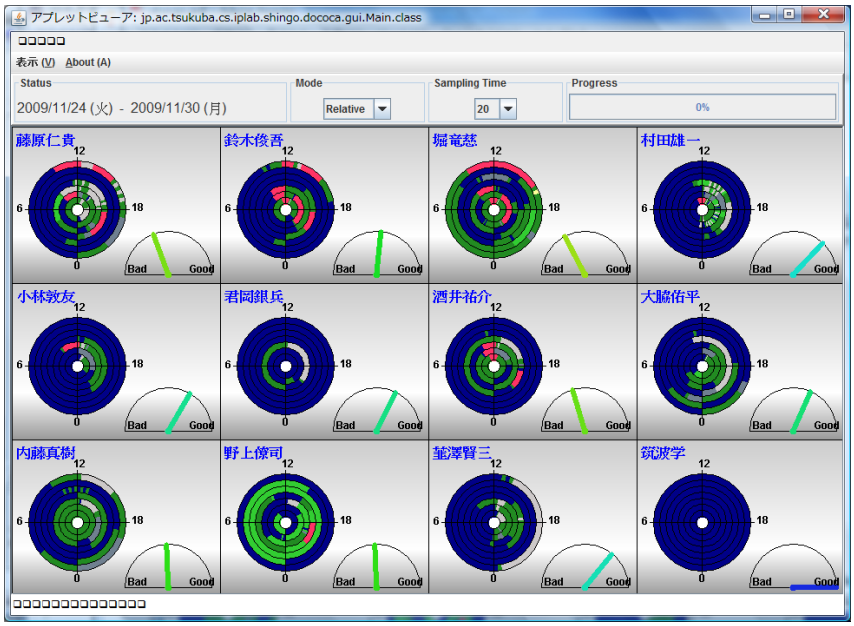

図 3.2: 提示期間を指定した直後の画面

比較方法の変更

この機能は、LifeRegularity の算出の際にどのように行方履歴を比較するかを設定する。詳しくは 4.5.4 節で述べる。比較方法は、LifeCircle 上部中央の Mode という名前のボーダーで

