筑波大学大学院博士課程

システム情報工学研究科特定課題研究報告書

進化的計算の実験可視化システムの開発
－テスト計画の立案と実践－

人見　圭一郎

修士（工学）

（コンピュータサイエンス専攻）

指導教員　田中　二郎

２０１４年　３月

概要

本報告書は、筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻高度 IT 人材育成のための実践的ソフトウェア開発専修プログラムにおける研究開発プロジェクトの成果をまとめたものである。

本プロジェクトの顧客は、筑波大学大学院システム情工学研究科で進化的計算を利用した研究を行っている Aranha Claus De Castro 助教である。進化的計算を利用した計算機実験は、「計算負荷が高い」、「実験に使用するパラメータ値の設定が経験的になり、パラメータを頻繁に変更して実験する」といった特徴があり、これらの特徴が顧客の進化的計算の実験の障害となっていた。本プロジェクトではこれらの問題を解決するために、実験に必要なファイル群を管理するための「実験パッケージ管理」機能、実験を実行・停止するための「実行管理」機能、実験の実行状態を監視する「実行状態監視」機能、パラメータ作成を支援するための「パラメータファイル作成」機能、実験結果を可視化する「実験結果可視化」機能を提供するシステムを開発した。開発活動は筆者を含めた学生 4 人で行い、開発手法には、アジャイル手法を利用した。

筆者は、このシステムの開発においてテスト計画の立案と実践、そして、2 つの機能を実現することにより開発活動に貢献した。テスト計画の立案では、テスト全体の計画を立てるとともに、システムの品質を保証する上で重要となるテストの十分性や網羅性を確保するための対策を検討した。開発面では、実験パッケージ管理機能、実行状態監視機能の一部である実験進捗表示機能を開発した。

目次

図目次

表目次

第1章　はじめに

1.1　プロジェクトの背景

本プロジェクトは、筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻における高度 IT 人材育成のための実践的ソフトウェア開発専修プログラムの一環として実施された研究開発プロジェクトである。研究開発プロジェクトでは、学生 4、5 名でチームを作り、実顧客を対象として顧客が抱える問題を解決するためのシステムを開発・提供する。

筆者が所属するチームは、同学に所属する進化的計算の研究者 Aranha Claus De Castro 氏を顧客として、顧客の実験における問題を解決するためのシステムの開発を行うこととなった。

1.2　進化的計算

進化的計算（Evolutionary Computation、以後 EC と略記） [1] [2]は、生物の進化のプロセスをモデル化した計算手法の総称を指す。EC は、最適化問題や地震予測モデルや生体医療情報の可視化などの分野で広く応用されている。代表的な EC の手法として、遺伝的アルゴリズム（Genetic Algorithm、以後 GA と略記）、遺伝的プログラミング（Genetic Programming）がある。本報告書では、EC の一つである GA を例に EC の説明していく。

GA は、初めに生物個体の遺伝子に見立てた解を要素とした集合を用意する。次に、この集合を初期集合として、解同士を混ぜ合わせる「交叉」、解の一部をランダムに変更する「突然変異」などの遺伝子操作をすることで、新たな遺伝子の集合を作り出していく。そして、新たに作成された集合の中から環境への適応度（評価関数値）が高い遺伝子を選択し、初期集団と同じ個数の次世代の遺伝子の集合を作り出す。 [3]これにより、前世代の遺伝子よりも優秀な遺伝子が作成される。

図 1-1 は、GA の流れを示したものである。この繰り返しのプロセスを繰り返していく（本論文では、この繰り返しの回数を世代数と呼ぶ）ことで、集合における各遺伝子の適応度が上昇し、より優秀な解を得ることができる。このように GA はヒューリスティックなアルゴリズムであるため一意的な正しい解を求めることは期待できないが、近似的に優良な解を求める方法として優れている。また、GA では、常に同じ解が得られるとは限らないので、同じ実験を複数回実験することが多い（この回数を本報告書では、反復回数と呼ぶ）。

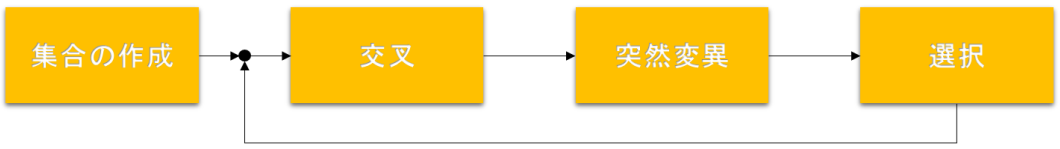


図 1-1　GA の流れ

1.3　進化的計算を利用した計算機実験の特徴

進化的計算を利用した計算機実験には 2 つの特徴がある。一つは計算コストが高いということである。GA は複数の解を保持しながらそれらを集団として改善していく多点探索型の手法である。一般に、多点探索型の手法は計算コストが高くなると言われている。加えて、

GA では反復計算を何度も行う。そのため、計算負荷が非常に高くなってしまうのである。[4] [5]

もう一つはパラメータの設定が経験的になってしまうということである。GA では、数多くのパラメータを利用するが、それらを決定する一般的な規則は存在しない。そのため、実験者の経験や勘によって決定するしかない。 [5] [6] [7]

1.4 顧客について

本プロジェクトの顧客は、課題担当教員でもある Aranha Claus De Castro 氏である。顧客は、筑波大学大学院システム情報工学研究科の助教である。顧客は、進化的計算や機械学習 [8]に関する研究を行っている。現在は特に、進化的計算手法を利用して金融ポートフォリオの最適化 [9]や多次元データの二次元座標への変換 [10]などの研究を行っている。

1.5 プロジェクトの目的

プロジェクト開始以前、1.3 節で挙げた EC の特徴により、顧客の EC を利用した計算機実験に 5 つの問題が生じていた。本プロジェクトの目的は、この 5 つの問題を解消し、顧客の EC を利用した計算機実験を効率化することである。本プロジェクトでは IT システム（以下、GAM と表記）を開発することで目的の実現を目指す。

本プロジェクトで対象としている問題は、EC の一般的な特徴により発生しているものが多い。そのため、顧客だけではなく他の EC の研究者に対しても十分に適用可能であると考えられる。しかし、本プロジェクトにおいては、まず GAM を利用するユーザは顧客に限定し、顧客が満足して利用できるものを開発することとした。

1.6 本報告書の構成

本節では、次章以降の構成について述べる。2 章では現状の顧客の実験方法やその問題点について考察する。3 章ではその問題を解決するためのアプローチや開発したシステムの概要を説明し、4 章では開発活動を実施する体制について述べる。5 章では開発活動における筆者の貢献を、6 章ではプロジェクト管理において筆者が貢献したテスト計画について説明する。最後に、7 章では開発したシステムに関する評価を述べる。

第2章　　現状の実験方法の分析

この章では、現在顧客がどのように EC を利用した計算機実験を行っているのか、また、その実験方法における問題を示す。

2.1　　現状の実験の流れ

現状の実験方法の流れを図 2-1 のように大別し説明する。まず初めに、実験者は実験を作成し、それ以降は「実行プログラムの実行」、「実行プロセスの監視」、「実行結果の分析」、「実行結果の分析」、「パラメータ値の変更」という実験プロセスを順に繰り返し実施する。以下で、個々のプロセスについて詳しく説明する。

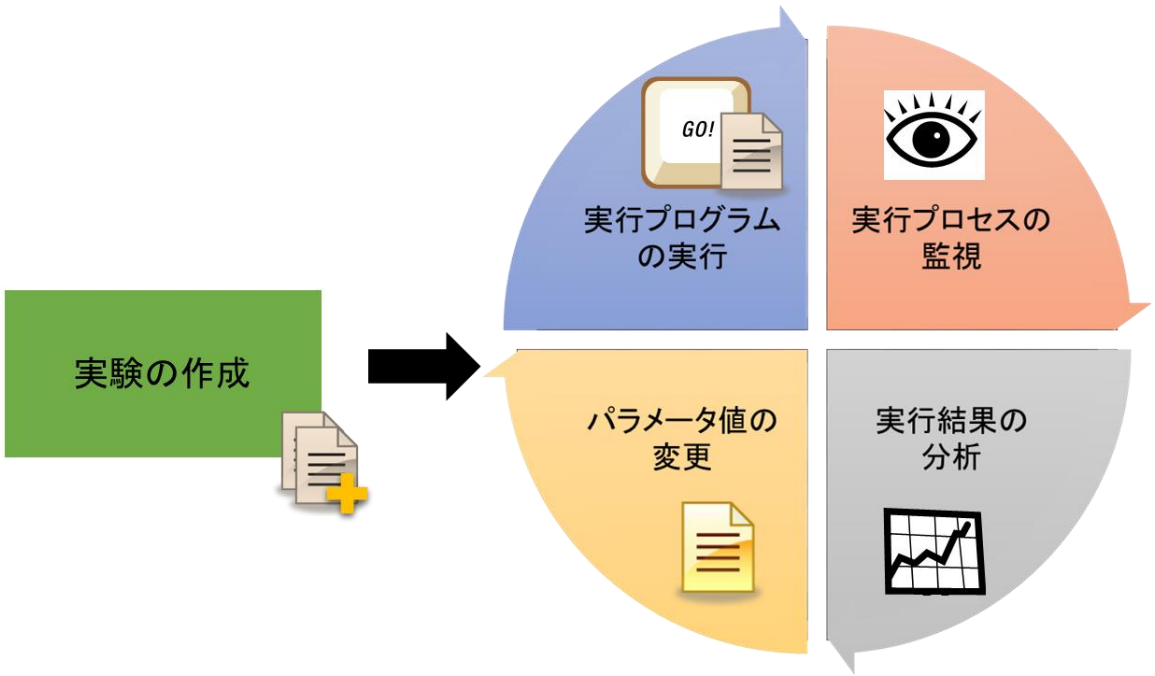


図 2-1　現状の実験の流れ

実験の作成

このプロセスでは、実験に必要なファイルの作成を行う。現在、顧客は EC の実験を行う際に、主に「実行ファイル」と「パラメータファイル」、「データファイル」を作成している。実験プログラムは EC のロジックが記述されているファイルである。パラメータファイルには実験プログラムで使用する世代数などのパラメータが記述されており、データファイルには分析対象の静的なデータが記述されている。顧客は、これらの実験に必要なファイルを一つのパッケージとしてまとめ、それを自身のサーバに保存している。

実行プログラムの実行

このプロセスは、サーバに保存した実行プログラムを実行するプロセスである。実験を実行する際には、まず自身のクライアント PC から SSH でサーバにログインする。その後、使用する実行ファイルとパラメータファイルを実行コマンドで指定し、実験を実行している。

実行プロセスの監視

このプロセスは、「実行プログラムの実行」プロセスで実行した実験が終了しているか、あるいは何らかのエラーにより実験が停止していないかを確認するプロセスである。この作業は、顧客がクライアント PC からサーバにログインし、動作している実行プロセスの状態を観察することで実験の状態を確認している。また、途中でエラーが発生した場合は、実験プログラムが出力する実験ログファイルを参考にエラーの原因を推測している。

実験結果の分析

このプロセスは、実験結果の分析を行うプロセスである。現在、実験結果はテキスト形式で出力されており、出力ファイルには世代ごとに個体の適応度などが記述されている。しかし、数値のみのデータから実験結果を考察することは難しいため、このテキストデータを可視化する必要がある。そのために、顧客はフリーの統計分析ツール R を利用して、実験結果のグラフ化や分析を行っている。ただし、テキスト形式のデータを分析ツールへかける作業は自動化されておらず、顧客が手動で行っている。

パラメータ値の変更

このプロセスは、世代数や反復回数、突然変異率（突然変異が発生する確率）などのパラメータ値の編集を行うプロセスである。パラメータを編集する際には、パラメータファイルの該当箇所をエディタで直接編集している。また、常にパラメータファイルと実験結果ファイルは対応付けて保存する必要があるため、そのための処理も編集のたびに手動で行っている。

2.2 現状の実験における問題と要望

2.2.1 コマンド入力による手間が大きい

現状では、すべての実験プロセスにおける作業を CUI によるコマンド操作で実現している。そのため、各作業プロセスで、その都度コマンドを入力しなければならない。加えて、実験を始める前にはサーバに SSH でログインする必要がある。これらの操作が各プロセスの作業を煩雑にしている。また、顧客からは Web ブラウザから実験作業をできるようにしてほしいという要望があった。

2.2.2 実験の実行プロセスの変化に対して即座に対応できない

1.3 節で述べた通り、EC の実験は一般に計算負荷が高く、計算時間が長くなる傾向にある。したがって、実験の開始から終了まで常に実行プロセスを監視し続けることは難しい。そのため、現在の「実行プロセスの監視」作業は、実験の終了しそうなタイミングで逐一クライアント PC からサーバの実験プロセスを観察することで対応している。この実験の終了しそうなタイミングとは顧客の経験則によるものであるため、実際には実験がまだ終了していない場合や終了してから時間が経ってしまっている場合もある。また、途中で実験プログラムなどのエラーによりプロセスが停止していても、実験者はサーバの実行プロセスを確認するまでプロセスの停止を認識することができない。

したがって、実験者は実行プロセスの変化に即座に対応することができず、「実行プロセス監視」において無駄な時間や作業を費やしてしまっている。

2.2.3 パラメータ設定の手間が大きい

1.3 節で述べた通り、EC の実験におけるパラメータの設定は、実験者の経験則により決めるしかない。そのため、パラメータ値を様々な値に変更して実験を行う必要がある。実験によっては、パラメータ値を少しずつ変化させ、パラメータの影響を細かく調べることもあり、その度にテキストエディタでパラメータファイルを作成・編集するのは非常に手間がかかる。

また、パラメータファイルと実験結果ファイル・ログファイルは常にセットで管理する必要があるため、その作業も手間になっている。

2.2.4 実験結果の分析の手間が大きい

1.3 でも述べた通り、EC の実験ではパラメータを頻繁に変更することがある。多くの場合、パラメータを変更すれば実験結果も変化するため、その度に実験結果を分析しなければならない。このように頻繁に起こる作業に対して、現状のように実験結果のファイルを手動で統計分析ツールにかけるという作業は非常に手間がかかる。

第3章　提案システム

3.1　問題に対するアプローチ

本プロジェクトでは、2.2 節の問題を解決するために、EC の実験を包括的に支援する Web アプリケーションを開発することとした。これは、2.2.1 項の問題を解決するためである。Web アプリケーションにすることで、面倒なコマンド操作をなくすことができる。また、図 2 で示した実験の 5 つのプロセス全てを Web ブラウザから実行できるようにすることで、プロセス間で発生するテキストエディタや分析ツールなどのツールの切り替えによるオーバーヘッドを減らすことができる。以下で、2.2.1 項を除いた 2.2 節の問題に対する個々のアプローチを述べる。

2.2.2 項の「実験の実行プロセスの変化に対して即座に対応できない」という問題に対しては、実験の実行プロセスの状態をメールで通知する機能を提供する。通知メールは実行プロセスの状態が変化したタイミングで実験者に送付する。そうすることで、実験者は実行プロセスの状態をサーバにアクセスせずとも、即座に実行プロセスの状態を知ることができる。これにより、プロセスの状態確認における無駄な作業や時間を軽減することができる。

2.2.3 項の「パラメータ設定の手間が大きい」という問題に対しては、2 つの機能を提供することで解決する。1 つは、パラメータファイルを自動で一括作成する機能である。この機能は、パラメータファイルに記述された任意のパラメータに対して範囲と刻み値を与えることで、その条件にあったパラメータファイルを自動作成することができる機能である。この機能があれば、パラメータ値を少しずつ変化させて実験をする際に、テキストエディタ起動しパラメータファイルを 1 つ 1 つ作成する必要がなくなる。もう 1 つの機能は、パラメータファイルと実験結果ファイル・ログファイルなどを自動で対応付けする機能である。この機能があれば、面倒なパラメータファイルと実験結果ファイル・ログファイルなどの対応付けを手動で管理する必要がなくなり、パラメータ設定における手間を削減することができる。

2.2.4 項の「実験結果の分析に手間がかかる」という問題に対しては、実験終了後に実験プログラムが出力するテキスト形式の実験結果ファイルをシステムが解析して自動でグラフ化する機能を提供することで解決する。この機能により、実験が終了する度に行っていた可視化・分析ツールにかける作業を行わずに済み、手間を削減することができる。

3.2　システム概要

この節では、前節の「問題に対するアプローチ」で挙げた機能について詳しく説明する。以下で開発した機能を 5 つのモジュールに分け説明する。

実験パッケージ管理モジュール

本プロジェクトでは、実験に関わるファイル群をまとめたディレクトリのことを「実験パッケージ」と呼んでいる。実験パッケージ管理モジュールは、ユーザが Web ブラウザから実験パッケージを管理できるようにした機能群である。この機能により、自動で実験ごとにパラメータファイルと実験結果ファイル・ログファイルが対応付けされる。また、実験ごとにいくつかの情報（実験の名前、作成者、バージョン、実験の説明）を付加し、Web で閲覧することもできるため、ユーザはファイルの中身を見ずに実験の概要を瞬時に知ることができる。

図 3-1 は、実験パッケージ管理モジュールにおける「実験パッケージを登録する」という機能を示したものである。まず、ユーザはクライアント PC で実験に必要な実験ファイル群を作成し、ディレクトリにまとめる。ディレクトリの構成は、GAM の規則に則って作成する必要がある。この点については、5.1 節で後述する。ディレクトリ作成後、ブラウザで GAM システムにアクセスし、ディレクトリを圧縮したファイルをアップロード/実験に関する情報（実験の名前、作成者、バージョン、実験の説明）を入力することで、GAM に実験パッケージを登録することができる。

表 3-1 は、実験パッケージ管理モジュールの機能を示したものである。実験パッケージの登録の他に、実験パッケージの削除、実験パッケージに関する情報の閲覧・編集を行うことができる。

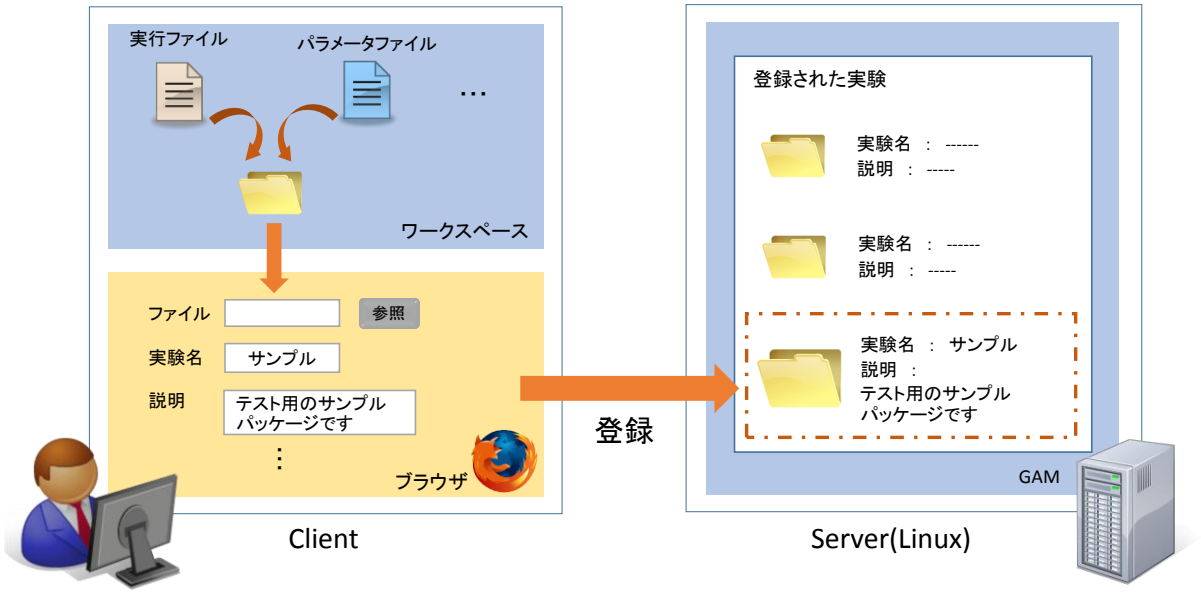


図 3-1　実験パッケージ管理モジュール「実験パッケージを登録する」機能の概要

表 3-1　実験パッケージ管理モジュールの詳細

機能名	概要
実験パッケージを登録する機能	ユーザが作成した実験ファイルをまとめた圧縮ファイルをアップロードしてシステムに登録する
実験パッケージの詳細情報を表示する機能	実験パッケージの名前や作者、バージョン、実験の説明などを表示する
実験パッケージの詳細情報を編集する機能	実験パッケージの名前や作者、バージョン、実験の説明などを表示する
実験パッケージを削除する機能	実験パッケージをシステムから削除する

実行管理モジュール

実行管理モジュールは、Web ブラウザから実験を実行・停止する機能群である。この機能を利用することで、CUI コマンドにより行っていた実行ファイルやパラメータファイルの指定を GUI で操作することができるようになる。また、実験の実行を予約することも可能で、予約した実験はその前に予約していた実験が終了しだい自動的に実行される。これにより、

ユーザが一つひとつの実験の終了を待たずに実験を実行することができる。図 3-2 は、「実験 1」、「実験 2」という実験を実行予約した後に、新たに「実験 3」という実験を実行予約している様子である。

表 3-2 は、実行管理モジュールの機能を示したものである。実験実行の他に、実行中の実験を停止するという機能も提供する。

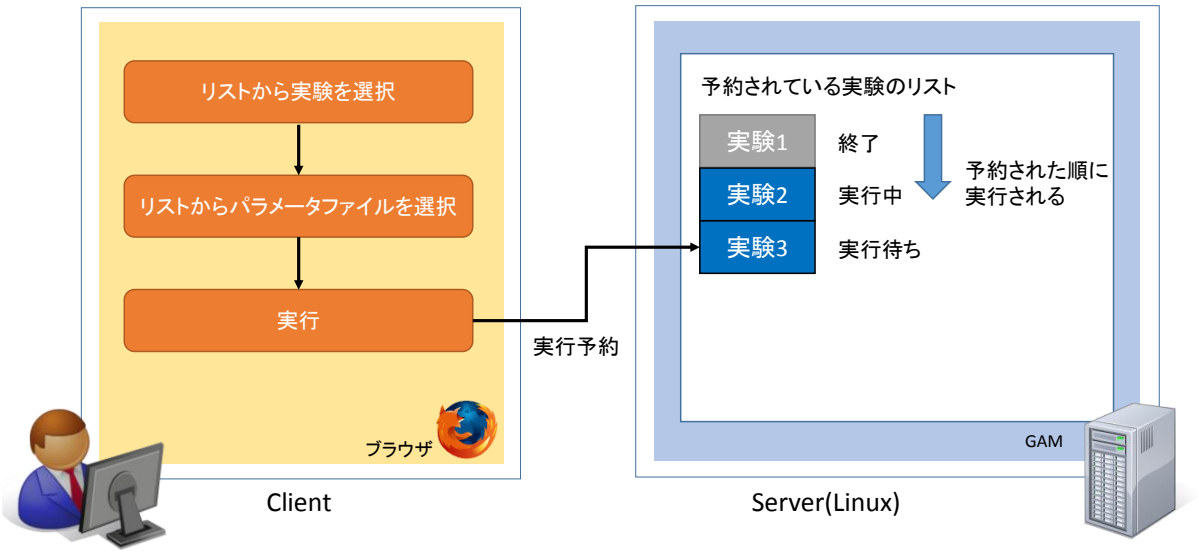


図 3-2　実行管理モジュールの概要

表 3-2　実行管理モジュールの詳細

機能名	概要
実験を実行する機能	実験を実行する
実験中の実験をキャンセルする機能	実行中の実験を停止する
実験中の全ての実験をキャンセルする機能	実行中の全ての実験を停止する

実行状態監視モジュール

実行状態監視モジュールは、実行予約された実験の実行状態を監視する機能群である。このモジュールは、表 3-3 にあるように、大きく 4 つの機能から構成される。1 つ目は、実験の実行状態をメールで通知する機能である。この機能により、ユーザは、実験の実行プロセスの状態変化を常に把握することができる。図 3-3 は、この機能を説明したものである。図 5 はシステムに「実験 1」、「実験 2」、「実験 3」の 3 つの実験が実行予約されており、そのうち「実験 1」が終了した状態を示している。実験 1 が終了した時点でシステムは、ユーザに実験 1 が終了したことをメールで伝える。

2 つ目の機能は、実行状態をブラウザから確認することができる機能である。「現在実行中の実験」や「終了した実験」、「実行待ちの実験」のリストを確認することができる。

3 つ目の機能は、EventLog を閲覧する機能である。これは、実行ファイルが出力する標準出力・標準エラー出力を表示する機能である。この機能を利用することで、実験途中にエラーが発生した際に、原因究明の一つの材料にすることができる。

