線形モジュール追加型抽象階層とAndroMDAによるオンライン会計システムの開発

奥野, 雄一 / OKUNO, Yuichi

(発行年 / Year)
2007-03-24
(学位授与年月日 / Date of Granted)
2007-03-24
(学位名 / Degree Name)
修士(理学)
(学位授与機関 / Degree Grantor)
法政大学 (Hosei University)

線形モジュール追加型抽象階層と AndroMDA による

オンライン会計システムの開発

法政大学　情報科学研究科　情報科学専攻
奥野雄一
yuichi.okuno.gu@gs-cis.hosei.ac.jp

abstract ソフトウェア開発において、バグは大きな問題をもたらしている。従来のソフトウェア開発技法においては、モデリングする際に理論的に表現できていなかったことと、プログラムを書く際の人為的なミスによって、バグが発生していた。この論文では IMAH(Incrementally modular abstraction hierarchy)と MDA(Model Driven Architecture)を組み合わせて利用し、このようなバグの削減方法を示す。IMAH は抽象的なレベルから具体的なレベルへと設計を段階的に進めながら、それぞれのレベルで不変量を線形に増加させることで、UML クラス図を理論的に表現できる。IMAH で得られたクラス図を基に、ここでは MDA 対応フレームワークの AndroMDA を用いて、プログラムを自動生成する。この方法を用いて会計システムを開発した。また、開発では、基本的な会計システム、元帳を利用できるようにする、仕訳伝票の承認をできるようにする、ログイン機能の追加というようにモジュールを追加していくことで開発した。その結果、AndroMDA の自動化できない部分においてのバグ以外は削減することができた。

1. 序論

1.1. 背景

現在、ソフトウェア開発において、バグとは切っても切れない関係にある。ソフトウェアが大規模になるにつれてバグの数は増えていく一方であり、2005 年 11 月に東証のシステムダウン、12 月にはみずほ証券の誤発注問題が起きた。また、自動車業界でもソフトウェアのバグによるリコールなどが起きている。損害賠償の請求を示唆されている事例もあり、これらのソフトウェアのバグによるトラブルが発生すれば、利用する企業に加え、ソフトウェア開発企業においても大きな被害をも[1]たらすといえる。バグのないシステムの開発を行うことが求められる。

Supervisor: Prof. Kenji Ohmori

ソフトウェアの開発手法の主流として、ウォーターフォール・モデルが使われている。これは要求仕様、設計、実装、テスト、の工程順に、開発を進めていく開発手法である。各工程では、上流工程から引き渡された成果物を元に作業が行われるので、上流工程におけるバグが、下流工程に拡大して影響する。また、バグが発生した場合、工程を後戻りしなければならないので、開発の遅延など問題が発生し、バグの見落としも懸念される。また、テストがソフトウェア開発全体の工数の中で占める割合は、少なくても 3 割、多い場合は 9 割にも達する。バグをつぶすための開発時間が大部分を占めており、品質の良いソフトウェアの開発が行える環境ではないと言える。これを解決する手段として、RUP (Rational Unified Process)などがある。ウォーターフォール・モデルでの工程を繰り返して、ソフトウェアを少しずつ開発していくことで、早い段階でバグを潰していこうというものである。これは工程の後戻りによる開発の遅延などに対応するようにはなるが、本質的にバグを消失させる方法論ではないといえる。

RUP においては、UML でのモデリングが必須である。クラス図の設計をする際には、開発を進めていくにつれて、多くのクラス図を加えていく必要がある。しかし、クラス図は理論的にどのようなクラスや関連が必要であるか、表現することが難しい。曖昧な表現で書かれたクラス図を基に、開発を進めていくと、バグの発生を引き起こしてしまうことになり、デバッグのために余計な反復を繰り返してしまう。

また、設計されたモデルを基にプログラムを実装する段階での、人為的なミスによってもバグが発生する。これはプログラムを書く故に発生するバグといえる。

ここでは、IMAH(Incrementally modular abstraction hierarchy)と MDA を組み合わせる手法を用いて会計システムを開発することで、システム開発におけるバグを削減することを目指した。

IMAH は、ホモトピーレベル、集合論レベル、位相空間レベル、接着空間レベル、セル空間レベル、表現レベル、可視レベルからなる。これらは順に抽象的なレベルから具体的なレベルへと変化していくものである。このレベル順に階層を下りながら、それぞれのレベルで不変量を線形に増加させていき、ソフトウェアを開発する。この方法を用いることでクラス図を論理的に表現できるようになり、曖昧さを取り除くことができる。

MDA（Model Driven Architecture）は OMG（Object Management Group）によって定義された、UML などでソフトウェアのモデルを作成し、これを基にアプリケーションを自動化、半自動化で生成する、次世代のオブジェクト指向開発体系である。ここでは、MDA に基づく開発フレームワークである AndroMDA を利用した。AndroMDA はカートリッジと呼ばれる変換定義を使用し、J2EE や.NET などのソースコードを自動生成することができる。自動生成されたコードは、これまで繰り返し使われてきた、信頼性の高いコードであるといえる。IMAH における表現、可視レベルにおいて、AndroMDA を利用することにより、ソフトウェアを生成する。UML が正しく定義されていれば、自動生成された部分において、人為的なバグの発生は避けることができる。現段階での AndroMDA では、ビジネスロジック

についてのみ記述する必要があり、その部分においてはテスト、デバッグが必要になる。しかし、全体の 10%程度であり、ほとんどの部分においてのバグを取り除くことができる。

1.2. 会計システム

最近は gmail や google calendar に代表されるような、OS に依存しない、ウェブアプリケーションが主流になってきている。利用する社員が、ローカルな場所に限定されず、どこからでも同じデータにアクセスすることで、システムの利便性を上げるために、ウェブアプリケーションで開発した。

ここでは、会計システムのアーキテクチャは Web+DB である。図 1-1 のように Web 側は、JSP を、DB 側は EJB を用いる。

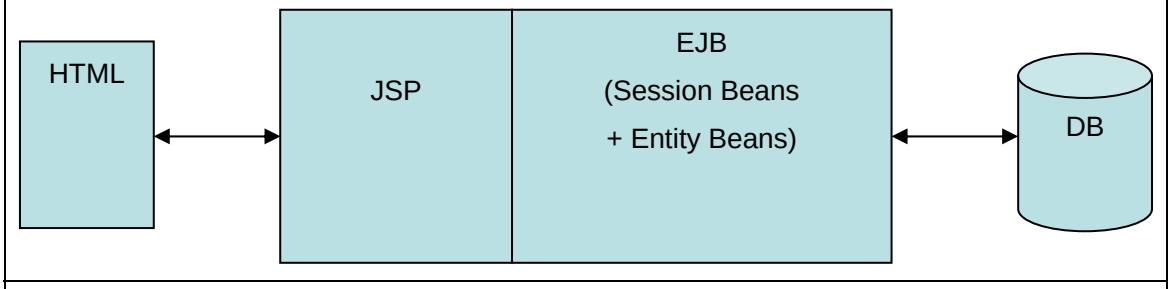

図 1-1　アーキテクチャ

会計システムは、それぞれの企業の活動を財務の面から見たものである。企業活動は、商取引により顕在化されるが、これは、例えば、表 1-1 のように示すことができる。この表で、企業 A は B から部品 P1,P2,P3 を 2 万、3 万、5 万円で購入したとする。