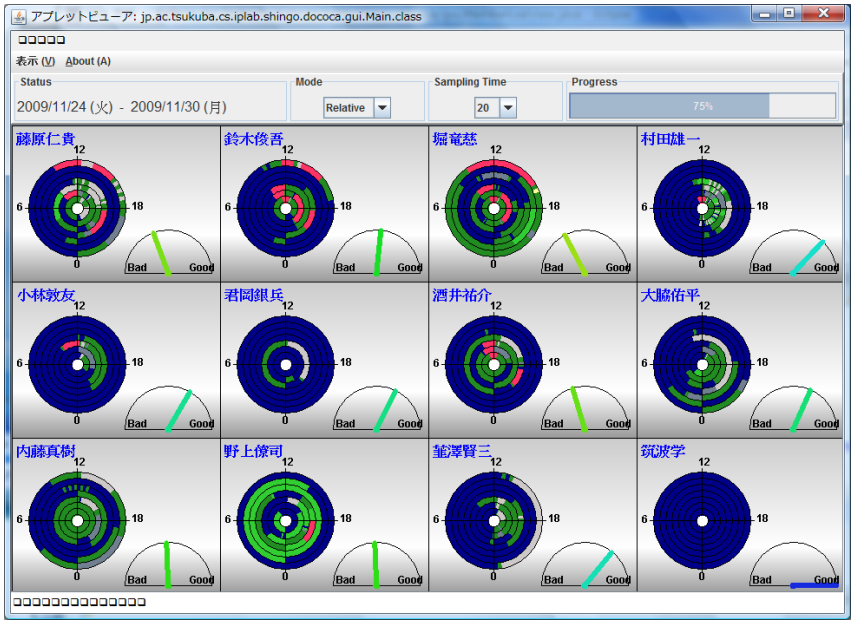

図 3.3: バックグラウンド処理中の画面

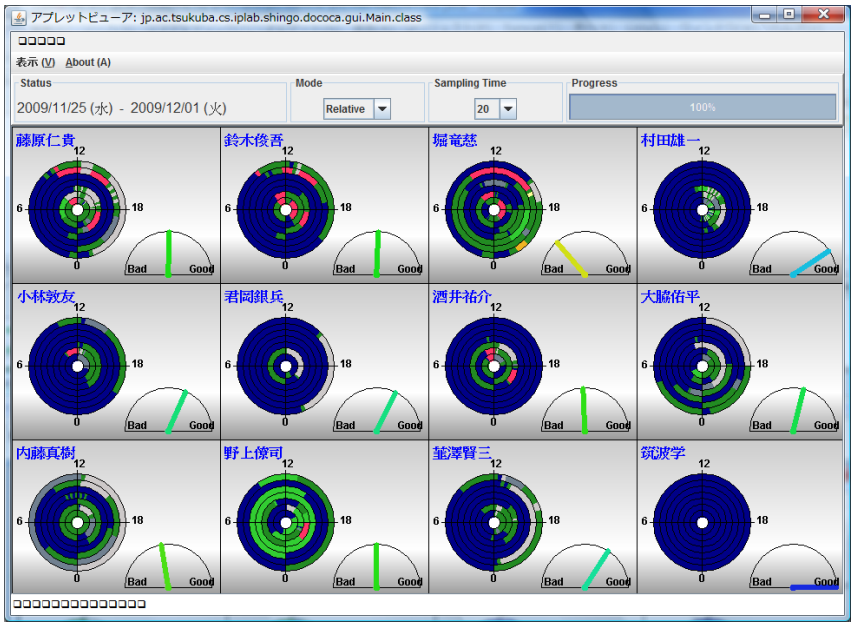

図 3.4: 再描画後の画面

囲まれたコンボボックスから選択する（図 3.5）。選択できるものには「Relative（相対比較）」「Absolute（絶対比較）」の 2 種類がある。

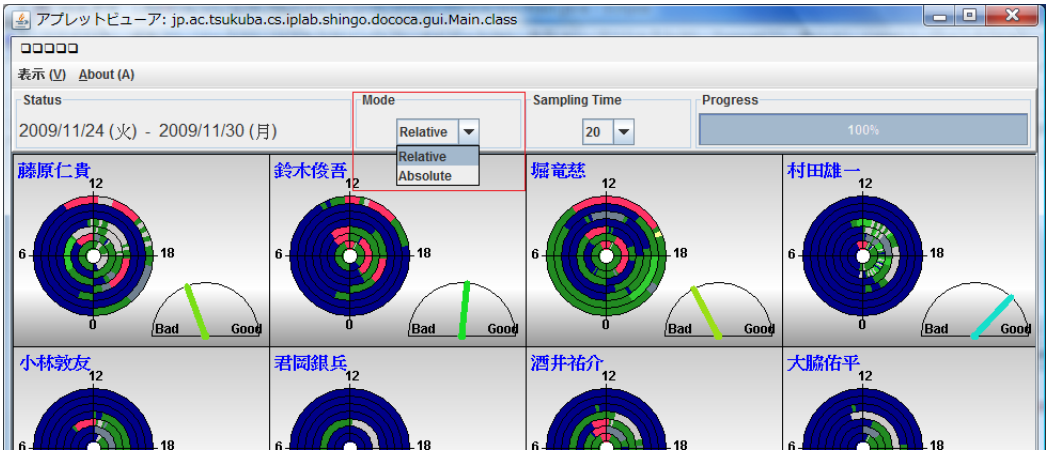

図 3.5: 比較方法の変更用コンボボックス

サンプリング時間の変更

この機能は、行方履歴をサンプリングする際の時間の間隔を設定する。設定可能な値は 10、20、30、60、120 である。単位は分である。これは、LifeCircle 上部中央やや右側にある Sampling Time という名前のボーダーで囲まれたコンボボックスから選択する（図 3.6）。

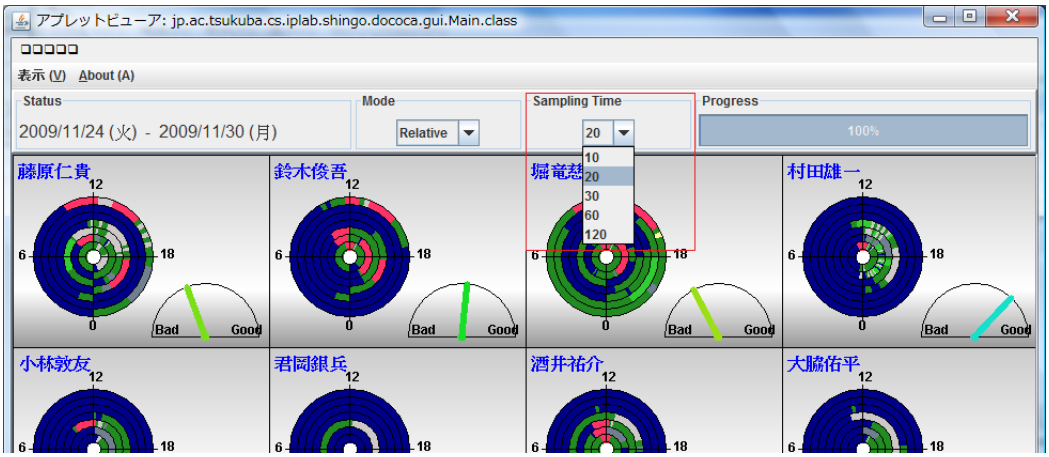

図 3.6: サンプリング時間の変更用コンボボックス

3.3 行方履歴の視覚化

LifeCircle では、一定期間の行方履歴を複数の同心円を各メンバーごとに並べることによって視覚化している。図 3.7 に行方履歴の視覚化例を示す。下側を 0 時、上側を 12 時とし、1 日

を円の 1 周として時計回りに行方履歴の描画を行っている。行方履歴は、内側の円から外側の円に向かうにつれて最近の行方履歴が並ぶように配置されている。各行方にはそれぞれ色が割り当てられており、その時刻にいた行方に併せて色が描画されるようになっている。

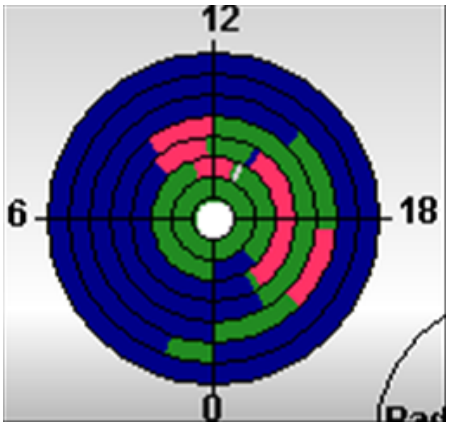

図 3.7: 行方履歴の視覚化例

行方履歴の視覚化に関して、各行方の色の対応は表 3.1 のようになっている。この色の割り当てには、その行方の活発さを考慮している。その行方が活発であるほど暖色系の色を、逆に活発ではない行方には寒色系の色になっている。

ゼミ室	研究室	学内	帰宅
赤	緑	灰	紺

表 3.1: 行方と色の対応

3.4 習慣の規則性の提示

LifeCircle では、習慣がどの程度規則的かということを図 3.8 に示す LifeIndicator によって視覚的に提示している。この LifeIndicator は、LifeRegularity の値と対応している。LifeRegularity は、行方履歴同士を一定の周期毎に分け、それらを比較することで計算を行っている。この計算の結果得られた LifeRegularity の値に対応させて、LifeIndicator の針の振れ具合と色を変化させて描画している。LifeRegulariry の具体的な計算方法については 4.5 節で述べる。

図 3.8: LifeIndicator による提示

第4章　LifeCircleの実装

この章では、本研究で作成したシステム LifeCircle の具体的な実装について述べる。

4.1　開発環境

LifeCircle を開発するにあたり、サーバー側の実装には PHP、クライアント側の実装には Java を用いている。このシステムは Java アプレットとして動作するため、利用するためには Java の実行環境である Java Runtime Environment(JRE) および Internet Explorer や Firefox などといった Web ブラウザが必要である。

4.2　システム構成

LifeCircle は、「通信モジュール」および「操作インタフェース」によって構成される。通信モジュールは、DOCoCa によって保存されている行方履歴データベースとの通信および取得したデータの変換処理を行っている。操作インタフェースでは、通信モジュールを用いて DOCoCa データベースと通信し取得したデータを基にして、行方履歴、LifeIndicator の操作および閲覧を行う。LifeCircle のシステム構成図は図 4.1 のようになっている。

4.3　DOCoCa

本研究では、DOCoCa のシステムを一部利用して実装している。ここでは、DOCoCa の概要について簡単に述べる。

4.3.1　DOCoCa の概要

作業時間にばらつきがあるオフィスなどでは、他のメンバーがいつ来て、いつ作業し、いつ帰るのかといった習慣を把握することは難しく、また習慣が分からないことをきっかけとしたコミュニケーションの欠如が問題となっている。

そこで、我々の研究室では、行方情報（誰が、いつ、どこにいるかを表す情報）を記録し、その行方履歴（行方情報の履歴）を可視化して提示する電子行方表システム DOCoCa を作成し、運用している。上記の問題に対し、現在の行方情報と同時に過去の行方履歴を提示し、メ

