AndroMDA 入門ガイド

第１．３版

Part 4 of 4

２００５／０８／１２

株式会社エクサ 技術部

＜Part 1 of 4＞

＜Part 2 of 4＞

＜Part 3 of 4＞

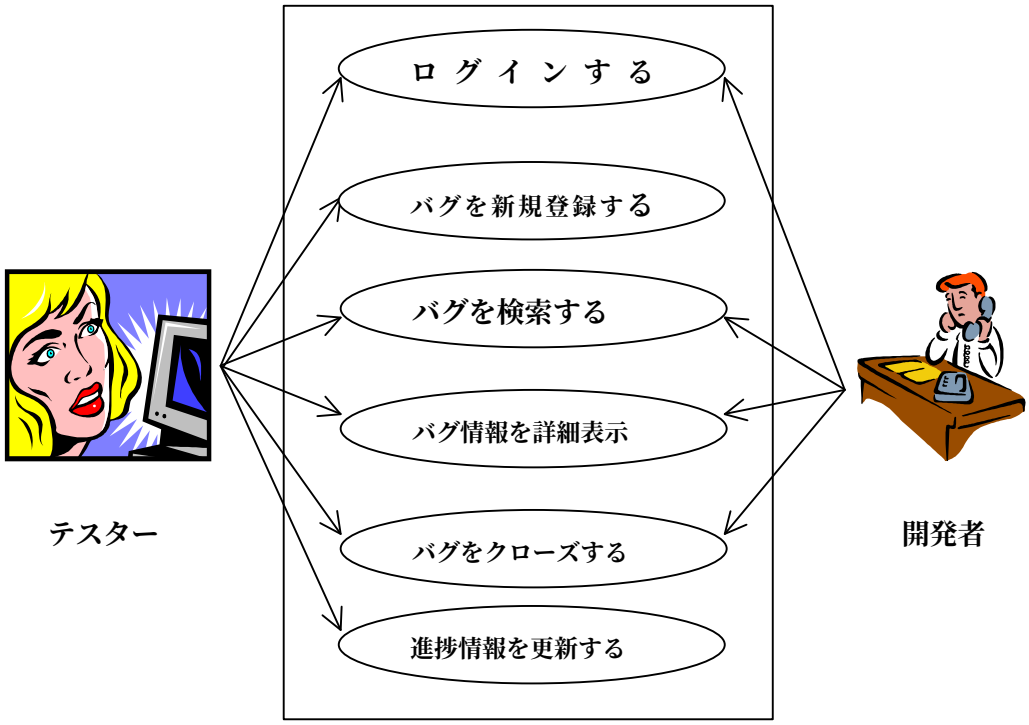
＜Part 4 of 4＞

３．３　事例紹介：バグ追跡掲示板アプリケーション

この章では、ユースケースや画面遷移、DB テーブルイメージなどを提示された状態で AndroMDA を使用してアプリケーションをパイロット的に開発したケースを事例紹介する。
本アプリケーションを新規に作成することが目的ではない。

尚、本アプリケーションでは JSP や CSS などについては AndroMDA で自動生成されたファイルをそのまま使用している。ユースケース、画面遷移、DB テーブルイメージは以下の通りである。

○ユースケース



＜U-1．ログインする＞

アクターは次の情報を選択し、『バグ登録』または『バグ検索』ボタンを選択する

登録済の自分の氏名

システムは、選択されたアクターの情報をセッションへ保持し、選択されたボタンに応じた画面へ遷移する

＜U-2．バグを新規登録する＞

アクター（テスター）は次の情報を画面から入力する

分類

タイトル

報告者氏名

バグ発生日時

バグ内容記述

アクターは『登録実行』ボタンをクリックする

システムはユニークなバグ ID を発番し、ステータス=新規で情報を登録する

アクターは手の空いている開発者に新規登録した、と声かけといたほうがいいかもしれない(システム外)

＜U-3．バグを検索する＞

アクター（テスターまたは開発者）は次の条件の組み合わせを画面から入力する

ステータス（新規、クローズ、調査中、対応中、修正結果確認中）

バグ発生日時 From

バグ発生日時 To

バグ ID

(条件を指定しない場合は全件検索を意味し、条件を複数指定した場合は AND 結合とする)

アクターは『検索実行』ボタンをクリックする

システムは検索を実行し、ヘッダ情報の一覧表及びバグクローズ用プルダウンを画面に表示する(バグクローズ用プルダウンにはステータスがクローズのバグ ID は表示されない)

アクターはバグ情報の詳細を確認するため、バグ検索結果一覧から対象のバグ ID を選択する

システムは選択されたバグ情報をデータベースより取得し、詳細情報画面を表示する(U-4.へ続く)

アクターはバグ情報をクローズするため、バグクローズプルダウンよりクローズするバグ ID を選択し、「バグクローズ」ボタンをクリックする

システムは選択されたバグ情報をデータベースより取得し、バグクローズ確認画面を表示する(U-6.へ続く)

＜U-4．バグを詳細表示＞

前提条件：バグ検索結果一覧から選択した 1 件の詳細表示が完了していること

アクターは表示されているバグ情報を確認し、情報を更新するために『情報更新』ボタンをクリックする

システムはバグ情報のステータスが「クローズ」でないことを確認し、質問や回答を追記するための更新画面を表示する(U-5.に続く)

システムはバグ情報のステータスが「クローズ」ならば更新画面へ遷移せず更新できない旨を表示する

＜U-5．進捗情報を更新する＞

前提条件：バグ情報の更新画面表示が完了していること

アクターは画面項目を編集する

アクターは『更新』ボタンをクリックする

システムは変更内容をデータベースへ反映する

＜U-6．バグをクローズする＞

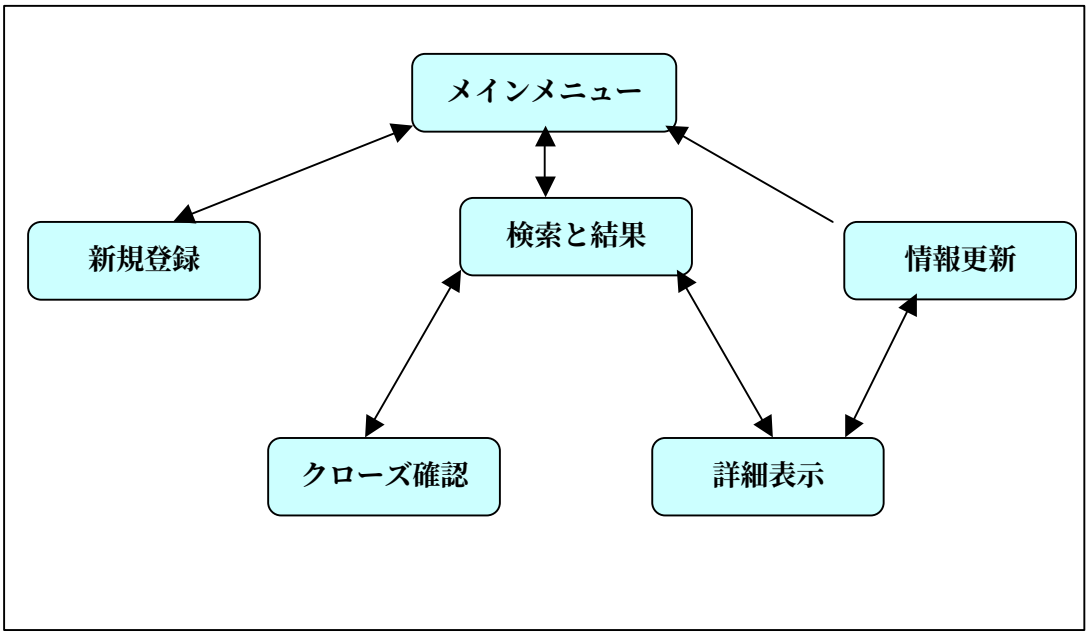
前提条件：バグクローズ確認画面表示が完了していること

アクターは表示された内容を確認し、『クローズ』ボタンをクリックする

システムは表示していたバグのステータスをクローズに更新する。

○画面遷移図

画面遷移は下記の通りである。



○DB テーブルイメージ

ヘッダ情報

	名前	型	サイズ	備考
1	バグ ID	INTEGER		主キー
2	分類コード	INTEGER		NOT NULL、分類名はハードコード
3	バグタイトル	CHAR	80	NOT NULL
4	報告者氏名	CHAR	20	NOT NULL
5	担当者氏名	CHAR	20	NOT NULL
6	バグ発生日時	TIMESTAMP		NOT NULL
7	クローズ日時	TIMESTAMP		NOT NULL
8	ステータスコード	CHAR	1	NOT NULL 、 “N”(新 規),”C”(ク ロ ー ズ),”A”(調 査 中),”M”(対 応 中),”T”(修正結果確認中)
9	バグ内容記述	CHAR	254	NOT NULL
10	更新日時	TIMESTAMP		NOT NULL

ボディー情報

	名前	型	サイズ	備考
1	バグ ID	INTEGER		外部キー
2	バグボディーID	INTEGER		主キー
3	入力者氏名	CHAR	20	NOT NULL
4	入力日時	TIMESTAMP		NOT NULL
5	メッセージ	CHAR	254	NOT NULL（報告者と開発者のやりとり）