表 1-1 商取引

日付	買手	品名	金額	売手
2006/12/1	A	P1	20,000	B
		P2	30,000	
		P3	50,000	

ここで企業 A を考えることにすると、この企業での仕訳伝票は表 1-2 に示すようなものとなる。この会計システムでは単式簿記を用いるので、貸方金額がマイナスで表される。

会計システムには、会計科目が用意されており、通常は、勘定科目ごとに、元帳(ledger)を作成する。元帳は表 1-3 のように作られるとする。

この元帳は科目ごとにその明細を見ることができる。

これを利用者について考慮し、ここで開発する会計システムは図 1-2 のようにする。

表 1-2　仕訳伝票

仕訳伝票

伝票番号

1

頭書

申請日付	備考	申請者	承認者	承認日付
2006/12/1	企業 B より部品 P1, P2, P3 を購入	A	B	2006/12/1

明細書

借方		貸方	
金額	勘定科目	勘定科目	金額
20,000	部品	現金	20,000
30,000	部品	現金	30,000
50,000	部品	現金	50,000

表 1-3 元帳

元帳

勘定科目	合計
現金	-80,000
部品	80,000

また、会計システムの概要を以下のように定める。

会計システムは、仕訳伝票を入力し、元帳を作成する。

１．仕訳伝票は、伝票番号、頭書、明細書から成り立っている。頭書は、申請者、申請処理日、承認者、承認処理日、備考で成り立っている。明細書はひとつ以上の明細からなり、明細は、明細番号(自動採番)、科目、金額で成り立っている。

２．会計システムの利用者の役割には４通りあり、申請者、承認者、管理者、会計士である。

３．申請者は、仕訳伝票を入力することができる。また、伝票が承認者によって承認・却下されていないときは、申請者はそれを変更、削除することができる。また、仕訳伝票の状態を見るために申請者と承認者はそれを参照することもできる。

４．承認者は、申請のあった仕訳伝票に対して認否を行う。認否が承認であったときは、

仕訳伝票から科目名ごとの元帳が作成される。

５．会計士は、元帳の参照することができる。

６．管理者は、会計システムの利用者、勘定科目、仕訳帳の管理を行う。

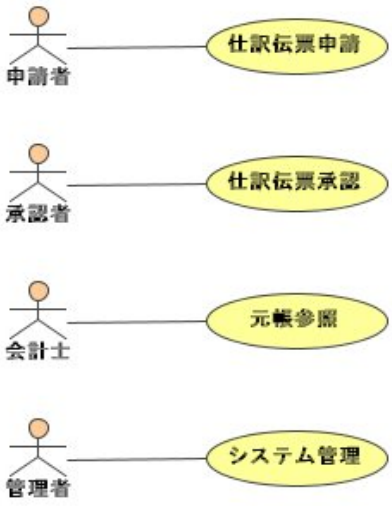

図 1-2　会計システムのユースケース図

システムの動作の詳細を利用者ごとに述べる。

- 申請者

・仕訳伝票を検索する

1. 伝票番号、日付、申請者を入力する。
2. 検索ボタンを押すことで、入力されたのもの組合せでデータベースから検索する。
3. システムは検索結果として、仕訳伝票番号、申請者、申請日、備考を表示する。

・仕訳伝票を生成する

1. 伝票番号、申請者、申請日付、備考、貸方金額、貸方科目コード、借方金額、借方科目コードを入力する。
2. 生成ボタンを押す
3. システムは仕訳伝票をデータベースに書き込む。

・仕訳伝票を削除する

1. 検索結果で表示された仕訳伝票から、削除したいものを選択する。

2. 削除ボタンを押す。
3. システムはデータベースから削除する。

・仕訳伝票を更新する

1. 検索された仕訳伝票に、更新したい値を入力する。
2. 更新ボタンを押す。
3. システムはデータベースを更新する。

・明細書を見る。

1. 検索された仕訳伝票の明細書ボタンを押す。
2. システムは明細を表示する。

・明細を生成する。

1. 金額、科目コードを入力する。
2. 生成ボタンを押す
3. システムはデータベースに書き込む。

・明細を削除する

1. 表示された明細から、削除したいものを選択する。
2. 削除ボタンを押す。
3. システムはデータベースから削除する。

・明細を更新する

1. 表示された仕訳伝票に、更新したい値を入力する。
2. 更新ボタンを押す。
3. システムはデータベースを更新する。

- 承認者

・仕訳伝票を承認する。

1. 承認する仕訳伝票の承認ボタンを押す。
2. システムはデータベースに承認者、承認日付、承認済を書き込む。

・仕訳伝票を却下する。

1. 却下する仕訳伝票の却下ボタンを押す。
2. システムはデータベースに却下が書き込む。

・仕訳伝票を検索する。

1. 伝票番号、日付、申請者を入力する。
2. 検索ボタンを押すことで、入力されたのもの組合せでデータベースから検索する。
3. システムは検索結果を表示する。

- 会計士

・特定の勘定科目の元帳を参照する。

1. 参照したい勘定科目コードを入力する
2. 参照ボタンを押す。
3. システムは該当の勘定科目の元帳を表示する。

・全ての勘定科目の元帳を参照する。

1. 全て参照ボタンを押す。
2. システムは元帳を表示する。

・元帳の詳細を参照する。

1. 参照された元帳の分解ボタンを押す。
2. システムは元帳が持つ詳細を表示する。

・詳細が書かれた仕訳伝票を見る。

1. 参照された詳細の仕訳伝票ボタンを押す。
2. システムは仕訳伝票を表示する。

- 管理者

・勘定科目を検索する

1. 科目名、コードを入力、親科目を選択する。
2. 検索ボタンを押すことで、入力されたのもの組合せでデータベースから検索する。
3. システムは検索結果として、科目名、コード、親科目を表示する。

・勘定科目を生成する

1. 科目名、コードを入力、親科目を選択する。
2. 生成ボタンを押す
3. システムはデータベースに書き込む。

・勘定科目を削除する

1. 検索結果で表示された仕訳伝票から、削除したいものを選択する。
2. 削除ボタンを押す。
3. システムはデータベースから削除する。

・勘定科目を更新する

1. 検索された仕訳伝票から、更新したいものを選択する。
2. 更新したい値を入力する。
3. 更新ボタンを押す。
4. システムはデータベースを更新する。

・社員を検索する

1. 社員名、コード、所属、役職を入力、システム権限を選択する。
2. 検索ボタンを押すことで、入力されたのもの組合せでデータベースから検索する。
3. システムは検索結果として、社員名、コード、所属、役職、システム権限を表示する。