ンバー間で共有することによって、メンバーの行方履歴から習慣を把握することを目的としている。

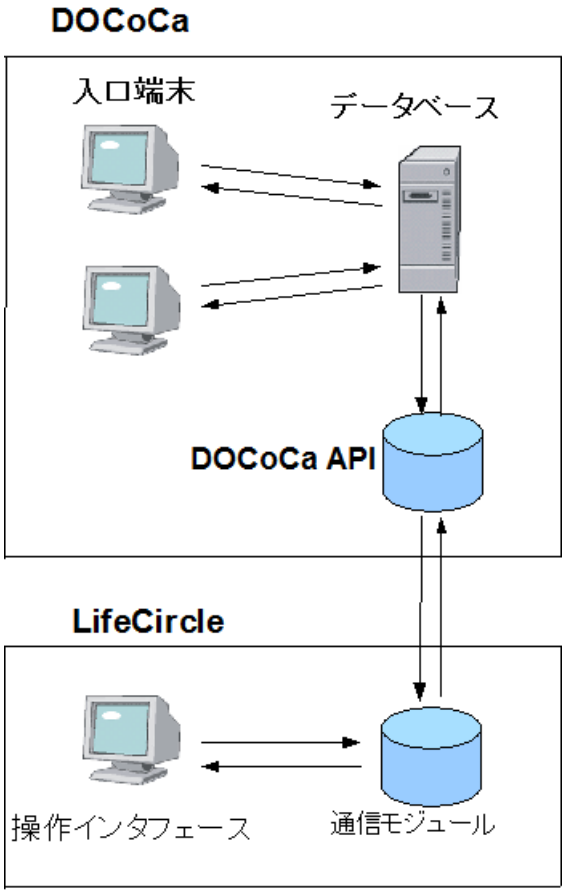

図 4.1: LifeCircle のシステム構成

4.3.2 行方情報の登録

行方情報の登録には専用の端末を用いて行っている。各研究室の入り口に、図 4.2 に示すような、FeliCa リーダを接続したタッチパネル付きの PC が設置されている。FeliCa リーダに学生証や携帯電話をかざすことで個人の認証を行い、次に画面上に表示される行方一覧から適した行方をタッチして選択することによって新たに行方情報が登録される。登録された行方情報は DOCoCa 用のデータベースに蓄積されていく。

本研究では、行方情報の登録と蓄積の部分を DOCoCa に任せることとし、DOCoCa データベースに蓄積した行方履歴を基に視覚化を行うというアプローチをとる。DOCoCa データベースには、ユーザー情報が格納された member テーブル、行方となる場所の情報が格納された

place テーブル、行方履歴が格納された timeline テーブルが存在する。LifeCircle では、必要に応じて通信モジュールを通して通信を行い、これらのテーブルの情報を取得している。

図 4.2: 部屋の入口に設置されている端末

4.4 通信モジュール

本研究では DOCoCa で記録されている行方履歴を対象としている。そのために、DOCoCa のデータベースと通信を行い、行方履歴の取得を行う必要がある。また、データベースから取得したデータはそのままでは扱いにくいため、Java で扱うことのできる形に変換することも重要である。

本研究では、DOCoCa データベースの操作には PHP を用いることとし、Java アプレット側から HTTP による通信を行うことでデータの取得を行っている。そして、得られたデータは Java クラスに変換する処理を施すことにより、オブジェクトとしてデータの操作が出来るようにした。これらは、「DOCoCa API」、「Connector API」、「DOCoCa JAXB API」、「Manager API」としてそれぞれの処理部分を API としてまとめている。DOCoCa API はサーバー側、Connector API、DOCoCa JAXB API 、Manager API はクライアント側のための API である。これら API の関係図を図 4.3 に示す。また、各 API の詳細については以下で述べる。

4.4.1 DOCoCa API

DOCoCa API は PHP を用いて実装している。この API からデータベース操作を行う際には、専用の URL にアクセスすることとし、その時にどの処理を行うかをクエリ（acrion=***）として指定する。現在利用することのできる処理には以下のようなものがある。