4 つ目は、実行中の実験の進捗を確認する機能である。この機能により、実行中の実験に関

して実験の進捗状況も確認することができるようになる。

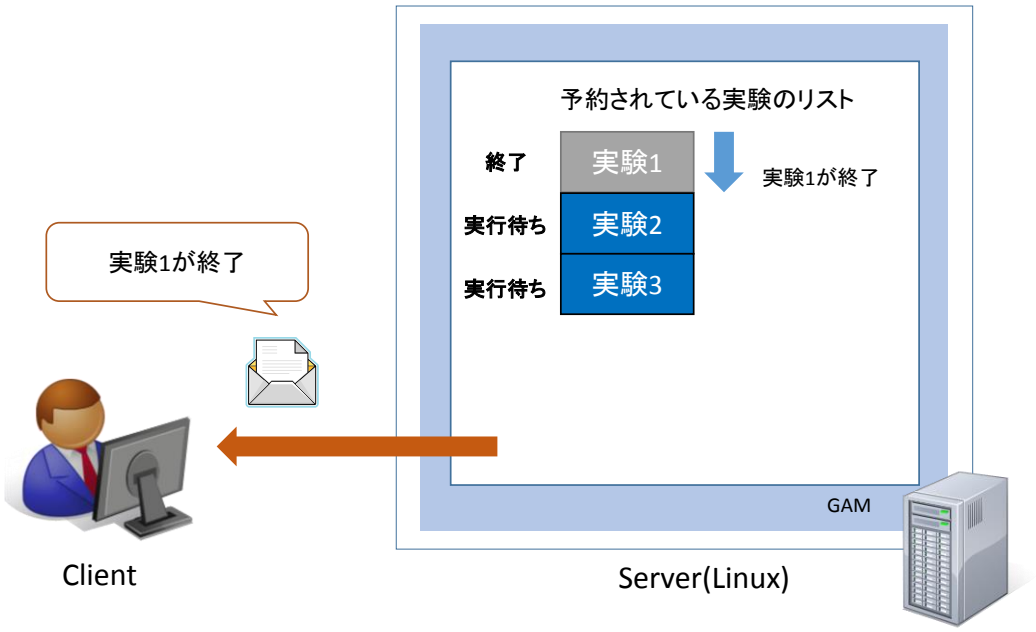


図 3-3　実行状態監視モジュール「実験の状態を通知する機能」の概要

表 3-3　実行状態監視モジュールの詳細

機能名	説明
実験の状態を通知する機能	実験の状態を実験者にメールで通知する
実験の状態を表示する機能	実行中の実験の実行状態を表示する
EventLog を表示する機能	実行ファイルが出力する標準出力、標準エラー出力を表示する
実験の進捗を表示する機能	実行中の実験の進捗を表示する

パラメータファイル作成モジュール

パラメータファイル作成モジュールは、パラメータファイル作成を支援する機能群である。この中でも特筆すべき機能は、パラメーター括作成機能である。

図 3-4 はパラメーター括作成機能をしたものである。GAM システムのパラメーター括作成画面にアクセスし登録されているパラメータファイルを選択すると、パラメータファイルの内容とともに「下限値」、「上限値」、「刻み値」を入力するテキストボックスが表示される。図では、それぞれのテキストボックスに、「300」、「500」、「100」と入力している。これは、世代数というパラメータを 300～500 の範囲で 100 ずつ変化させることを意味している。そして、その条件に合うパラメータファイルが自動生成される。図 3-4 の場合だと、世代数が 300、400、500 となるパラメータファイルがそれぞれ一つずつ作成される。これにより、パラメータ値を少しずつ変化させたパラメータファイルを容易に作成することができる。

また、この機能の他に、クライアント PC にあるパラメータファイルを GAM にアップロードしたり、GAM 上でパラメータファイルを編集して新たなパラメータファイルを作成することも可能である。このモジュールの詳細な機能リストは、表 3-4 の通りである。

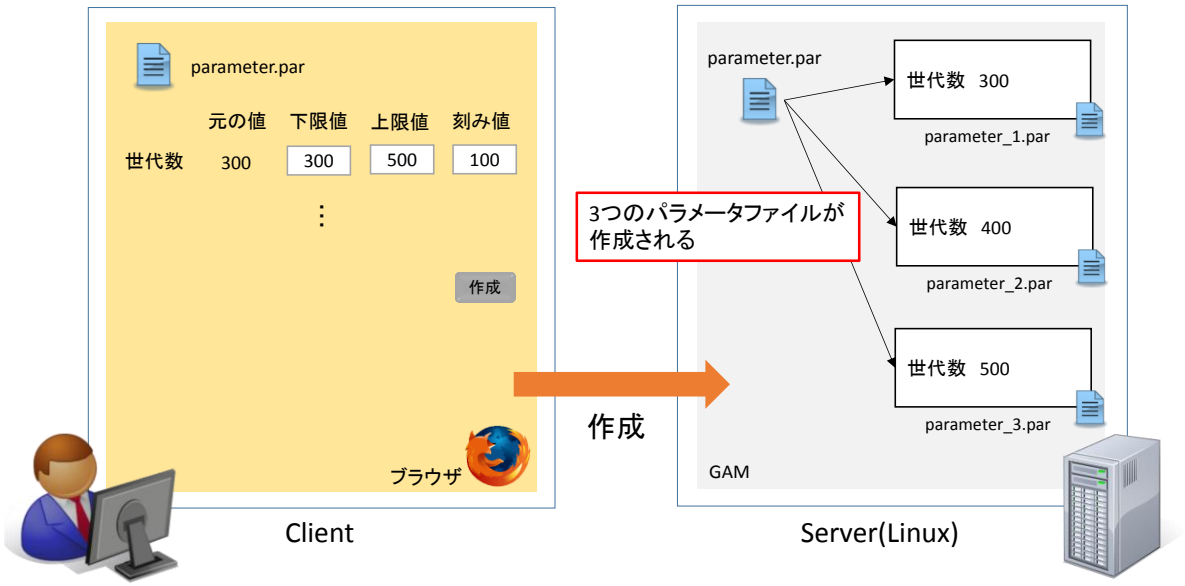


図 3-4　パラメータファイル作成モジュール

表 3-4　パラメータファイル作成モジュールの詳細

機能名	概要
パラメータファイルを一括作成する機能	条件に合うパラメータファイルを一括作成する
既存のパラメータファイルを編集して新しいパラメータファイルを作成する機能	あるパラメータファイルを元に新たなパラメータファイルを作成する
パラメータファイルをアップロードする機能	パラメータファイルをアップロードする
パラメータファイルを削除する機能	システムに登録されているパラメータファイルを削除する

実験結果可視化モジュール

実行結果可視化モジュールは、実験結果をグラフ化する機能群である。実行ファイルは、実験終了後にテキスト形式の実験結果ファイルを出力している。GAM ではこのテキストデータをグラフ化して表示する。これにより、ユーザは実験結果を分析しやすくなる。グラフ化の際には、テキスト形式の実験結果ファイルを参照しているため、この実験ファイルは、規定のフォーマットで記述されている必要がある。GAM では、適応度、個体の多様性、適応度の標準偏差をグラフ化対象のデータとし、それらを規定のフォーマットで出力するための API を用意している。ユーザは、実行ファイル内でこの API を利用することで実験結果のフォーマットを考慮せずに実験結果をグラフ化することができる。

グラフは大きく分けて 3 つ出力される。1 つ目は、世代ごとの適応度をプロットしたグラフである。2 つ目は、個体の多様性のグラフである。3 つ目は、各反復の最大適応度を表示する機能である。この値は、各反復で適応度に振れ幅があるかを分析するために必要となるグラフである。図 3-5、図 3-6、図 3-7 は顧客から提供されたサンプルプログラムの実行結果である。上から順に、各世代の適応度のグラフ、個体の多様性のグラフ、各反復の最大適応

度のグラフを表している。

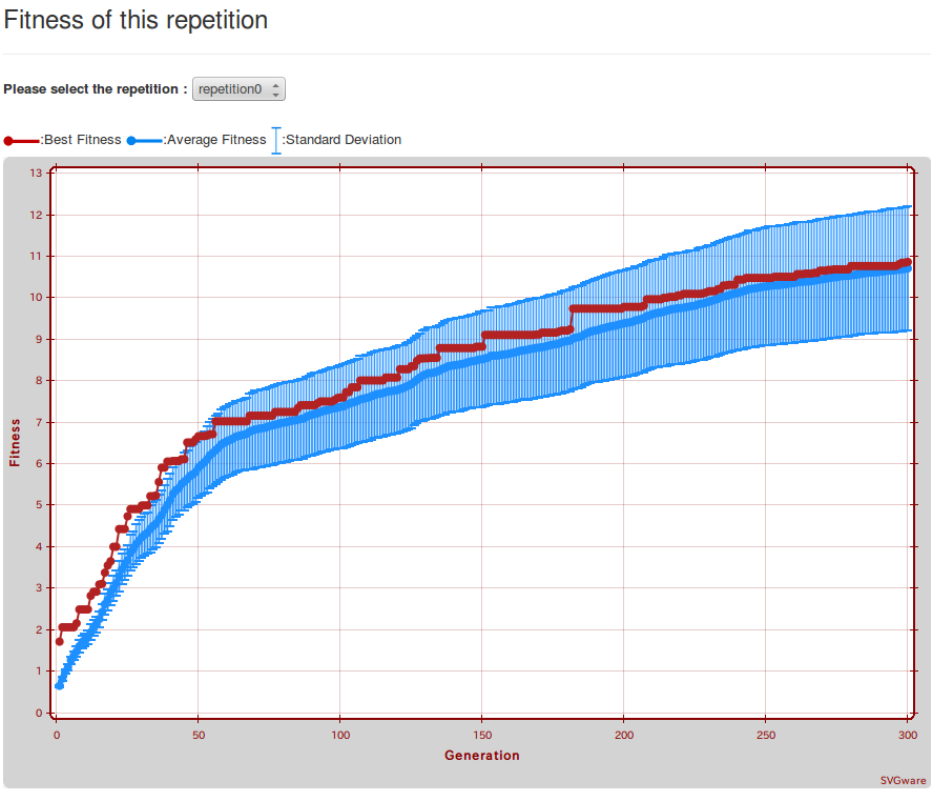


図 3-5　各世代の適応度のグラフ

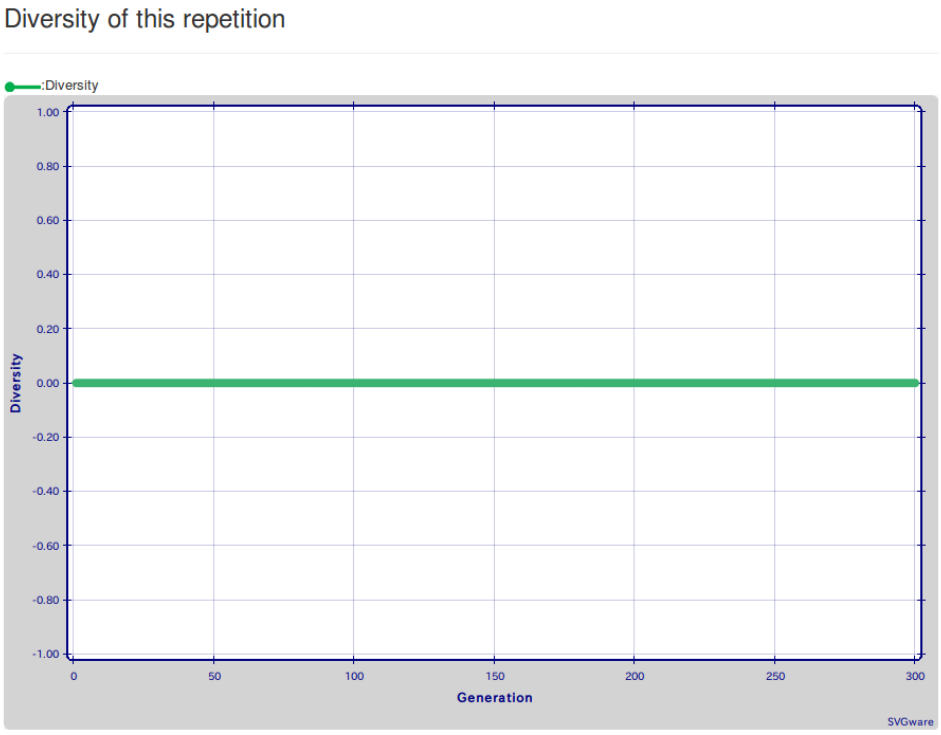


図 3-6　個体の多様性のグラフ

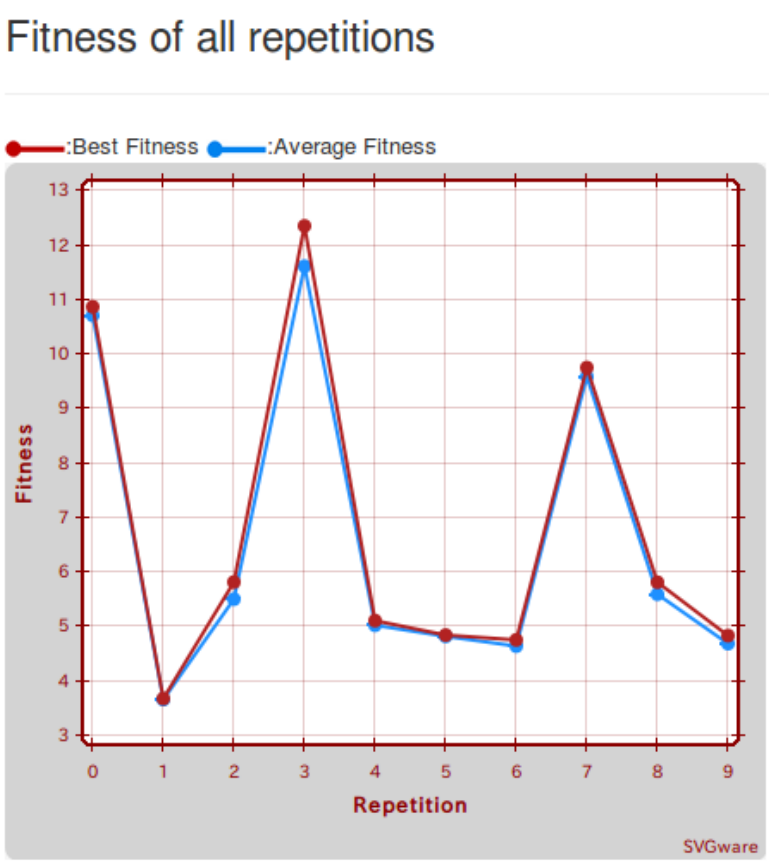


図 3-7　各反復の最大適応度のグラフ

表 3-5　実験結果可視化モジュールの詳細

機能名	説明
システム API（Java バージョン）により実験結果をシステムに渡す機能	システム API の Java バージョン。Java で書かれた実験プログラムはこの API により実験結果をシステムに渡すことができる
実行後の適応度グラフの表示機能	実験結果により適合度グラフを画面上で描画する
実行後の多様性グラフの表示機能	実験結果により多様性グラフを画面上で描画する

開発スコープから外れた機能

本プロジェクトでは、顧客の要求を全て実装することは開発期間内には難しいと判断しており、当初より顧客に全ての機能に対して優先順位をつけていただいていた。下記の機能はいずれも優先度が低いとされていた機能である。顧客の合意のもと、下記の機能については本プロジェクトでは実装していないこととなった。

表 3-6　開発スコープから外れた機能

機能名	説明
一括作成したパラメータファイルの概要を表示する機能	一括作成したパラメータファイルの概要を表示する
メール通知の時間間隔を設定する機能	短時間で大量な通知メールが発生する場合には、まとめて送付する
マシンリソースを表示する機能	サーバの状態（CPU・メモリ使用率など）を画面上に表示する機能
パラメータファイルの項目を追加、削除する機能	パラメータ項目を追加、削除する
データファイルの追加アップロード機能	データファイルを追加でアップロードする
リアルタイムグラフ表示機能	実行中の実験の適応度グラフをリアルタイムで表示する
実験パッケージのダウンロード機能	システムに登録された実験パッケージをダウンロードする
パラメータファイルのダウンロード機能	システムに登録されたパラメータファイルをダウンロードする
スナップショット機能	実行中の実験の状態を保存し、復元、再開などの操作ができる機能
個体の木構造グラフを表示する機能	実験結果の個体の木構造グラフを表示する
EventLog 画面のフィルタリング機能	EventLog 画面でユーザが見たい情報だけを表示できる機能
大容量ファイルをアップロードできる機能	大容量のファイルをアップロードした後でも、システムを使用できるようにする
システムの Debian パッケージ化	システムを Debian パッケージにし、インストールを容易にする
ANOVA 分析機能	実験結果を自動的に ANOVA 分析する
統計分析機能	実験結果を自動的に統計分析する
実験パッケージのディレクトリ構成を表示する機能	実験パッケージのディレクトリ構成を表示する
実験の並列実行機能	複数の実験を並列で実行する
サーバを直接操作できる機能	サーバを直接操作した内容が、システムに反映されるようにする

第4章　　プロジェクト体制

4.1　　基本方針

開発手法には，アジャイル開発手法のスクラム [11]を採用し，反復型開発を行った。図 4-1 は，開発の流れを示したものである。最初の要求分析工程でユーザストーリを用いて機能を抽出し，顧客に機能一つひとつに優先順位をつけていただく。その後，設計，実装，テスト，レビューを一つのイテレーションとして，全ユーザストーリ中で優先順位の高いものから開発していく。レビューフェーズでは，そのイテレーションで開発した動作するシステムを顧客に見せ，フィードバックをいただきながら，仕様の確認や今後実装していく機能について議論していくこととした。

本プロジェクトでアジャイル開発を取り入れた理由は，顧客の要求の変化に対応していくという狙いと言葉による誤解を防ぐためである。ウォーターフォール開発では，仕様をドキュメントで顧客と確認するが，文章や言葉だと開発チームと顧客の考えているイメージに誤解が生じやすい。特に，本プロジェクトのように，開発チームと顧客で母国語が異なる場合は，このような問題は生じやすい。アジャイル開発では，実際に動作するシステムをプロジェクト途中で顧客と確認するため，言葉による誤解を起こりにくくすることができると考えた。

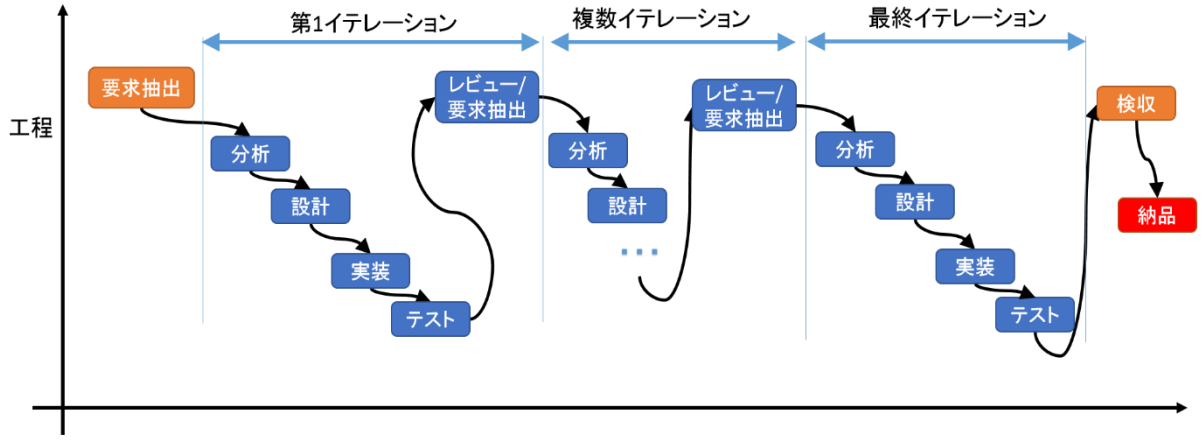


図 4-1　アジャイル開発手法の流れ

4.2　　開発メンバ

表 4-1 に開発メンバと各自の役割を示す。

表 4-1　開発メンバと役割

氏名	主な担当
中西　陽平	開発環境の整備 継続的インテグレーション 品質メトリクスの分析
岡谷　亮	スケジューリング実行機能
栗　克	パラメータ管理機能 実験結果の可視化機能

人見　圭一郎	実験パッケージの管理機能 テスト計画の立案と実施

4.3　実績

本プロジェクトの開発スケジュールは表 4-2 のとおりである。基本的には、1 スプリントを 2 週間に設定している。3 章で挙げた各機能を 1 スプリントで完成させることは難しいため、実際には 一つの機能を更に分割し、複数のスプリントで 一つの機能を完成させている。

基本的には、1 スプリントで 機能の設計からテストまで実施しているが、画面設計については、顧客の要求を早い段階で明確にするため、例外的に一つ前のスプリントで実施している。赤字部分は、筆者が担当した仕事である。バグ修正に関しては、メンバ全員で手分けして実施した。

表 4-2　開発実績

スプリント	実施内容	期間
要求抽出	・要求の抽出 ・プロジェクト管理手法の調査	5/21 ~ 7/5
スプリント 1	・プロトタイプの作成 ・実験パッケージ管理モジュール、パラメータファイル作成モジュール、実行管理モジュールの画面設計 ・開発環境の構築 ・関連技術/ソフトウェアの調査	7/8 ~ 7/21
スプリント 2	・「実験パッケージを登録する」機能の設計/実装/単体テスト ・「実験パッケージを削除する」機能の設計/実装/単体テスト ・「実験パッケージの詳細情報を表示する」機能の設計/実装/単体テスト ・「実験パッケージの詳細情報を編集する」機能の設計/実装/単体テスト ・「パラメータファイルを編集する」機能の設計/実装/単体テスト ・「パラメータファイルをアップロードする」機能の設計/実装/テスト ・「パラメータファイルを削除する」機能の設計/実装/単体テスト ・「実験を実行する」機能の設計/実装/単体テスト ・継続的インテグレーションツールの導入 ・バグ修正	7/22 ~ 8/2
スプリント 3	・実験パッケージ管理モジュール全体の修正/単体テスト ・「実験を実行する」機能の修正/単体テスト ・実行状態監視モジュール、実験結果可視化モジュールの画面設計 ・DBMS の切り替え	8/26 ~ 9/6

	・バグ修正	
スプリント 4	・「実験の状態を通知する」機能の設計/実装/単体テスト ・実験パッケージ管理モジュール全体の修正/単体テスト ・実験結果可視化機能モジュールの設計 ・テスト計画の立案/ツールの調査・環境構築 ・画面デザイン（CSS フレームワーク）の変更 ・バグ修正	9/9 ~ 9/20
スプリント 5	・「実験の状態を通知する」機能の修正/単体テスト ・「システム API により実験結果をシステムに渡す」機能の実装 ・「実行後の適応度グラフを表示する」機能の実装/単体テスト ・テスト計画の立案/ツールの調査・環境構築 ・バグ修正	9/23 ~ 10/5
スプリント 6	・「実験の状態を通知する」機能の修正/単体テスト ・「システム API により実験結果をシステムに渡す」機能の修正 ・「実行後の適応度グラフを表示する」機能の修正/単体テスト ・「実行後の多様性グラフを表示する」機能の実装/単体テスト ・スプリント 5 までに実装した機能の結合テスト ・「実験中の実験をキャンセルする」機能の設計/実装/テスト ・バグ修正	10/7 ~ 10/18
スプリント 7	・「実験の状態を通知する」機能の修正/単体・結合テスト ・「実験の進捗を表示する」機能の設計/実装/単体・結合テスト ・「実行後の適応度グラフを表示する」機能の修正/単体テスト ・「実行後の多様性グラフを表示する」機能の修正/単体テスト ・「実行中のすべての実験をキャンセルする」機能の設計/実装/単体・結合テスト ・「パラメータファイルを一括作成する」機能の設計/実装/単体テスト・結合テスト ・「Eventlog を表示する」機能の設計/実装/単体・結合テスト ・バグ修正	11/11 ~ 11/23
スプリント 8	・導入マニュアルの作成 ・運用マニュアルの作成 ・総合テストの準備と実施 ・受け入れテストの準備 ・バグ修正 ・結合テストの修正と実施	11/25 ~ 12/06
スプリント 9	・受け入れテストの実施 ・評価アンケートの作成 ・受け入れテストで発生したバグへの対処	12/11 ~ 12/20

※進捗管理/要求管理は全スプリントで実践しているため省略する。

4.4　　アジャイルプラクティス

ここでは、プロジェクト運営にあたり利用したアジャイルプラクティスを述べる。

デイリースクラム

デイリースクラム [12]とは、毎日時間を決めた時間でステークホルダーが顔をあわせ、お互いの進捗状況などを短時間で報告しあうものである。これにより、開発者同士の情報共有の漏れを防ぎ、現在起きている問題の把握・対処を適切なタイミングで行うことができる。

本プロジェクトでは、平日の 14:00～17:00 をコアタイムとして設定し、コアタイム開始時に互いの進捗状況や現在困っていること・発生している問題などを報告しあい、情報の共有を行った。