・社員を生成する

1. 社員名、コード、所属、役職を入力、システム権限を選択する。
2. 生成ボタンを押す
3. システムはデータベースに書き込む。

・社員を削除する

1. 検索結果で表示された社員から、削除したいものを選択する。
2. 削除ボタンを押す。
3. システムはデータベースから削除する。

・社員を更新する

1. 検索された社員から、更新したいものを選択する。
2. 更新したい値を入力する。
3. 更新ボタンを押す。
4. システムはデータベースを更新する。

・元帳を検索する

1. 金額、勘定科目を入力する。

2. 検索ボタンを押すことで、入力されたのもの組合せでデータベースから検索する。
3. システムは検索結果として、金額と勘定科目を表示する。

・元帳を生成する

1. 金額、勘定科目を入力する。
2. 生成ボタンを押す
3. システムはデータベースに書き込む。

・元帳を削除する

1. 検索結果で表示された元帳から、削除したいものを選択する。
2. 削除ボタンを押す。
3. システムはデータベースから削除する。

・元帳を更新する

1. 検索された元帳から、更新したいものを選択する。
2. 更新したい値を入力する。
3. 更新ボタンを押す。
4. システムはデータベースを更新する。

なお、仕訳伝票の変更に当たっては、仕訳伝票を取ってきたとき一度セッションが切断され、更新しようとするときに新たなセッションが作られる。このため、切断の間に、別の人が更新している可能性がある。このような場合、取ってきた伝票の記録時間と、データベースにある伝票の記憶時間を比較し、同じでなければ書き込みを行わずエラーで戻す。これは、多くのオンライン・チケット予約システムと同じである。

ここでは、仕訳帳だけからなるきわめて基本的な会計システムから開発し、元帳を利用できるようにする、仕訳伝票の承認をできるようにする、というようにモジュールを追加することで開発していった。

開発には UML からウェブプリケーションを自動生成することのできる、AndroMDA を利用した。AndroMDA は Web の部分をユースケース図とアクティビティ図で、EJB の部分をクラス図で表す。また、アプリケーションサーバとしては JBOSS、JBoss はウェブサーバ（Struts）、データベースサーバ（Hypersonic）としての機能も有しており、ここではそれを用いた。また、MagicDraw（http://www.magicdraw.com/）を UML モデル記述用ツールとして使用した。

2. 基本的な会計システム

2.1. ホモトピーレベル

このレベルでは、それぞれの空間が定義される。

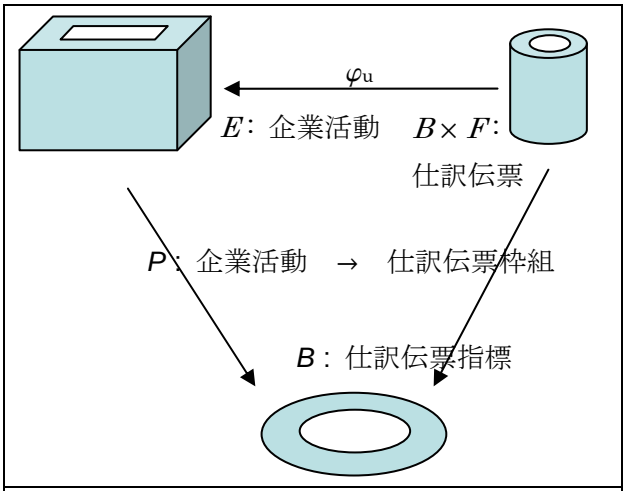

図 2-1　基本的な会計システム

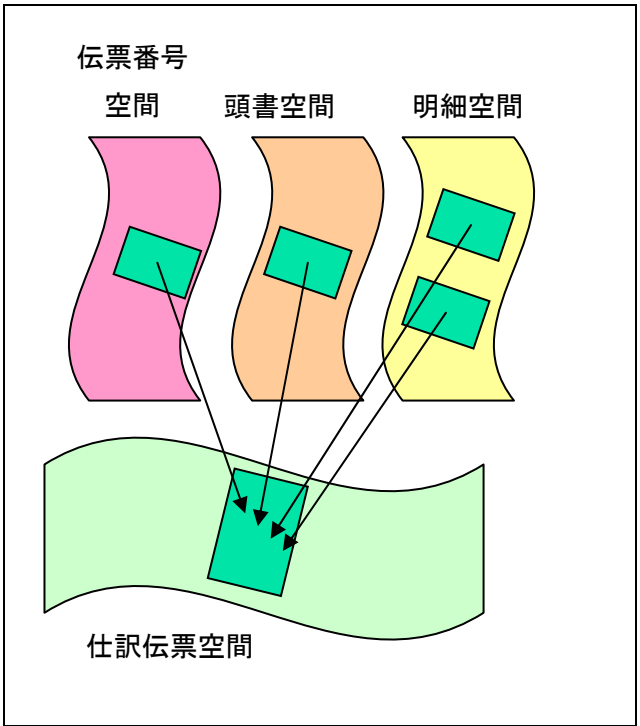

図 2-2 基本的な会計システムの空間

会計システムは、それぞれの企業の活動を財務の面から見たものである。企業活動は、商

取引により顕在化される。商取引が行われると、仕訳伝票が発行される。仕訳伝票は伝票番号、頭書、明細書からなる。これらをファイバー束に写像する。
企業活動から、仕訳伝票に移すまでの作業をファイバー束$\xi = (E, B, F, p)$で表わすと図 2-1 のようになる。ここで、E は全空間、B は底空間 F はファイバーである。

2.2. 集合論レベル

このレベルでは、それぞれの空間の要素を定義する。

仕訳伝票を集めたものを仕訳帳と呼ぶことにする。仕訳帳 J は、仕訳伝票 j_i の集合となっているので、次のように記述することが出来る。

$$J = \{j_1, j_2, j_3, \ldots, j_n\}$$

仕訳伝票 j は、生成更新時間 st、伝票番号 s、頭書 h、明細書 DS で構成されるので、次のように記述できる。

$$j_i = (st_i, s_i, h_i, DS_i) \in J$$

$st \in$ 時間

これとは別に、日付 t, 申請者 a, 備考 r を要素にもつ頭書のリスト H と、金額 da、勘定科目 di で構成される明細のリスト D を別途用意しておく。すなわち、

$$H = \{h_1, h_2, \ldots, h_s\},$$
$$h_j = (t_j, a_j, r_j) \in H,$$
$$t \in \text{Date}, a \in \text{Name}, r \in \text{String},$$
$$D = \{ d_1, d_2, \ldots, d_t \},$$
$$d_j = (da_j, di_j) \in D,$$
$$da \in \text{Money}, di \in \text{勘定科目のリスト}.$$