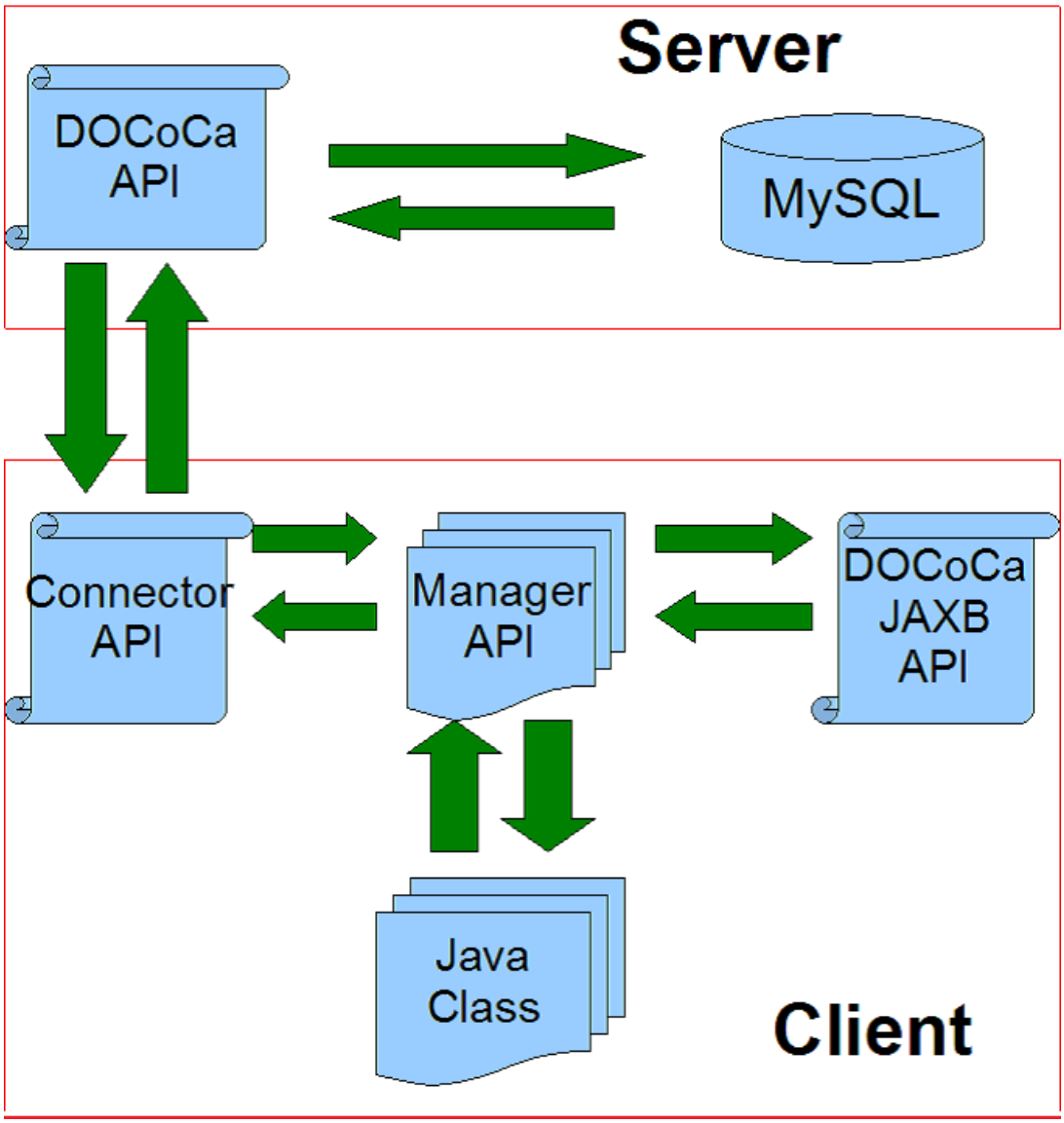

図 4.3: 各 API 間の関連図

getallmembers

DOCoCa に登録されているすべてのメンバーの情報を取得する。

getallplaces

DOCoCa に登録されているすべての場所の情報を取得する。

gettimelineswhere

DOCoCa に登録されているタイムラインの中で、where で指定された条件を満たすタイ

ムラインの情報を取得する。

例として、DOCoCa に登録されている全てのメンバーの情報が欲しい場合には、「action=getallmembers」を URL のクエリ文字列として付加しアクセスする。ここで、仮に API の URL を url とした場合には、

http://url?action=getallmembers

と入力して API にアクセスすれば良い。同様に、全ての場所の情報を得る場合は action 以下の部分を「action=getallplaces」とすれば良い。

ただし、gettimelineswhere の場合には新たなクエリとして where を要求し、データベースからデータを選択する際の WHERE 句を指定する。例えば、「メンバーの ID（nameid）が 5 で、日付（time）が 2009 年 12 月 1 日から 2009 年 12 月 8 日までのタイムラインを得たい」とする。このとき条件文は、

nameid=5 and time>=’2009-12-01 00:00:00’ and time<=’2009-12-08 23:59:59’

となるので、API にアクセスする際には、

http://url?action=gettimelineswhere&where=WHERE+nameid=5+and+time>=’2009-12-01+00:00:00’+and+time<=’2009-12-08+23:59:59’

とする。

この API は、データベースからデータを取得した後で、その得られたデータを XML 形式に変換して返すようになっている。返される XML の形式は以下のとおりである。

getallmembers

members 要素の中に member 要素が複数存在する。member 要素には、属性として固有の ID（nameid）、名前（name）、アカウント名（accountname）、不可視にするかを決めるフラグ（visible）、並び替えのための値（orderno）を持っている。

getallplaces

places 要素の中に place 要素が複数存在する。place 要素には、属性として固有の ID（placeid）、場所の名前（placename）、行方に対応させる色（color）、不可視にするかを決めるフラグ（visible）、並び替えのための値（orderno）を持っている。

gettimelineswhere

timelines 要素の中に timeline 要素が複数存在する。timeline 要素には、属性として固有の ID（timelineid）、メンバーの ID（nameid）、行方の ID（placeid）、登録された時刻（time）を持っている。

このような方針を採ることにより、DOCoCa API の利用者は状況に応じてクエリ（action、where）を変えることで、DOCoCa で保持されている様々なデータを取得することができる。また、API 内部で行われている処理を意識することなく、DOCoCa API が XML の仕様を知っ

ているだけで利用できるという点も API 化することの利点となっている。

4.4.2 Connector API

Connector API は、DOCoCa API と Java プログラムとの通信部分を扱いやすいようにラップした Java クラス群である。DOCoCa データベースからデータを取得するためには、4.4.1 節に述べたように、DOCoCa API に対して HTTP による通信を行う必要があり、DOCoCa API の仕様に適した URL に送信しなければならない。しかし、Connector API では DOCoCa API との通信処理が隠ぺいされており、この API で提供されているメソッドを呼び出すことにより DOCoCa データベースからのデータ取得を可能となっている。

Connector API は、DOCoCa API が提供する機能それぞれに対応するメソッドを持っている。この API が提供するメソッドには以下のようなものがある。

Connector.getAllMembers()
DOCoCa データベースに登録されているすべてのメンバーの情報を取得する。

Connector.getAllPlaces()
DOCoCa データベースに登録されているすべての行方の情報を取得する。

Connector.getTimeline(String where)
DOCoCa データベースに登録されている行方履歴のうち、where で指定された条件を満たす行方履歴の情報を取得する。

これらのメソッドは、呼び出されると DOCoCa API に HTTP リクエストを送信し XML を取得するという処理を行っている。Connector.getAllMembers、Connector.getAllPlaces、Connector.getTimelines メソッドはそれぞれ DOCoCa API の action である getallmembers、getallplaces、gettimelineswhere に対応している。このクラスによって得た XML は DOCoCa JAXB API に渡される。

4.4.3 DOCoCa JAXB API

DOCoCa JAXB API は、Manager API から呼び出され、Connector API によって得られる XML から Java クラスに変換する処理を行う。この API では、Java 標準の XML 操作 API である JAXB の技術を利用している。

JAXB とは Java Architecture for XML Binding のことを指し、Java のクラスを XML で表現可能にする仕様のことである。この JAXB を用いることにより、Java のオブジェクトを XML にシリアライズすること、また逆に XML から Java オブジェクトにデシリアライズすること

ができる。そのため、Java プログラム内に XML を処理するためのルーチンを新たに実装する必要が無くなる。この技術を利用するためには、XML と Java クラスとの対応関係を示すための XML スキーマファイルが必要になる。

DOCoCa JAXB API では、JAXB によって DOCoCa データベースに対応する XML スキーマファイル（member.xsd、place.xsd、timeline.xsd）を記述し、「xjc」コマンドにより変換された Java クラスを利用している。それぞれの XML スキーマファイルには、XML の論理的構造が記述してある。例えば、DOCoCa データベースの member テーブルに対応する Java クラスを作成するための XML スキーマ（member.xsd）は以下のようになる。

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="members">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="member" maxOccurs="unbounded" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="member">
      <xs:complexType>
        <xs:attribute name="nameid" type="xs:int" />
        <xs:attribute name="name" type="xs:string" />
        <xs:attribute name="accountname" type="xs:string" />
        <xs:attribute name="visible" type="xs:int" default="1"/>
        <xs:attribute name="orderno" type="xs:int" default="0"/>
      </xs:complexType>
    </xs:element>