個人情報

	名前	型	サイズ	備考
1	個人 ID	INTEGER		主キー
2	名前	CHAR	20	NOT NULL

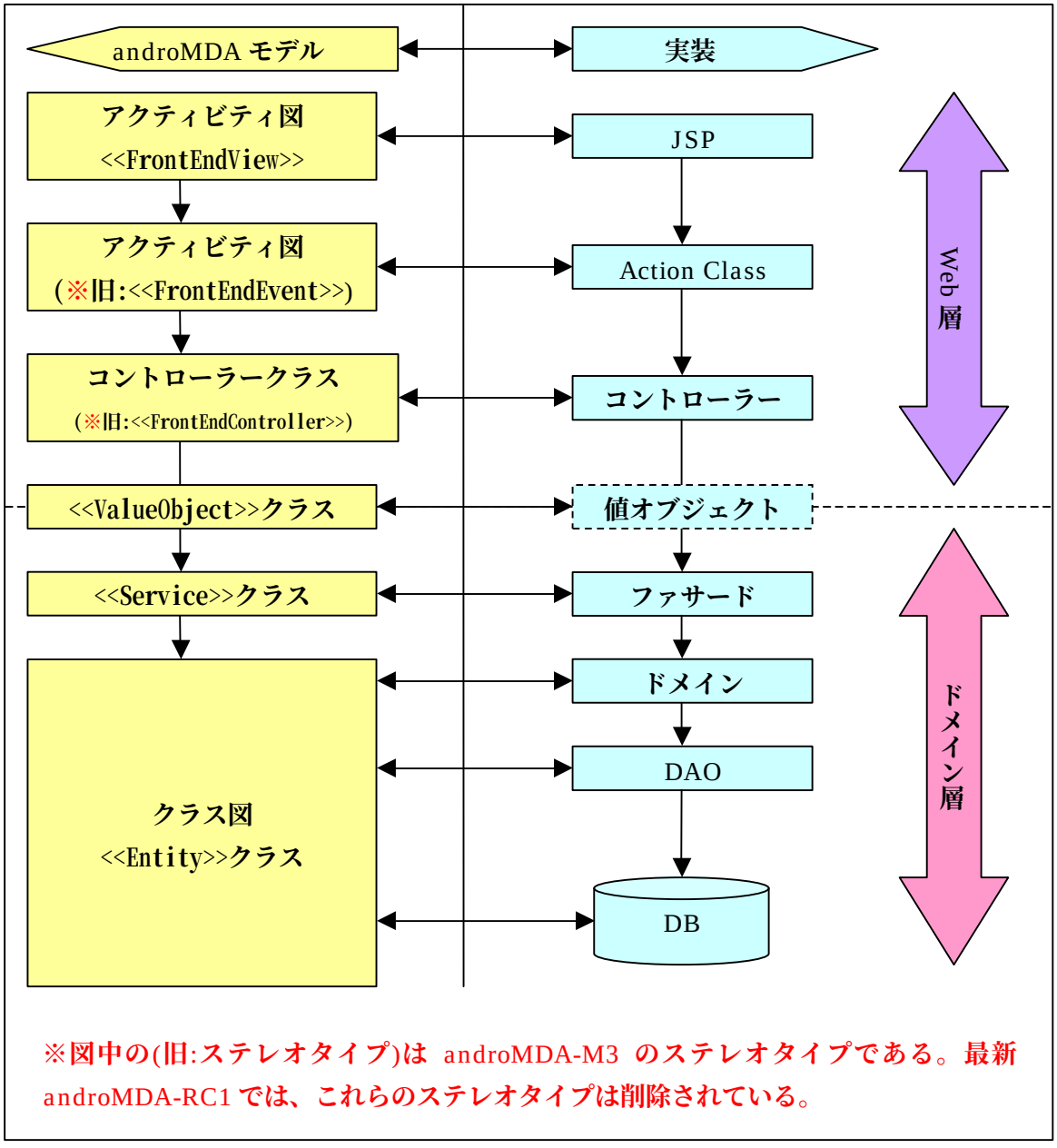
３．３．１　作成手順

本アプリケーションを作成した際の手順は下記のようになる。

1) ユースケースやテーブルイメージを元にドメイン層クラス図作成
2) ユースケースや画面遷移を元に Web 層クラス図、ユースケース図、アクティビティ図を作成
3) 1)と 2)で作成したモデルよりコードを生成
4) 不足部分について実装追加
5) サーバ(JBoss)へデプロイ

３．３．２　アーキテクチャ

本アプリケーションにおけるアーキテクチャを説明する。



※図中の(旧:ステレオタイプ)は androMDA-M3 のステレオタイプである。最新 androMDA-RC1 では、これらのステレオタイプは削除されている。

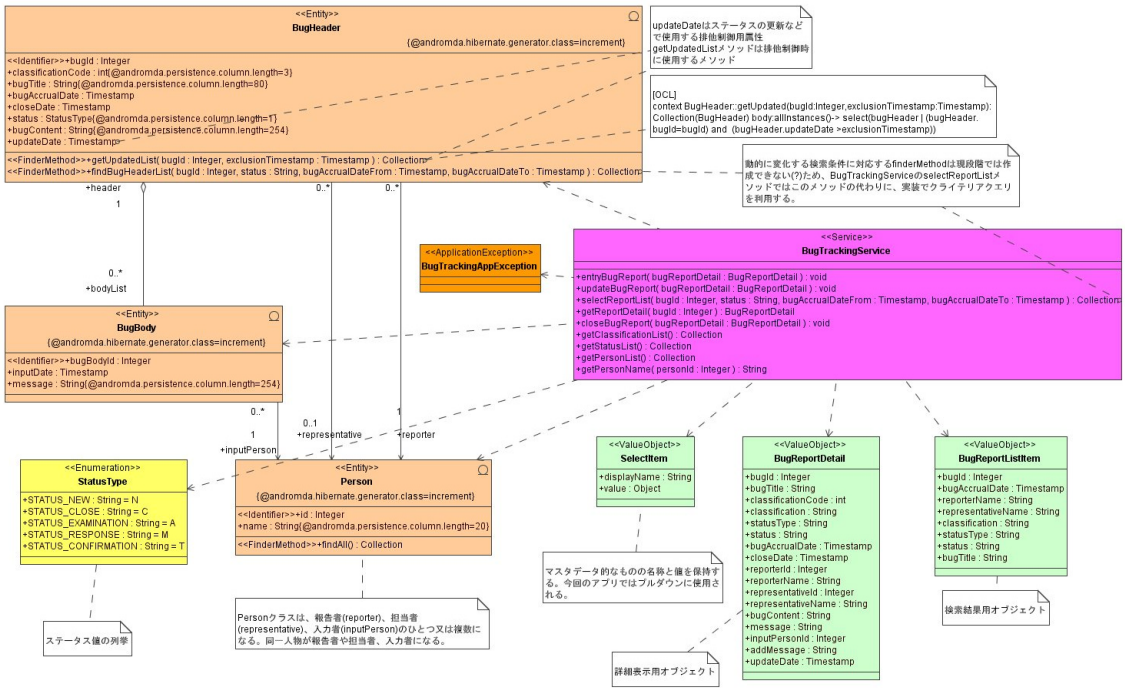
下記の表は、各実装の役割である。

名前	役割	補足説明
JSP	画面表示(レイアウト等)	
Action Class	コントローラーに対するロジックの呼び出しや画面の遷移を制御する	
コントローラー	画面表示データの設定やドメイン層のサービスメソッドの呼び出し	form からデータ取り出し
値オブジェクト	Web 層とドメイン層の間でやり取りする値を格納。構造的には Web 層に合わせた形。	JavaBeans 仕様 (シリアライズ可能)
ファサード	サービスメソッドを用意し、DAO などを利用して DB よりデータを取得し、要求された形に加工して戻す。	EJB を用いる場合は Stateless Session Bean (※hibernate カートリッジ：各サービスのメソッドのはじめで session を取得し、終了時に閉じられる。明示的なトランザクションの取得は行われていない。 例外時は、ファサードクラスが Throwable 型で catch し、EJBException を throw する。 ただし、独自の Exception クラスをモデル中で参照している場合は、その Exception を catch し、SessionContext クラスの setRollbackOnly メソッドを呼び出し、catch した Exception を再び throw するロジックが追加される。)
ドメイン	永続化されるドメインクラス DB のテーブルにマップ	POJO or EJB(EntityBean)
DAO	DB へのアクセスを制御する	EJB EJB+Hibernate、 Spring+Hibernate のどれか(カートリッジの選択による)
DB	データを永続的に保持する	HSQLDB 等

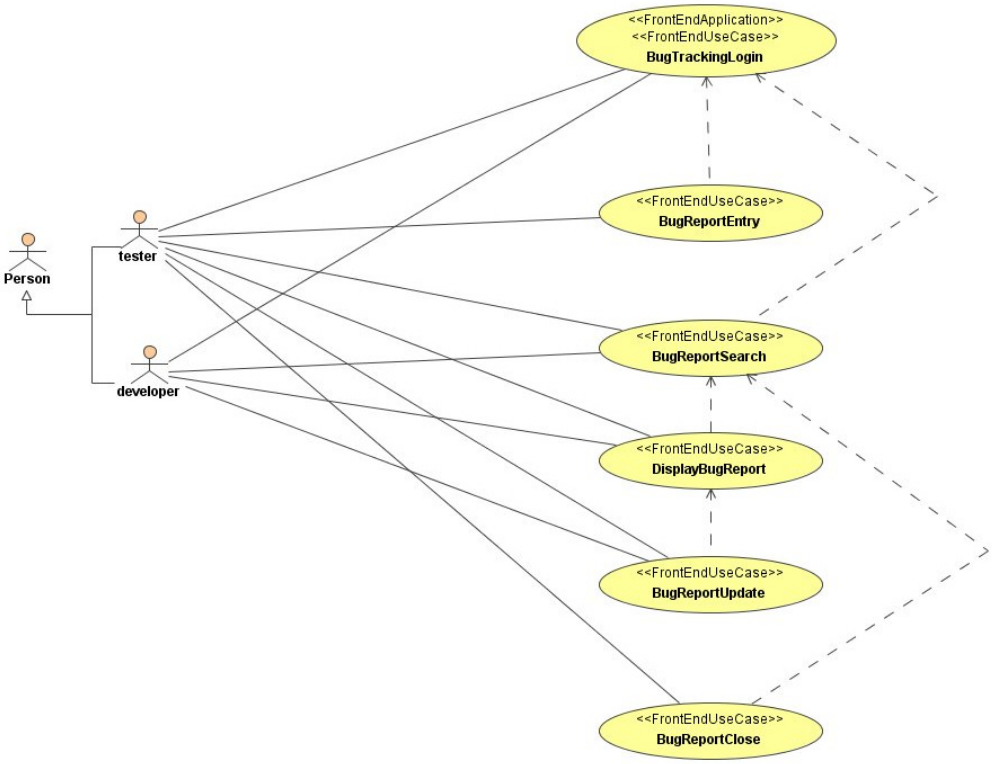
３．３．３　モデル図

本アプリケーション作成時に作成したモデル図を紹介する。

○　ドメインモデル(クラス図)