このようにすると、仕訳伝票 j_i を構成する伝票番号 s_i、頭書 h_i、明細書 DS_i は次のように表すことができる。

$$s_i \in \mathbb{N},$$
$$h_i \in H,$$
$$DS_i \subset D.$$

2.3. 位相空間レベル

このレベルでは位相を導入する。

位相空間は、近いという概念を抽象的に表したものである。連続した空間のときは、*Hausdorff* 空間のような形で、直感的にわかりやすい考え方を導入することができるが、離散的な空間の場合には、位相空間の定義を満たす集合で考えることになる。従って、本来、有していた近いという概念は消失されるが、連続写像などの重要な概念はそのまま保持さ

れる。

ホモトピーレベルで定義された空間は、すべてが離散的である。そのため、ここでは、離散集合での位相空間を次のように定義する。集合 S に導入された位相空間 T は、集合 S の要素のいかなる集まりも位相空間の要素とする。すなわち、集合 S のべき集合をその位相空間 T とする。例えば、仕訳帳 J に対する位相空間 TJ は、

$$TJ = \{\phi, j_1, j_2, j_3, \ldots, j_n, (j_1, j_2), (j_1, j_3), \ldots, (j_{n-1}, j_n), \ldots\ldots, (j_1, j_2, j_3, \ldots, j_n)\}.$$

これからは、集合に導入された位相空間だけで議論するので、集合の記号をそのまま位相空間の記号として用いることとする。例えば、仕訳帳 J に導入された位相空間はそのまま J とする。

ところで、ここまで進んでくると、ファイバー束の性格がもう少し明らかになってくる。底空間 B は、

$$B = \{\phi, j_1, j_2, j_3, \ldots, j_n, (j_1, j_2), (j_1, j_3), \ldots, (j_{n-1}, j_n), \ldots\ldots, (j_1, j_2, j_3, \ldots, j_n)\}$$

で仕訳伝票の指標を表す。ファイバーF は

$$F = S \times H \times D.$$

である。これから、$B \times F$ は $j_i \in B$ に対して、s_i, h_i, DS_i を与えることとなる。これを示したのが図 2-3 である。

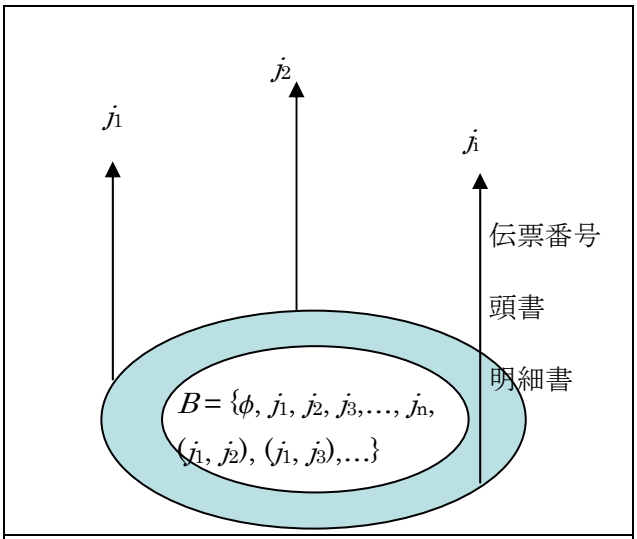

図 2-3　基本的な会計システムのファイバー

2.4. 接着空間レベル

商取引がまだ一つも発生していない状態、例えば会社が設立されたとき、から始めて、次第に商取引が行われるような状態を考えるとする。これは、仕訳伝票がまったくない状態から、仕訳伝票が次第に積み重ねられていく。即ち、仕訳帳が $J = \{\}$から始まって、$J = \{j_1, j_2, j_3, \ldots, j_n\}$へと変わっていく状況を示している。

このとき、$B \times F$ では何が起こっているのであろうか。次のように考える。B は伝票と考

えてよい。商取引が発生していない状態では、何も書かれていない伝票の山と考えてよい。コンピュータの世界で考えると、これは仕訳伝票に与える指標である。具体的に、データベースで考えるのであればプライマリー・キーであり、オブジェクト指向プログラムで考えるのであればクラスである。F は、実際に発生した商取引である。コンピュータの世界で考えると、伝票の中身である。具体的に、データベースで考えるのであればフィールドに入るべきデータであり、オブジェクト指向プログラムではインスタンスを作るためのデータである。ところで、

$$B = J,$$

$$F = S \times H \times D.$$

であるため、商取引がまだ一つも発生していない状況は、空間 J と空間 $N \times H \times D$ とは全く分離している状態にあると考えることが出来る。即ち、$J \sqcup (S \times H \times D)$で表すことが出来る。（少し、片手落ちのようにも考えられるが、常に、$J \cap (S \times H \times D) = \phi$、即ち共通部分を持つことは考えられない状況を扱っているので、これで正しいことになる）これは、$(J \sqcup S) \times (J \sqcup H) \times (J \sqcup D)$と書くことも出来る。

ここで、初めて一つの商取引が成立したとする。このとき、その内容が、仕訳伝票に書かれることになる。これは、次のように表すことが出来る。何も書かれていない仕訳伝票を一つ取り出し、これを j_{i} とする。これに、実際の商取引の内容 $s_{\mathrm{i}}, h_{\mathrm{i}}, DS_{\mathrm{i}}$ が書き込まれる。これは、コンピュータの世界で考えれば、インスタンスの生成であり、データベースへの書き込みである。

この行為を接着空間で表すと、次のようになる。少し、汎用的にするため、同時に複数の商取引が生じたとし、そのために用意した複数の伝票は

$$J_0 = \{j_{\mathrm{i}1}, j_{\mathrm{i}2}, \ldots, j_{\mathrm{in}}\} \subset J.$$

これに、伝票番号、頭書、明細書が書き込まれる。これは、関数 f を用いて記述する。ここでの関数の役割は、記帳という行為である。ただし、記入間違いが生じる場合もあるし、更には、商品の戻しなどによるキャンセルなどもあるので、記入事項が消し去られる可能性も残っている。これは、一般的に考えると、伝票と商取引は一時的にくっついていると考えることが出来る。また、新しい伝票であればどれを用いて書いても良いので、関数はこの性質も持ち合わせている必要がある。このような性質を有しているのが、接着関数である。

$$f : J_0 \to S \times H \times D.$$

まず、明細リスト DS について考えることにする。伝票 j_{ij} には、$DS_{\mathrm{j}} = \{d_{\mathrm{j}1}, d_{\mathrm{j}2}, \ldots, d_{\mathrm{jm}}\}, d_{\mathrm{jk}} \in D$ が書き込まれる。この状況を説明したのが図 2-4 である。