</xs:schema>
```

このスキーマは、「members 要素の中には 0 以上の member 要素があり、member 要素には属性として nameid、name、accoutname、visible、orderno を持っている」ということを表している。place.xsd、timeline.xsd の場合も member.xsd と同様に、それぞれ place テーブル、timeline テーブルに対応するスキーマを記述している。place.xsd および timeline.xsd のソースコードは、member.xsd と併せて本論文の付録として載せているので参照してほしい。

DOCoCa JAXB API は、XML スキーマを基に生成されたクラス群と Java API である javax.xml.bind パッケージのクラス群とを組み合わせて作成している。この API により、Connector API に

よって得られる XML を DOCoCa のデータベースに対応した Java クラスに変換することが可能になる。

4.4.4 Manager API

Manager API は、Connector API と JAXB API の仲介を行う Java クラス群である。MemberManager、PlaceManager、TimelineManager の 3 つのクラスで構成されている。それぞれのクラスは、DOCoCa データベースの member、place、timeline テーブルに対応している。Connector API によって得られた XML を JAXB API に渡し、対応する Java クラスに変換させ、そしてそのインスタンスを管理する機能を持っている。

この Manager API が管理するクラスは、JAXB API によって生成されるクラスをさらに変換したものである。JAXB API によって得られるクラスは、XML スキーマによって定義された属性がクラスのフィールドに対応しているが、そのフィールドは文字列型や Integer 型といった基本的な型になっている。本研究で扱う場合にはより扱いやすくするために、JAXB API で得られるクラスを基にして、さらにラップしたクラスを生成するようにしている。

4.5 LifeIndicator による視覚化

本研究では、以下の手順に従って LifeRegularity の算出および LifeIndicator による視覚化を行っている。次節より、それぞれのステップで行う処理の詳細について述べる。

1. LifeRegularity 算出期間の指定
2. 対象期間のサンプリング
3. LifeRegularity の算出
4. LifeIndicator による視覚化

4.5.1 LifeRegularity 算出期間の指定

まず最初のステップとして、行方履歴及び LifeIndicator を視覚化し提示する期間を指定する。

期間の指定にはキーボードの上下左右キーを用いる。ユーザはこれらのキーを使用することで、行方履歴の可視化及び LifeRegularity の計算対象となる期間を選択する。

各キーに割り当てられている機能は以下のとおりである。

↑
: 現在表示されている期間を 1 週間前にずらして行方履歴及び LifeIndicator の提示を行う。

↓

現在表示されている期間を 1 週間後にずらして行方履歴及び LifeIndicator の提示を行う。

←

現在表示されている期間を 1 日前にずらして行方履歴及び LifeIndicator の提示を行う。

→

現在表示されている期間より 1 日後にずらして行方履歴及び LifeIndicator の提示を行う。

4.5.2 対象期間のサンプリング

LifeRegularity の算出対象とする期間を指定すると、予め設定された時間（サンプリングタイム）単位（例えば 10 分、20 分など）毎に行方履歴のサンプリングを行う。サンプリングの例を図 4.4 に示す。この図は、一定の間隔で行方履歴を分割し、同じ時刻における行方のサンプリングを行っている。色が同じ部分は同じ行方を表している。

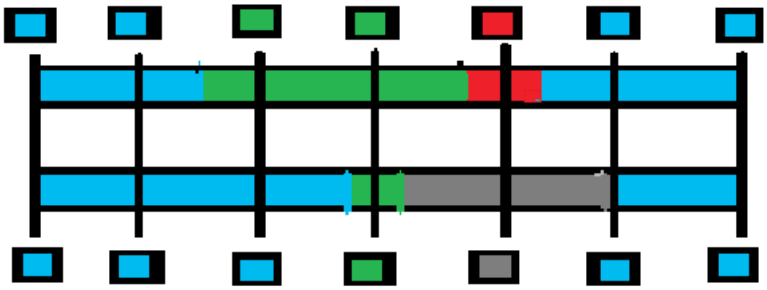

図 4.4: 行方履歴のサンプリング例

このサンプリングタイムは、LifeCircle の操作インタフェースにある設定画面から変更が可能である。この値を小さくする（サンプリングを細かくする）ほど、最終的により精度の高い LifeRegularity を得ることができるが、その代償として計算コストがかかるようになる。

4.5.3 類似度算出

このステップでは、前ステップにおいてサンプリングされた行方履歴を比較し、類似度を算出する処理を行う。

類似度とは、サンプリングされた行方同士を比較したときに、どれだけその行方同士が性質的に近いかを表す値である。本研究では、類似度を「行方同士の内積」と定義することとした。

各行方（帰宅、ゼミ室、学内など）には、それぞれ 2 次元の単位ベクトル p を設定している。本論文では、このベクトルを行方ベクトルと呼ぶことにする。例えば、「帰宅」は $(1, 0)$、「ゼミ室」は $(0, 1)$、「学内」は $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ のようにしている。図 4.5 に行方ベクトルの例を示す。図中の赤い矢印で表現されているのが「ゼミ室」、青い矢印が「帰宅」、そして灰色の矢印が「学内」を表す行方ベクトルとなっている。

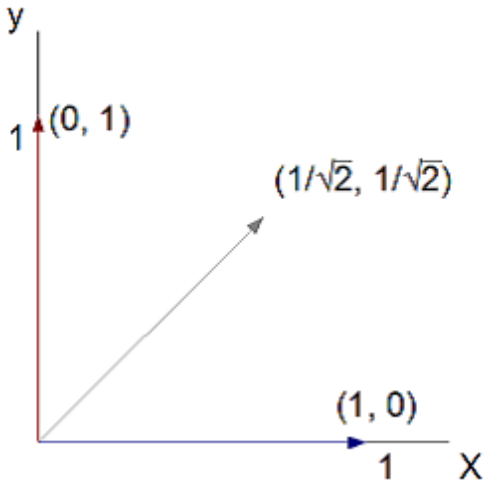

図 4.5: 行方ベクトルの例

通常内積を求める際には以下の式を用いる。x、y を 2 次元のベクトル $x : (x_1, x_2)$、$y : (y_1, y_2)$ としたとき、

$$x \cdot y = x_1 * y_1 + x_2 * y_2 \tag{4.1}$$

または、

$$x \cdot y = ||x||||y|| \cos\theta \tag{4.2}$$

と表すことができる。ここで、$\cos\theta$ の値域が、

$$-1 \leq \cos\theta \leq 1 \tag{4.3}$$

であることを考慮すると、式 4.2 より、