かんばん

かんばん [13]とは、一つのユーザストーリに対して、図 4-2 のように細かいタスクを洗い出し、そのタスクの進捗（「新規」、「進行中」、「レビュー」、「終了」）を一目で確認できるようにしたものである。これにより、チームメンバ間の作業の進捗などを一目で把握できるようになる。本プロジェクトでは、Redmine のかんばんツールを利用した。

図 4-2　Redmine のかんばん

KPT

KPT [14]は、現在までに行ってきた活動を振り返る際に、「Keep（今後も継続すべきこと）」、「Problem（良くなかったこと）」、「Try（今後、実践すべきこと）」といった観点で活動を評価・改善するフレームワークのことである。KPT とは、Keep、Problem、Try の頭文字をとったものである。本プロジェクトでは、各スプリントの最終日に活動の振り返りを実施し、その際に KPT を使って活動を評価し、改善を図った。

4.5　　開発環境

本プロジェクトでは、開発を円滑に進めるために、フリーソフト「Git」や「Redmine」、「Jenkins」を利用している。Git は、オープンソースの分散型バージョン管理システムであ

る。自分以外の開発者がソースコードに加えた修正点を自分のソースコードに取り込んだり、ソースコードの変更点を記録し任意の時点のソースコードを復元したりする機能などを持っている。本プロジェクトでは、各個人の開発用 PC に Git クライアントソフト、統合用マシンに Git サーバソフトをインストールしている。コーディングやドキュメントの作成は各個人の開発用 PC で行い、作成後にそれを統合用サーバに統合していくことで、メンバ間で整合性のとれたソースコードやドキュメントを管理した。

Redmine はプロジェクト管理用のフリーツールで、本プロジェクトでは主にタスク管理で利用している。Jenkins は、継続的インテグレーションツールと呼ばれるもので、自動でビルド・テスト・コードの品質検査などを継続的に行うものである。このソフトを利用することで、システムの品質確保を図った。Redmine と Jenkins については統合用マシンにインストールし、開発用 PC のブラウザから利用した。

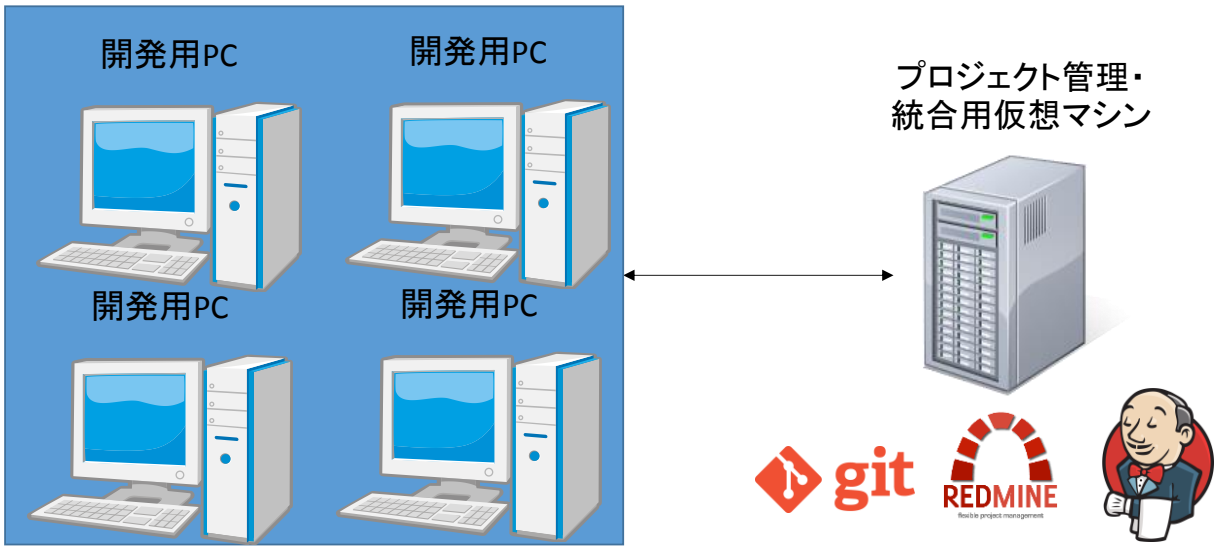


図 4-3　開発環境

第5章　開発活動における筆者の貢献

筆者は主に、実験パッケージ管理モジュールと実験状態監視モジュールの「実験の進捗を表示する」機能の実現を行った。この章では、筆者が開発活動における貢献について述べる。

5.1　実験パッケージ管理モジュールの実現

本モジュールの開発において、クラス設計・データベース設計は中西が作成しており、筆者は画面設計と実装を担当した。この節では、実験パッケージ管理モジュールの仕組みや画面設計、実装において工夫した点とそれらについての考察を述べる。

5.1.1　実験パッケージ管理モジュールの仕組み

ここでは、実験パッケージ管理モジュールの仕組みについて述べる。実験パッケージ管理モジュールの概要については、3.2 節の通りである。このモジュールの仕組みについて説明する前に、まず GAM を利用する際の実験パッケージ規則について説明する。GAM システムを利用するためには、GAM システムの規則にしたがって実験パッケージを作成する必要がある。図 5-1 はその規則を示したものである。この規則は大きく分けて下記の 3 つである。

- 「実行ファイル（図における foo.jar)」と「パラメータファイルを格納するディレクトリ（図における parameters)」は親ディレクトリ直下に配置する
- 「ユーザ自身で必要なファイルもしくはディレクトリ（図における Other Files or Directories)」は、親ディレクトリ直下に配置する
- 「パラメータファイル」は全て parameters ディレクトリの直下に配置する

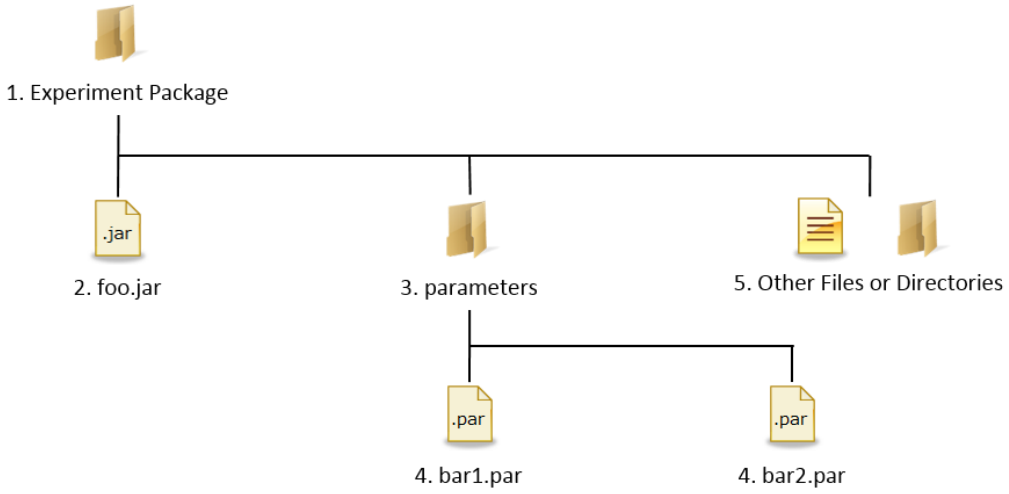


図 5-1　実験パッケージのディレクトリ構成

この規則に則って実験パッケージを作成すれば、GAM を利用することができる。

次に、実験パッケージ管理モジュールの仕組みの説明をする。ここでは、3.2 節同様、「実験パッケージを登録する」機能を例に説明する。図 5-2 が仕組みを示したものである。まず、ユーザは実験パッケージの規則に則り実験パッケージを作成し、そのファイルを圧縮する。圧縮の形式は表 5-1 の形式をサポートしている。この圧縮形式は顧客からヒアリングし、それらの形式に対応できるように実装した。また、ファイル圧縮するソフトによっては、圧縮

ファイル直下にファイルを展開するものと圧縮ファイル直下にディレクトリを作成しその中にファイルを展開するものがある。これらのソフトの違いにも対応できるように設計している。実験パッケージを作成した後、ユーザは、圧縮ファイルと実験パッケージ詳細情報（実験名、説明、作者、バージョン）を入力し、作成ボタンを押す。すると、それらの情報は GAM システムに渡り、GAM システムは実験パッケージを解凍し、実験パッケージを保管するディレクトリにファイルを展開する。また、ユーザが入力した実験詳細情報は、実験パッケージが展開されたディレクトリパスと合わせてデータベースに保存される。これにより、実験パッケージと実験パッケージの詳細情報が対応付けられて管理されることになる。以上が実験パッケージを登録する機能の仕組みである。

その他の「実験パッケージの詳細情報を表示する」機能や「実験パッケージ情報を編集する」機能、「実験パッケージを削除する」機能などは、対応付けられたデータベースの情報と実験パッケージのディレクトリを適宜参照することで実現している。

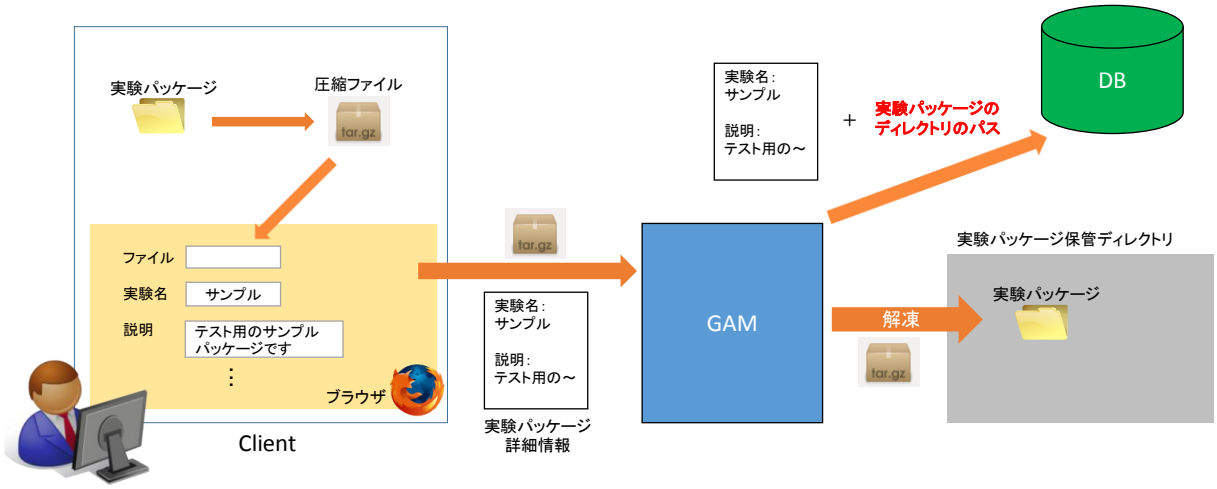


図 5-2　「実験パッケージを登録する」機能の仕組み

表 5-1　圧縮形式の対応表

圧縮形式	拡張子
Zip	.zip
RAR	.rar
Gzip	.tar.gz,tgz
Bzip2	.tar.bz2

5.1.2　画面設計

筆者は、実験パッケージ管理モジュールの画面設計において、画面遷移と画面モックアップの作成を行った。図 5-3　実験パッケージ管理モジュール・実行管理モジュールの画面遷移図 5-3 は、実験パッケージ管理モジュールと実行管理モジュールの画面遷移図である。青色の四角がひとつ一つの画面を表している。赤線で囲まれた部分が実験パッケージ管理モジュールである。緑線で囲まれた部分は実行管理モジュールを表している。筆者は、実験パッケージ管理モジュールの他に実行管理モジュールとパラメータファイル作成モジュールの画面設計（画面遷移と画面モックアップの作成）を行った。

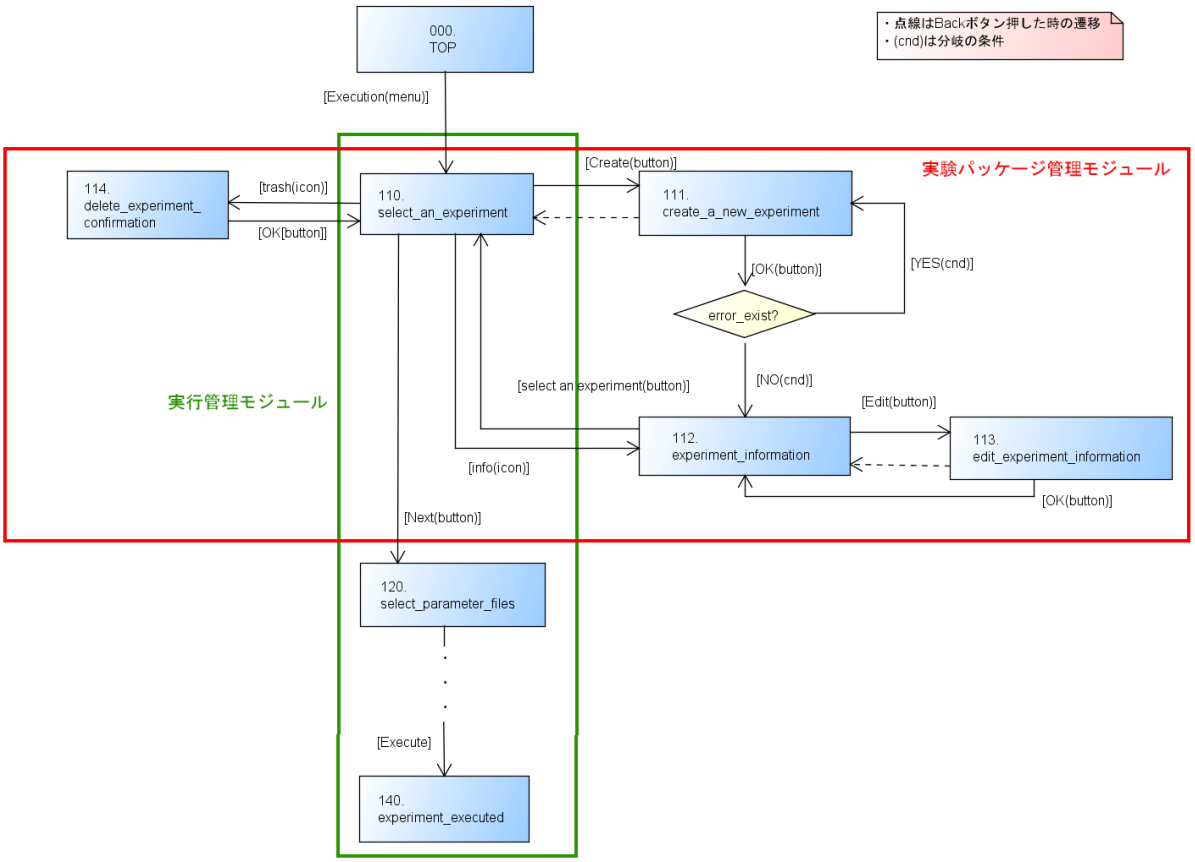


図 5-3　実験パッケージ管理モジュール・実行管理モジュールの画面遷移

5.1.3　　実装における工夫

筆者は、この機能を実現するにあたり、RESTful な設計を意識して実装した。REST [15] の世界では、ネットワーク上のリソースをすべて一意な URL で表現する。そして、これらの URL に対して、HTTP メソッドである GET（取得）や POST（作成）、PUT（更新）、DELETE（削除）を使ってアクセスする。つまり、何（リソース）をどうするか（HTTP メソッド）を表現する書き方である。GAM における実験パッケージ管理機能に適用するならば、実験パッケージがこのリソースに相当する。 [16]

このような設計にすることで、より統一感のあり、かつ意味がつかみやすい URL を設計することができる [16]。また、GAM システムの開発には Ruby on Rails というプログラミング言語を使用しており、この言語は RESTful な設計と親和性が高く、様々な便利機能を利用することができる。これにより、実装にかかる時間や手間を削減することができた。

5.1.4　　考察

筆者は 5.1.2 項で述べたように、実験パッケージ管理モジュールと実行管理モジュールの画面遷移を設計した。図 5-3 はその画面遷移図を表しているが、この画面遷移には一つ問題がある。それは、実験パッケージ管理モジュールの画面が実行管理モジュールの画面遷移の中にあることである。ユーザが TOP ページから実験パッケージの登録・削除・表示・編集といった操作を行うには、一度実験を実行するモジュールに移動しなければならないのである。つまり、ユーザが利用したい機能が違う機能群の流れの中にあることになる。実行する実験を選択する際に、実験パッケージを登録・削除・編集・表示することもあるが、このような設

計は望ましくない。TOP ページから実験パッケージ管理モジュールへ直接遷移するような設計に見直すべきである。あるいは、現状の画面遷移は残したまま、TOP ページから実験パッケージ管理モジュールへのリンクを追加するという対策を実施すべきである。

5.2 実験の進捗表示機能の実現

5.2.1 実験の進捗に関する要求

実行状態監視モジュールの中には「実験の状態を表示する機能」という機能が存在する。しかし、この機能は実験が「実行中」、「実行待ち」あるいは「終了」のいずれかの状態にあるということしか分からない。1.3 節でも述べたように、EC を利用した実験は一般に長くなる傾向にあるため、「実行中」というという情報だけでは十分ではない。その実験が終わりそうなのか、それともまだ時間がかかるのか、そういった情報が分からないのである。したがって、現在実行している実験がどの程度進んでいるか、その進捗を把握できるというのはユーザにとって有益である。

5.2.2 実験の進捗表示機能の仕組み

実験の進捗を表示するためには、そのための指標が必要である。この指標に「現在の反復回数」、「現在の世代数」、「総反復回数」、「総世代数」を用いて、実験の進捗表示機能を実現した。以下で、その指標とその指標を利用した実現方法を説明する。

現状では、実行ファイルの中で「現在の反復回数」、「現在の世代数」を保持している。実行ファイルは一つの世代についての適応度の計算が終了した時点で、「現在の反復回数」や「現在の世代数」、適応度、多様性などの情報を実験結果ファイルに出力している。図 5-4 は、顧客からいただいたサンプル実行ファイルの実験結果ファイルを一部抜粋したものである。ファイル上部に書かれている文字列は各列のタイトルである。1 列目、2 列目がそれぞれ「現在の反復回数」、「現在の世代数」を表している。3 列目から 6 列目までは本機能とは関係ないので割愛する。

```
repetition generation best_fitness avg_fitness fitness_stdev diversity
0 1 1.321878524266623 0.6200301417892377 0.01817367102801295 0.0
0 2 1.3474807116466145 0.7870381856648596 0.040342156609744315 0.0
0 3 1.4531009567035333 0.9213690072707919 0.057544745187894944 0.0
                                        .
                                        .
                                        .
5 100 3.931549782690915 3.659909918642008 0.4924244260892236 0.0
5 101 3.931549782690915 3.668959968211689 0.4924244260892236 0.0
5 102 3.931549782690915 3.6870557143160907 0.4949321049407343 0.0
                                        .
                                        .
                                        .
9 298 3.985100121612561 3.900865319849004 0.5483671757914492 0.0
9 299 3.985100121612561 3.9019538777734963 0.549546030299686 0.0
9 300 3.985100121612561 3.9034628981097512 0.549546030299686 0.0
```

図 5-4　実験結果ファイル（API を利用して出力されたもの）

図 5-5 に進捗機能の大まかな仕組みを示す。パラメータファイルには「総反復回数」と「総世代数」が記述することになっている（進捗を表示しなくても良い場合は、記述する必要はない）。1 世代の計算が終了したタイミングで実行ファイルから実験結果ファイルに、「現在の反復回数」、「現在の世代数」が出力される。GAM システムは、パラメータファイルから「総世代数」と「総反復回数」を、実験結果ファイルから「現在の反復回数」、「現在の世代数」を取得する。そして、それらの数値から、反復回数と世代数の進捗をそれぞれ、「現在の反復回数 / 総反復回数」、「現在の世代数 / 総世代数」とし、表示している。

図 5-6 は、GAM システムの実画面である。赤い丸で囲まれた部分が反復回数の進捗を表しており、緑の丸で囲まれた部分が反復回数の進捗を表している。

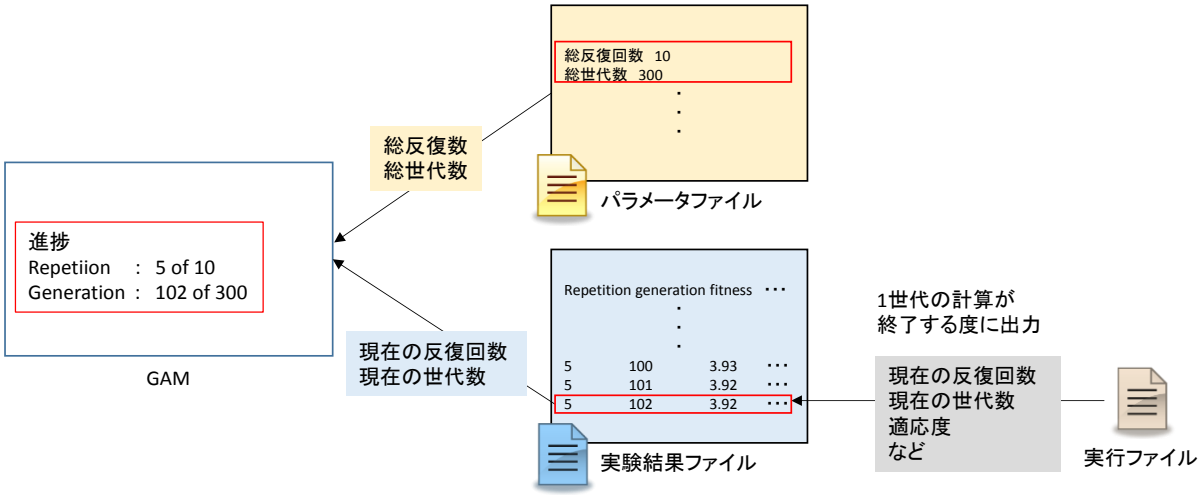


図 5-5　実験の進捗機能の仕組み

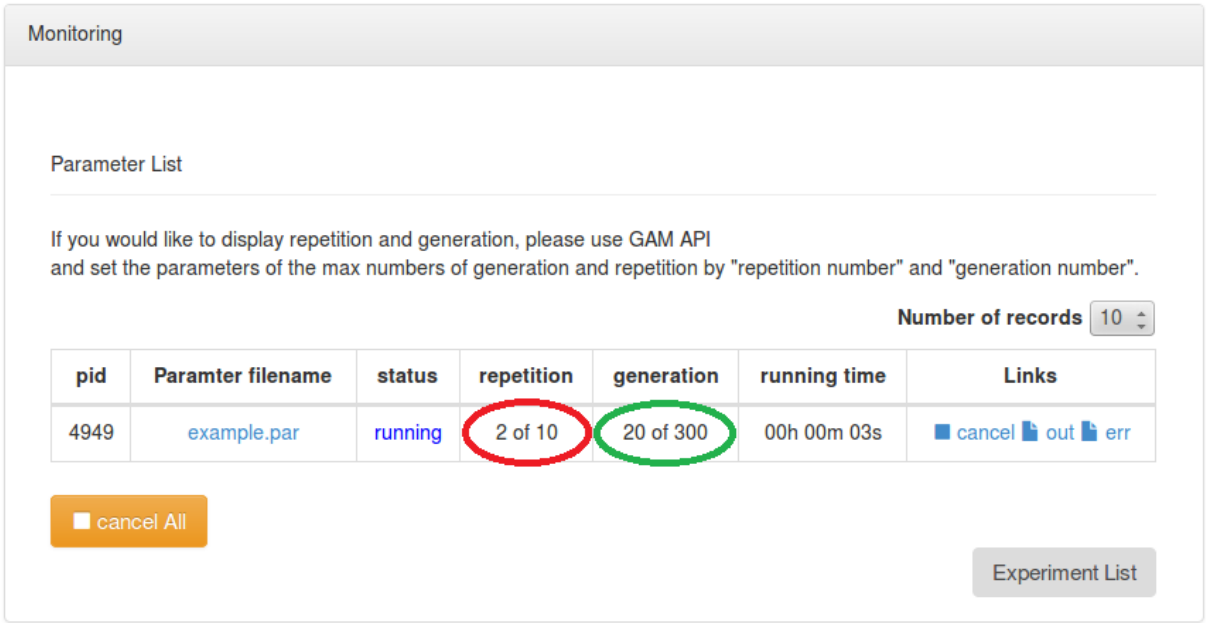


図 5-6　実験の進捗機能画面

5.2.3 考察

現状では、実験の進捗を反復回数と世代数という 2 つの指標を用いて、それぞれ別々に進捗を表示している。そのため、実験全体として何%程度進んでいるのかが直感的に把握しにくいものとなっている。それを解決するための方法としては、反復回数と世代数を掛け合わせた値を進捗として利用するという方法が考えられる。具体的には、(現在の反復回数×現在の世代数）÷（総反復回数×総世代数）という値を算出し、この値を進捗の指標として用いる。例えば、図 5-6 のような状態（repetition: 2 of 10、generation: 20 of 300）であれば、（2×40）÷ （10×300）= 0.27 という値である。この値を見れば、全体として 30%程度実験が進んでいることが瞬時に分かり、現在までに経過した時間を鑑みれば、実験がいつ頃終了するのかも簡単に推測できる。