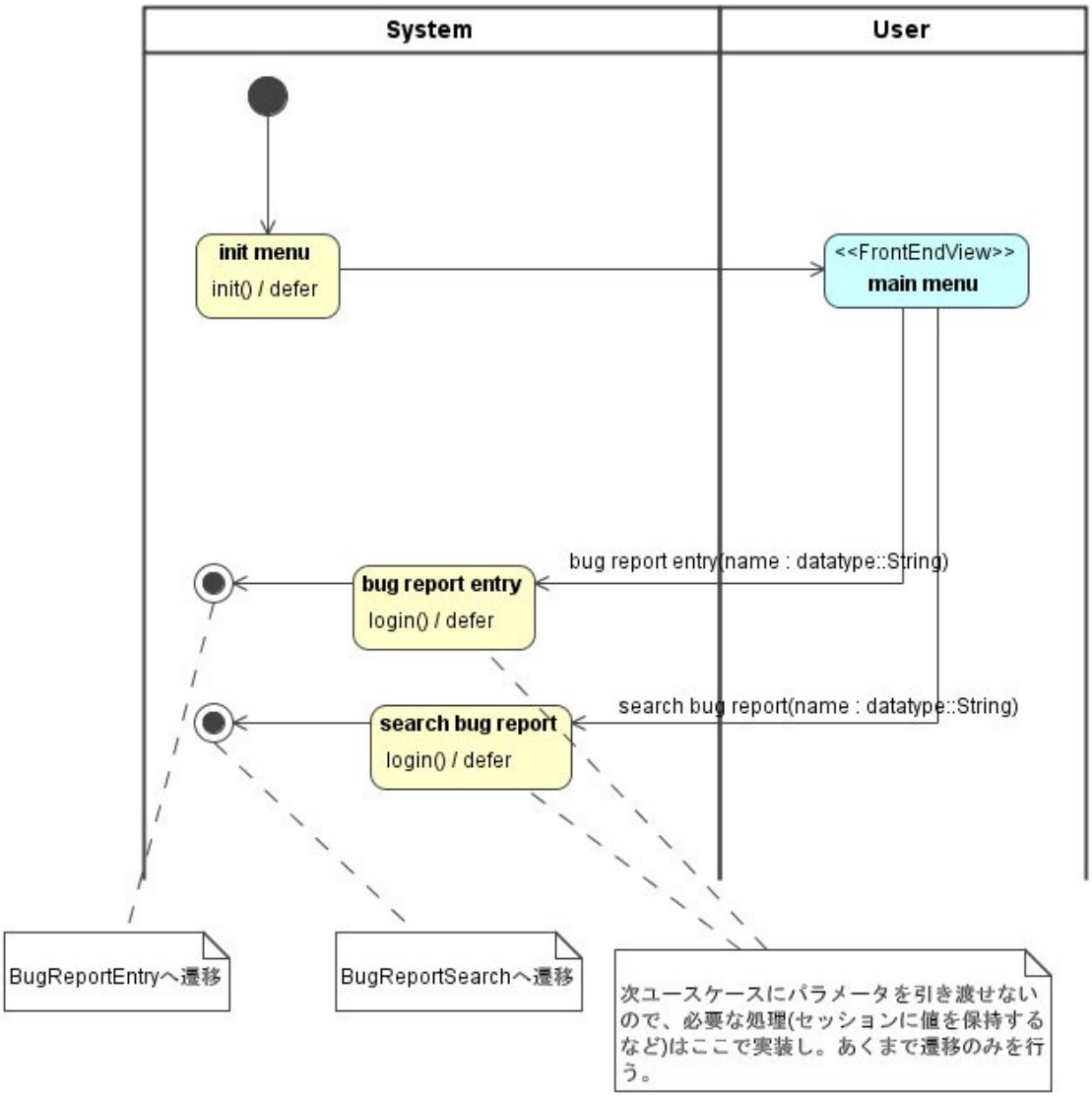
○　ユースケース図



各ユースケースごとにアクティビティ図(画面遷移図)を作成している。

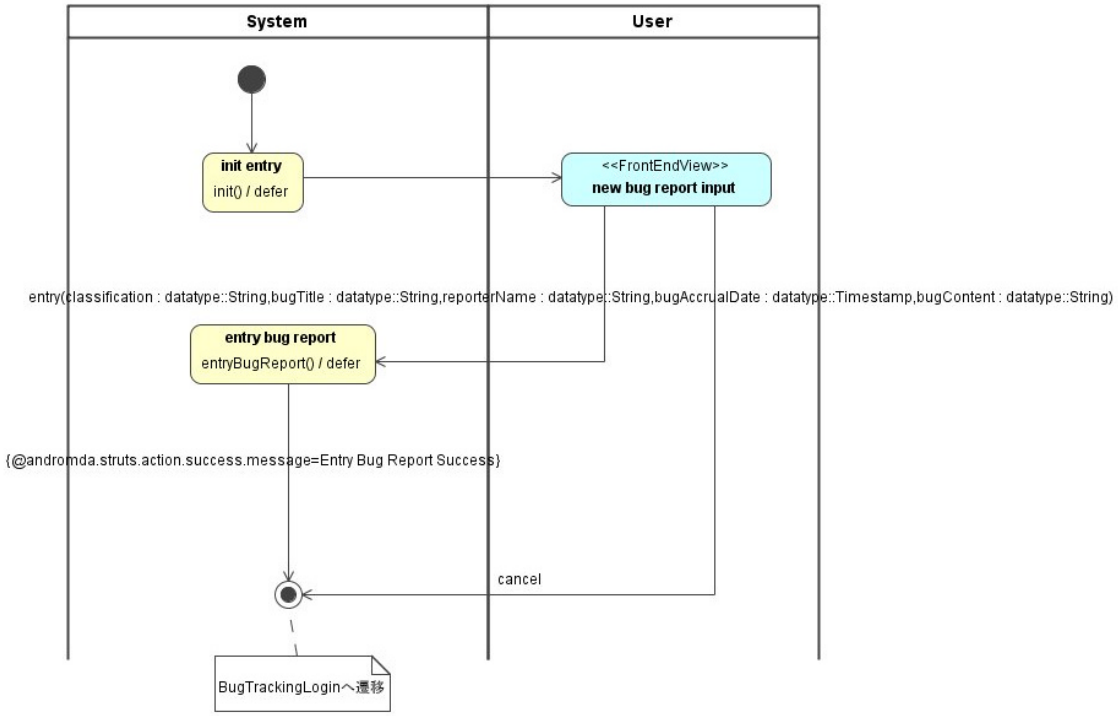
○ アクティビティ図(画面遷移図)