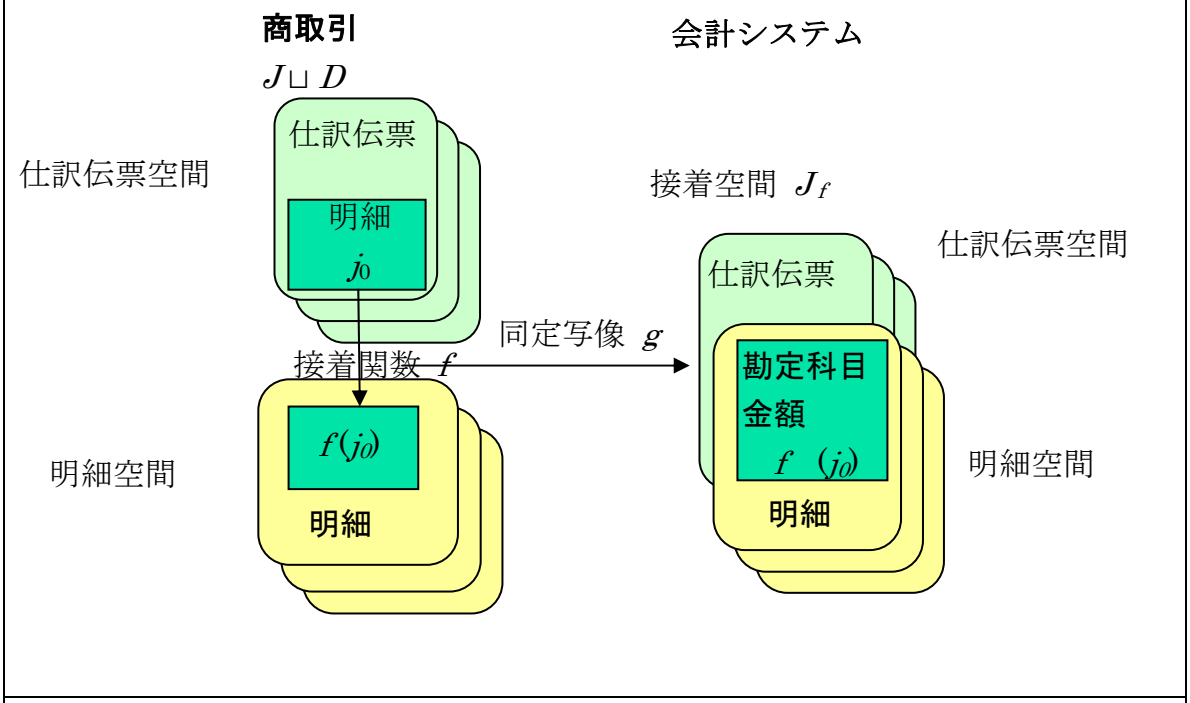

図 2-4　仕訳伝票と明細リストの接着

接着空間 D_f は

$$D_f = D \sqcup J / \sim$$
$$= D \sqcup_f J$$
$$= D \sqcup J / (j_j \sim f(y) \mid j_i \in J, \forall y \in D_0).$$

また、同定(Identification)写像 g は次のように定義できる。

$$g : D \sqcup J \to D_f = D \sqcup J / \sim$$
$$= D \sqcup_f J$$
$$= D \sqcup J / (j_j \sim f(y) \mid j_i \in J, y \in D_0).$$

もちろん、これは商取引が生じたことによる変化を示すものである。

同様に、伝票番号 S と頭書 H の間にも以下のような接着関数が定義できる。

$$H_f = H \sqcup J / \sim$$
$$= H \sqcup_f J$$
$$= H \sqcup J / (j_j \sim f(y) \mid j_i \in J, y \in H_0).$$
$$S_f = S \sqcup J / \sim$$
$$= S \sqcup_f J$$
$$= S \sqcup J / (j_j \sim f(y) \mid j_i \in J, y \in S_0).$$

2.5. セル空間レベル

このレベルでは、セルを利用することで、物理的な構造を構築する。

伝票番号 s_i は 1 つの整数型の変数を持ち、$\mathcal{B}_{si}{}^{1}$ と表現できる。伝票番号の空間 S は伝票番号の互いに素な集合であるので、$\sqcup_i \mathcal{B}_{si}{}^{1}$ によって表される。頭書 h_i 明細 d_i はそれぞれ 2 つと 3 つの変数を持つので、それぞれの空間 H、D は、$\sqcup_i \mathcal{B}_{hi}{}^{3}$、$\sqcup_i \mathcal{B}_{di}{}^{2}$ で表される。仕訳伝票 j_i はこれらの要素の入れ物であり、生成更新時間の変数を持つので、空間 J は $\sqcup_i \mathcal{B}_{i}{}^{1}$ で表される。

接着空間のところで見たように、これらは接着関数 $f_1, f_2, f_3,$ で関連付けられているので、これらの関係は次のように表すことが出来る。

$f_1 : \partial^1\mathcal{B}_{sj}{}^{1} \to \partial^1\mathcal{B}_{j}{}^{1},\ or\ \partial^1\mathcal{B}_{j}{}^{1} \sqcup_{f1} \mathcal{B}_{sj}{}^{1} /\sim.$

$f_2 : \partial^3\mathcal{B}_{hj}{}^{3} \to \partial^1\mathcal{B}_{j}{}^{1},\ or\ \partial^1\mathcal{B}_{j}{}^{1} \sqcup_{f2} \mathcal{B}_{hj}{}^{3} /\sim.$

$f_3 : \sqcup_f \partial^2\mathcal{B}_{djj}{}^{2} \to \partial^1\mathcal{B}_{j}{}^{1},\ or\ \partial^1\mathcal{B}_{j}{}^{1} \sqcup_{f3} (\mathcal{B}_{djj}{}^{2}) /\sim.$

これらを用いて、CW 空間を構成することが出来る。

まず、明細リストが作る空間 *DSi* を考えることにする。*DSi* は開いた球を使い $\sqcup_j \partial^2\mathcal{B}_{djj}{}^{2}$ と表すことができる。明細はその変数として、勘定科目と、金額を持っており、それぞれを明確に区別するため、インデックスも必要となる。これらは *DSi* のスケルトン $X_{\mathrm{detail}}{}^{0}$ として使用される。

$$X_{detail}{}^{0} = \{\, e_{did}{}^{0}{}_{1}, \ldots, e_{did}{}^{0}{}_{k}, e_{item}{}^{0}{}_{1}, \ldots, e_{item}{}^{0}{}_{k}, e_{amount}{}^{0}{}_{1}, \ldots, e_{amount}{}^{0}{}_{k} \,\}$$

スケルトン $X_{detail}{}^{1}$ に関しては、インデックス、勘定科目、金額の中の 2 つが 1 次元のセルを通して接着される。

$$X_{detail}{}^{1} = \{X_{detail}{}^{0}, e_{diditem}{}^{1}{}_{1}, \ldots, e_{diditem}{}^{1}{}_{k}, e_{itemamount}{}^{1}{}_{1}, \ldots, e_{itemamount}{}^{1}{}_{k}, e_{amountdid}{}^{1}{}_{1}, \ldots, e_{amountdid}{}^{1}{}_{k} \mid f_1: \partial e_{diditem}{}^{1}{}_{i} \to e_{did}{}^{0}{}_{i},\ f_2: \partial e_{diditem}{}^{1}{}_{i} \to e_{item}{}^{0}{}_{i},\ f_3: \partial e_{itemamount}{}^{1}{}_{i} \to e_{item}{}^{0}{}_{i},\ f_4: \partial e_{itemamount}{}^{1}{}_{i} \to e_{amount}{}^{0}{}_{i},\ f_5: \partial e_{amountdid}{}^{1}{}_{i} \to e_{amount}{}^{0}{}_{i},\ f_6: \partial e_{amountdid}{}^{1}{}_{i} \to e_{did}{}^{0}{}_{i} \}.$$