第6章　　テスト計画の立案

本章では、筆者の担当部分であるテスト計画について述べる。
6.1 節でテスト全体の方針を述べ、6.2~6.5 節で各テストフェーズの詳細について述べる。6.6~6.7 節ではテストの十分性と網羅性を確保するために筆者が導入した方法について説明する。6.8 節では、テスト結果の評価と考察を述べる。

6.1　　テスト方針

本プロジェクトでは、テスト工程における「テスト設計」、「テスト実装」、「テスト実施」の全てを専任する担当者は用意していない。これらの作業は、開発メンバ全員で分担している。筆者は、分担して行うテスト作業に対してメンバ間で共通の認識を持つために、本プロジェクトで行うテストの枠組みを計画した。本章では、筆者が計画した本プロジェクトにおけるテストの枠組みとその中で検討した品質を守るための対策について述べる。

本プロジェクトでは、テストを「単体テスト」、「結合テスト」、「総合テスト」、「受け入れテスト」の 4 つのレベルに分け実施する。図 6-1 は、各テストの実施フェーズを示したものである。単体テストは各スプリントの実装フェーズと並行して実施し、結合テストは単体テスト完了後に実施する。総合テストと受け入れテストについては納品前にそれぞれ 1 回ずつ行う。以下で、それぞれのテストの詳細を述べる。

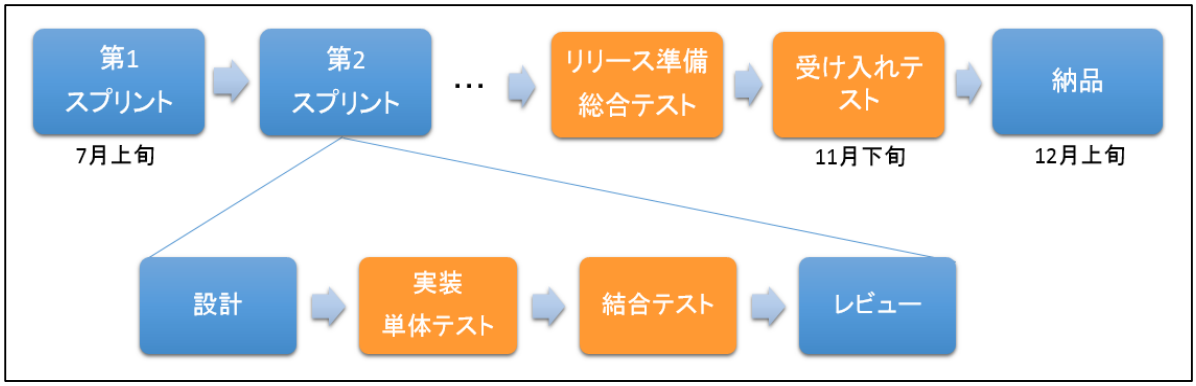


図 6-1　テストの流れ

6.2　　単体テスト

本プロジェクトにおける単体テストの概要を表 6-1 に示す。一般に単体テストは、ソフトウェアを構成する最小単位である関数やメソッドについての品質を確認する作業であると言われている。本プロジェクトの単体テストでは、MVC モデルにおけるモデルクラス、コントローラクラスの各メソッドを対象として、仕様通りの処理が実装されているかを確認する。また、バリデーションチェックも同時に行う。ビュークラスのテストについては、結合テスト工程で行うこととした。その理由は、結合テストのツールを用いても同等のテストをすることができ、かつ簡単にテストコードを実装できると考えたためである。

単体テストの担当者は、実装者が担当することにした。テスト専任者を設け、専任者がテストを実施するのが、実装者の思い込みによる要件漏れがないか防ぐ意味でも理想である。しかし、その場合テスト担当者は実装コードを読んで理解しなければならず、大幅な時間のロスにつながる。本プロジェクトでは、単体テストにおける実装者の思い込みや要件漏れを防

ぐためにいくつかの対策を実施することを想定しているため、実装者が行うこととした。ここで述べた対策については、節で説明する。

単体テストでは、RSpec というテスト自動化ツールを利用している。テスト自動化ツールを導入するとテストコードを書く手間が増えるが、ソースコードが改変されたときでもツールを実行するコマンド一つで全てのテストを再実行することができるというメリットがある。本プロジェクトではアジャイル手法を利用していることもあり、改変のリスクは非常に高いと判断し、導入を決定した。Rspec の導入は中西が行った。

表 6-1　単体テストの概要

対象	下記のクラスのメソッド ・コントローラクラス ・モデルクラス
確認内容	・メソッドが仕様通りに動作するか ・バリデーションチェック
担当者	実装者
環境	開発環境
ツール	RSpec

6.3　結合テスト

本プロジェクトにおける結合テストの概要を表 6-2 に示す。本プロジェクトにおける結合テスト工程では、単体テストが完了したモジュールを結合させて、一つの機能が仕様通りに動作すること確認する。具体的には、ブラウザ上で GUI 操作（ボタンのクリック、テキストフィードへの入力）をしていき、一つひとつの機能が動作することを確認する。これにより、ユーザ視点での各機能の振舞、インタフェース間の整合性が取れているかを検証する。

単体テストとは異なり、担当者を、該当機能を実装していないものを割り当てた。その理由は、この工程におけるテストはトップダウンのテストであるため、実装コードを理解していなくても十分に実施可能だからである。テストの実施者が実装の担当者以外であれば、実装時に漏れてしまった機能がチェックされることが期待できるからである。

結合テストも単体テスト同様、自動テストツールを利用している。その理由は、単体テストと同様である。Turnip,Capybara というツールを利用すると、GUI 操作を自然言語で表現することができる。例を挙げる。図 6-2 は「実験パッケージを削除する」という機能に対する Turnip のテストコードを一部抜粋したものである。テストコード全てが自然言語で記述することがわかる。自然言語で記述されているため、実装部分を知らない担当者や顧客もテストコードを読むことができ、どのようなテストを行っているのかを理解することができる。

```
機能: #195_実験パッケージを削除する機能

  シナリオ: 実験パッケージを削除する
    前提 “JavaTest”サンプルパッケージを作成する
    もし 実験選択画面を表示する
    ならば "JavaTest"テキストを表示する
    もし "JavaTest"の行の"del"リンクをクリックする
    ならば “JavaTest“がリストから消える
```

図 6-2　Turnip テストの例「実験パッケージを削除する機能」

表 6-2　結合テストの概要

対象	機能モジュール
確認内容	・シナリオ通りに機能が動作するか ・画面遷移
担当者	・該当機能を実装していないメンバ ・イテレーション 1~5 までの機能に関しては、例外的に筆者が担当
環境	開発環境
ツール	Turnip,Capybara

6.4　総合テスト

表 6-3 は、総合テストの概要をまとめたものである。

単体テスト・結合テストは開発環境でテストを実施するが、総合テストでは本番環境でテストを行う。その理由は、ハードウェアや OS、ミドルウェア（Web サーバや DBMS など）のバージョンの違いによるトラブルや個別の障害が発生しないかを確認するためである。また、テストデータは本番で使用するデータを用いる。本番用のデータを使うことにより、想定外のデータの存在によりエラーが発生しないかを確認する。また、十分なパフォーマンスが得られているかどうかも確認する。

テストケースは、結合テスト時に使用したものと同じものを使用する。ただし、総合テストではツールを利用せずに手動で動作を確認する。これは、より本番に近い状況を想定してテストするためである。

一般には、システムが複数人の利用に耐えられるかの負荷テストなども行うが、本プロジェクトでは基本的にユーザは一人で利用することを想定しているため、複数人が同時アクセスした場合の挙動のテストは行わない。

表 6-3　総合テストの概要

対象	システム全体
確認内容	・本番環境で全ての機能がシナリオ通りに動作するか

担当者	・栗（実施） ・人見（準備）
実施フェーズ	リリース準備
ツール	手動

6.5　受け入れテスト

表 6-4 は、受け入れテストの概要をまとめたものである。このテスト工程では、実際に顧客にシステムを利用していただき、顧客の要求した通りの機能が実装されているかの最終チェックを行う。また、それと同時にシステムの使いやすさなども評価していただくこととした。

受け入れテストのテストケースは、基本的に結合テストのシナリオと同じである。ただし、すべてのテストシナリオを顧客に実行していただくのは大きな負担となってしまう。そのため、各機能の正常系のテストだけを行っていただくこととした。結合テストのシナリオは自然言語で書かれているため、結合テストのシナリオをそのまま顧客が読んで実施することができる。

表 6-4　受け入れテストの概要

対象	システム全体
実施内容	・全ての機能がシナリオ通りに動作するか ・顧客の要求とあっているか ・使いやすさ
担当者	顧客
実施フェーズ	納品前
ツール	手動

6.6　テストの十分性

テストをどこでやめるか、その判断は非常に難しい。テストではバグがあることは証明できても、バグがないことは証明できないからである。また、開発期間も限られている。テストの十分性を確認するために一般に用いられる方法は、(1)ホワイトボックステストでカバレッジを計測する(2)バグの検出目標値を設定し、目標の達成状況を評価する(3)バグの検出状況をグラフ化して、潜在バグ数を予測し信頼性の向上度を確認する、などの方法がある。ただし、(1),(2),(3)のいずれについてもいくつかの欠点が指摘されている。 [17](1)については、プログラム作成者が誤った判断で機能の実装をしていなかった場合には、数値に反映されない。(2)については、組織が成熟しており十分な過去の実績値を保持していないと信頼性のある予測をすることができない。(3)については、テストの実施順序の偏りがある場合や開発者の力量のバラツキが大きいと判断を誤る場合がある。また、(2),(3)については管理も非常に難しい。

これらの対策の費用対効果を考慮し、本プロジェクトでは、”ホワイトボックステストでカバレッジを計測する”、”レビューによる十分性チェック”の 2 点でテストの十分性チェックを行うこととした。レビューによる十分性チェックは、実装者の思い込みなどにより実装がされていない機能をチームレビューによって見つけるという方法である。具体的には、テスト観点表やテストケースを見て、漏れがないかをメンバ間でチェックしていく。これにより、ホワイトボックステストのカバレッジ計測ではカバーすることができない部分の十分性チェ

ックを行うことができる。

十分性チェックの指標として、その他にも試験密度（どのくらいのボリュームでテストをしたか）、バグ密度（テストの結果、どのくらいのバグを出したか）などのメトリクス指標も存在する。これらのメトリクスを利用することも検討したが、これらの数値はあくまで、テスト項目数、バグ件数が少ないモジュールの場所を特定するための判断材料にしかならない。そのため、基本的には”カバレッジ計測”、”レビュー”の 2 軸でテストの十分性を確保することとした。

6.7 テストケースの網羅性

ここでは、テストケースの網羅性を確保するために筆者が行った対策について説明する。6.1 節で述べたように、本プロジェクトでは、テストケースの作成からテストの実施までを、メンバで分担して行っている。そのため、テストケースの網羅性は個人の裁量によって大きく変化する。それを防ぐために、筆者は「テスト観点表」を作成した。図 6-3 はその表を簡略化し、その一部を切り取ったものである。この表の列には、すべての機能で共通してチェックすべきテスト観点がならんでおり、行には機能が記述してある。担当者は、テストケースを洗い出す際に、”どういう観点でテストする必要があるか”一つずつテスト観点を確認していく。テストする必要がある項目は空欄で、確認する必要がない観点に対してはハイフンを入力していく。そして、テストの実施後に、テストが実施された項目に、アルファベットの”D”を入力するようにする。この表に文字を入力していく過程で、担当者は、テスト観点を一つ一つ確認するため、テストケースに漏れを防ぐことが期待できる。しかし、問題が一つある。それは、あらかじめ列に列挙したテスト観点に漏れがある場合である。この場合、チェック項目が欠落していることになるので、結果的にテストすべき項目が抜け落ちてしまっていることになる。それを防ぐために、筆者はこのテスト観点を洗い出す際に、マインドマップを用いたテスト観点抽出法 [18]を利用し、できるだけ網羅的にテスト観点を上げる事ができるように工夫した。図 6-4 はその一部である。しかし、それでも完全にテスト観点の漏れを防ぐことはできない。そのため、テスト担当者が表に記載されていないテスト観点を見つけた時点で、表に随時追加するようにした。以上の取り組みをすることで、テストの網羅性が向上するよう努めた。

		入力							
機能		テキストフィールド							
大項目 必須ではない：- 必須だがやっていない：空欄 実施済み：D	小項目	最大文字数以上	最小文字数未満	最大値以上	最小値未満	未入力	数値のみ	文字のみ	指定値以外
実験パッケージを編集する機能		D		-	-	D	-	-	D

図 6-3 テスト観点表

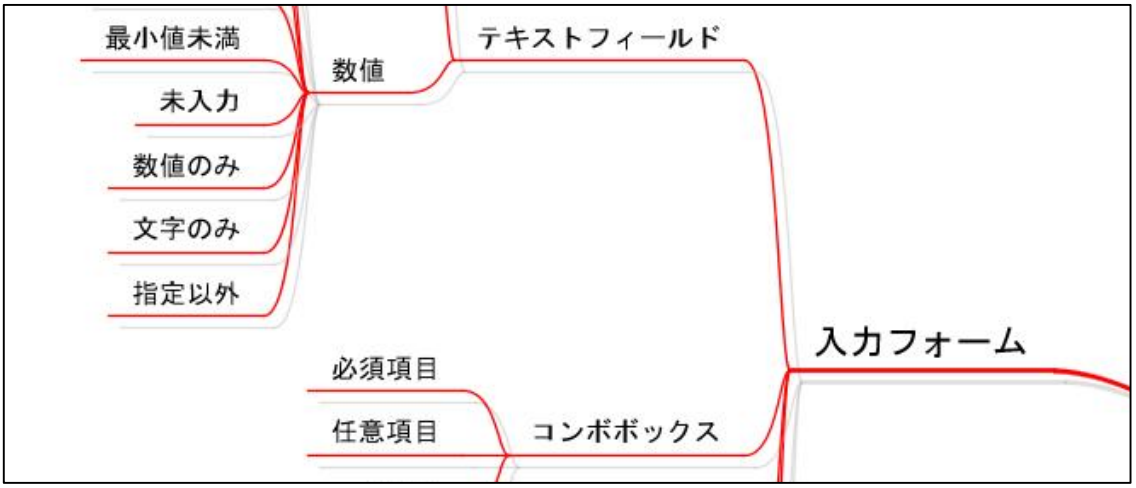


図 6-4　テスト観点を洗い出すためのマインドマップ

6.8　　テストの考察

この節では、テスト結果の考察を述べる。まず、6.8.1、6.8.2 項でテストの品質向上のために筆者が導入した施策に対する考察を述べ、6.8.3 項でテスト全体の振り返りを述べる。

6.8.1　　テストの十分性について

本プロジェクトでは、「カバレッジ率を 100%」、「メンバ間のレビュー」を実施することで、テストの十分性を確保することとしていた。図 6-5 はプロジェクト終了時のカバレッジ率別のクラス数を示したもので、Simplecov というテストコードからカバレッジ率を自動で算出するツールを用いて計測したものである。図を見てみると、カバレッジ率 100%をテストの終了条件の一つとしていたが、100%になっていないクラスが見受けられる。100%に達しているものは、全クラス 29 中 18 個である。それ以外の 11 個のクラスについては、カバレッジ率 100%を満たしていなかった。その理由について考察する。カバレッジ率を満たしていないクラスを分析してみると、以下の 3 つが原因であった。

- 自動テストできないソースコード
- 論理的には通り得ないソースコード
- テストケース漏れ

自動テストできない箇所とは、例えばメールの送信などである。メールがユーザに通知されたかどうかをテストするためには、ユーザのメールボックスを見て判断する必要がある。しかし、その処理を自動テストのコードで表現することは困難である。そのため、その部分のテストコードは記述することができず、カバレッジ率を低下させてしまっていた。しかし、このコードに対しては自動テストではなく手動で確認しているため、システムの品質には影響していない。

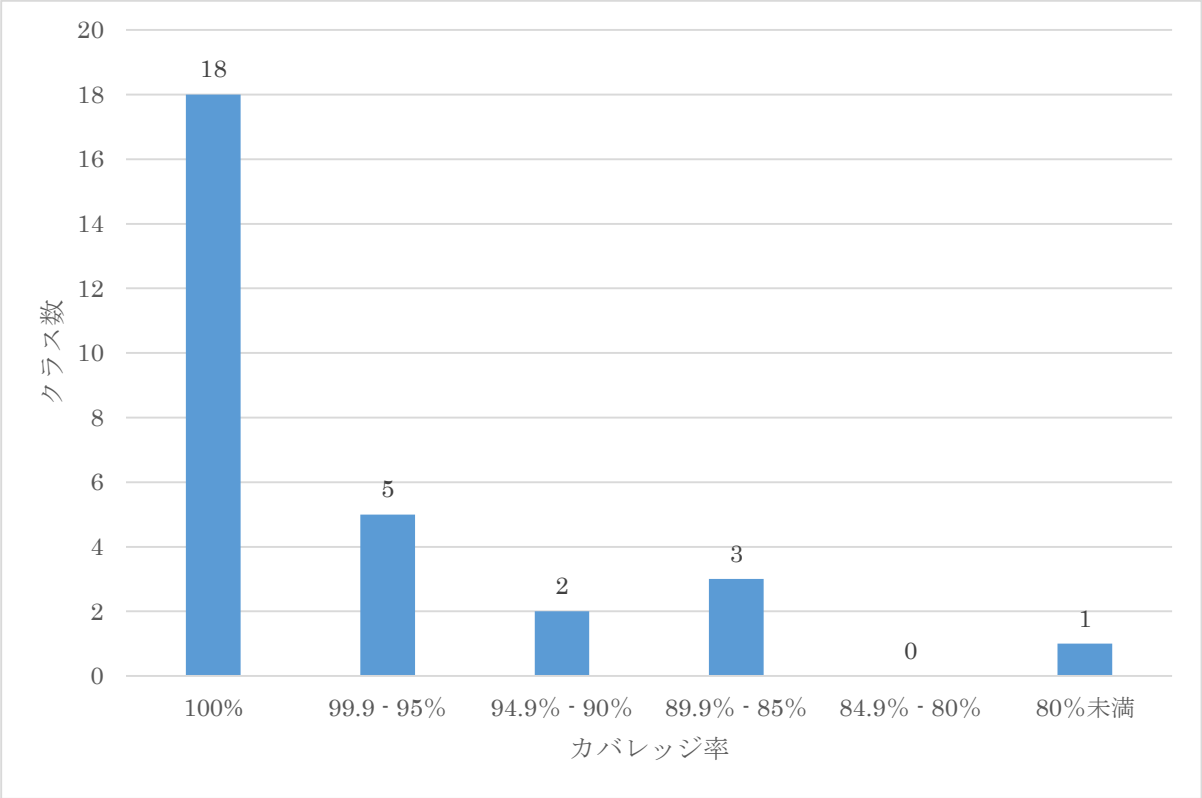


図 6-5　カバレッジ別のクラス数

論理的には通り得ないコードとは、if 文や switch 文などの条件分岐において論理的に実行されることがないコードである。その分岐を通るテストコードは書くことは困難である。このようなコードの存在により、カバレッジ率が低下していた。

最後は、単純なテストケース漏れがあったケースである。開始当初はカバレッジ率 100%を保っていたが、プロジェクト中盤から上記 2 つの原因などによりカバレッジ率を 100%に保つことが難しくなった。これにより、開発者は、カバレッジ率が低下していても上記 2 点によりカバレッジ率が低下しているのだと勘違いしてしまうといったことが考えられる。このような理由からメンバ間でのチェックがおろそかになってしまった。テスト担当者として、定期的にそのようなテストケース漏れを監視し、発見後タスクとして発行するといった対策が必要だった。

6.8.2　テストの網羅性について

6.7 項で述べたように、筆者はテストの網羅性確保のためにテスト観点表を作成した。しかし、他の開発メンバからは実施すべきだと感じていたが使用できなかったという報告を受けた。それは、膨大なテストケースに対して、エクセルシートのテスト観点を一つずつチェックし、記入していくのは非常に煩雑で時間がかかるという理由からであった。また、想定していたよりもテストコードの記述に時間がかかり、進捗に遅れをきたしていたということも理由としてあげられる。結果的に、単体テストは、個人の判断によってテストケースを考えだすこととなったしまった。この問題の対応策は今後の課題であるが、開発者にとってもっと容易にチェックできる仕組みを考える必要があると考えている。

6.8.3　テスト全体の振り返り

まず、反省しなければならないことは、テスト計画立案にとりかかるのが遅くなってしま

ったことである。表 4-2 でも示したように、本プロジェクトでは、機能の開発を始めてから 2 か月後の 9 月からテスト計画を立て始めた。それまでは、単体テストしか実施しておらず、スプリントごとに品質が確保された状態のシステムを提供するというアジャイルの指針に沿っていなかった。また、その単体テストにおいても明確なガイドラインが存在せず、全て実装者の裁量で実施されていた。そのため、7 月～9 月までに実装した機能に対して、9 月以降に単体テストコードの修正や結合テストの実施を行う時間が必要になり、大きなオーバーヘッドになってしまった。テスト計画については、機能の実装を始める 7 月以前に十分に検討しておく必要があった。

次に、テストに関するレビューについて述べる。本プロジェクトでは、テストの終了条件としてレビューを行うことを計画していた。しかし、実際には実施が徹底されていなかった。それには 2 つの原因があったと筆者は考えている。一つは、1 スプリントで多くの機能を実装しようとしすぎたことである。想定していたよりも実装に多くの時間を費やしてしまい、テストに充てる時間を十分に確保できなかった。その結果、テストケースのレビューまで手が回らなくなってしまった。もう一つの理由は、レビューをするための体制が整っていなかったことである。テストのレビューは基本的にテストコードを見て、テストケースに漏れがないかを確認することとしていた。しかし、テストコードだけを見ただけではテストケースが網羅されているかを判断するのは難しい。それは、テストコード実装者がどのようなテスト観点を考慮し、その観点に対してどのようなテストケースをあげたかがレビュー者にはわからないからである。このような状態でレビューを実施してもテスト漏れを発見するのは難しい。以上のような理由から、メンバのレビューに対する意識が下がり、実施されなくなってしまったであると筆者は考えている。

本プロジェクトのテスト計画立案を担当して感じたのは、定期的にテスト計画を見直し修正していく必要があるということである。最初に立てた計画がすべて上手くいくとは限らない。定期的なテスト実施状況のモニタリングや、メンバに話を聞くなどして、問題を把握・改善していく必要があると感じた。

最後に、各テストフェーズのテストケース数を表 6-5 に示す。

表 6-5　テストフェーズ別のテストケース数

テストフェーズ	テストケース数
単体テスト	425
結合テスト	82
総合テスト	82
受け入れテスト	16

第7章　　システム評価と課題

この章では、受け入れテスト後に顧客に実施していただいたアンケートの結果とそれに対する考察を述べる。

7.1　　アンケート結果

受け入れテスト後に実施したアンケートの結果を表 7-1 に示す。各アンケート項目に対して、顧客に 5 段階で評価していただいている。表中の括弧内の数値は 5 段階評価における評価値を示している。

表 7-1　アンケート結果

アンケート項目	評価
実験パッケージ管理モジュール	
実験プログラムを管理機能について、使いやすいですか？	とても使いやすい(5)
実験プログラムの管理機能を使うことで、従来の方法で実験の情報を管理するときよりも管理にかかる手間が改善されましたか？	とても手間が減った(5)
パラメータファイル作成モジュール	
パラメータファイルの作成機能は使いやすいですか？	どちらともいえない(3)
パラメータファイルの作成機能を利用することで、従来と比べてパラメータファイルの作成にかかる手間は軽減されましたか？	少し手間が軽減された(4)
パラメータファイルを一括作成できる機能は使いやすいですか？	すこし使いやすい(4)
パラメータファイルを一括作成できる機能を利用することで、従来と比べてまとめてパラメータファイルを作成するときにかかる手間は軽減されましたか？	手間が軽減された(5)
実行管理モジュール	
実験を実行できる機能は使いやすいですか？	とても使いやすい(5)
実験を実行できる機能を利用することで、従来のコンソールから実験を実行する場合と比べて、どの程度手間が軽減されましたか？	少し手間が軽減された(4)
実行状態監視モジュール	
実験の実行状態をメールで通知する機能は使いやすいですか？	とても使いやすい(5)
実験の実行状態をメールで通知する機能を利用することで、従来の実行状態がわからなかった場合と比べると、監視にかかるコストや失敗した時のリスクはどの程度改善されたと思いますか？	とても良くなった(5)
実験の実行状態を Web から確認できる機能は使いやすいですか？	とても使いやすい(5)
実験の実行状態を Web から確認できる機能は使いやすいですか？	とても使いやすい(5)
実験の実行状態を Web から確認できる機能を利用することで、従来の実行状態がわからなかった場合と比べると、監視にかかるコストや失敗した時のリスクはどの程度改善されたと思いますか？	良くなった(4)
実験結果可視化モジュール	
実験結果のグラフ化機能は使いやすいですか？	どちらともいえない(3)
実験結果のグラフ化機能で表示されるグラフは見やすいですか？	どちらともいえない(3)
実験結果のグラフ化機能を利用することで、従来のグラフ化方法と比べて、どの程度手間が軽減されましたか？	少し手間が軽減された(4)