a) ログインする

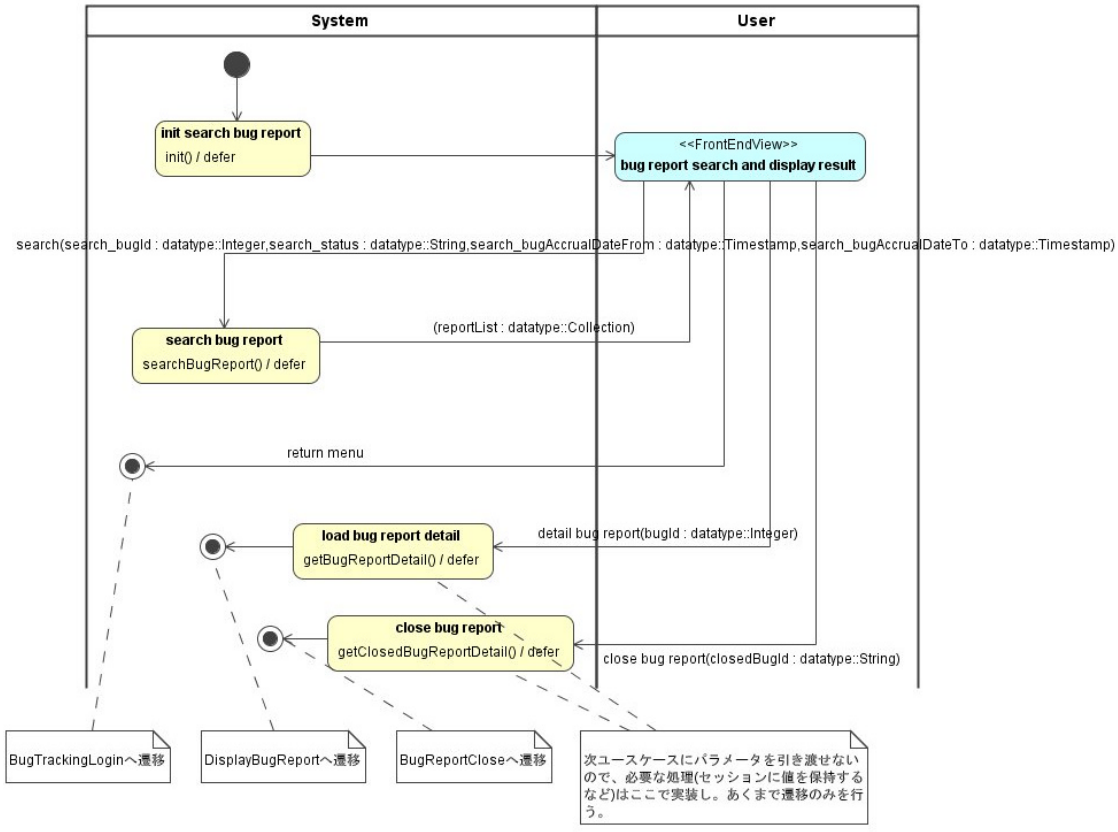


モデル中のコメントは任意の要素であり、コード生成には影響を与えない。

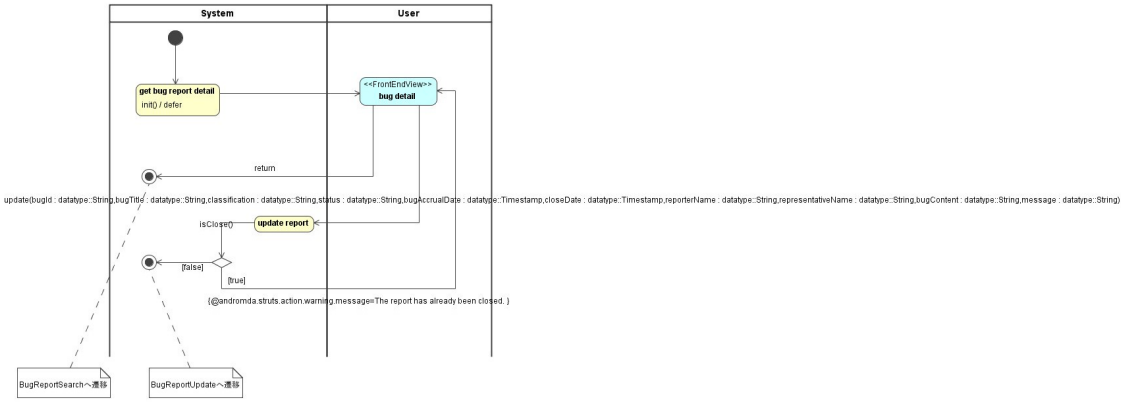
b) バグを新規登録する



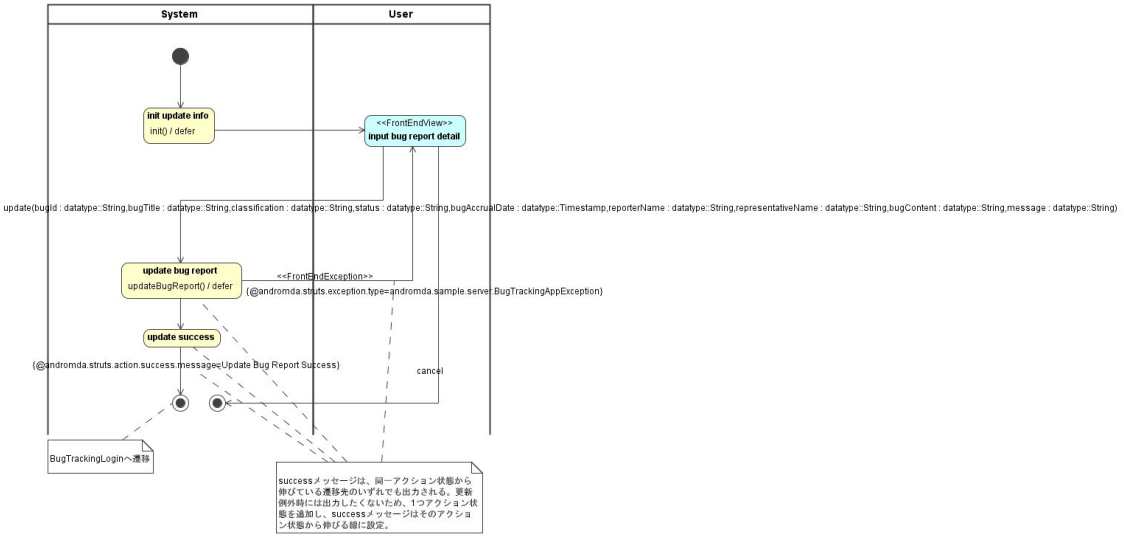
c) バグを検索する



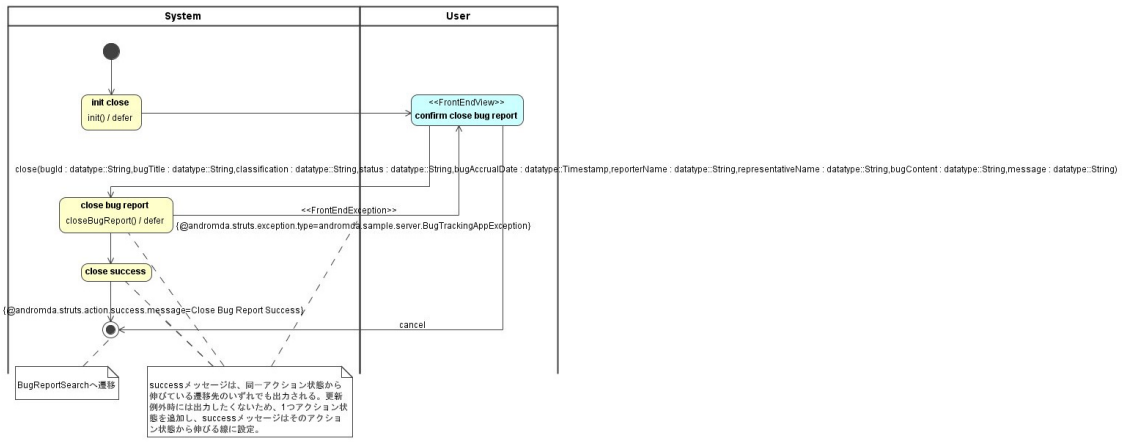
d) バグを詳細表示



e) 進捗情報を更新する

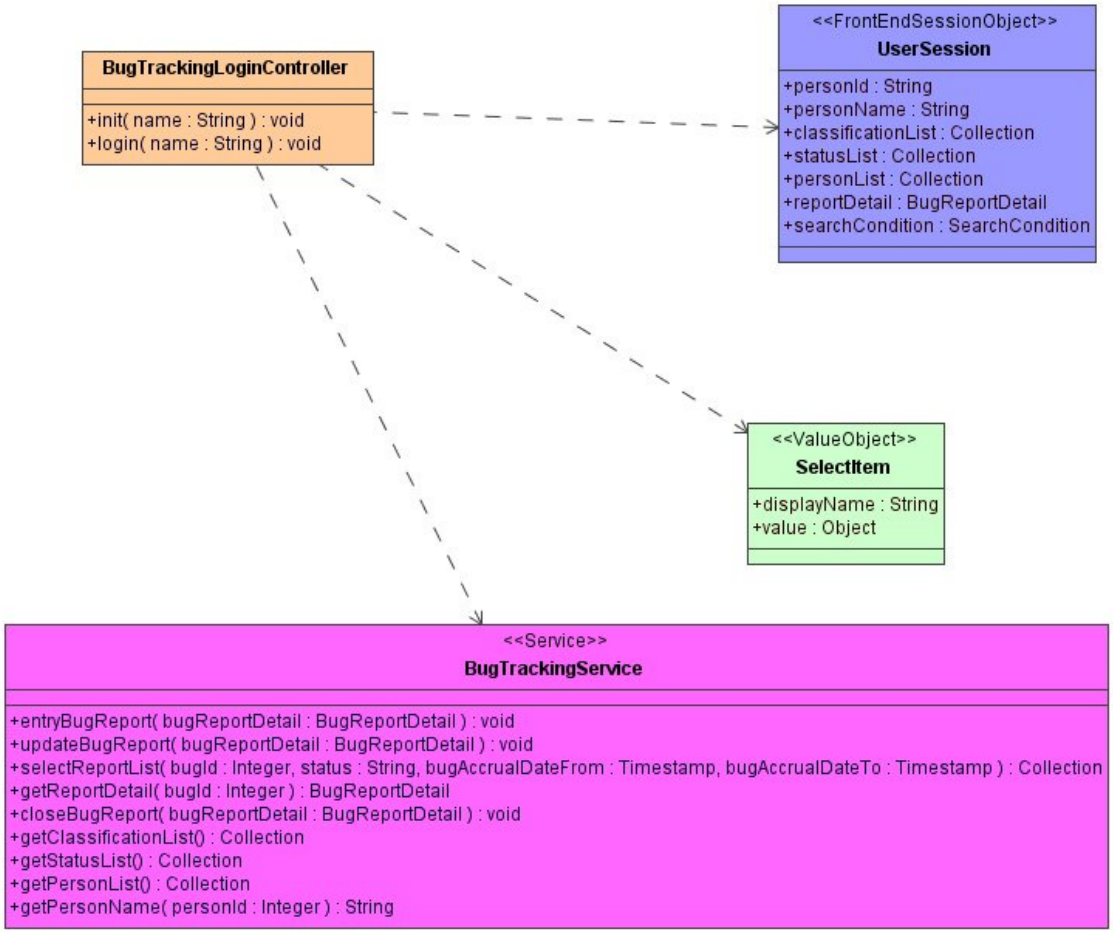


f) バグをクローズする

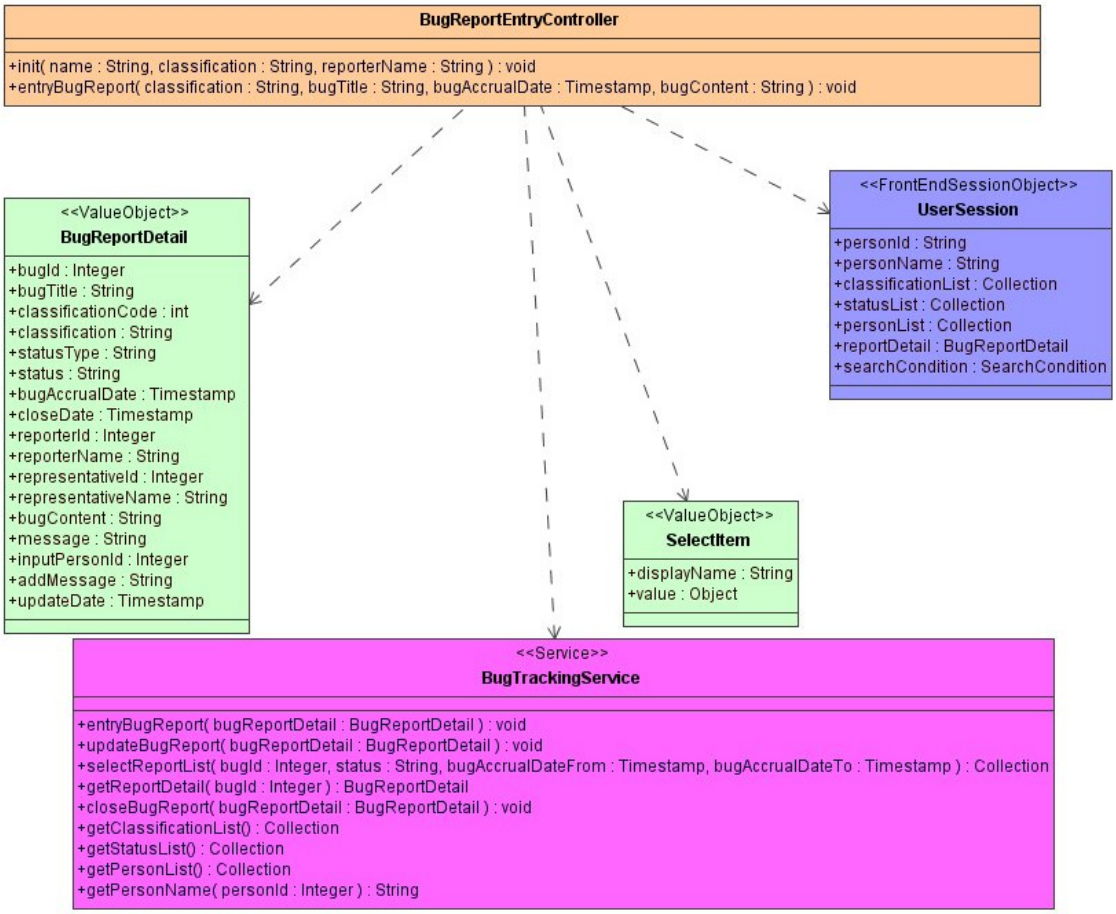


○ Web 側のコントロールクラス図

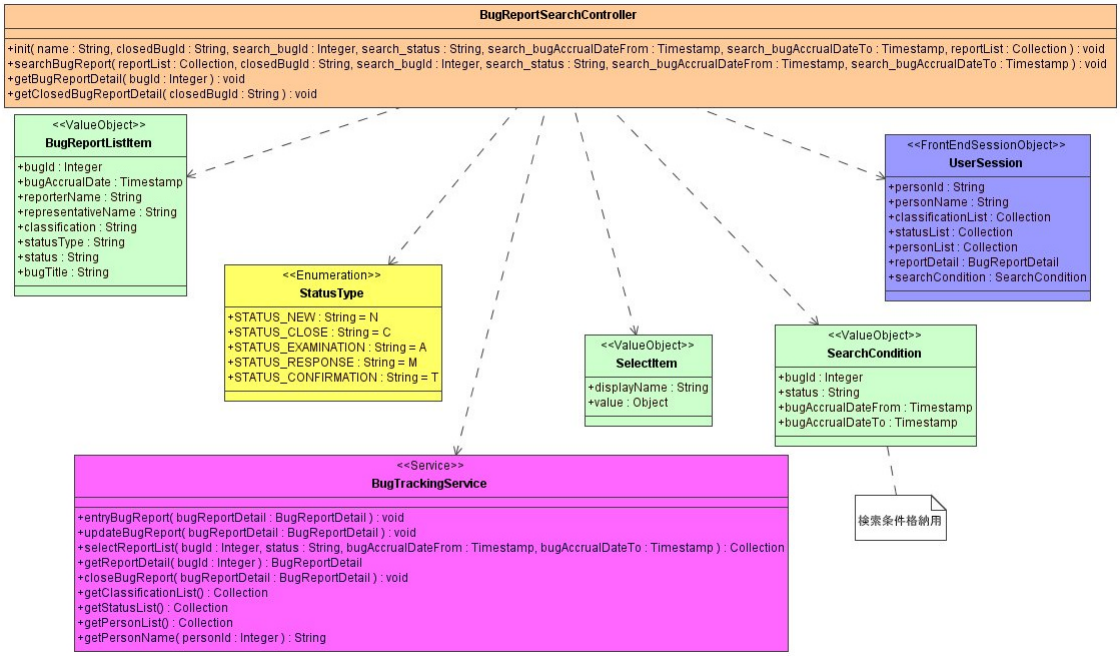
a) ログインする



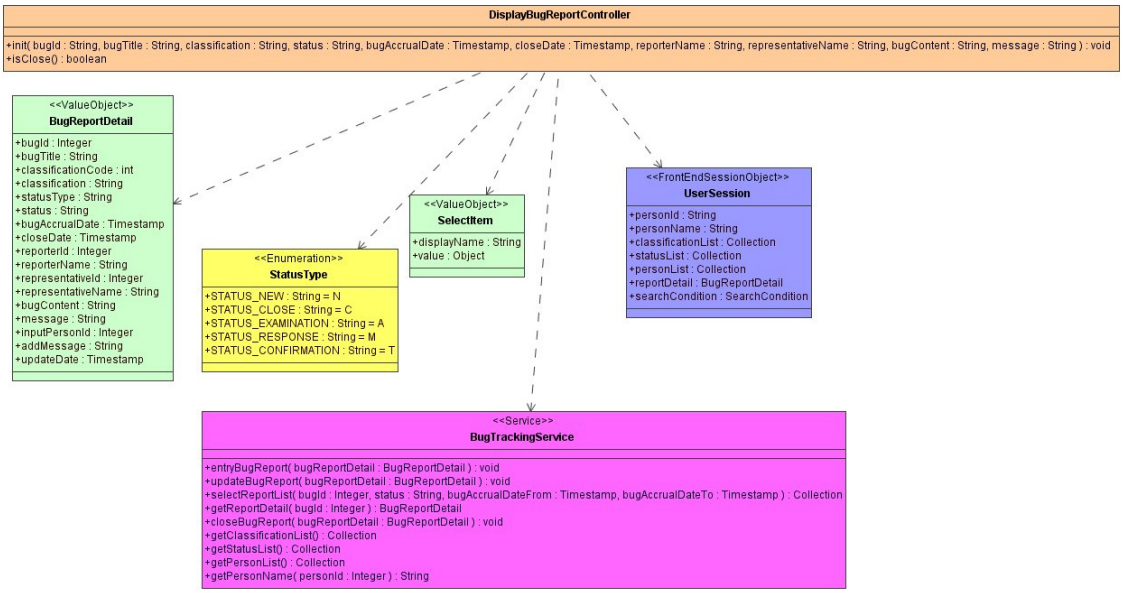
b) バグを新規登録する


BugReportEntryController
+init(name : String, classification : String, reporterName : String) : void
+entryBugReport(classification : String, bugTitle : String, bugAccrualDate : Timestamp, bugContent : String) : void
<<ValueObject>>
BugReportDetail
+bugId : Integer
+bugTitle : String
+classificationCode : int
+classification : String
+statusType : String
+status : String
+bugAccrualDate : Timestamp
+closeDate : Timestamp
+reporterId : Integer
+reporterName : String
+representativeId : Integer
+representativeName : String
+bugContent : String
+message : String
+inputPersonId : Integer
+addMessage : String
+updateDate : Timestamp
<<FrontEndSessionObject>>
UserSession
+personId : String
+personName : String
+classificationList : Collection
+statusList : Collection
+personList : Collection
+reportDetail : BugReportDetail
+searchCondition : SearchCondition
<<ValueObject>>
SelectItem
+displayName : String
+value : Object
<<Service>>
BugTrackingService
+entryBugReport(bugReportDetail : BugReportDetail) : void
+updateBugReport(bugReportDetail : BugReportDetail) : void
+selectReportList(bugId : Integer, status : String, bugAccrualDateFrom : Timestamp, bugAccrualDateTo : Timestamp) : Collection
+getReportDetail(bugId : Integer) : BugReportDetail