$$-||x||||y|| \leq x \cdot y \leq ||x||||y|| \tag{4.4}$$

となる。つまり、x と y の内積 $x \cdot y$ の値域は式 4.4 の範囲になるということが分かる。このとき、x、y がそれぞれ単位ベクトルであるという条件を加えると、

$$||x|| = ||y|| = 1 \tag{4.5}$$

となるので、式 4.4 より

$$-1 \leq x \cdot y \leq 1 \tag{4.6}$$

と表すことができる。すなわち、ベクトル x、y を正規化する、もしくは単位ベクトルを用いることによって、内積 $x \cdot y$ の値域が式 4.6 で表される範囲にあることが保証される。

この性質を用いると、次のステップで行うこととなる LifeRegularity を算出する際の計算量を減らすことができる。本研究では、各行方ベクトル p を 2 次元の単位ベクトルとすることにしている。

このベクトル p は、その行方の活発さを考慮して決定している。現状では著者がそれぞれの値を設定している。このベクトルの与え方によって、異なる行方同士を比較した場合に、類似度の値に違いが出てくる。

例えば、(1)「学内」と「ゼミ室」、(2)「帰宅」と「ゼミ室」、が比較対象となったとする。(1) の場合では「学内」「ゼミ室」の両方が大学内の行方情報である。それに対して、(2) の場合では「ゼミ室」は大学内であるが、「帰宅」は大学外の行方情報となっている。このとき、活発さという観点から見ると、(1) の方が (2) よりも活発であるといえる。言い換えると、(1) は活発さの差が小さく、(2) は活発さの差が大きいということを意味している。こうした活発さの情報を計算に含ませるために、活発さの差が小さい行方ベクトル同士がなす角度を小さく、逆に差が大きいベクトル同士がなす角度が大きくなるように配置している。図 4.6、4.7 にそれぞれ活発さの差が小さい場合と大きい場合の行方ベクトルのイメージを示す。図 4.6 ではベクトル同士がなす角度が小さくなることで、ベクトル同士の内積をとったときの値は 1 に近づく。逆に、図 4.7 のようなベクトル同士の内積をとると値は 0 に近づくことになる。このようにすることで、ある時刻における行方情報に違いが生じた際に、ベクトルの内積が各行方同士の活発さの差に応じて変化するため、比較対象となる行方同士の活発さの差を考慮した行方同士の類似度を算出することが可能となる。

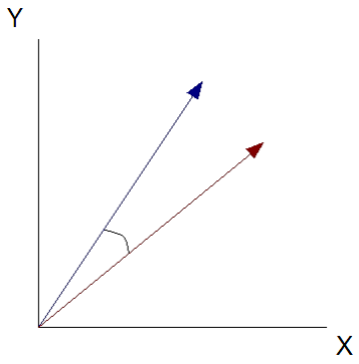

図 4.6: 活発さの差が小さい行方ベクトルの設定例

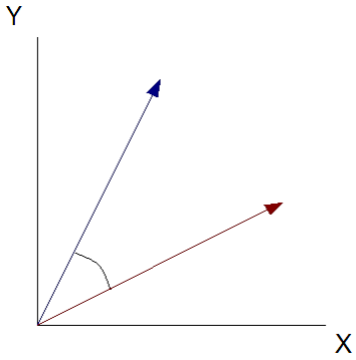

図 4.7: 活発さの差が大きい行方ベクトルの設定例

4.5.4 LifeRegularity の算出

LifeRegularity の算出には自己相関関数を参考にした計算方法を採用している。自己相関関数は式 4.7 のように表現される。

$$C(\tau) = \frac{1}{N}\sum_{n=1}^{N} x(t_n) \cdot x(t_n + \tau) \tag{4.7}$$

この式は、ある信号 $x(t)$ について、タイムラグ τ があるときに n が 1 から N の範囲まで変化するときの $x(t_n)$ と $x(t_n + \tau)$ の間にどの程度相関があるのかを求めるために使用される。本研究では、$x(t)$ に行方履歴を対応させ、一定の周期 τ だけずらした行方履歴と比較することによって LifeRegularity の算出を行っている。

自己相関関数を基にした LifeRegularity の計算手順は以下のようにして行っている。

1. サンプリングされた行方履歴を 1 日毎のリストに分解。
2. 周期ごとに行方履歴のリストをグループ化。
3. 各グループ間で比較を行い、類似度の総和を算出。
4. 全体のサンプリング回数で割る。

LifeCircle では、行方履歴の比較の方法として「絶対比較」と「相対比較」の 2 種類が行えるようにしている。各比較方法のイメージを図 4.8 に示す。t1 から t4 はそれぞれ比較する行方履歴の 1 周期を表している。相対比較では、比較対象の行方履歴を隣接するもの同士にし、順番に 1 つずらして隣接する行方履歴を評価していくものである。図の例では、t1 と t2、t2 と t3、t3 と t4 を比較する。絶対比較は、基準となる履歴を順番に変えていくのは相対比較を変わらないが、比較対象とする履歴は比較期間のサイクルの内最も最新のものとする。図の例では、t1 と t4、t2 と t4、t3 と t4 を比較する。

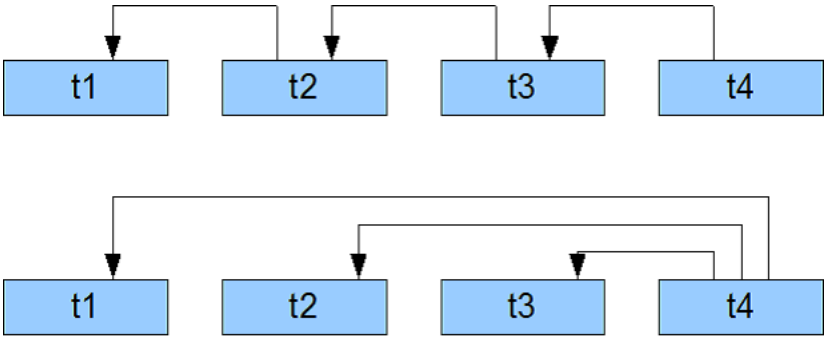

図 4.8: 相対比較（上）と絶対比較（下）

4.5.5 LifeIndicator による視覚化

このステップは、前ステップにおいて得られた LifeRegularity を用いて LifeIndicator による視覚化を行っている。LifeIndicator では、LifeRegularity の値に対応させて針の振れ具合及び針の色を調節して提示している。そのため、この針の振れ具合をユーザが閲覧することによって、視覚的に自分がどの程度規則的か、または不規則なのかを確認することができる。

LifeIndicator は半円の左側に針がいくと不規則、右側に針がいくと規則的な事を表す。前のステップにおいて得られる LifeRegularity の値は 0〜1 となっており、これは最も不規則なときを 0、最も規則的なときを 1 として数値化している。この LifeRegularity を LifeIndicator に対応させる際には、最も左側を 0、最も右側を 1 になるようにしている。また、色の対応に関しては、LifeIndicator の左側に針が寄るほど針の色が赤くなっていき、右側に寄っていくほど青になっていく。色の変更には HSV 表色系を用いることで赤から青へと滑らかに色が変化するようにしている。

この LifeIndicator によって LifeRegularity を視覚化した例を図 4.9、4.10 に示す。図 4.9 では、LifeIndicator の針がやや左側に寄っており、針の色も黄緑になっている。これは、生活習慣がやや不規則な傾向にあることを示している。それに対し、図 4.10 では針が右側に傾き、その色が水色になっている。この場合は、さきほどの例とは逆に生活習慣が規則的であることを示している。