マニュアル	
運用マニュアルから目的の情報を簡単に見つけることができましたか？	どちらともいえない(3)

7.2 アンケート結果の考察

実験パッケージ管理モジュール、実行管理モジュール、実行状態監視モジュールについては「使いやすさ」、「手間の軽減」という点で概ね良い評価（5 段階評価中 5 または 4 点）を得ることができた。しかし、パラメータファイル作成モジュール、実験結果可視化モジュールについては、5 段階評価中 3 点という評価がいくつか見られた。この点については、自由記述欄でご意見をいただいている。

まずは、パラメータファイル作成モジュールについて述べる。このモジュールについては「パラメータファイルの編集ができない」、「パラメータ一括作成をする際に、パラメータ値に負の値を入力できない」といった 2 つのご意見をいただいた。GAM が提供する「パラメータファイルを編集する機能」というのは、既存のパラメータファイルを編集して新たなパラメータファイルを作成するというものである。つまり、既存のファイルが編集した内容で上書きされるわけではなく、別のファイルとして保存されるようになっている。その理由は、実験の実行中に、使用しているパラメータファイルを編集してしまうとパラメータファイルと実験結果の整合性がとれなくなってしまうからである。研究をするうえでは、実験結果とその実験結果を与えたパラメータを常にセットで保存しておく必要がある。しかし、実行中にパラメータファイルを編集してしまうと、実験結果に対応するパラメータファイルが上書きされ消えてしまうことになる。このような理由から、上書き編集機能は作らないこととした。この点については、受け入れテスト後に、この理由を説明しご納得いただけた。2 つ目のパラメータ一括作成時の問題については、顧客との仕様の確認が取れていなかったことが原因であった。受け入れテスト後に負の値を入力できるように変更を加えた。

次に実験結果可視化モジュールについて述べる。このモジュールについては、「実験結果のテキストファイルを取得する手段がない」、「グラフ上でマウススクロールすると、グラフが拡大・縮小してしまう」というご指摘をいただいた。一つ目の問題に対しては、実験結果ファイルを閲覧できるページを追加することにより対応した。2 つ目の問題については、グラフ描画ライブラリの拡大・縮小機能をオフにすることにより対処した。これら 2 つの問題に対処することで、可視化モジュールに対してもご満足を得ることができた。

運用マニュアルについては、どちらともいえないという回答を得た。これは、運用マニュアルを日本語で作成したためであると考えられる。当初、顧客からは英語で作成してほしいとの要望をいただいたが、開発期間内に作成することは難しく日本語で作成することとなってしまった。この点については今後の課題としたい。

第8章　おわりに

筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻高度 IT 人材育成のための実践的ソフトウェア開発専修プログラムにおける研究開発プロジェクトにおいて、筆者は、同学の Aranha Claus De Castro 助教を顧客として、「進化的計算の実験可視化システム」を開発した。

本プロジェクトの目的は、顧客の EC を利用した計算機実験における「コマンド入力による手間が大きい」、「実験の実行プロセスの変化に対して即座に対応できない」、「パラメータ設定の手間が大きい」、「実験結果の分析の手間が大きい」といった問題を解決することであった。その実現のために、本プロジェクトでは、実験に必要なファイル群を管理するための「実験パッケージ管理」機能、実験を実行・停止するための「実行管理」機能、実験の実行状態を監視する「実行状態監視」機能、パラメータ作成を支援するための「パラメータファイル作成」機能、実験結果を可視化する「実験結果可視化」機能をもつ GAM システムを開発した。

開発したシステムの「実験パッケージ管理」、「実行管理」、「実行状態監視」機能については概ね良い評価を得ることができた。「パラメータファイル作成」、「実験結果可視化」機能についても顧客からのご意見をもとに改良することで、ご納得いただけた。

筆者は、このシステムの開発活動において、テスト計画の立案・実践と 2 つのモジュール・機能の開発を行うことでチームに貢献した。テスト計画の立案では、テスト全体の計画を立てテストの円滑化を図るとともに、品質を保つための方法を検討した。開発面では、「実験パッケージ管理」機能、「実行状態監視」の一部である「実験進捗表示機能」を開発し、顧客からは概ね良い評価を得ることができた。

謝辞

本プロジェクトを進めるにあたり、多くのご助言とご指導を頂きました顧客兼課題担当教員の Aranha Claus De Castro 助教，指導教員の田中二郎教授ならびに本プロジェクトに関わって下さった他教員の皆様に深く感謝致します。

本プロジェクトのチームメンバである中西陽平君、岡谷亮君、栗克君には、広くお世話になりました。皆と同じチームで一つのプロジェクトを完了できたことを誇りに思います。本当にありがとうございました。深く感謝致します。

最後に、様々な面でご支援いただきました家族、友人、大学生活でお世話になった全ての方々に心より感謝申し上げます。

参考文献

[1] Thomas Back and Hans-Paul Schwefel, "An Overview of Evolutionary Algorithms for Parameter Optimization," *Evolutionary Computation,* vol.1, no.1, pp.1-23, 1993.

[2] Thomas Back, David B. Fogel, and Zbigniew Michalewicz, "Hand book of evolutionary computation," *Oxford University Press,* 1997.

[3] 樋口哲也, 北野宏明, “遺伝的アルゴリズムとその応用,” *情報処理,* vol.34, no.7, pp. 871-883, 1993.

[4] 廣安知之, 三木光範, 佐野正樹, 谷村勇輔, 濱崎雅弘, “2 個体分散遺伝的アルゴリズム,” *計測自動制御学会論文集,* vol.38, no.11, pp.990-995, 2002.

[5] 三木光範, 畠中一幸, “並列分散 GA による計算時間の短縮と解の高品質化,” *JSME 最適化シンポジウム講演論文集,* 1998.

[6] 斉天, 川名学, 平松友康, 宮崎道雄, “間欠性カオス写像を用いた遺伝的アルゴリズムのパラメータ動的制御,” *電気学会論文誌. C, 電子・情報・システム部門誌,* vol.117, no.9, pp.1238-1244, 1997.

[7] 坂和正敏, 田中雅博, 遺伝的アルゴリズム, 朝倉書店, 1995.

[8] Pat Langley and Jaime G. Carbonell, "Approaches to Machine Learning," *Journal of American Society for Information Science,* vol.35, pp.306-316, 1984.

[9] Claus Aranha, Carlos R.B. Azevedo, and Hitoshi Iba, "Money in trees: How memes, trees, and isolation can optimize financial portfolios," *Information Sciences,* pp.184-198, 2011.

[10] Rodrigo T. Peres, Claus Aranha, and Carlos E. Pedreira, "Optimized Bi-Dimensional Data Projection For Clustering Visualization," *Information Sciences,* no.232, pp.104-115, 2013.

[11] Hirotaka Takeuchi and Ikujiro Nonaka, "The new new product development game," *Harvard Business Review,* pp.285-305, 1986.

[12] 平鍋健児, 天野勝, “プロジェクトファシリテーション 実践編 朝会ガイド,” 2013. [オンライン], http://objectclub.jp/download/files/pf/MorningMeetingGuide.pdf.

[13] 平鍋健児, 野中郁次郎, アジャイル開発とスクラム, 翔泳社, 2013.

[14] 天野勝, “プロジェクトファシリテーション 実践編 ふりかえりガイド,” 2013. [オンライン], http://www.objectclub.jp/download/files/pf/RetrospectiveMeetingGuide.pdf.

[15] Sam Ruby, Dave Thomas, and David Heinemeier Hansson, Rails によるアジャイル Web アプリケーション開発, オーム社, 2012.

[16] 山田祥寛, Ruby on Rails 3 アプリケーションプログラミング, 技術評論社, 2011.

[17] 飯田志津雄, “基幹システムにおけるテスト十分性の確保,” *Unisys 技法,* vol.28, no.4, pp.513-525, 2009.

[18] 池田暁, 鈴木三紀夫, ソフトウェアテスト入門, 技術評論社, 2011.

付録

付録 A：導入マニュアル

対象環境

GAM システムは以下の環境でのインストールを対象としています。なお、GAM システム上で実行する実験プログラムによって必要な性能は左右されますので、必要に応じて性能を調整してください。

OS: Ubuntu Server 12.04LTS
Memory: 1GB 以上
Disk: 1GB 以上

インストール手順

1　事前準備（システム管理者アカウントによる作業）

1.1　依存パッケージのインストール

GAM システムでは以下のパッケージを利用しています。GAM システムのインストールの前にインストールを済ませておいてください。

コマンド例

```
$ sudo apt-get install git gcc make build-essential automake zlib1g-dev libssl-dev libreadline6-dev libyaml-dev libxml2-dev libxslt-dev libmysqld-dev libcurl4-openssl-dev apache2 apache2-prefork-dev libapr1-dev libaprutil1-dev mysql-server postfix
```

1.2　JRE のインストール

GAM システムはユーザがアップロードした Java 形式または Python 形式の実行ファイルを実行します。Java 形式のファイルを実行するために、JRE をインストールしてください。

コマンド例

```
$ sudo apt-get install openjdk-7-jre-headless
```

1.3　mysql に gamadmin ユーザを作成

GAM システムではデータの保存に MySQL を利用しています。利用するデータベース名は[gam]となっています。gam データベースに対してすべての実行権限を持つ gamadmin ユーザを作成し、パスワードに Xe7AnfZ3 を設定してください。

コマンド例

```
$ mysql -u root (-p)
mysql>GRANT ALL PRIVILEGES ON gam.* TO 'gamadmin'@'localhost' IDENTIFIED BY 'Xe7AnfZ3';
```

```
mysql>exit
```

1.4 gam-system ユーザの作成

GAM システムを配置するにあたり、gam-system ユーザを作成してください

コマンド例

```
$ sudo adduser gam-system
```

2 インストール作業（gam-system ユーザでの作業）

システム管理者アカウントでログインしている場合は、gam-system にログインし直してください。

コマンド例

```
su gam-system
```

2.1 rbenv および ruby-build のインストール

Ruby のバージョン管理システムである rbenv を利用します。これにより、既存の Ruby のバージョンに依存することなく GAM システムのインストールを行うことができるようになります。

2.1.1 ダウンロード

以下の通りコマンドを実行し、rbenv および ruby-build をダウンロードします。

コマンド例

```
$ git clone https://github.com/sstephenson/rbenv.git ~/.rbenv
$ git clone https://github.com/sstephenson/ruby-build.git
~/.rbenv/plugins/ruby-build
```

2.1.2 rbenv の設定

~/.bashrc ファイルの末尾に以下の内容を追加してください。

追記内容

```
export PATH="$HOME/.rbenv/bin:$PATH"
eval "$(rbenv init -)"
```

2.1.3 rbenv の有効化

~/.bashrc の再読み込みを行います。

コマンド例

```
$ source ~/.bashrc
```

2.2 ruby のインストール

GAM システムは Ruby on Rails で動作するため、専用の Ruby をインストールします。以下の通りにコマンドを実行してください。

コマンド例

```
$ rbenv install 1.9.3-p484
$ rbenv global 1.9.3-p484
$ rbenv rehash
```

2.3 bundler のインストール

GAM システムで利用する外部ライブラリを管理するためのツールとして、bundler をインストールします。以下のようにコマンドを実行してください。

コマンド例

```
$ gem install bundler
$ rbenv rehash
```

2.4 GAM システムのインストール

GAM システムをダウンロードし、利用可能にするための準備を行います。なお、ここでは~/www/gam にインストールすることを前提としています。他のパスへインストールする場合は以後適宜読み替えてください。

コマンド例

```
$ git clone http://vm12.sit.cs.tsukuba.ac.jp/git/gam ~/www/gam
$ cd ~/www/gam
$ bundle install --without development test
$ rake db:create RAILS_ENV=production
$ rake db:migrate RAILS_ENV=production
```

2.5 passenger のインストール

Apache2 Web Server 上で GAM システムを動作させるために、Phusion Passenger モジュールをインストールします。以下のようにコマンドを実行してください。

コマンド例

```
$ gem install passenger
$ rbenv rehash
$ passenger-install-apache2-module
```

※インストール中に 2 度 Enter を押す確認が出てきます。
1 度目はインストールをするかどうかの確認です。Enter を押すことでインストールが実行されます。2 度目は Apache のモジュールに関する設定情報が出てきます。Enter を押す前に設定情報をメモしてください。Enter を押すことでインストールが完了します。

2 度目の Enter を押すかどうかの確認メッセージ

以下のような内容が表示されますので、
LoadModule,PassengerRoot,PassengerDefaultRuby の行をメモしておいてください。

出力例

```
The Apache 2 module was successfully installed.

Please edit your Apache configuration file, and add these lines:

    LoadModule passenger_module /home/gam-system/.rbenv/versions/1.9.3-p484/lib/ruby/gems/1.9.1/gems/passenger-4.0.26/buildout/apache2/mod_passenger.so
    PassengerRoot /home/gam-system/.rbenv/versions/1.9.3-p484/lib/ruby/gems/1.9.1/gems/passenger-4.0.26
    PassengerDefaultRuby /home/gam-system/.rbenv/versions/1.9.3-p484/bin/ruby

After you restart Apache, you are ready to deploy any number of Ruby on Rails applications on Apache, without any further Ruby on Rails-specific configuration!
.----------------------------------------------
Press ENTER to continue.
```

3　Apache の設定（システム管理者アカウントによる作業）
GAM システムを Apache2 Web Server 上で動作させるための設定を行います。gam-system でログインしている場合は、システム管理者アカウントにログインし直してください。

3.1　Passenger モジュールの設定
2.4 でメモした内容を以下のようにそれぞれ設定ファイルに書き出します。

3.1.1 /etc/apache2/mods-available/passenger.load

2.4 でメモした LoadModule の行のみを記入した/etc/apache2/mods-available/passenger.load というファイルを作成します

例

```
LoadModule passenger_module /home/gam-system/.rbenv/versions/1.9.3-p484/lib/ruby/gems/1.9.1/gems/passenger-4.0.26/buildout/apache2/mod_passenger.so
```

3.1.2 /etc/apache2/mods-available/passenger.conf

2.4 でメモした内容を以下の通り編集し、/etc/apache2/mods-available/passenger.conf というファイルを作成します
1 行目に<IfModule mod_passenger.c>
2 行目に PassengerRoot の行
3 行目に PassengerDefaultRuby の行
4 行目に</IfModule>
を書き出します。

例

```
<IfModule mod_passenger.c>
PassengerRoot /home/gam-system/.rbenv/versions/1.9.3-p484/lib/ruby/gems/1.9.1/gems/passenger-4.0.26
PassengerDefaultRuby /home/gam-system/.rbenv/versions/1.9.3-p484/bin/ruby
</IfModule>
```

3.1.3 モジュールの有効化

以下のコマンドを実行し Passenger モジュールを有効化します

コマンド例

```
$ sudo a2enmod passenger
```

3.2 Apache のサイト設定用ファイルの作成

Apache2 Web Server 上でサイトを構築するための設定情報を書き出します。
すでに Apache 上で別の Web サイトが動作しているかどうかで設定が変わりますので、3.2.1 もしくは 3.2.1 のみを実行してください。

3.2.1 Apache に既存 Web アプリケーションが存在しない場合

/etc/apache2/sites-available/gam-system に以下の内容を書き出します。

/etc/apache2/sites-available/gam-system の内容

```
<VirtualHost *:80>
```

```
ServerName （サーバの FQDN）
DocumentRoot /home/gam-system/www/gam/public
RailsEnv production
<Directory /home/gam-system/www/gam/public>
AllowOverride all
Options -MultiViews
</Directory>
</VirtualHost>
```

3.2.2 Apache で既存 Web サイトを運用している場合

既存 Web サイトを運用するにあたり作成したセッティングファイルを、以下のように編集してください。

```
<VirtualHost *:80>
ServerName ...(既存の設定)...
DocumentRoot  ...(既存の設定)...

Alias /gam /home/gam-system/www/gam/public
<Directory /home/gam-system/www/gam/public>
RailsEnv production
PassengerBaseURI /gam
PassengerAppRoot /home/gam-system/www/gam
AllowOverride all
Options -MultiViews
</Directory>
</VirtualHost>
```

3.3 サイトの有効化

以下のコマンドを実行し、サイトを有効化してください。

コマンド行例

```
$ sudo a2ensite gam-system
$ sudo service apache2 reload
```

動作確認

Web クライアントからシステムにアクセスし、正常に表示されればインストール完了となります。

アンインストール手順

1. gam-system ユーザの削除

   gam-system ユーザとそのホームディレクトリをすべて削除します。

2. mysql からデータの削除
MySQL 上に作成されている gamadmin ユーザを削除し、gam データベースを drop します。

```
$ mysql -u root (-p)
mysql>REVOKE ALL PRIVILEGES ON gam.* FROM 'gamadmin'@'localhost';
mysql>DROP DATABASE gam
```

3. Apache2 Web Server の設定を削除
以下の手順で Apache2 Web Server から GAM システムの設定情報を削除します。

実行例

```
$ sudo a2dismod passenger
$ sudo rm /etc/apache2/mods-available/passenger.load
$ sudo rm /etc/apache2/mods-available/passenger.conf
$ sudo a2dissite gam-system
$ sudo rm /etc/apache2/sites-available/gam-system
```

付録 B：ユーザマニュアル

事前準備

GAM で実験を行うには、実行ファイルとパラメータファイルを作成し、それらを適切なディレクトリ構成に配置する必要があります。また、GAM 用の出力ライブラリを用いることで、計算結果を自動でグラフ表示できるようになります。このページでは、これらの作成規則について以下の順に説明します

- 実行ファイルの作成

  GAM で実験を行うには、実行ファイルの第一引数にパラメータファイルのパスを指定する必要があります。
  実行ファイルを作成する際は、進化的計算のパラメータとして用いる値を、第一引数に指定するパラメータファイルの内容から参照して用いるようにしてください。

  GAM が実行ファイルを起動する方法は、実行ファイルのファイル形式に応じて下表のコマンドを実行することで行います。
  下表のコマンドで起動できるように、実行ファイルを作成してください。

ファイル形式	実行コマンドの例
Java 形式	java –jar foo.jar bar.par
Python 形式	py foo.py bar.par

- パラメータファイルの作成

実行ファイルに用いるパラメータファイルは、PAR 形式で記述する必要があります。
PAR 形式のファイルを作成するには、以下の 3 つの記述ルールに従ってファイル内にテキストを記述し、拡張子を”.par”として保存してください。

1. ルール１：パラメータ項目とパラメータ値
   パラメータ項目とパラメータ値は、”パラメータ項目= パラメータ値”の形式で記述します。

   例: example1.par

```
Algorism = GA
Generation = 200
```

2. ルール 2：コメント
   行頭に#を付けることで、GAM で パラメータファイルを編集する際に無視されるコメントを記述できます。
   パラメータ項目の行の一つ上の行にコメントを記述することで、画面上にコメント文を表示することができます。

例: example2.par

```
#Evolutionary parameters
Algorism = GA
Generation = 200
```

3. ルール 3：世代数と反復回数の最大値（任意）

パラメータ項目名を”repetition number”、”generation number”とすることで、GAM で実験の進捗を確認する際に世代数と反復回数の最大値を表示することができます。

```
#Repetition Number
repetition number = 10

#Generation Number
generation number = 200
```

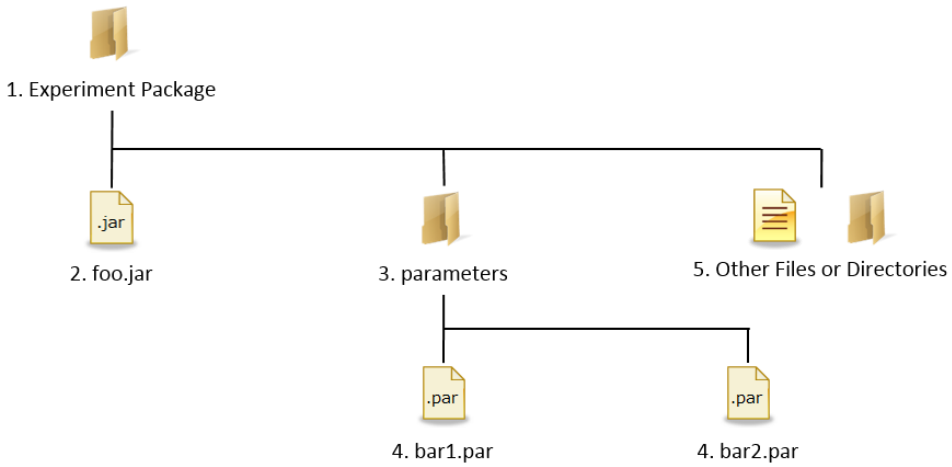
4. 注意

- ”repetition number”と”generation number”の大文字、小文字は区別されません
- ”repetition number”と”generation number”のそれぞれの単語間のスペースの有無は任意です

- ディレクトリ構成

作成した実行ファイルとパラメータファイルを下図の<ディレクトリ構成>と<ファイル規則>に従って配置し、下表の<対応圧縮形式>の中の圧縮形式で保存してください。GAM では、これにより作成した実験に関係するファイルを含んだ圧縮ファイルを実験パッケージと呼んでいます。実験パッケージを GAM にアップロードすることで、GAM で実験が行えるようになります。

<ディレクトリ構成>



<ファイル規則>

番号	項目名	ファイルの種類	命名規則	必須ファイル	説明
1	実験パッケージ	ディレクトリ	任意	-	実験パッケージのディレクトリ
2	実行ファイル	jar または py	任意	○	プログラムの実行ファイル
3	パラメータディレクトリ	ディレクトリ	parameters	-	パラメータファイルを格納するディレクトリ。複数のパラメータファイルを格納できる
4	パラメータファイル	.par	任意	-	実行ファイルで使用するパラメータを記述したファイル。アップロード機能により後から追加することも可能
5	その他のファイル	任意	任意	-	ユーザが任意に配置できるファイルやディレクトリ

<対応圧縮形式>

圧縮形式	拡張子
Tar	.tar
Gzip	.tar.gz , .tgz
Bzip	.tar.bz2 , .tbz2
ZIP	.zip
RAR	.rar

<注意>

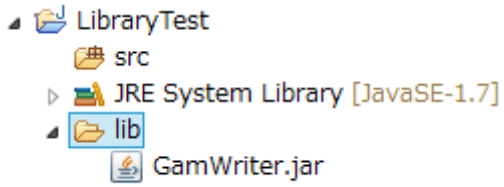
- 実行ファイルが、実験パッケージに 1 つも存在しない場合、2 つ以上存在する場合はエラーになります。
- 実験に用いる外部データは、5.その他ファイルとして配置してください。
- その他ファイルのパスを実行ファイルやパラメータファイル内で指定する場合は、実験フォルダの階層から相対パスで記述してください。

- GAM 用の出力ライブラリ

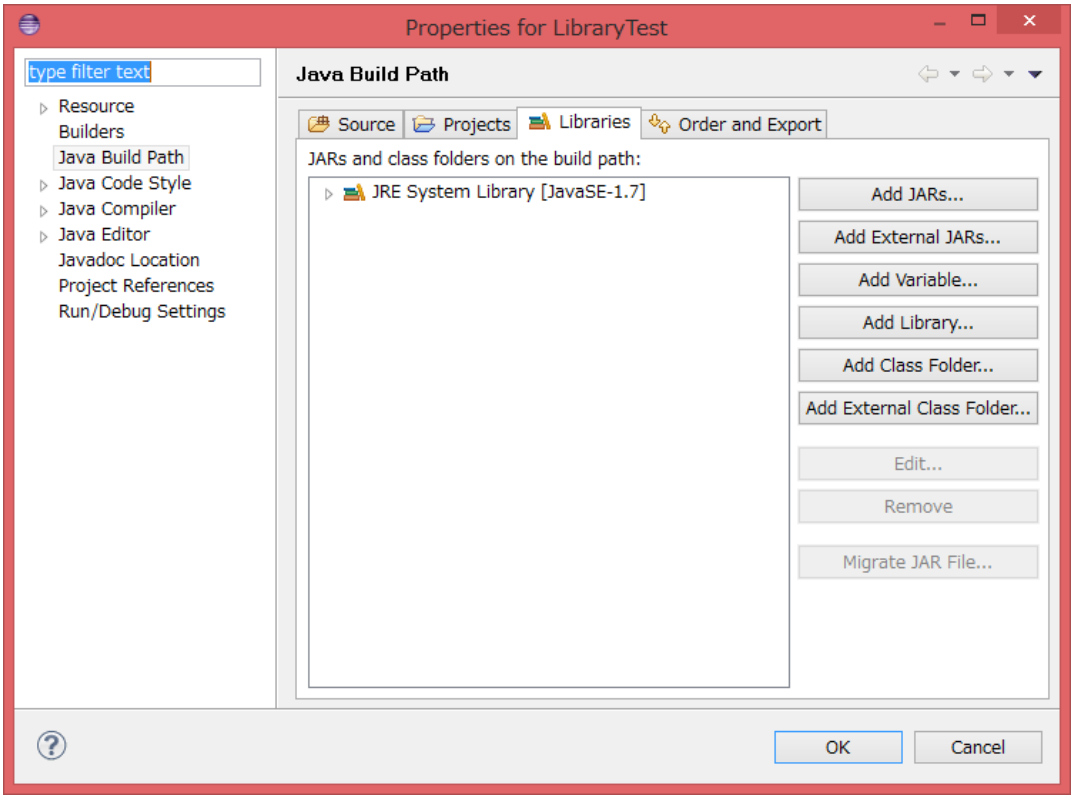
GAM は、進化的計算の結果出力用の Java クラスライブラリとして GAM Writer を提供しています。
実行ファイルの出力を GAM Writer を用いて行うことで、実験の出力結果を GAM 上でグラフ表示することができます。