スケルトン $X_{detail}{}^{2}$ は以下のように得られる。

$$X_{detail}{}^{2} = \{X_{detail}{}^{1}, e_{detail}{}^{2}{}_{1}, \ldots, e_{detail}{}^{2}{}_{k} \mid f_1: \partial e_{detail}{}^{2}{}_{i} \to e_{diditem}{}^{1}{}_{i},\ f_2: \partial e_{detal}{}^{2}{}_{i} \to e_{itemamount}{}^{1}{}_{i},\ f_3: \partial e_{detail}{}^{2}{}_{i} \to e_{amountdid}{}^{1}{}_{i} \}.$$

同様に頭書リストについて考えると、以下のようにあらわせる。

$$X_{header}{}^{0} = \{\, e_{hid}{}^{0}{}_{1}, \ldots, e_{hid}{}^{0}{}_{k}, e_{date}{}^{0}{}_{1}, \ldots, e_{date}{}^{0}{}_{k}, e_{applicant}{}^{0}{}_{1}, \ldots, e_{applicant}{}^{0}{}_{k}, e_{remarks}{}^{0}{}_{1}, \ldots, e_{remarkst}{}^{0}{}_{k} \,\}$$

$$X_{header}{}^{1} = \{X_{header}{}^{0}, e_{hiddate}{}^{1}{}_{1}, \ldots, e_{hiddate}{}^{1}{}_{k}, e_{dateapplicant}{}^{1}{}_{1}, \ldots, e_{dateapplicant}{}^{1}{}_{k}, e_{applicantremarks}{}^{1}{}_{1}, \ldots, e_{applicantremarks}{}^{1}{}_{k}, e_{remarkshid}{}^{1}{}_{1}, \ldots, e_{remarkshid}{}^{1}{}_{k}, e_{hidapplicant}{}^{1}{}_{1}, \ldots, e_{hidapplicant}{}^{1}{}_{k}, e_{dateremarks}{}^{1}{}_{1}, \ldots, e_{dateremaks}{}^{1}{}_{k} \mid f_1: \partial e_{hiddate}{}^{1}{}_{i} \to e_{hid}{}^{0}{}_{i},\ f_2: \partial e_{hiddate}{}^{1}{}_{i} \to e_{date}{}^{0}{}_{i},\ f_3: \partial e_{dateapplicant}{}^{1}{}_{i} \to e_{date}{}^{0}{}_{i},\ f_4: \partial e_{dateapplicant}{}^{1}{}_{i} \to e_{applicant}{}^{0}{}_{i},\ f_5: \partial e_{applicantremarks}{}^{1}{}_{i} \to e_{applicant}{}^{0}{}_{i},\ f_6: \partial e_{applicantremarks}{}^{1}{}_{i} \to e_{remarks}{}^{0}{}_{i},\ f_7: \partial e_{remarkshid}{}^{1}{}_{i} \to$$

$e_{remarks}{}^{0}{}_{i}$, $f_{8:}\ \partial e_{remarkshid}{}^{1}{}_{i} \rightarrow e_{hid}{}^{0}{}_{i}$, $f_{9:}\ \partial e_{hidapplicant}{}^{1}{}_{i} \rightarrow e_{hid}{}^{0}{}_{i}$, $f_{10:}\ \partial e_{hidapplicant}{}^{1}{}_{i} \rightarrow e_{applicant}{}^{0}{}_{i}$, $f_{11:}\ \partial e_{dateremarks}{}^{1}{}_{i} \rightarrow e_{date}{}^{0}{}_{i}$, $f_{12:}\ \partial e_{dateremarks}{}^{1}{}_{i} \rightarrow e_{remarks}{}^{0}{}_{i}$ }.

$X_{header}{}^{2}$= {$X_{header}{}^{1}$, $e_{hiddateappicant}{}^{2}{}_{1}$, … ,$e_{hiddateapplicant}{}^{2}{}_{k}$, $e_{dateapplicantremarks}{}^{2}{}_{1}$, … ,$e_{dateapplicanremarkst}{}^{2}{}_{k,}$ $e_{applicantremarkshid}{}^{2}{}_{1}$, … ,$e_{applicantremarkshid}{}^{2}{}_{k,}$ $e_{remarkshidapplicant}{}^{2}{}_{1}$, … ,$e_{remarkshidapplicant}{}^{2}{}_{k,}$ | $f_{1:}\ \partial e_{\ hiddateappicant}{}^{2}{}_{i} \rightarrow e_{hiddate}{}^{1}{}_{i}$, $f_{2:}\ \partial e_{\ hiddateappicant}{}^{2}{}_{i} \rightarrow e_{dateapplicant}{}^{1}{}_{i}$, $f_{3:}\ \partial e_{\ hiddateappicant}{}^{2}{}_{i} \rightarrow e_{hidapplicant}{}^{1}{}_{i}$, $f_{4:}\ \partial e_{dateapplicantremarks}{}^{2}{}_{i} \rightarrow e_{dateapplicant}{}^{1}{}_{i}$, $f_{5:}\ \partial e_{dateapplicantremarks}{}^{2}{}_{i} \rightarrow e_{applicantremarks}{}^{1}{}_{i}$, $f_{6:}\ \partial e_{dateapplicantremarks}{}^{2}{}_{i} \rightarrow e_{dateremarks}{}^{1}{}_{i}$, $f_{7:}\ \partial e_{\ applicantremarkshid}{}^{2}{}_{i} \rightarrow e_{rapplicantremarks}{}^{1}{}_{i}$, $f_{8:}\ \partial e_{\ applicantremarkshid}{}^{2}{}_{i} \rightarrow e_{remarkshid}{}^{1}{}_{i}$, $f_{9:}\ \partial e_{\ applicantremarkshid}{}^{2}{}_{i} \rightarrow e_{remarkshid}{}^{1}{}_{i}$, $f_{10:}\ \partial e_{remarkshidapplicant}{}^{2}{}_{i} \rightarrow e_{remarkshid}{}^{1}{}_{i}$, $f_{11:}\ \partial e_{remarkshidapplicant}{}^{2}{}_{i} \rightarrow e_{hidapplicant}{}^{1}{}_{i}$, $f_{12:}\ \partial e_{remarkshidapplicant}{}^{2}{}_{i} \rightarrow e_{applicantremarks}{}^{0}{}_{i}$ }.