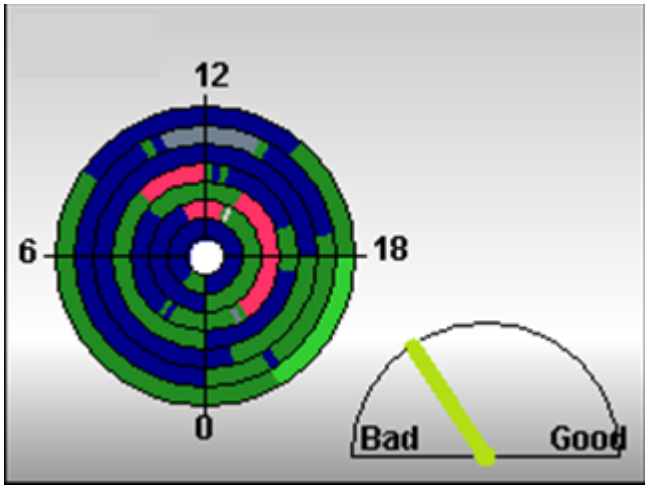

図 4.9: 習慣が不規則な例

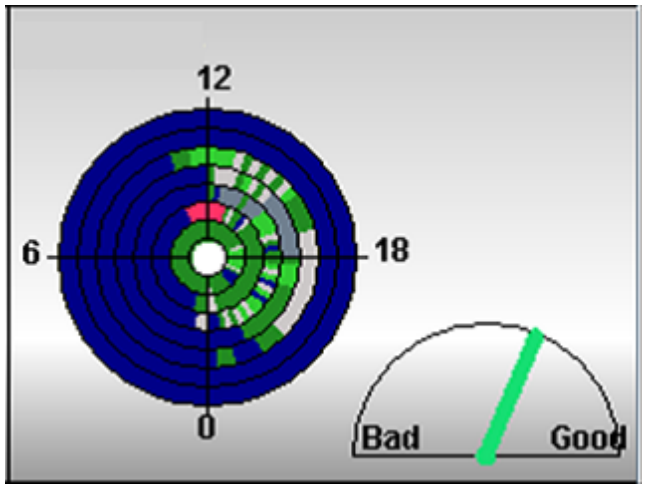

図 4.10: 習慣が規則的な例

第5章　考察

本研究では、行方履歴と LifeIndicator を同時に提示し、メンバー間で共有することにより生活の振り返りを支援することを目的としている。実際に運用してみると、行方履歴から過去の行動を思い出し、LifeIndicator による針の振れ具合によって、どの程度規則的な生活をしているかを視覚的に確認することはできる。

しかしながら、LifeIndicator の視覚化に関して問題点が 2 つあることが分かった。1 つは、行動の時間が多少ずれた場合に LifeIndicator が不規則な傾向になってしまうという点、そしてもう 1 つはあまり望ましくない生活習慣をしている場合でも、LifeIndicator が規則的であると評価されることがあるという点である。

習慣が規則的であるということには、毎日同じ時間に同じ行動をすることも含まれる。ただ、多少時間がずれたとしても、同様の行動をしていれば規則的であるというべきである。現在の実装では、LifeRegularity を算出する際に、ある周期 τ だけずらした後で同じ時刻同士における行方のサンプリングを行ったうえで LifeRegularity を算出している。そのため、行方履歴全体の推移が同様でも時間のずれの分だけ LifeRegularity の値も低くなってしまう。運用結果から得られた履歴の中で、不規則な傾向にあると判断されている例を図 5.1 に示す。また、同様の変更推移があるにも関わらず不規則に評価されてしまう例を図 5.2 に示す。

図 5.2 の場合、t1 と t2 の間にはサンプリング 1 回分の時間のずれがあるものの、行方履歴の推移は同じである。同一時刻でサンプリングを行うと 8 回のサンプリングの内 4 回が不一致となってしまい（図の X 印の部分）、結果として得られる LifeRegularity の値が低くなってしまう。この問題に対する対策方法としては、行方履歴の変更パターンを LifeRegularity の算出の際に考慮に入れ、変更パターンが一致したときには LifeRegularity の値に補正をかけるという手段がある。

また、行方履歴の比較をする際に、あまり望ましくない生活習慣（例えば夜型の生活）をしている場合でも、基準となる履歴と比較対象の履歴が一致すると規則的であると評価される問題にも対処する必要がある。例として、1 週間全く学校に来ていないにも関わらず LifeIndicator による評価は最大値になっている場合を図 5.3 に示す。生活が規則的であるというのは、通常朝起きて昼間に活動をして夜に寝るという昼型の行動パターンを指す。夜型の生活を続けていると、人間が本来持っている生体リズムが崩れ、その結果体調を損ねる原因になる。この問題には、予め理想的な生活パターンを設定しておき、行方履歴を比較する場合にはその理想的なパターンと比較することによって、夜型の生活をしている場合には LifeRegularity を低くすることができるようになると考えられる。

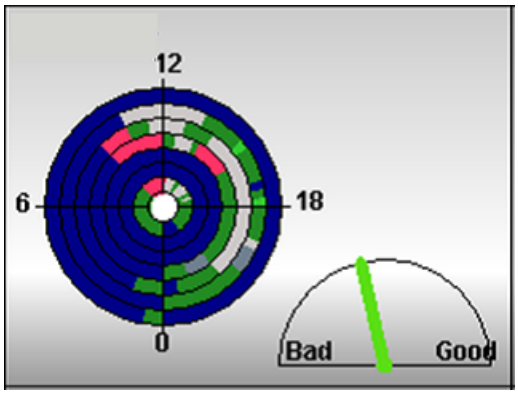

図 5.1: 不規則な傾向にあると判断されている例

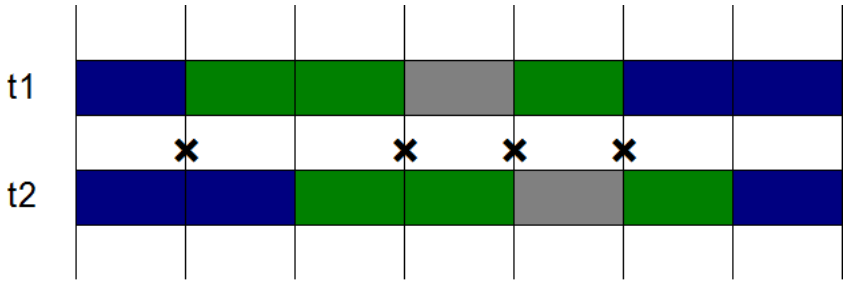

図 5.2: 同様の変更推移でも不規則となる例

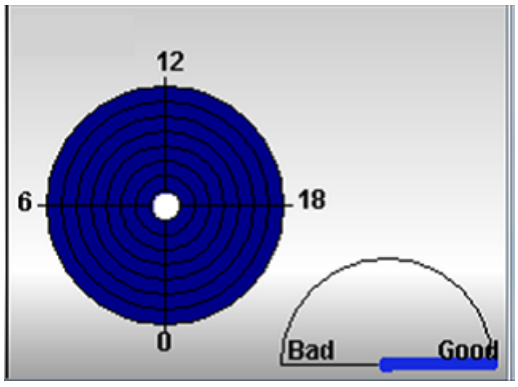

図 5.3: 1 週間全く学校に来ていない例

第6章　まとめと今後の課題

本研究では、過去の行方履歴を参照しつつ生活の規則性を視覚的に提示するシステムである LifeCircle の実装を行った。このシステムにより、過去の行方を確認しながら生活の振り返りを支援するシステムを構築した。

今後の課題としては、LifeRegularity 算出アルゴリズムの改良が挙げられる。現在の実装では、一定の周期をずらしたあとは、同一時刻における行方をサンプリングし比較していくという仕様になっている。そのため、本来は規則的であると評価されるべき行方履歴に対して得られる LifeRegularity の値が低めに算出されてしまい、それにともなって LifeIndicator による提示の際には針が不規則な側へと傾いてしまうといった問題がある。この問題に対しては、行方の変更パターンや各行方の滞在時間を利用するなどの時間のずれを考慮することによって対処ができると考えられる。

また、行方履歴同士の比較を行う際、基準となる行方履歴と比較対象となる行方履歴が一致すれば高い LifeRegularity が得られるのであるが、この時に基準となる行方履歴があまり好ましくない状況であっても高評価となってしまうという問題もある。こちらに対しては、本来好ましいとされる手本となる行方履歴を基準として LifeRegularity を算出するようにすることによって対処できると考える。

今後は、上記に述べた問題点を改善するために LifeCircle の修正、および改良を行っていきたい。

謝辞

本論文を執筆するにあたって、指導教員である田中二郎先生をはじめ、志築文太郎先生、高橋伸先生、三末和男先生には、丁寧な指導および貴重な御意見をいただきました．特に、志築先生には、研究に関するいろはから具体的なアドバイス、懇切丁寧な指導をいただきました．ここに深く感謝いたします。

また、インタラクティブプログラミング研究室（IPLAB）のメンバー、特に WAVE チームの皆様には、チームゼミや普段の研究生活を通して、研究に関する様々な御意見、アドバイスをいただきました。ここに深く感謝いたします。最後に、ここまで学生生活を支えてくださった家族、友人やお世話になった全ての皆様に感謝いたします。本当にありがとうございました。

参考文献

[1] 佐藤 陽治, 斎藤 滋雄, 上岡 洋晴. 大学生の精神的健康度とライフスタイルとの関係. 学習院大学スポーツ・健康科学センター紀要 6, pp. 9–30, 1998.

[2] 森本 兼曩. ライフスタイルと健康. 全日本鍼灸学会雑誌, 2003 年第 53 巻 2 号, pp. 141–149, 2003.