1. Java プロジェクト内に新しいフォルダを作り、そこに Gam Writer の.jar ファイ

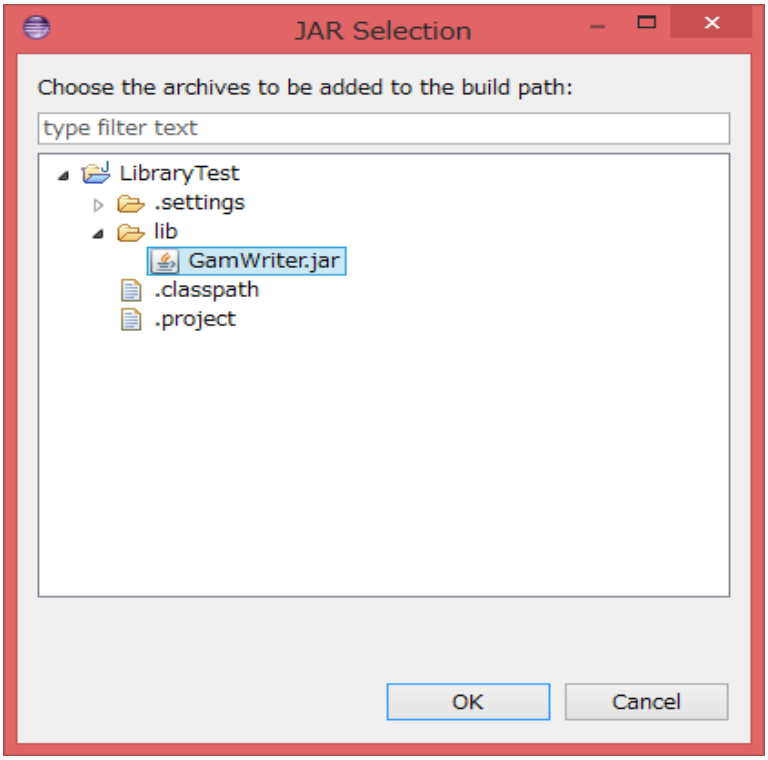
ルをコピーします。



2. Java プロジェクトの"properties"メニューを開き、“Java Build Path” -> “Libraries” -> “add JARs…”をクリックします。



3. コピーした.jar ファイルを選択して、OK をクリックします。これでインポートの操作が完了です。



- GAM Writer を用いてプログラムを記述する
  1. RWriter クラスを import します。

```
import gam_writer.RWriter;
```

  2. RWriter クラスをインスタンス化して、各世代の結果を RWriter の Setter メソッドを用いてインスタンスに渡します。
     その後、RWriter の"write_line()"メソッドを呼び出して、GAM 用の結果ファイルを出力します。

```
RWriter r_writer = new RWriter();
r_writer.set_repetition_no(repetition_no);
r_writer.set_generation_no(generation_no);
r_writer.set_best_fitness(best_fitness);
r_writer.set_avg_fitness(avg_fitness);
r_writer.set_fitness_stdev(fitness_stdev);
r_writer.set_diversity(diversity);
r_writer.write_line();
```

     <記述例>

```
public void RunAndWriteout()
{
    RWriter r_writer = new RWriter();
    for(int rep = 0; rep < max_repetition; rep ++)
    {
        Population pop = new Population();
        r_writer.set_repetition_no(rep);
        for(int gen = 0; gen < max_generation; gen ++ )
        {
            pop.runGeneration();
            pop.updateStatus();

            r_writer.set_generation_no(gen);
            r_writer.set_best_fitness(pop.best_fitness[gen]);
            r_writer.set_avg_fitness(pop.avg_fitness[gen]);
            r_writer.set_fitness_stdev(pop.fitness_stdev[gen]);
            r_writer.set_diversity(pop.diversity[gen]);
            r_writer.write_line();
        }
    }
}
```

使用方法

1　実験を選択する

左のメニューバーの Executions ボタンを押し、実験選択画面を表示します。
実験選択画面のテーブルから実行する実験を選択し、Next ボタンを押します。
実験選択画面では、以下のことを行うことができます。

- 実験を作成する
- 実験を編集する
- 実験を削除する

2　パラメータファイルを選択する

パラメータ選択画面のテーブルから実験に用いるパラメータファイルをチェックし、Next ボタンを押します。
複数チェックの場合は、それぞれのパラメータを引数として 1 つずつ実行ファイルが実行されます。
パラメータ選択画面では、以下のことを行うことができます。

- パラメータファイルをアップロードする
- 複数のパラメータファイルを一括作成する
- パラメータファイルを編集する
- パラメータファイルを削除する

3　通知設定を行う

通知設定画面で通知設定を行い、Execute ボタンを押すと、実験の実行が完了します。通知設定画面では、以下の設定を行うことができます。

- イベントログを記録する
- 通知メールを送信する

<注意>
Execute ボタンを押してから実際に実験がサーバ上で実行されるまで、最大で 5 秒ほど時間がかかる場合があります。
これは、サーバ上で予約されている実験が存在するかのチェックを 5 秒間隔で行っているためです。

4　実験後の実験について
左メニューの Monitoring ボタンを押すと、実行中の実験情報を確認することができます。
- 実行予定の実験一覧
- 実行中のプロセスの状態
- 実行中の実行プロセスの世代と反復回数(要 GAM Writer)

左メニューの Analysis ボタンを押すと、実行後の実験情報を確認することができます。
- 実験終了後の実験一覧
- 実験の標準出力と標準エラー出力
- 適応度と多様性のグラフ(要 GAM Writer)

機能一覧

- 実験を作成する
  1. 左メニューの Execution ボタンを押します。
  2. 実験選択画面の右上にある Create ボタンを押します。
  3. 実験作成画面の右上にある Browse ボタンをクリックして、実験パッケージをアップロードします。
  4. 入力フォームに情報を入力して、OK ボタンを押します。

- 実験を編集する
  1. 左メニューの Execution ボタンを押します。
  2. 実験選択画面の Action の列の info アイコンをクリックします。
  3. 実験情報表示画面の右下にある Edit ボタンを押します。
  4. 入力フォームに変更情報を入力して、OK ボタンを押します。

- 実験を削除する
  1. 左メニューの Execution ボタンを押します。
  2. 実験選択画面の Action の列の del アイコンをクリックします。
  3. 実験削除確認画面で実験の削除に関連して消去される情報を確認して、OK ボタンを押します。

- 実験の実行を予約する
  1. 実験の実行方法を参考に、実験を実行します。

2. その時点で未完了の実験が存在する場合は、それらの実験の終了後に実行が開始します。

- パラメータファイルをアップロードする
  - 事前に実験パッケージ内に配置する方法
    1. パラメータファイルを参考に、パラメータファイルを作成します。
    2. 実験パッケージの parameters ディレクトリ内にパラメータファイルを配置します。
    3. 実験を作成するを参考に、パラメータファイルを含めた実験パッケージを用いて、実験を作成します。

  - パラメータファイルを個別にアップロードする方法
    1. パラメータファイルを参考に、パラメータファイルを作成します。
    2. 左のメニューから Execution ボタンを押します。
    3. 実験選択画面でアップロードする対象の実験を選択し、OK ボタンを押します。
    4. パラメータ選択画面の右上にある参照ボタンをクリックして、パラメータファイルをアップロードします。

- パラメータファイルを編集する
  1. 左のメニューから Execution ボタンを押します。
  2. 実験選択画面で編集するパラメータファイルが含まれる実験を選択し、OK ボタンを押します。
  3. パラメータ選択画面のテーブルの Actions 列にある edit アイコンをクリックします。

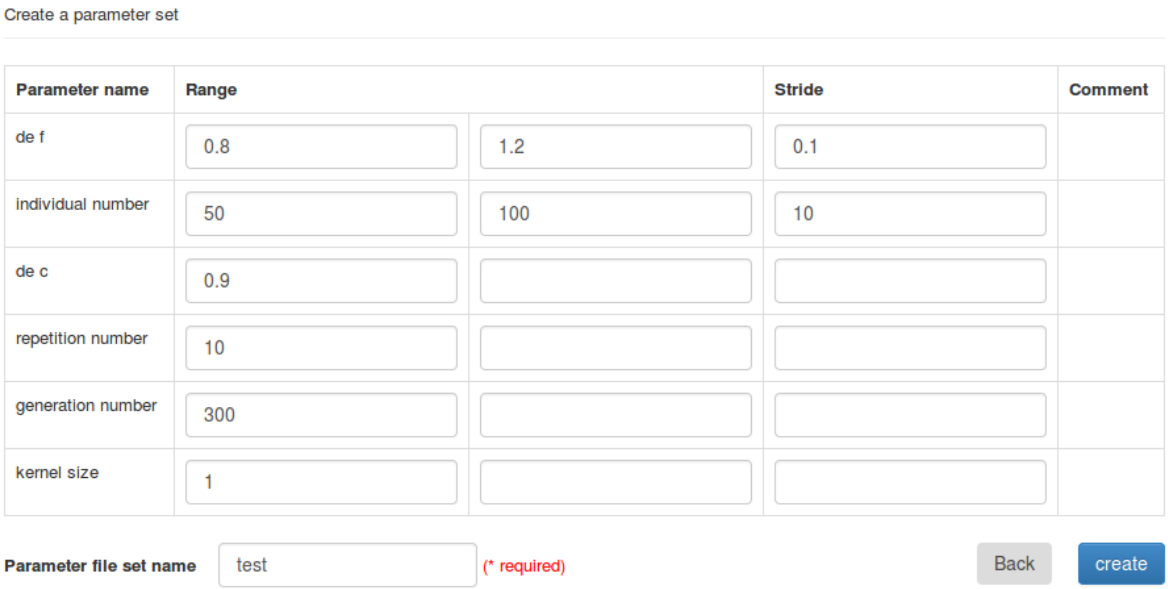
  <注意>
  パラメータファイルの項目の追加・削除を行いたい場合や、コメント文を変更したい場合は、パラメータファイルを再作成してください。

- パラメータファイルを削除する
  1. 左メニューの Execution ボタンを押します。
  2. 実験選択画面で、削除するパラメータが含まれる実験を選択して、OK ボタンを押します。
  3. パラメータ選択画面で、テーブルの Action 列の del アイコンをクリックします。
  4. 削除確認画面でパラメータの削除に関連して削除される情報を確認し、Delete ボタンを押します。

- 複数のパラメータファイルを一括作成する
  1. パラメータファイルを編集する場合と同様に操作して、パラメータファイル編集画面を表示します。
  2. パラメータ編集画面で"create a parameter set"ボタンをクリックします。
  3. 変更を加えたいパラメータの範囲と刻み値を指定します。

4. パラメータセット名を入力し、"create"ボタンをクリックします。
5. 生成されるパラメータファイルの個数を確認し、OK ボタンをクリックします。

<複数パラメーター一括作成の例>



- 実験の一覧を表示する
  - 左メニューの Execution ボタンを押すと、実験パッケージの一覧が表示されます。
  - 左メニューの Monitoring ボタンを押すと、実行中および予約中の実験の一覧が表示されます。
  - 左メニューの Analysis ボタンを押すと、完了した実験の一覧が表示されます。

- 実験の進捗を表示する
  1. 左メニューの Monitoring ボタンを押します。
  2. 進捗を確認したい実験の details アイコンをクリックします。

  <注意>
  - 最大世代数と最大反復数を表示するには、パラメータファイルに"Generation Number"と"Repetition Number"が含まれている必要があります。
  - 現在の世代数と反復数を表示するには、実行プログラムが GAM Writer を用いて記述されている必要があります。

- 実験の出力を表示する
  1. 左メニューの Monitoring または Analysis ボタンを押します。
  2. 実験一覧から details をクリックします。
  3. 実行プロセス一覧の stdout または stderr をクリックすると、実行プログラムの標準出力と標準エラー出力を確認できます。

- 実験結果のグラフを表示する
  1. 左メニューの Analysis ボタンを押します。
  2. 実験一覧から details をクリックします。
  3. 実行プロセス一覧から charts をクリックします。

  以下のグラフを確認することができます。
  - 反復ごとの適応度の最大適と平均値
  - 世代ごとの適応度の最大値と平均値、標準偏差
  - 世代ごとの多様性の値

- 通知メールを送信する
  1. 左のメニューから Execution ボタンを押します。
  2. 実行する実験を選択し、Next ボタンを押します。
  3. 用いるパラメータファイルをチェックし、Next ボタンを押します。
  4. 表示される通知設定画面で、その実験のメール送信設定を行うことができます。

  - 通知設定について
    - Send Email をチェックすることで、通知メールを送信できます
    - メールアドレスは、カンマで繋げて記述することで、複数の宛先を指定することができます。
      例) someone@example.com,another@example.com

  < Email settings のメール送信条件>

イベント名	通知メール送信条件
Experiment started	実験が開始する
Experiment successfully completed	実験が終了し、全プロセスが正常終了している
Experiment canceled	実験/プロセスがキャンセルされ、全プロセスがキャンセル状態である
Experiment finished but error process(es) exist	実験が終了し、エラー状態のプロセスが存在する
Process started	プロセスが開始する
Error: process abnormally terminated	プロセスが終了し、終了ステータスが 0 以外である
Error: process terminated by signal	プロセスが signal により終了する
Error: process unknown status	プロセスの終了を GAM が検知できない

- イベントログを表示する
  1. 左メニューの Event Log をクリックすると、イベントログの一覧が表示されます。

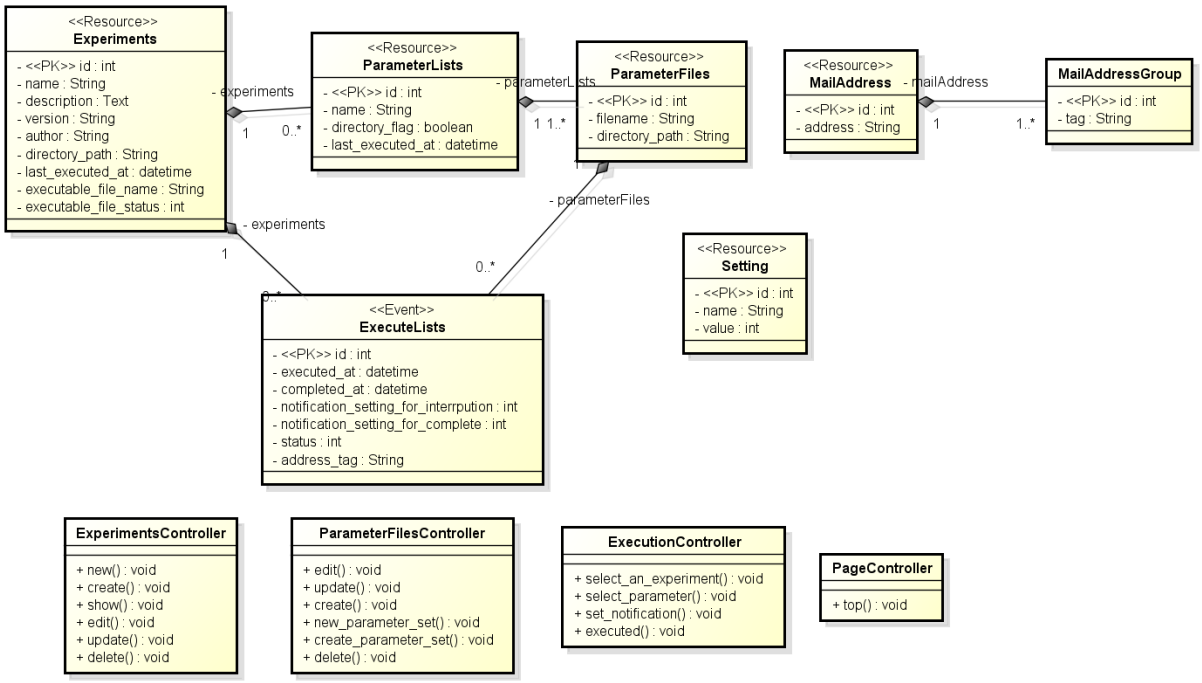
  <イベントログのメッセージと記録条件の一覧>

メッセージ	記録条件

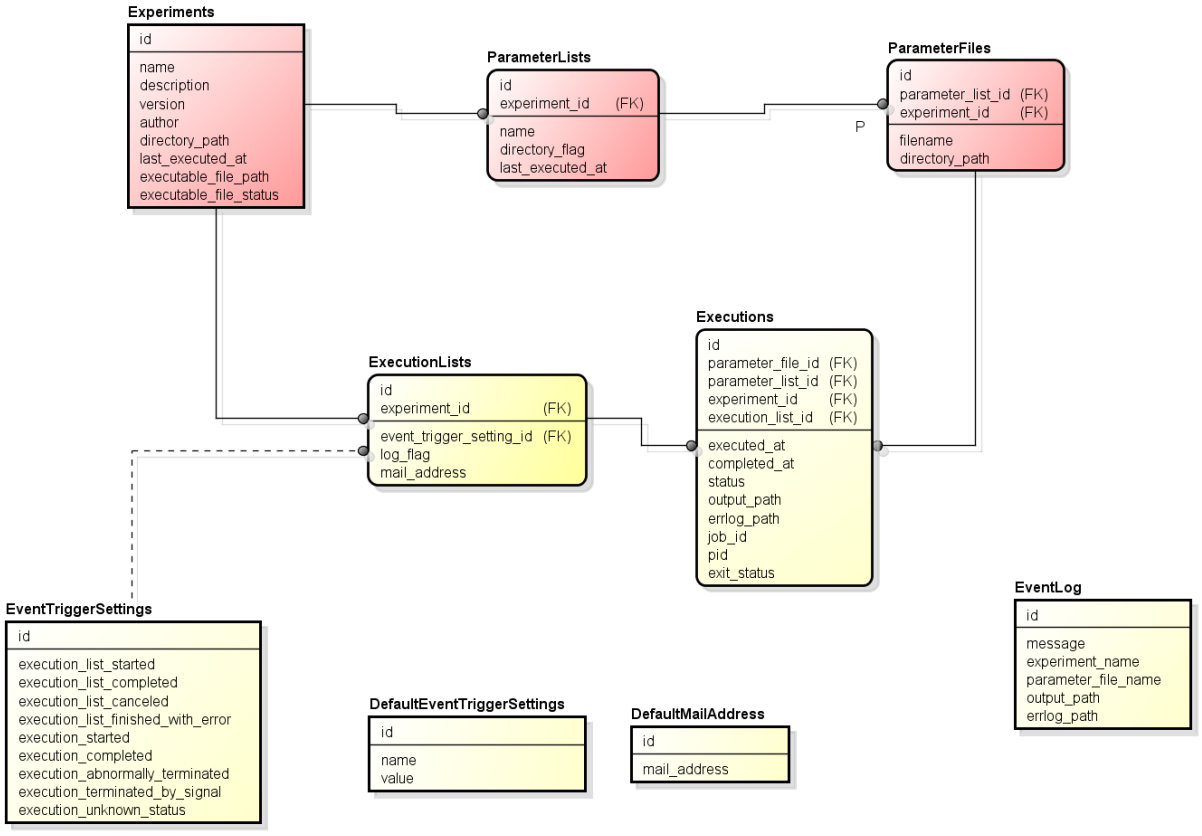
Experiment started	実験が開始する
Experiment successfully completed	実験が終了し、全プロセスが正常終了している
Experiment canceled	実験/プロセスがキャンセルされ、全プロセスがキャンセル状態である
Experiment finished but error process(es) exist	実験が終了し、エラー状態のプロセスが存在する
Process started	プロセスが開始する
Process canceled	プロセスがキャンセルされる
Process expired	プロセスが最大実行時間(120 時間)を超える
Error: process abnormally terminated	プロセスが終了し、終了ステータスが 0 以外である
Error: process terminated by signal	プロセスが signal により終了する
Error: process unknown status	プロセスの終了を GAM が検知できない