+closeBugReport(bugReportDetail : BugReportDetail) : void
+getClassificationList() : Collection
+getStatusList() : Collection
+getPersonList() : Collection
+getPersonName(personId : Integer) : String

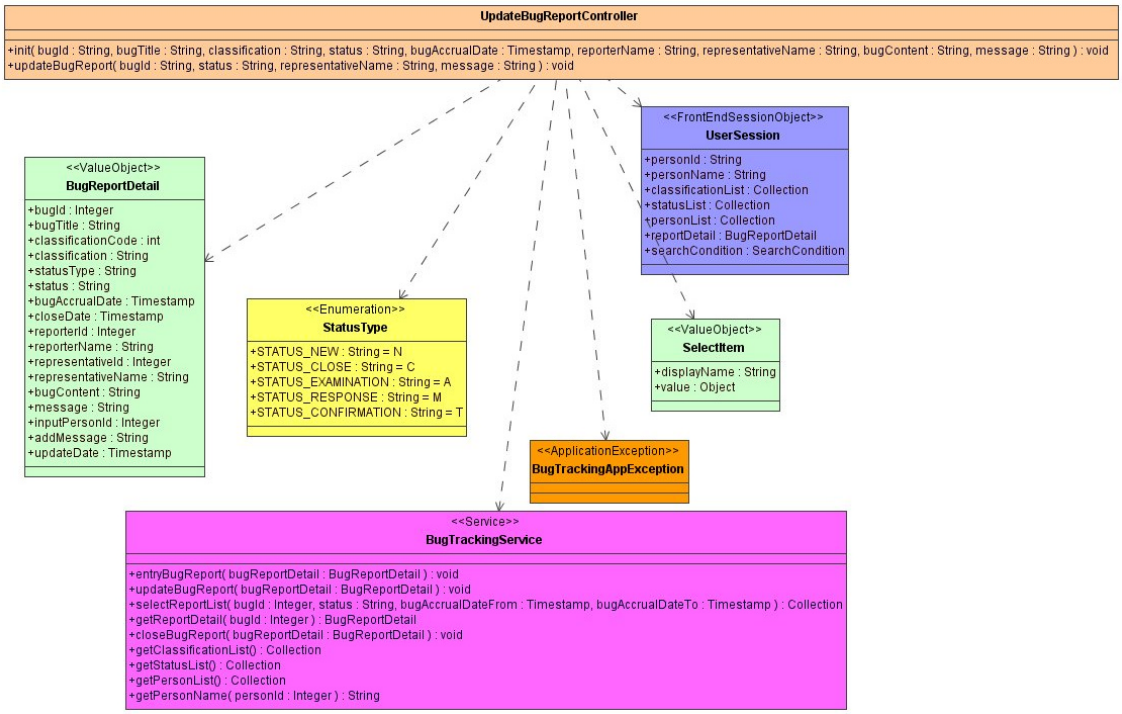
c) バグを検索する



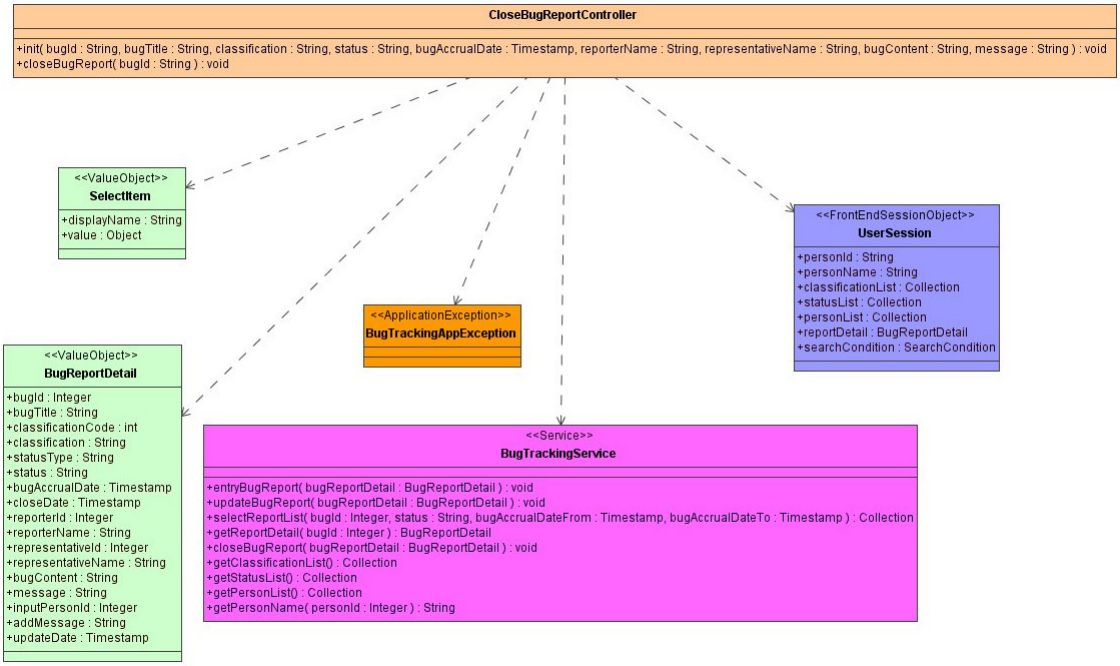
d) バグを詳細表示



e) 進捗情報を更新する



f) バグをクローズする



３．３．４　実装ファイル

本アプリケーション作成時に追加実装を加えたソースファイルは以下の通りである。

ソースファイル名(フルパス)	追加実装 ステップ数
bug_tracking\core\src\java\andromda\sample\server\ **BugTrackingServiceBeanImpl.java**	190
bug_tracking\web\src\java\andromda\sample\web\bug\tracking\menu\ **BugTrackingLoginControllerImpl.java**	19
bug_tracking\web\src\java\andromda\sample\web\bug\tracking\report\entry\ **BugReportEntryControllerImpl.java**	21
bug_tracking\web\src\java\andromda\sample\web\bug\tracking\report\search\ **BugReportSearchControllerImpl.java**	136
bug_tracking\web\src\java\andromda\sample\web\bug\tracking\report\display\ **DisplayBugReportControllerImpl.java**	18
bug_tracking\web\src\java\andromda\sample\web\bug\tracking\report\update\ **UpdateBugReportControllerImpl.java**	52
bug_tracking\web\src\java\andromda\sample\web\bug\tracking\report\close\ **CloseBugReportControllerImpl.java**	14

３．３．５　アプリケーションの実行及び操作方法

ここでは、本アプリケーションの実行と操作方法について説明する。

○　アーカイブの展開

アーカイブファイルをディレクトリ構造がフラットにならないように、圧縮操作ツールのオプションに注意して解凍・展開する。

○　アプリケーションの実行

ビルド、デプロイ、JBoss 起動の手順はこれまでに述べたものと同一である。

すなわち、コマンドプロンプトを開き、本アプリケーションのプロジェクトルートディレクトリへ移動する。本アプリケーションのプロジェクトルートディレクトリ名は「bug_tracking」である。

下記のコマンドを実行する。

```
>maven
```

「BUILD　SUCCESSFUL」と表示されればビルドが完了する。

次に JBoss へデプロイを行う。下記のコマンドを実行する。

```
>maven deploy
```

このコマンドにより、ファイル「bug_tracking\app\target\bug_tracking-app-1.0.ear」が「%JBOSS_HOME%\server\default\deploy」にコピーされる。これでデプロイが完了する。