$X_{header}{}^{3}$= {$X_{header}{}^{2}$, $e_{header}{}^{3}{}_{1}$, … ,$e_{header}{}^{3}{}_{k}$ | $f_{1:}\ \partial e_{\ header}{}^{3}{}_{i} \rightarrow e_{hiddateaplicant}{}^{2}{}_{i}$, $f_{2:}\ \partial e_{\ header}{}^{3}{}_{i} \rightarrow e_{dateapplicantremarks}{}^{2}{}_{i}$, $f_{3:}\ \partial e_{\ header}{}^{3}{}_{i} \rightarrow e_{dateapplicantremarks}{}^{2}{}_{i}$, $f_{4:}\ \partial e_{header}{}^{3}{}_{i} \rightarrow e_{remarkshidapkicant}{}^{1}{}_{i}$ }.

伝票番号のリストに対しては、

$X_{number}{}^{0}$ = { $e_{nid}{}^{0}{}_{1}$, … , $e_{nid}{}^{0}{}_{k,}$ $e_{serial}{}^{0}{}_{1}$, … , $e_{serial}{}^{0}{}_{k}$ }

$X_{number}{}^{1}$ = { $X_{number}{}^{0}$, $e_{number}{}^{1}{}_{1}$, … , $e_{number}{}^{1}{}_{k}$ | $f_{1:}\ \partial e_{\ number}{}^{1}{}_{i} \rightarrow e_{nid}{}^{0}{}_{i}$, $f_{2:}\ \partial e_{\ number}{}^{1}{}_{i} \rightarrow e_{serial}{}^{0}{}_{i}$ }

最後に、仕訳帳について考える。

$X_{\text{journal}}{}^{0}$ = {$e_{jid}{}^{0}{}_{1}$, … , $e_{jid}{}^{0}{}_{k,}$ $e_{time}{}^{0}{}_{1}$, … , $e_{time}{}^{0}{}_{k}$ }

$X_{journal}{}^{1}$ = { $X_{journal}{}^{0}$, $e_{journal}{}^{1}{}_{1}$, … , $e_{journal}{}^{1}{}_{k}$ | $f_{1:}\ \partial e_{journal}{}^{1}{}_{i} \rightarrow e_{jid}{}^{0}{}_{i}$, $f_{2:}\ \partial e_{journal}{}^{1}{}_{i} \rightarrow e_{time}{}^{0}{}_{i}$ }

ここに伝票番号、頭書、明細を接着する。

$X_{journal}{}^{2}$ = { $X_{journal}{}^{1}$, $e_{journalnumber}{}^{2}{}_{1}$, … , $e_{journalnumber}{}^{2}{}_{k}$ | $f_{1:}\ \partial e_{journalnumber}{}^{2}{}_{i} \rightarrow e_{journal}{}^{1}{}_{i}$, $f_{2:}\ \partial e_{journalnumber}{}^{2}{}_{i} \rightarrow e_{number}{}^{1}{}_{i}$ }

$X_{journal}{}^{5}$ = { $X_{journal}{}^{2}$, $e_{journalheader}{}^{5}{}_{1}$, … , $e_{journalheader}{}^{5}{}_{k}$ | $f_{1:}\ \partial^{3} e_{journalheader}{}^{5}{}_{i} \rightarrow e_{journalnumber}{}^{2}{}_{i}$, $f_{2:}\ \partial^{2} e_{journalheader}{}^{5}{}_{i} \rightarrow e_{header}{}^{3}{}_{i}$ }

$X_{journal}{}^{7}$ = { $X_{journal}{}^{5}$, $e_{journal}{}^{7}{}_{1}$, … , $e_{journal}{}^{7}{}_{k}$ | $f_{1:}\ \partial^{2} e_{journal}{}^{7}{}_{i} \rightarrow e_{journalheader}{}^{5}{}_{i}$, $f_{2:}\ \partial^{5} e_{journal}{}^{7}{}_{i} \rightarrow e_{detail}{}^{2}{}_{i}$ }

これを表したものが図 2-5 である。

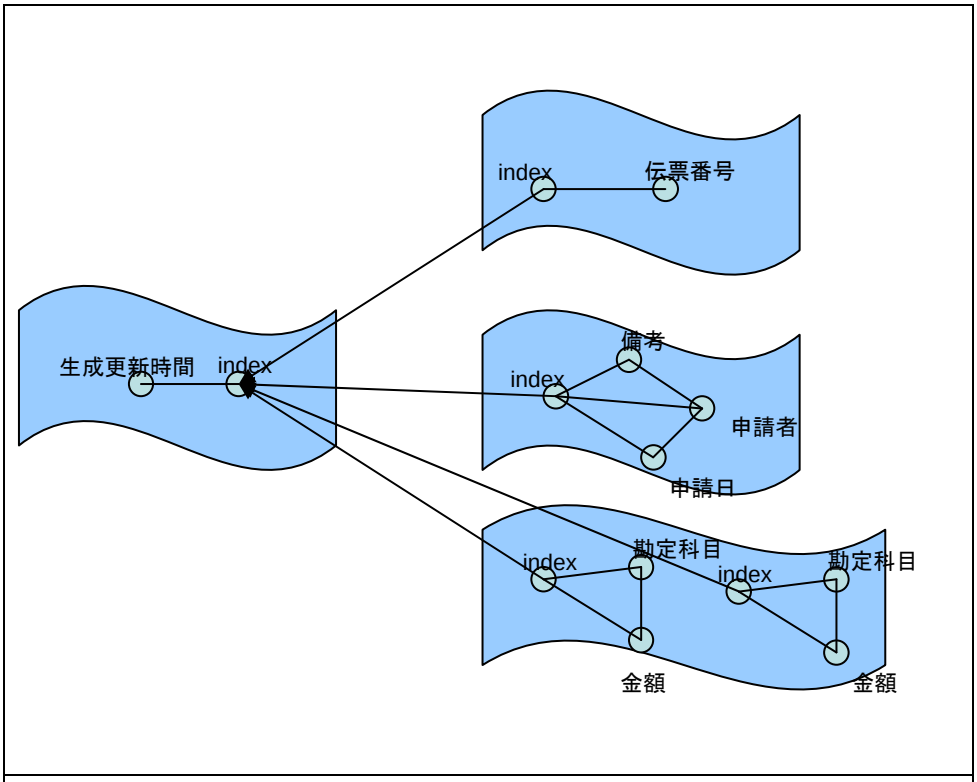

図 2-5　基本的な会計システムのセル空間

セル空間レベルでは、接着空間レベルまでに定められた不変量を維持するとともに、次元に対する不変量を定めている。

2.6. 表現レベル

ここまでに決まった空間の関係をクラス図で表すことが出来る。図 2-6 はそれを示したものである。

ファイバー束での底空間としての仕訳帳の空間 *J* は仕訳というクラスで表されている。また、ファイバーを構成する空間に対しては、伝票番号の空間 *S* は伝票番号というクラスで、頭書のリストの空間 *H* は頭書というクラスで、明細のリストの空間 *D* は明細というクラスで実現されている。また、接着関数は、クラス間を結ぶ関連(Association links)で表されている。これは、接着空間レベルで定められた保存量を維持しているものである。伝票番号、頭書に対する多重度は 1 であるが、明細への多重度は *n* となっていて、複数許される。これは、セル空間レベルで決められた次元に対する不変量を保持している。