付録 C：設計書関連

クラス図

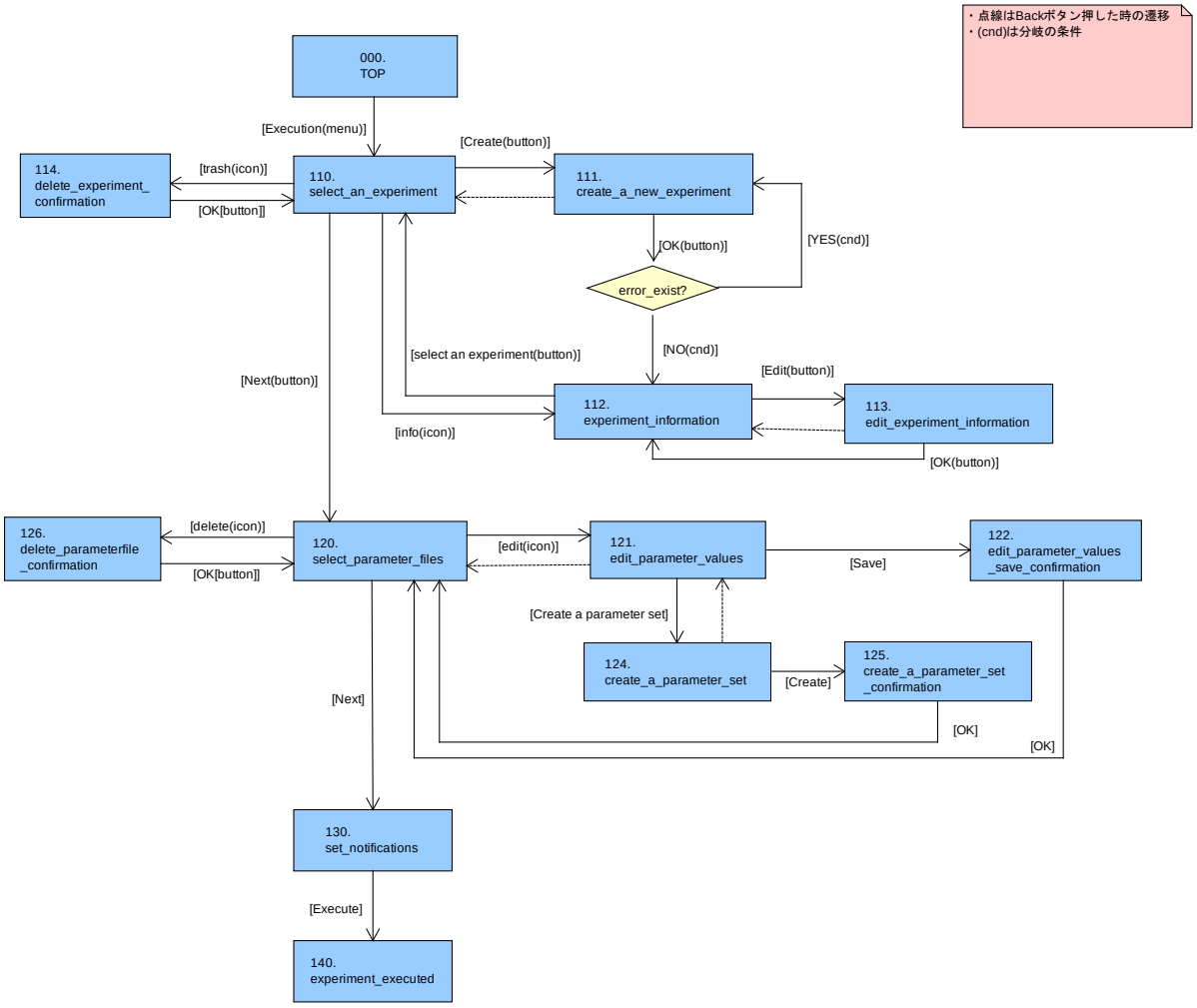


ER 図

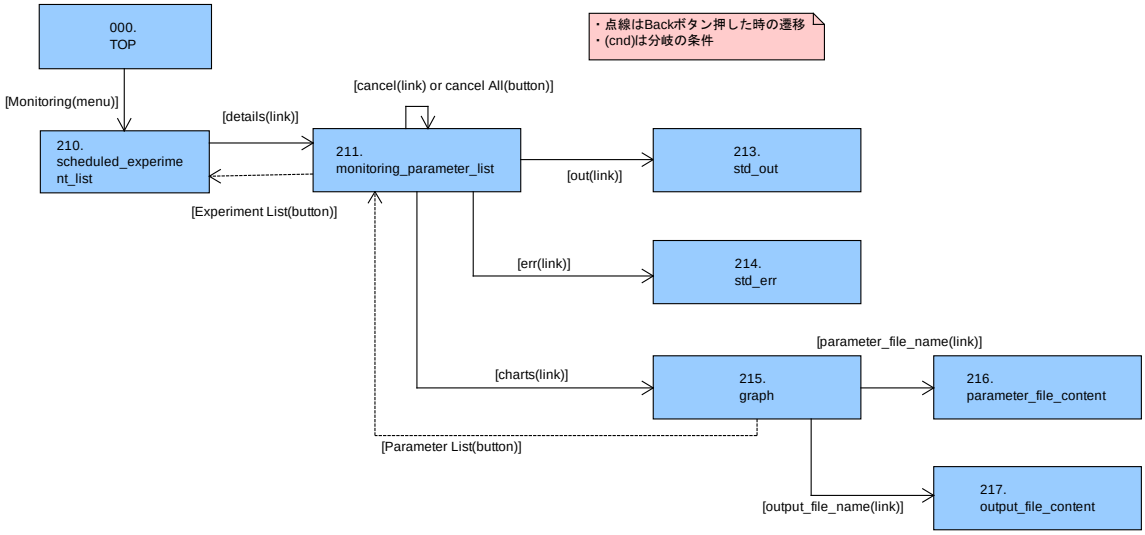


画面遷移図

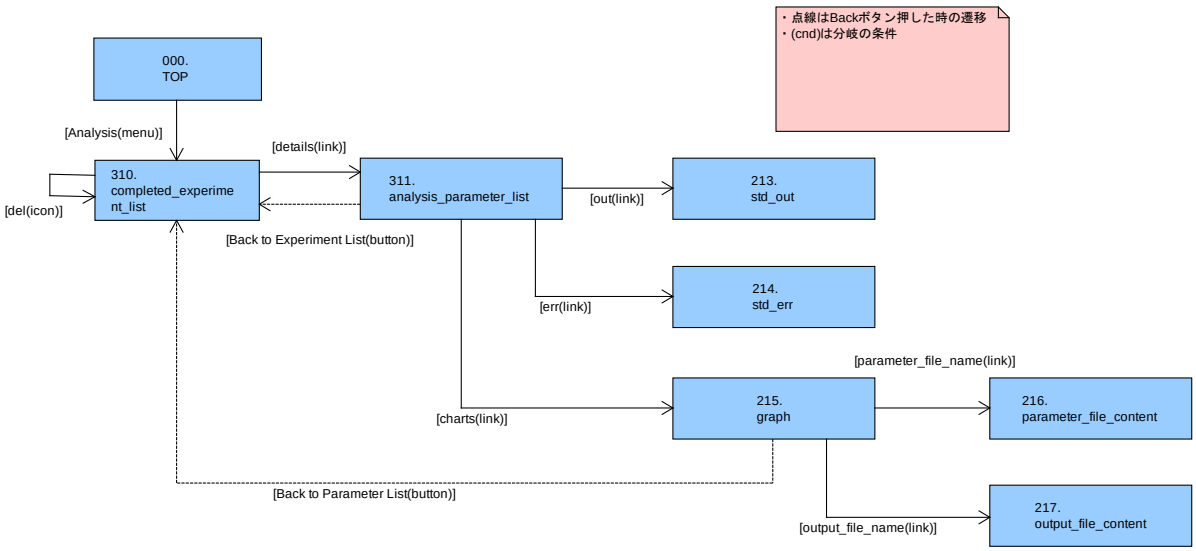
- 実験実行、実験パッケージ管理



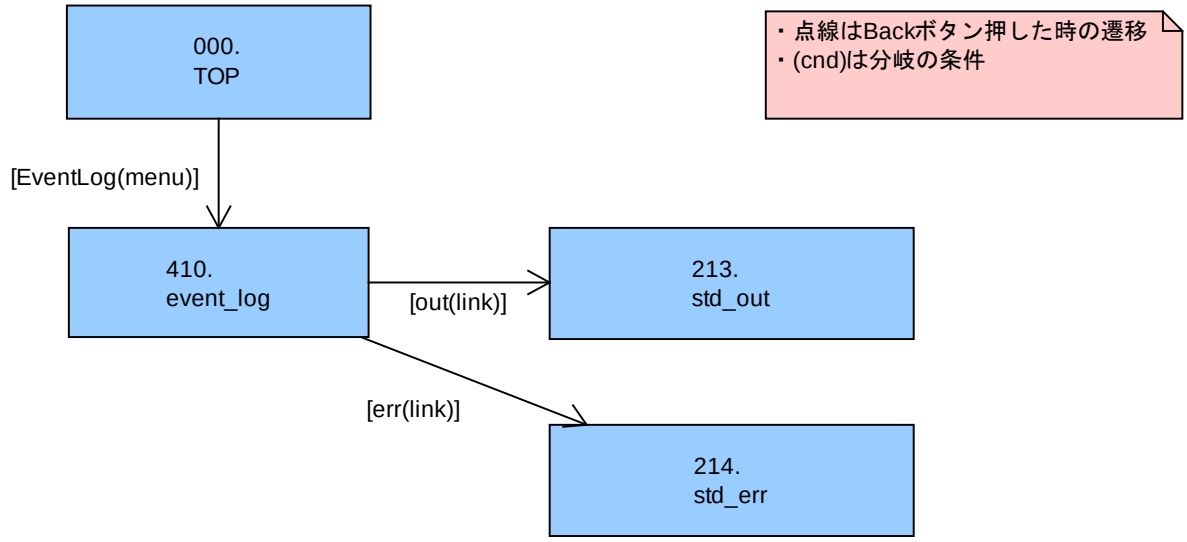
- 実験状態表示



- 実験結果表示



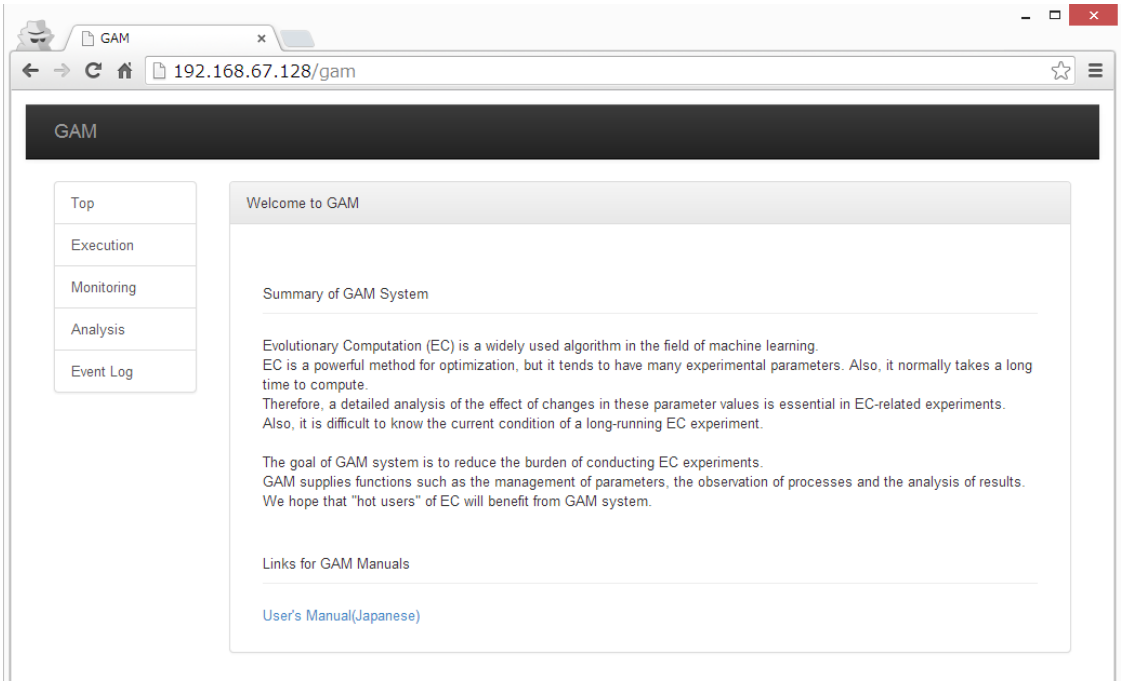
- イベントログ表示



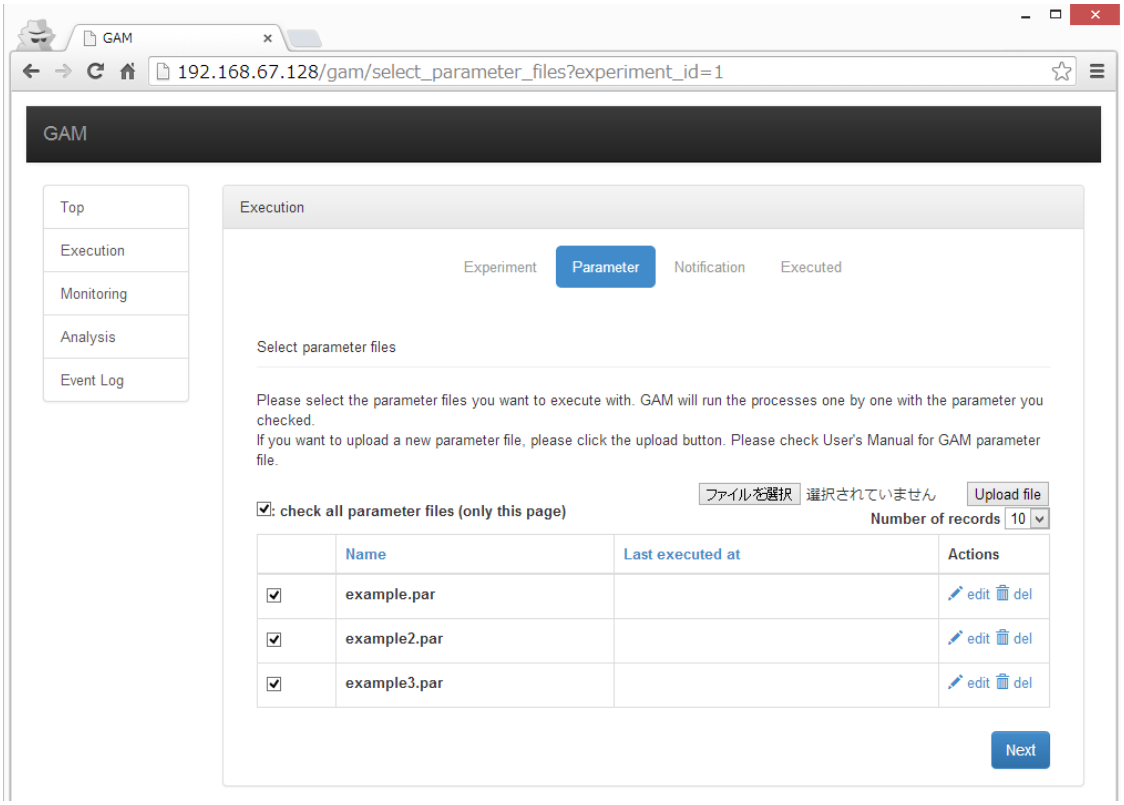
付録 D：システム画面

ID は付録 C の画面遷移図の画面 ID と対応している。

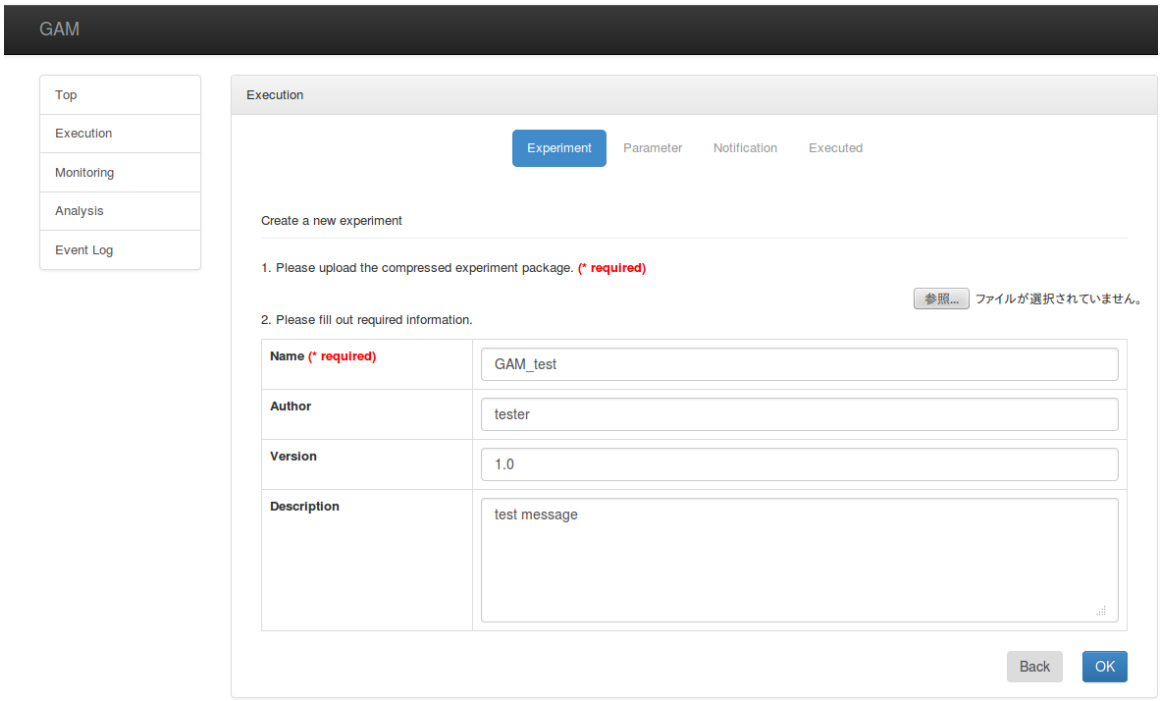
- TOP（ID:000）



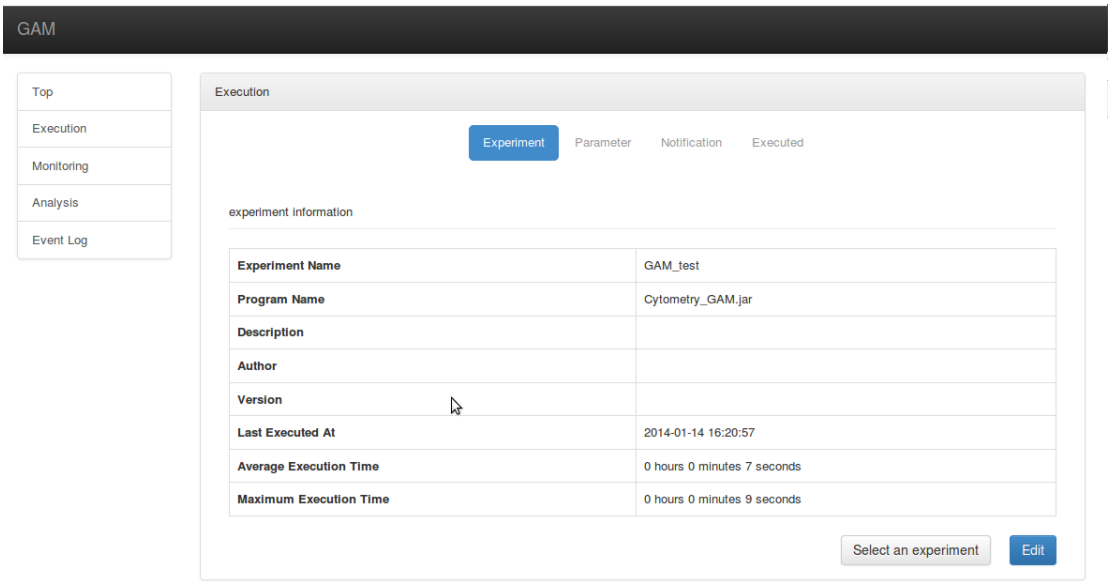
- select_an_experiment (ID:110)



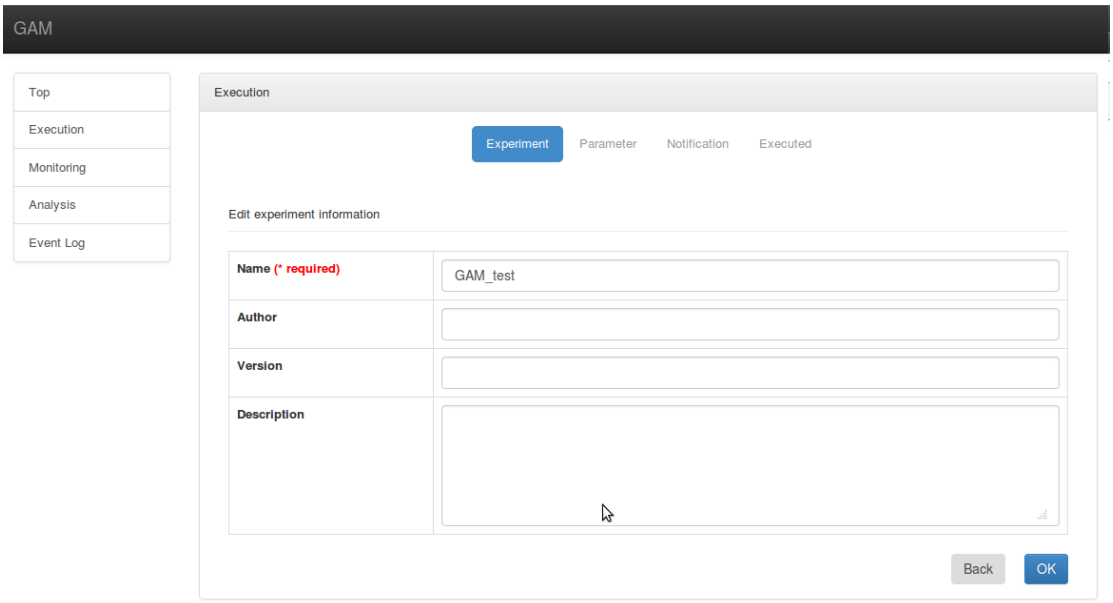
- create_a_new_experiment (ID:111)



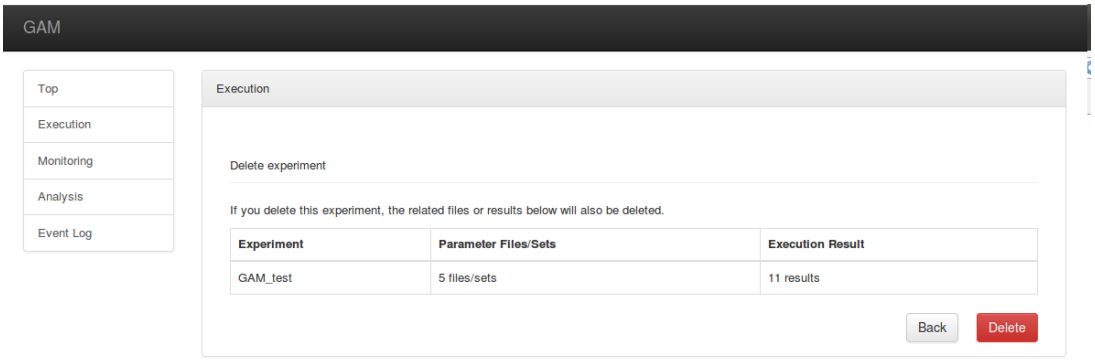
- experiment_information (ID:112)



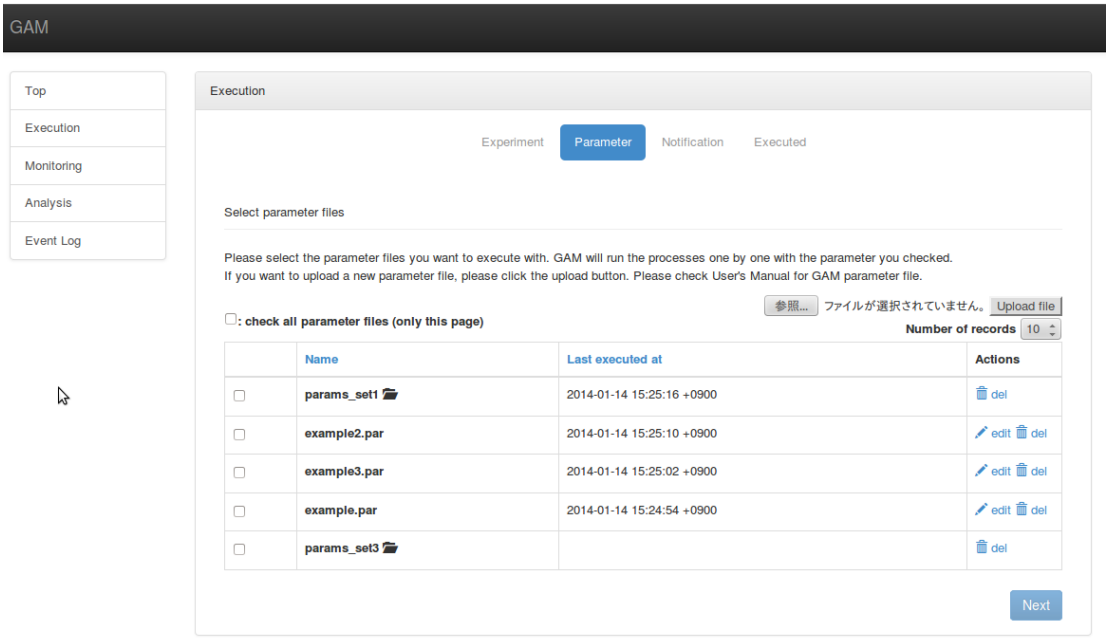
- edit_experiment_information (ID:113)



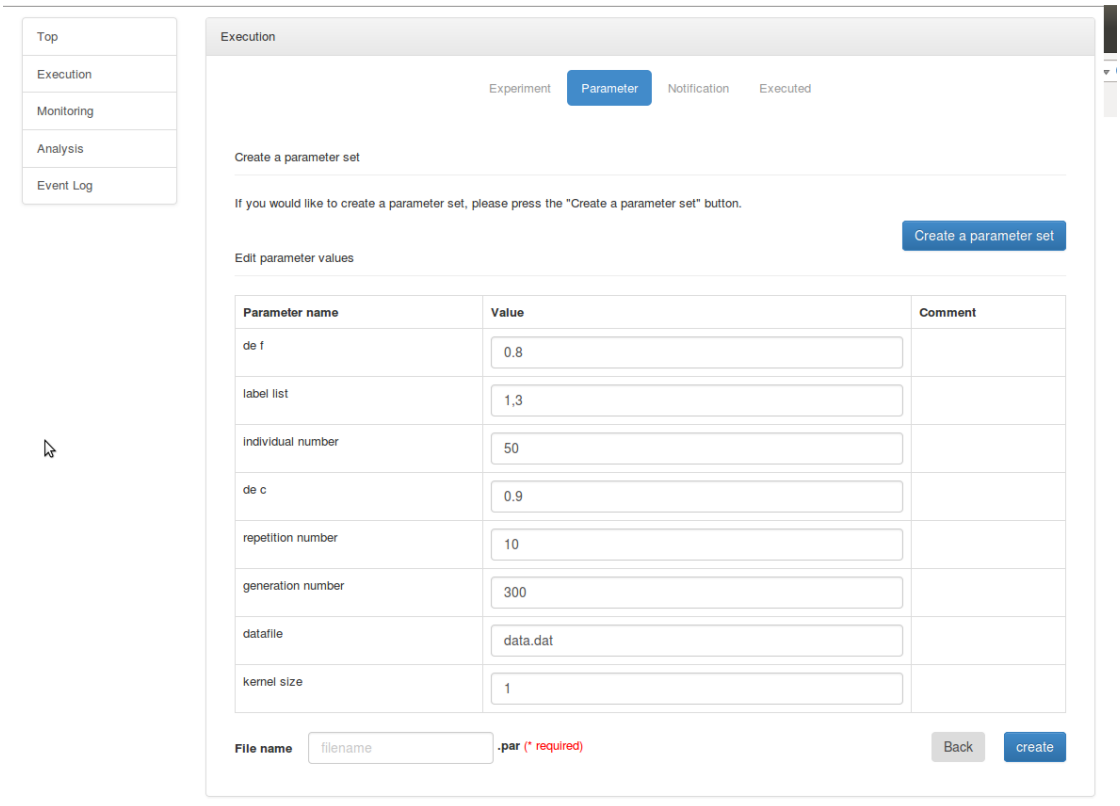
- delete_experiment_confirmation (ID:114)