コマンドプロンプトを開き、「%JBOSS_HOME%\bin」へ移動する。

下記のコマンドを実行し JBoss を起動する。

```
>run
```

アプリケーション固有のテーブル作成及びデータ登録を行うため、別のコマンドプロンプトを開き、「bug_tracking\app\target」へ移動する。

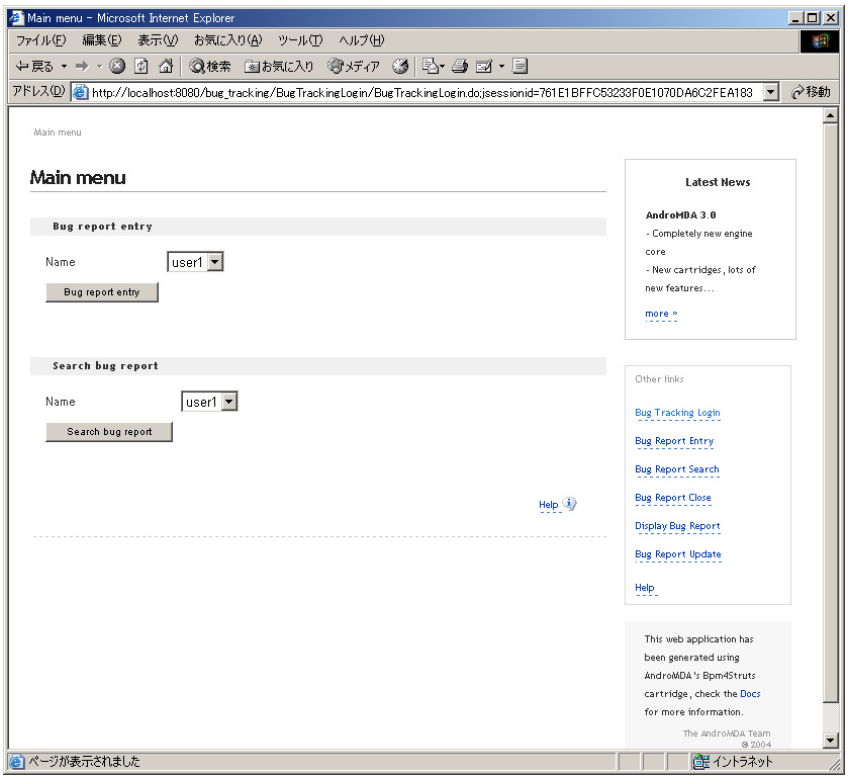
ディレクトリにある下記のファイルを実行する。

```
>initializeSchema.cmd    ※テーブル作成
>create_user_data.cmd    ※個人データ登録
```

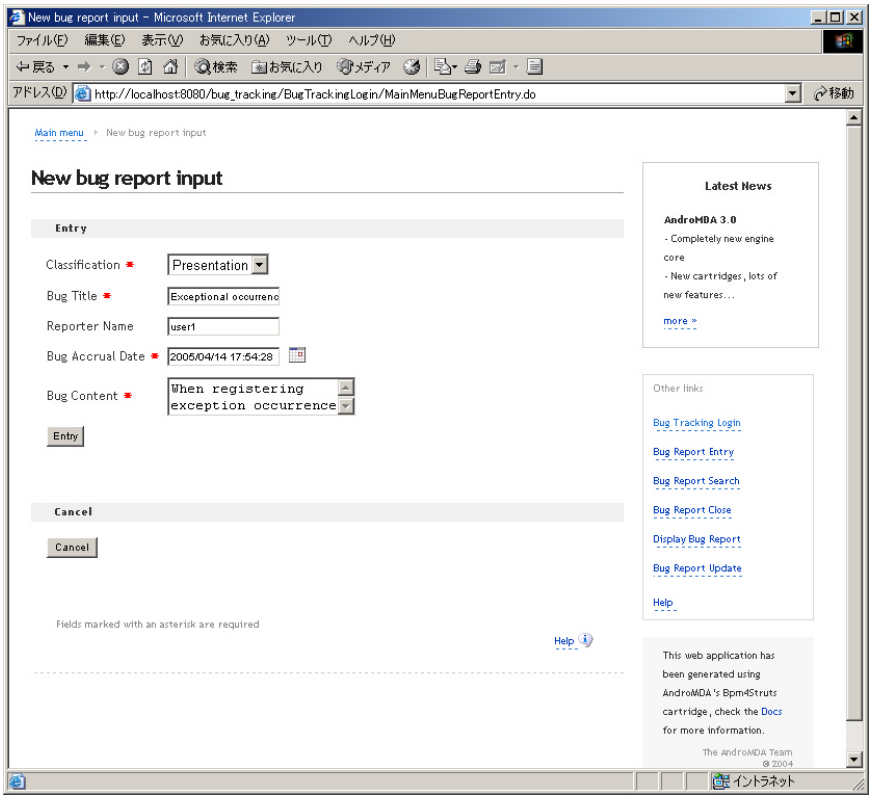
下記の URL へアクセスする。

http://localhost:8080/bug_tracking

○ アプリケーションの操作方法



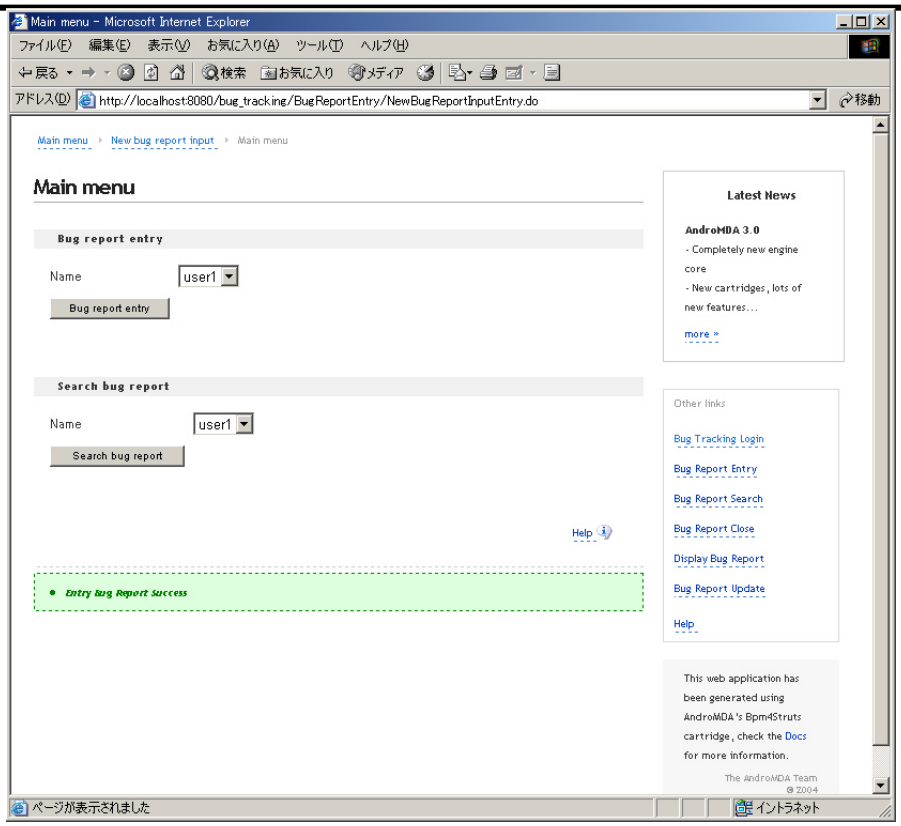
メインメニュー画面では、新規にバグを登録したければ「Name」プルダウンをからログインするユーザーを選択し、「Bug report entry」ボタンを押下する。
バグの検索を行いたい場合は、「Name」プルダウンをからログインするユーザーを選択し、「Search bug report」ボタンを押下する。

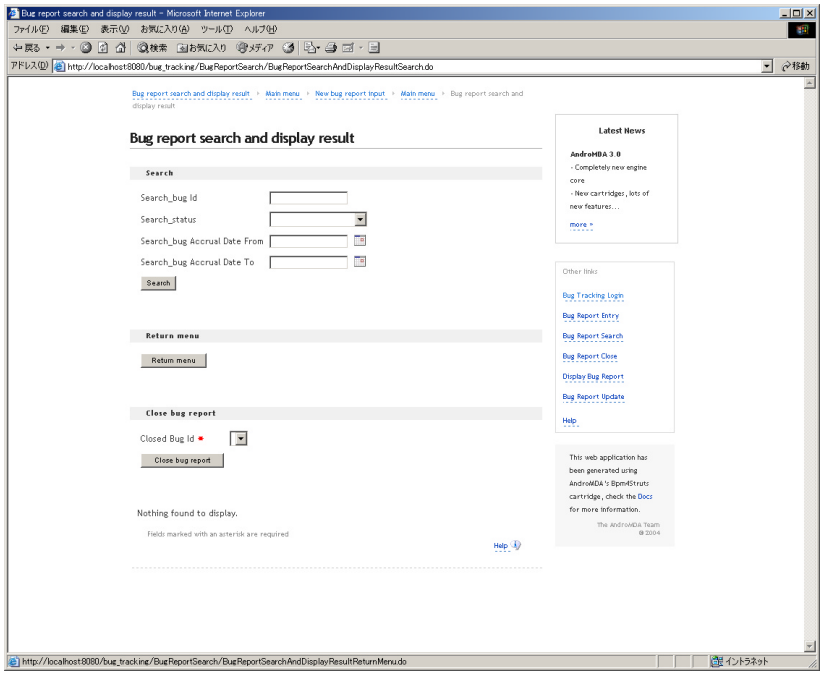


表示されている項目は、

- Classification(分類)
  「Presentation(画面系)」、「Service(サーバ系)」の 2 つ選択することが出来る。
- BugTitle(バグタイトル)
  バグ情報のタイトルを入力する。
- ReporterName(報告者名)
  バグの報告者が表示される。表示される名前はログインユーザー名である。
- BugAccrualDate(バグ発生日)
  バグの発生した日時を入力する。項目の右にあるアイコンをクリックすると日付選択用カレンダダイアログが表示される。
- BugContent(バグ内容)
  発生したバグの内容を入力する。

必要な情報を入力し、「Entry」ボタンを押下するとバグ情報を DB へ登録する。





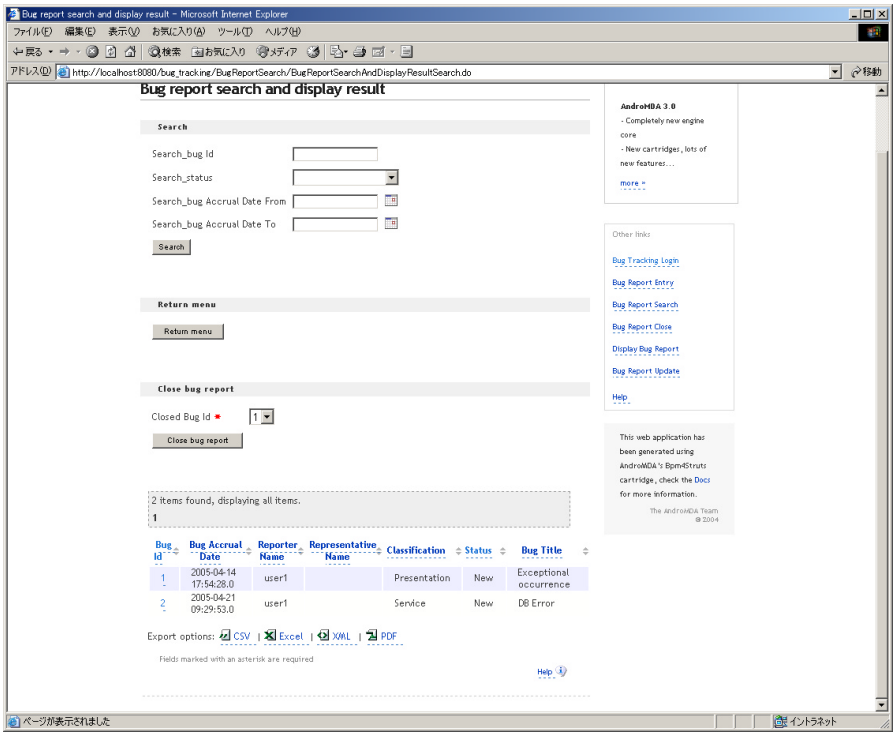
検索の条件として以下の項目がある。

- Search_bugId(バグ ID)
- Search_status(ステータス)
  「New(新規)」、「Close(クローズ)」、「Under Examination(調査中)」、「Response(対応中)」、「Confirmation(修正結果確認中)」をプルダウンで選択できる。
- Search_bugAccrualDateFrom(バグ発生日 From)