[3] 青木 茂樹, 岩井 嘉男, 大西 正輝, 小島 篤博. 人物の位置・姿勢に注目した行動パターンの学習・認識と非日常状態検出への応用. 電子情報通信学会論文誌, Vol. J87-D-II, No.5, pp. 1083-1093, 2004 年 5 月.

[4] 山越 恭子. ワークリズムを用いた面会支援システムの構築. HIS (ヒューマンインタフェースシンポジウム) 2003, pp. 7441–7444, 2003.

[5] 土持 幸久, 高橋 伸, 田中 二郎. プライバシを考慮しつつユーザの状況・状態を推定と提示を行うシステム. マルチメディア, 分散, 協調とモバイルシンポジウム（DICOMO2006）論文集, 情報処理学会, pp. 497–500, 2006.

[6] M. Webera, M. Alexa, W. Muller. Visualizing Time-Series on Spirals. IEEE Symposium on Information Visualization 2001, pp. 7–14, 2001.

[7] J. V. Carlis, J. A. Konstan. Interactive visualization of serial periodic data. Proceedings of the 11th annual ACM symposium on User interface software and technology, pp. 29–38, 1998.

[8] 藤原 仁貴, 村田 雄一, 堀 竜慈, 鈴木 俊吾, 志築 文太郎, 田中 二郎. DOCoCa: 行方履歴を用いてメンバーの習慣を可視化する電子行方表. 第 17 回インタラクティブシステムとソフトウェアに関するワークショップ (WISS2009), 日本ソフトウェア科学会, pp. 115–116, 2009.

付録

member.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema version="1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="members">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="member" maxOccurs="unbounded" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="member">
      <xs:complexType>
        <xs:attribute name="nameid" type="xs:int" />
        <xs:attribute name="name" type="xs:string" />
        <xs:attribute name="accountname" type="xs:string" />
        <xs:attribute name="visible" type="xs:int" default="1"/>
        <xs:attribute name="orderno" type="xs:int" default="0"/>
      </xs:complexType>
    </xs:element>

</xs:schema>
```

place.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<xs:schema version="1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
  <xs:element name="places">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="place" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="place">
    <xs:complexType>
      <xs:attribute name="placeid" type="xs:int"/>
      <xs:attribute name="placename" type="xs:string"/>
      <xs:attribute name="color" type="xs:long" />
      <xs:attribute name="visible" type="xs:int" default="1"/>
      <xs:attribute name="orderno" type="xs:int" default="0"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

timeline.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema version="1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="timelines">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="timeline" maxOccurs="unbounded" minOccurs="0" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="timeline">
      <xs:complexType>
        <xs:attribute name="timelineid" type="xs:int"/>
```

```
        <xs:attribute name="nameid" type="xs:int"/>
        <xs:attribute name="placeid" type="xs:int"/>
        <xs:attribute name="time" type="xs:string"/>
      </xs:complexType>
    </xs:element>

</xs:schema>
```