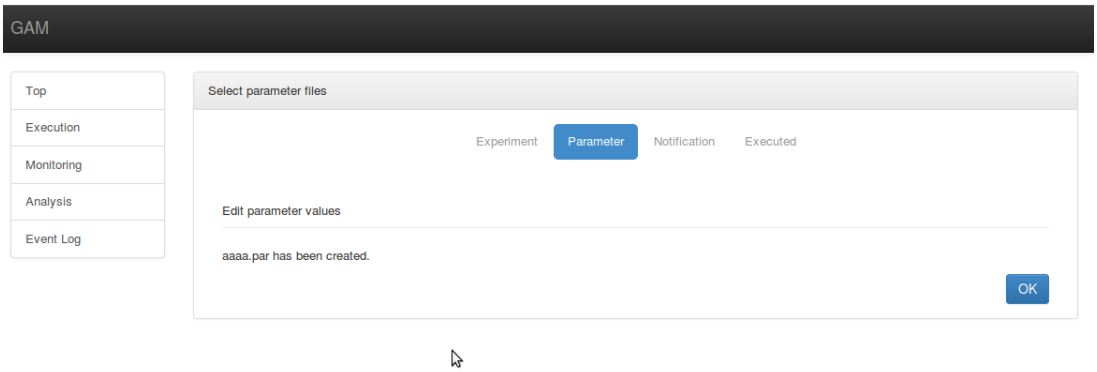
- select_parameter_files (ID:120)



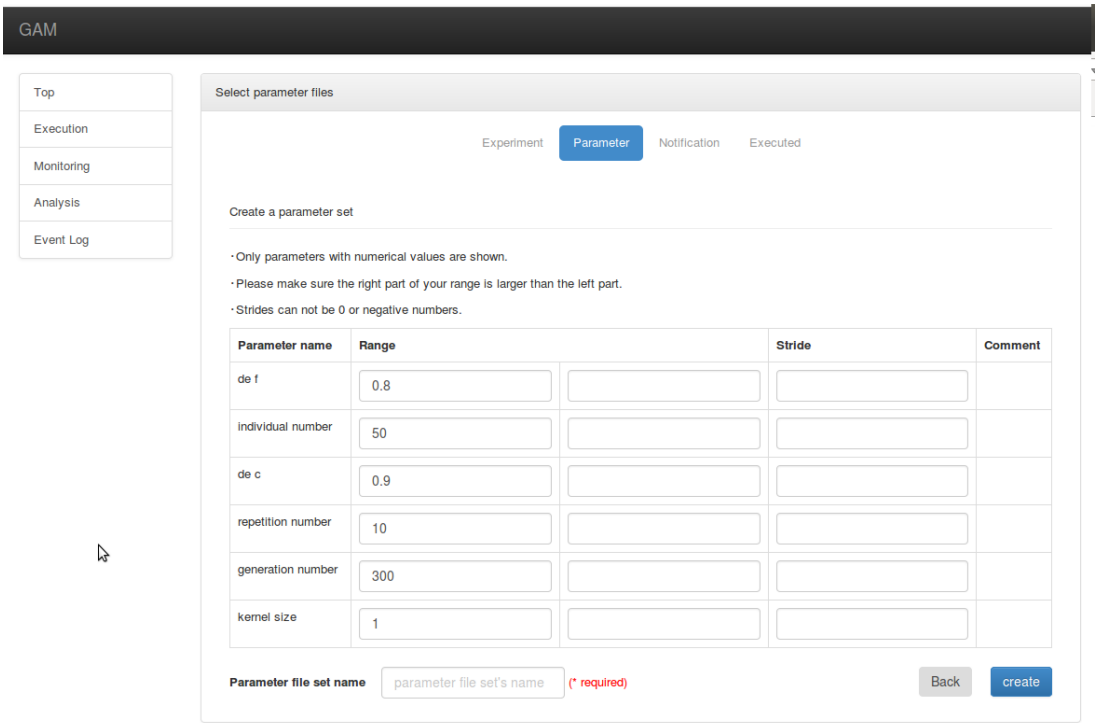
- edit_parameter_values (ID:121)



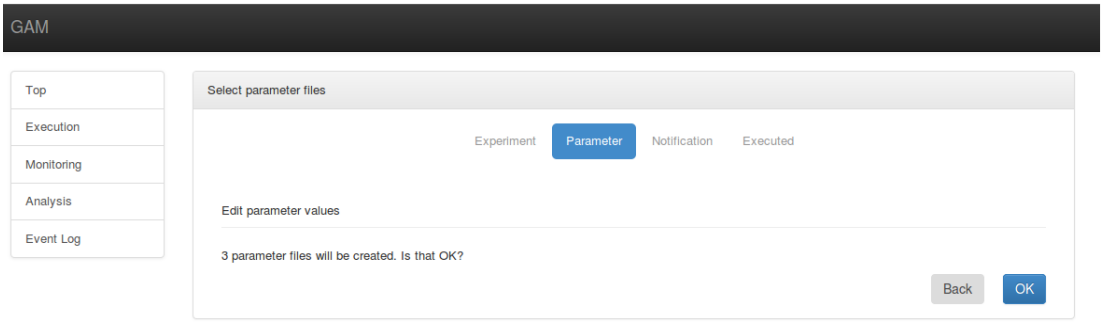
- edit_parameter_values_save_confirmation (ID:122)



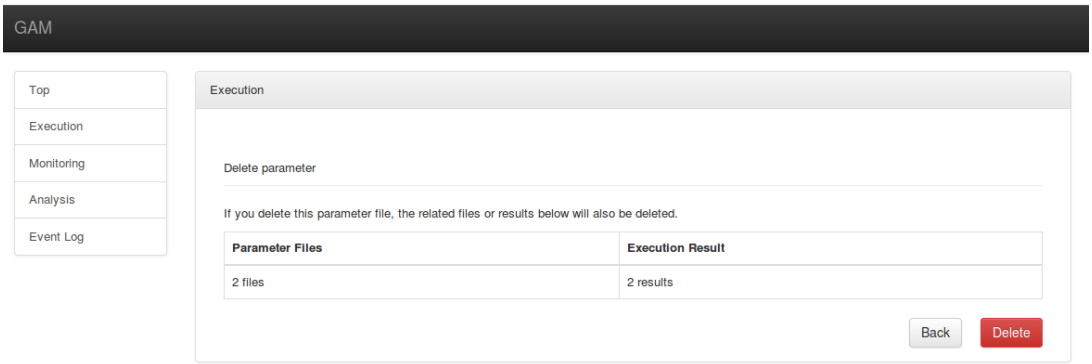
- create_a_parameter_set(ID:124)



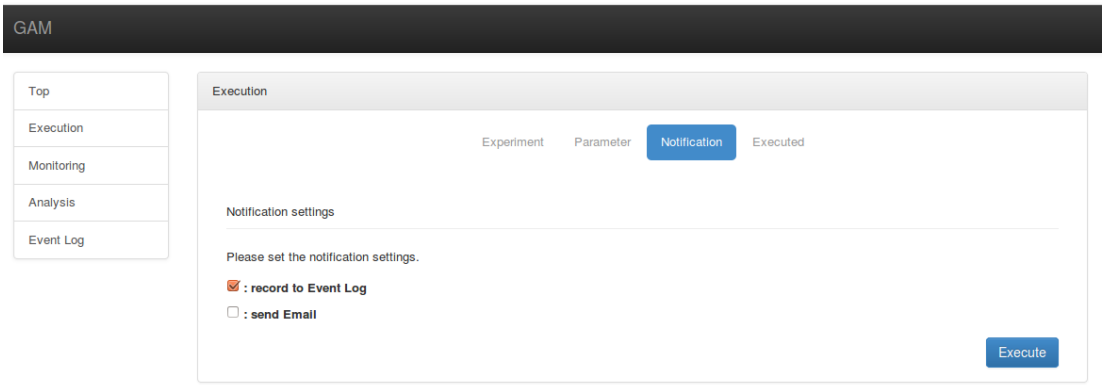
- create_a_parameter_set_confirmation (ID:125)



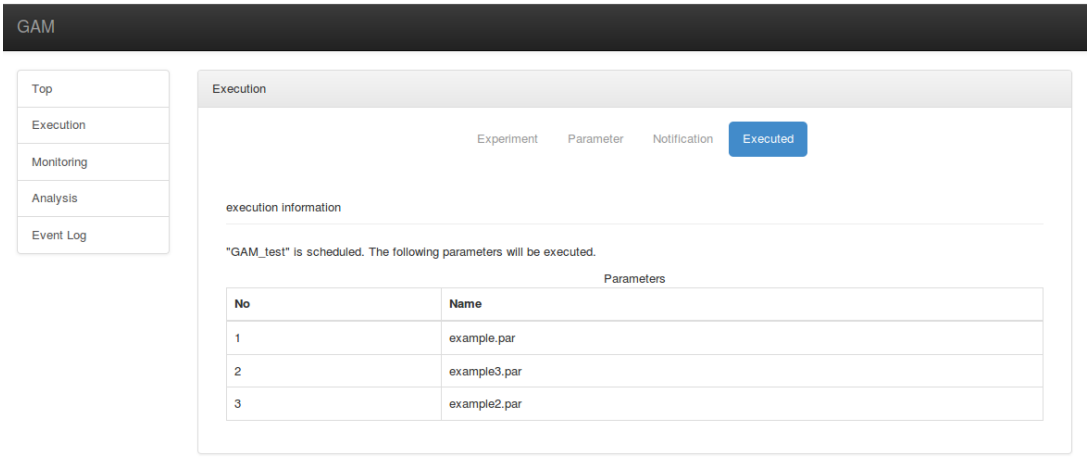
- delete_parameter_ file_confirmation (ID:126)



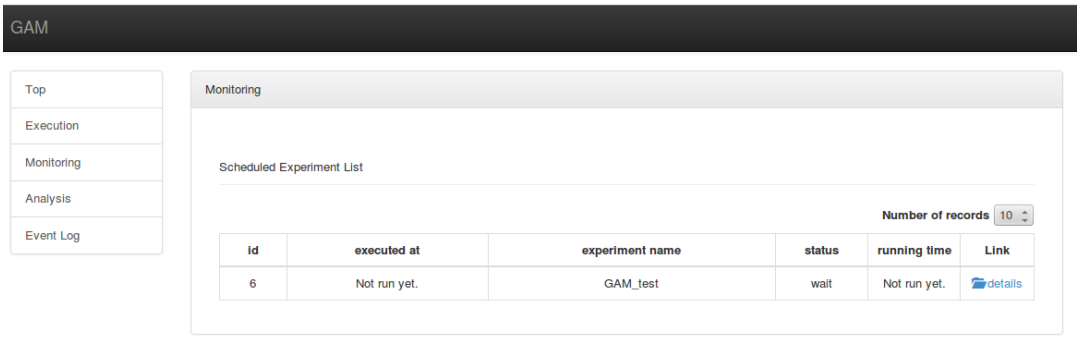
- set_notification (ID:130)



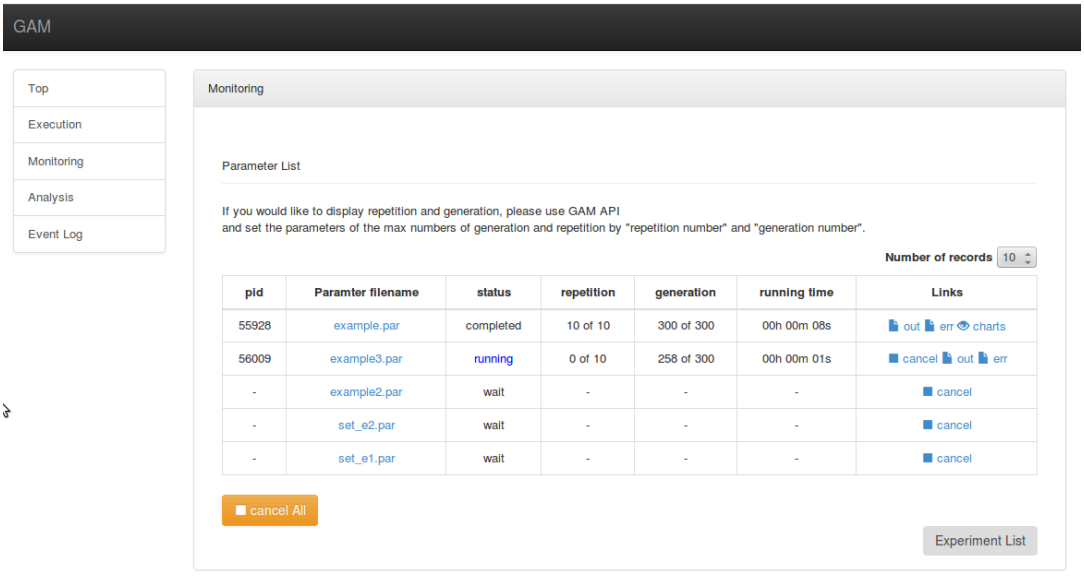
- experiment_executed (ID:140)



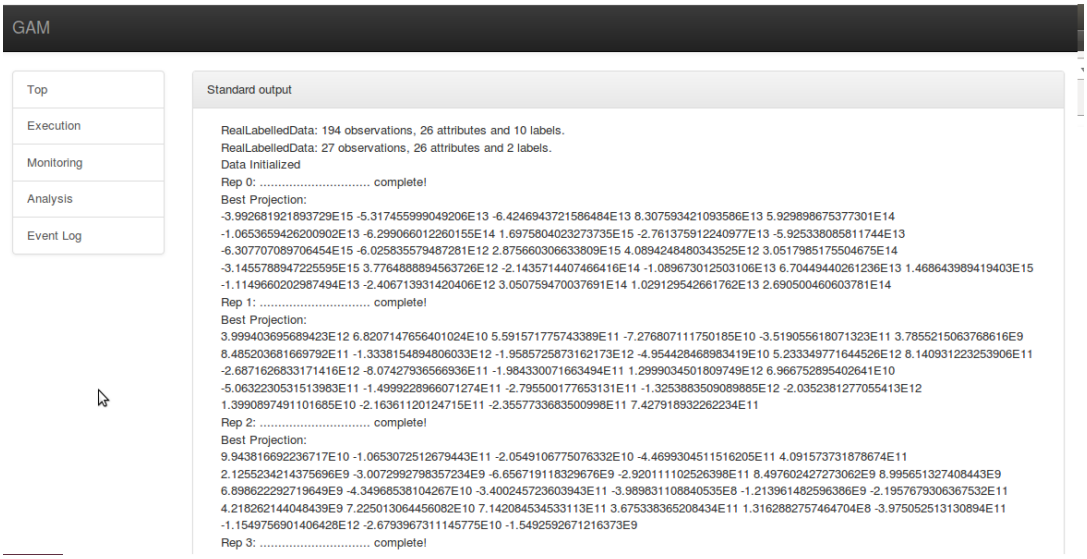
- scheduled_experiment_list (ID:210)



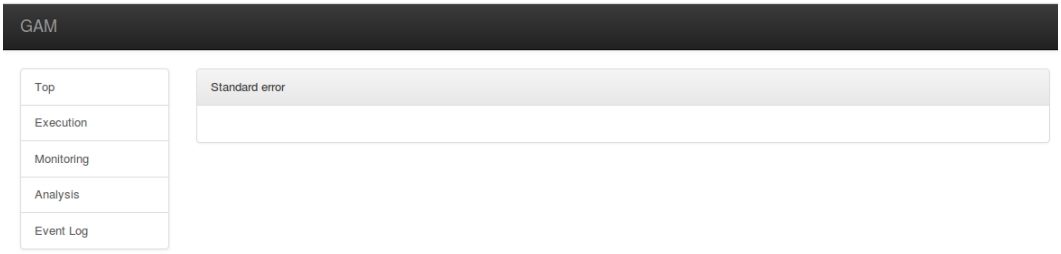
- monitoring_parameter_list (ID:211)



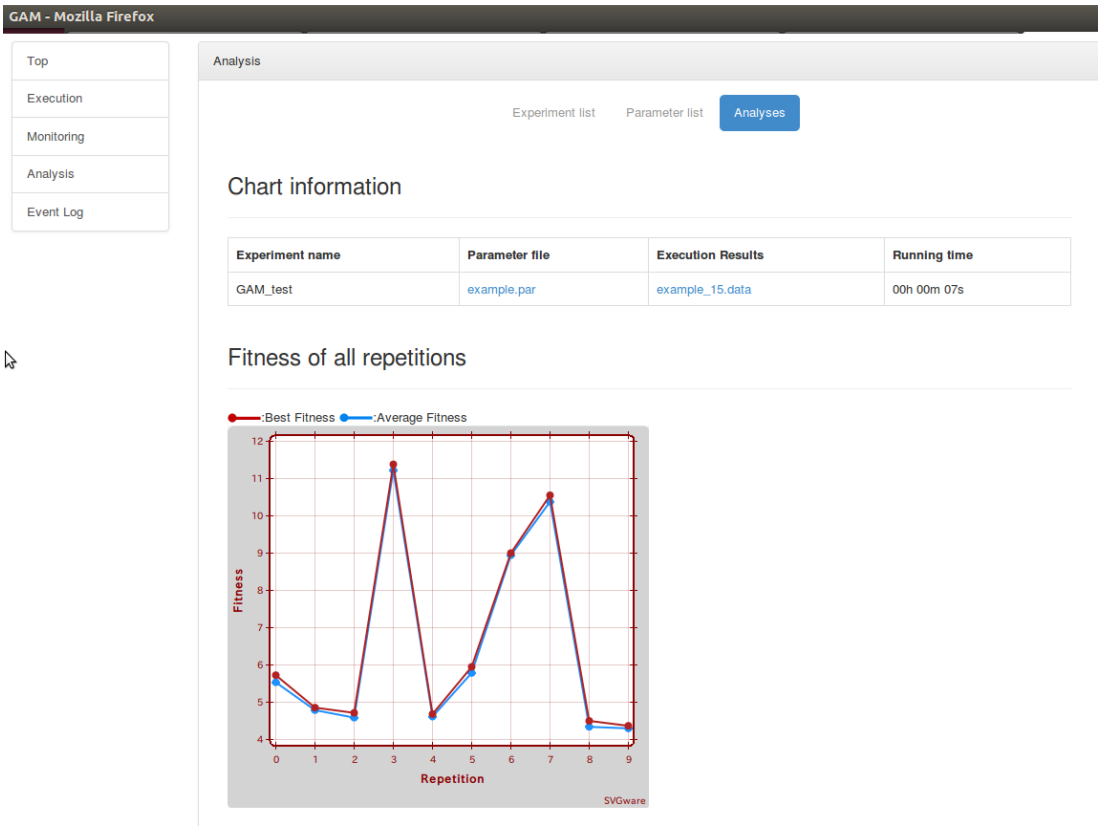
- std_out (ID:213)



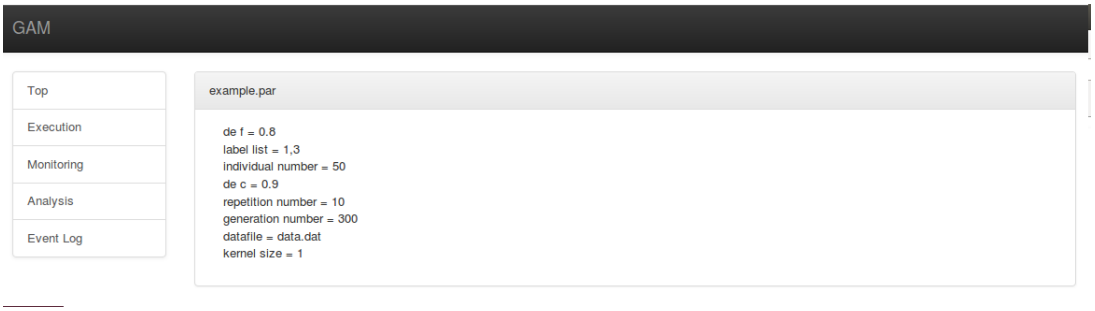
- std_err (ID:214)



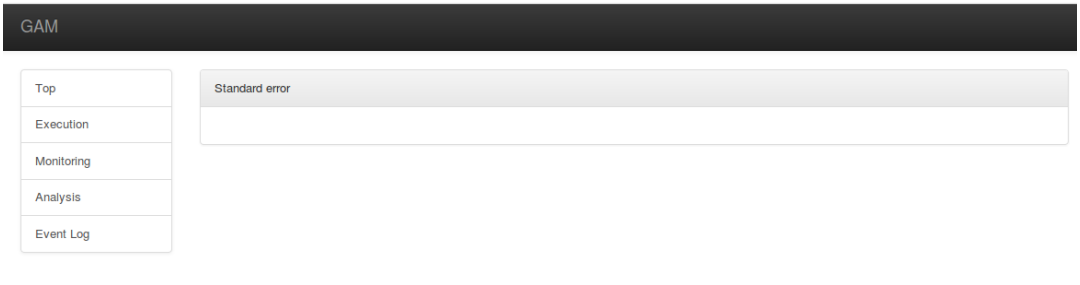
- graph (ID:215)



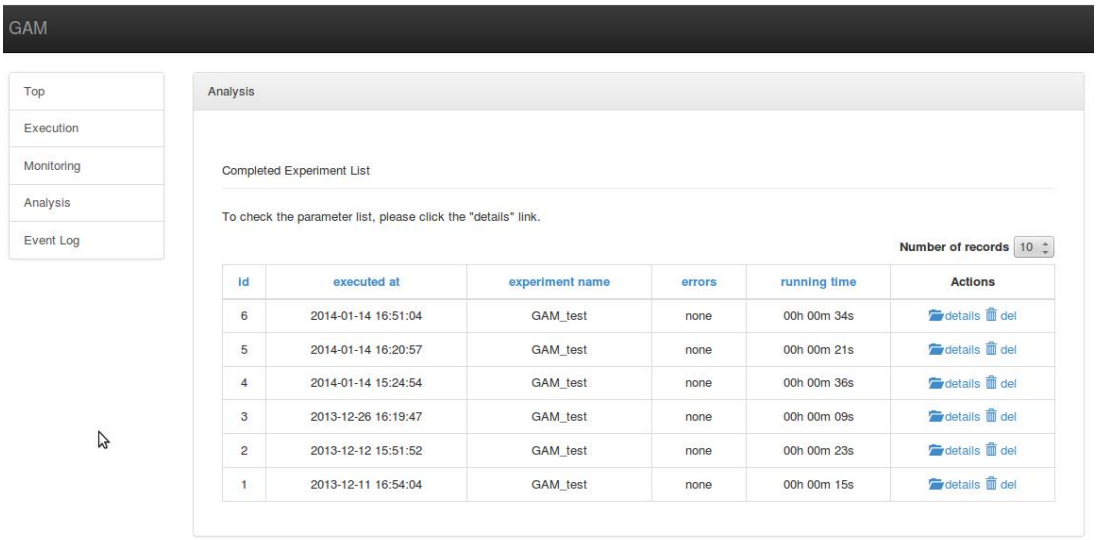
- parameter_file_content (ID:216)



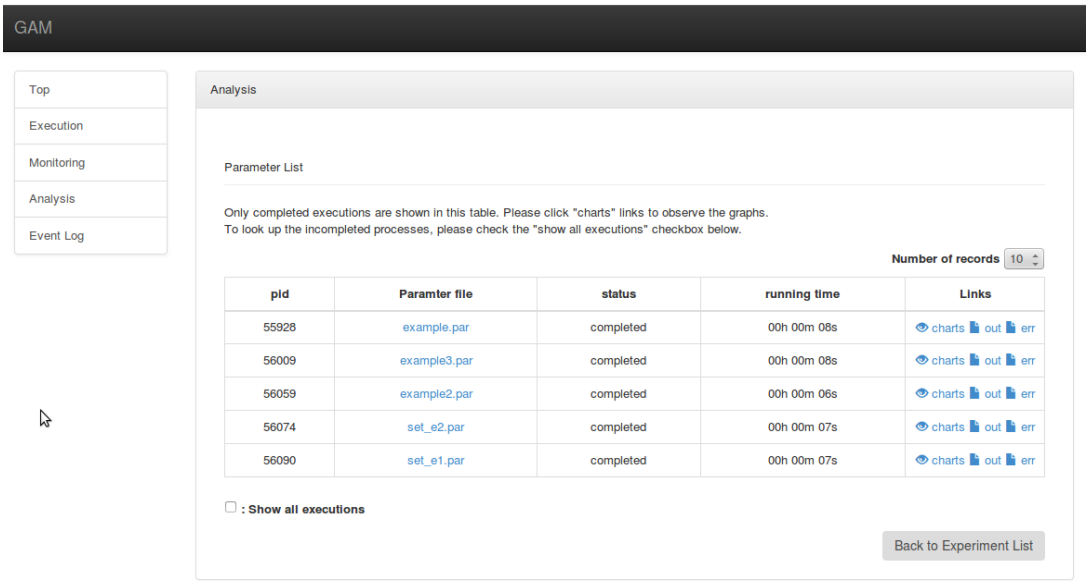
- output_file_content (ID:217)



- completed_experiment_list (ID:310)



- analysis_parameter_list (ID:311)



- event_log (ID:410)