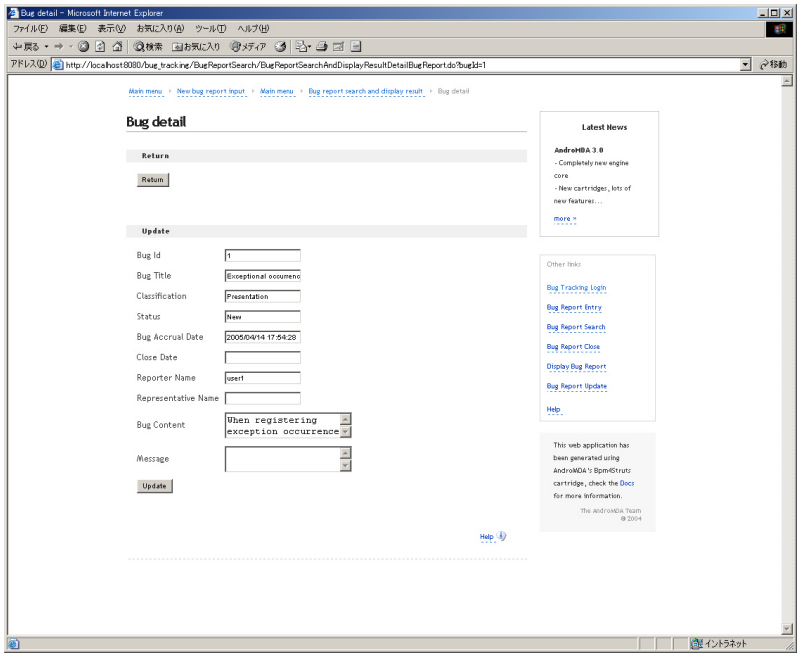
バグ発生日について入力日時以降という条件を設定できる。

- Search_bugAccrualDateTo(バグ発生日 To)
  バグ発生日について入力日時以前という条件を設定できる。

以上の検索条件を必要に応じて設定し、「Search」ボタンを押下することによって検索を実行する。



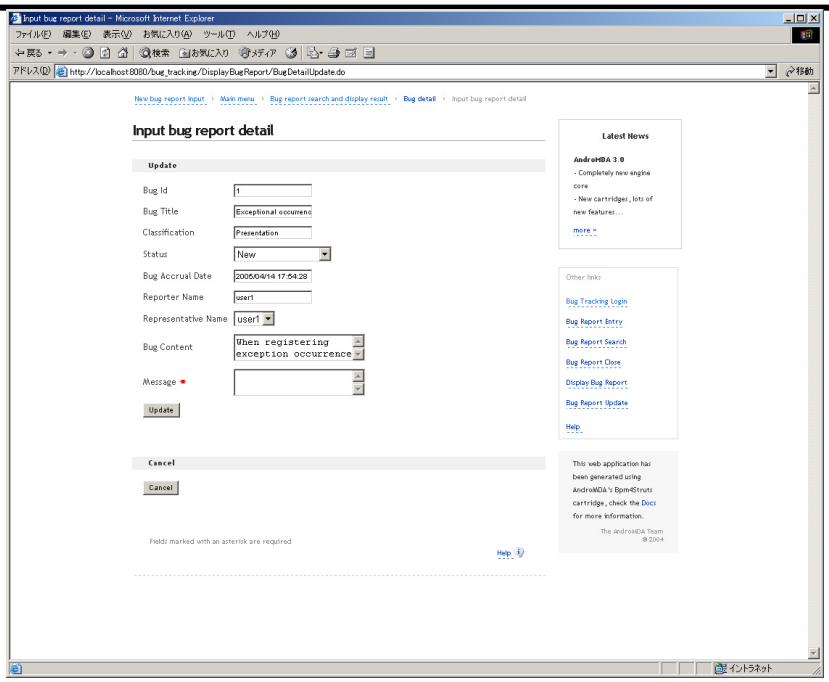
検索結果の「BugId(バグ ID)」カラムのリンクをクリックすると対象バグ情報の詳細画面へ遷移する。



この画面の項目はすべて読み取り専用になっている。表示されている項目は、

- BugId(バグ ID)
- BugTitle(バグタイトル)
- Classification(分類)
- Status(ステータス)
- BugAccrualDate(バグ発生日)
- CloseDate(クローズ日時)
- ReporterName(報告者)
- RepresentativeName(担当者)
- BugContent(バグ内容)
- Message(メッセージ)

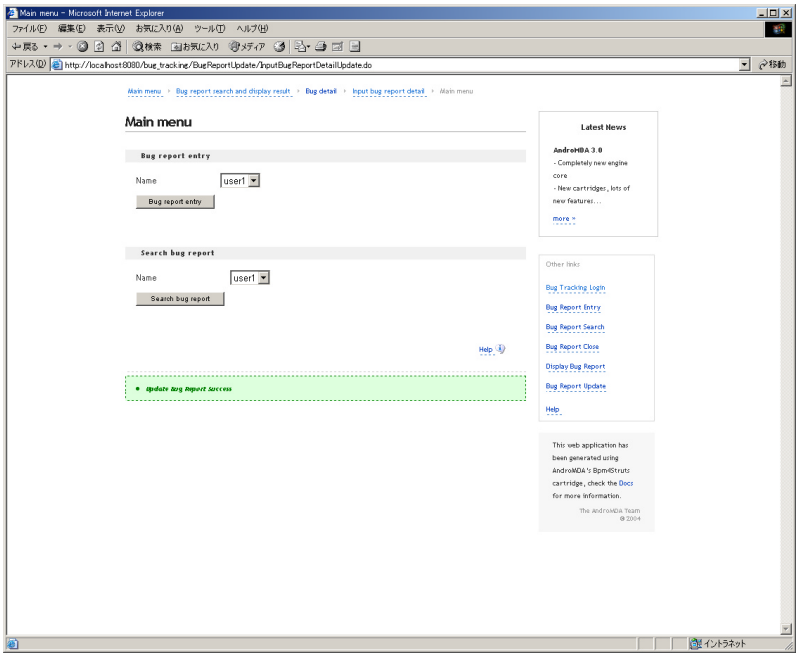
内容を更新したい場合は「Update」ボタンを押下する。押下すると更新画面へ遷移する。

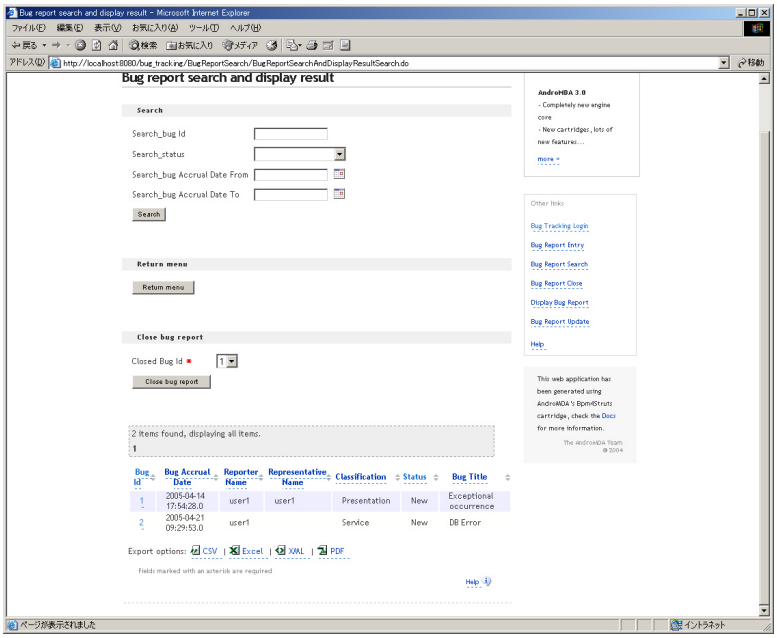


内容を変更できる項目は下記の項目

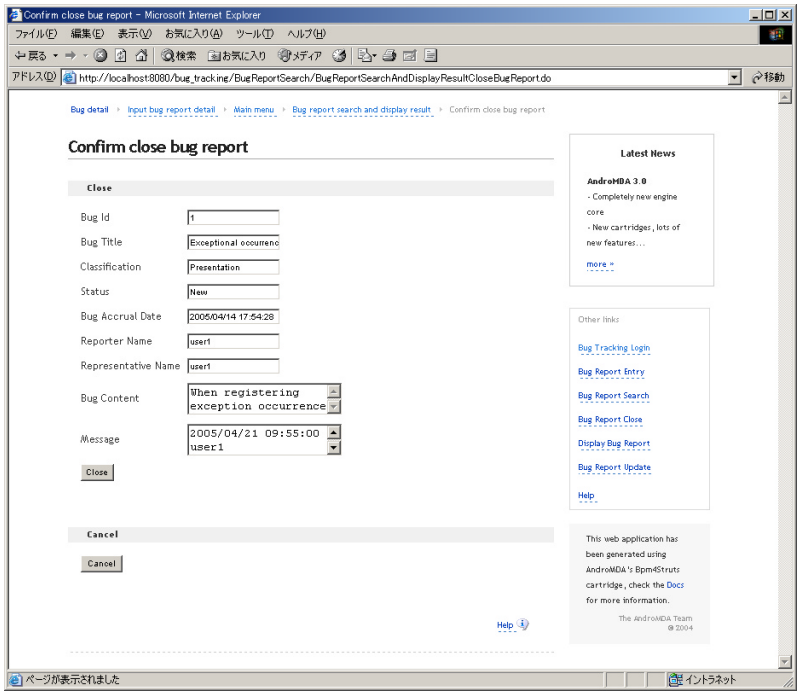
- Status(ステータス)
  「New(新規)」、「Under Examination(調査中)」、「Response(対応中)」、「Confirmation(修正結果確認中)」をプルダウンで選択できる。
- RepresentativeName(担当者)
  登録済みの個人から選択する
- Message(メッセージ)
  この項目は変更ではなく追加するメッセージを入力する。

必要な項目を変更し「Update」ボタンを押下するとバグ情報を更新する。更新を行ったログインユーザー情報はバグボディテーブルへ入力者として登録する。

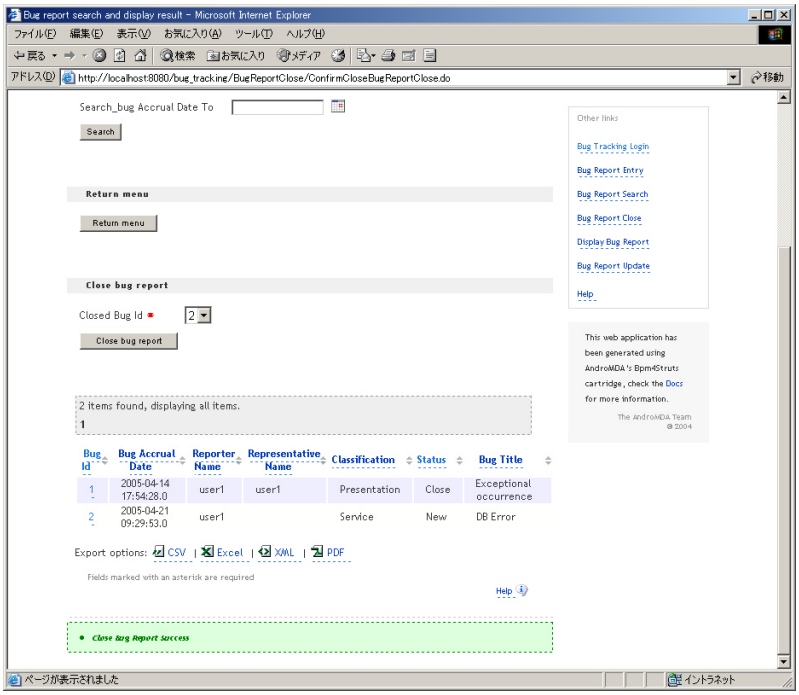




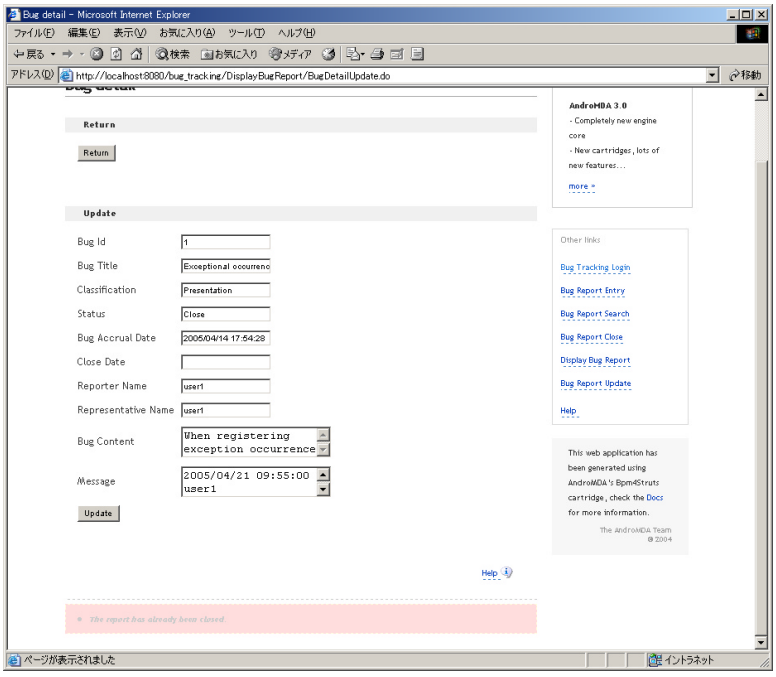
検索画面で検索を実行すると、ステータスが「クローズ」でないバグ情報のバグ ID が「ClosedBugId」にプルダウンで表示される。クローズしたいバグ情報のバグ ID を選択し「Close bug report」ボタンを押下するとクローズ確認画面へ遷移する。



内容を確認しクローズして問題なければ「Close」ボタンを押下する。
ステータスが「Close(クローズ)」に変更され、クローズ日時を登録する。



クローズされたバグ情報は「ClosedBugId」プルダウンに表示されなくなる。
クローズされたバグ情報は詳細画面から更新画面へ遷移することができなくなる。



以上がバグ追跡掲示板アプリケーションの操作方法である。

３．３．６　使用したステレオタイプとタグ付き値

下記に本アプリケーションで使用したステレオタイプとタグ付き値を記述する。

ステレオタイプ	内容
FrontEndApplication	アプリケーションの開始点を意味する
FrontEndUseCase	設定したユースケースをアプリケーションに含めることを意味する
Entiry	テーブルをマッピングしたクラスを表す
ValueObject	値オブジェクトを表す
Service	ファサードの役割を行うクラスを表す
Identifier	Entity の id(PK)を表す
FinderMethod	検索メソッドを表す hibernate カートリッジでは、検索クエリが生成される
ApplicationException	例外クラスを表す
Enumeration	列挙型のクラスを表す
FrontEndView	JSP を表す
FrontEndException	例外処理遷移を表す
FrontEndSessionObject	セッションオブジェクトを表す

タグ付き値	内容
@andromda.hibernate.generator.class	Entity の id 生成方法について設定する
@andromda.persistence.column.length	桁数を設定する
@andromda.struts.view.field.required	必須項目かどうかを設定する
@andromda.struts.view.table.column	テーブルに表示する項目を設定する。カンマで区切る。
@andromda.struts.view.table.maxrows	テーブル表示の最大行数
@andromda.struts.view.table.sortable	テーブルにソート機能を付加するかを設定
@andromda.struts.view.field.tablelink	リンクを作成するテーブルのカラムを設定する。 設定方法は、 テーブルパラメータ.カラム (例：resultList.id)
@andromda.struts.view.field.type	項目の種別を設定する
@andromda.struts.view.field.readonly	読み取り専用項目かどうかを設定する
@andromda.struts.action.success.message	成功メッセージを設定する
@andromda.struts.action.warning.message	警告メッセージを設定する
@andromda.struts.view.field.format	項目のフォーマットを設定する
@andromda.struts.exception.type	例外のタイプを設定する

4　AndroMDA を用いる場合の制約事項

現時点(2005 年 4 月末日)での AndroMDA のバージョンは「3.0-RC1」である。AndroMDA はまだ開発途中でもあり、使用する際にいくつか制約事項がある。この章ではいくつか制約事項を記述する。

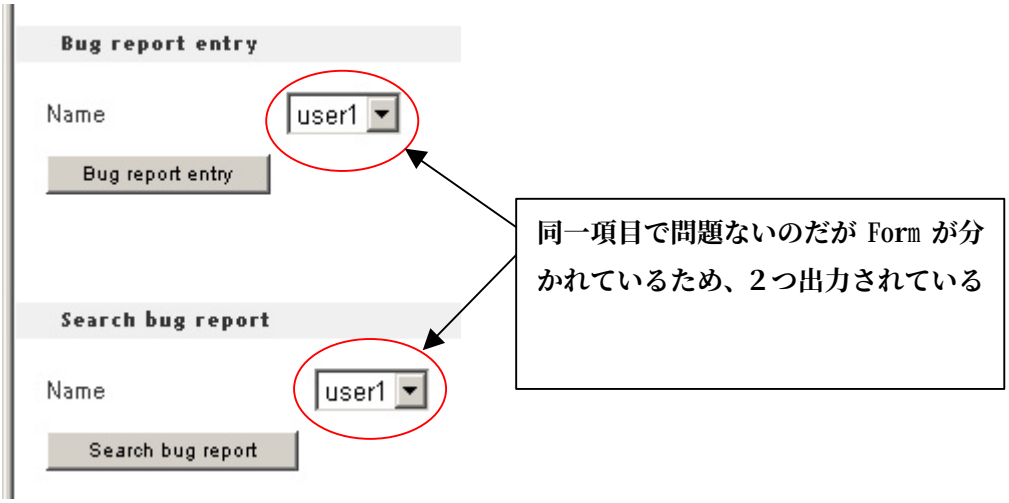
4．3　画面レイアウトの制限

AndroMDA デフォルトで出力される JSP については画面のいくつか制限がある。

下記にいくつかあげる。

- ひとつの Form に２つ以上の Submit ボタンを配置できない

  アクティビティ図で画面遷移を設定するが、遷移の線１本に付き 1Form になる。そのため、複数 Submit ボタンでひとつの項目を利用することができず、１つの Submit ボタンに付き１つ同一項目を用意される。



- テーブル表示制御ができない

  テーブルにチェックボックスやラジオボタンをつけることは可能だが、Submit ボタンが出力されないため、その情報をリクエストとして投げることができない。

Id	Title	Author	Available
id-1	title-1	test-1	☐
id-2	title-2	test-1	☐
id-3	title-3	test-1	☐
id-4	title-4	test-1	☐
id-5	title-5	test-1	☐

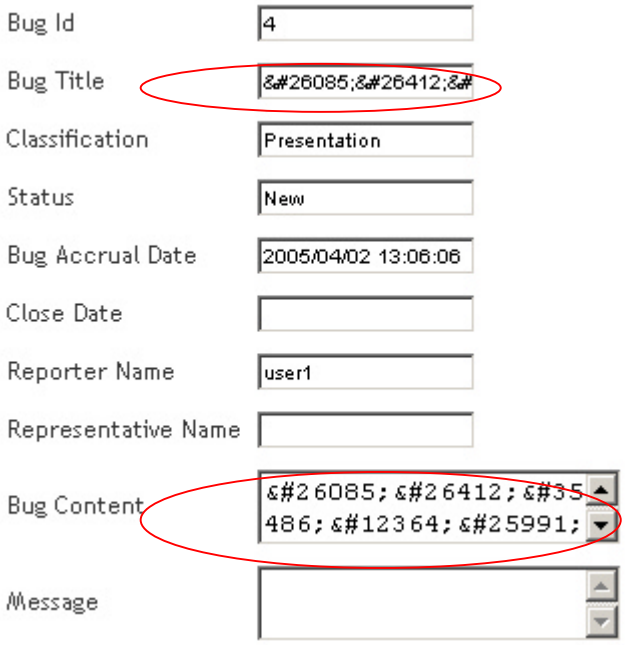
テーブルからイベントを発生させるには、テーブルの１カラムをリンクにしてイベントを作成するしかできない。

・ 項目に対して設定できる表示形式が少ない
右寄せや左寄せ、カンマ付け、項目の表示サイズなどの設定はできず、デフォルトまま。
ただし、日付などのフォーマットについては設定できる。

4．4　日本語は文字化けする

Bug Id	Bug Accrual Date	Reporter Name	Representative Name	Classification	Status	Bug Title
1	2005-04-14 17:54:28.0	user1	user1	Presentation	Close	Exceptional occurrence
2	2005-04-21 09:29:53.0	user1		Service	New	DB Error
3	2005-04-22 10:18:56.0	user1		Presentation	New	日本語バグ
4	2005-04-02 13:06:06.0	user1		Presentation	New	日本語文字化け

テーブルに表示した場合は文字化けないが、



テキストボックスなどに表示すると文字化けする。

4．5　データベーステーブルの項目設定の制限

カラムの名前に日本語を使用するとカラム名が文字化けするなど、いくつか項目の設定に制限がある。

※hibernate カートリッジを使用

4．6　OCL の使用制限

AndroMDA は OCL をサポートしているが、現在のところステレオタイプ「FinderMethod」を設定したメソッドにしか適用できない。さらに、「allInstances() -> select」しか宣言できない。
詳細は、http://www.andromda.org/andromda-ocl-query-library/modeling.html を参照。

5 まとめ

以上で AndroMDA のバンドルカートリッジを用いた Web アプリケーション開発をマスターしていただけたと思う。近い将来、ソフトウェア開発は全面的に MDA になるだろうか。それとも特定の分野だけで MDA が適用されるのだろうか。読者の担当分野にどのように MDA を取り入れていくか、より正確な感触を得るために、本書が寄与できれば幸である。