Web システムのプログラムは Web の部分と EJB の部分からなる。EJB は Session Beans と Entity Beans と呼ばれる性格の異なる二種類のクラスで構成されている。これまで述べてきたクラスは Entity Beans と呼ばれるもので、データベースへのアクセスを行う。これに対し

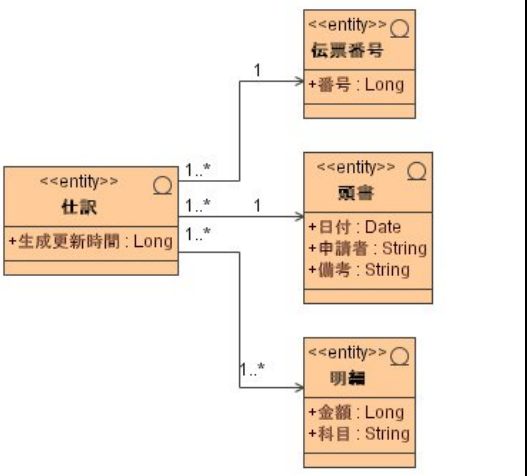

図 2-6　基本的な会計システムのクラス図

て、Session Beans は Web システムのほうから要求を受けて、あるいは、Web Service からの要求を受けて Entity Beans との間で必要な処理を行い、Web あるいは Web Service が要求しているものを返す。この会計システムでは、伝票に対する生成、削除、更新、検索の機能が必要となる。つまり、それらの機能を定義する Session Beans のクラスが必要となる。クラスの種類はステレオタイプで行う。また、AndroMDA ではクラスに実装されるメソッドや変数の性質を定める必要があるときは、タグ付き値で行う。
それらのクラスを足すとクラス図は図 2-8 のようになる。
　追加したクラスは以下の通りである。

- SlipController
  Web 側の操作を定義するクラス。
- SlipService
  EJB 側の操作を定義するクラス。
  ステレオタイプ<<Service>>を付与する。
- Sliplist、SearchList、DetailList
  データ転送用のクラス。
  ステレオタイプ<<ValueObject>>が付与する。
- SlipSessionState
  ページをまたいで同じデータを保持するクラス。
  ステレオタイプ<<FrontEndSessionObject>>

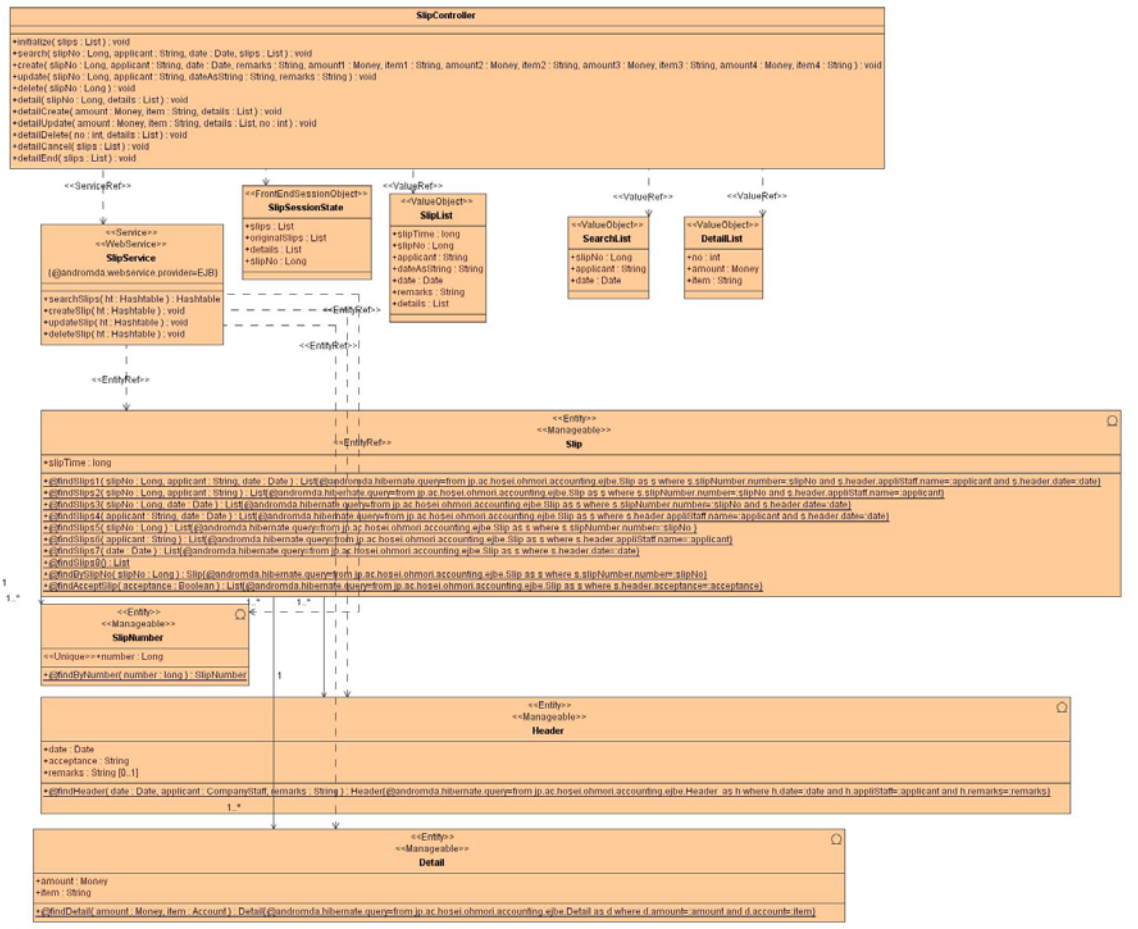

図 2-8　基本的な会計システムのクラス図

一般に Web システムは、有限状態機械(A Finite State machine)と考えることができる。ある画面で入力を与えると次の画面に移る。一つ一つの画面に対して、それに対応する状態があると考え、画面はその状態からの出力と考える。また、書き込まれた内容や押されたボタンは入力と考える。

一般に、有限状態機械を表現するためのムーアマシンは

$$Q = (S_{t+1}, S_t, \boldsymbol{I}, \boldsymbol{O}, f, g);$$

$$S_{t+1} = f(S_t, \boldsymbol{I});$$

$$\boldsymbol{O} = g(S_t);$$

で表される。ここでは仕訳伝票のみからなるシステムを考えているので、このシステムでの要求に合わせて設計すると、図 2-7 のような状態遷移図を得ることができる。

また、この状態遷移に基づいてアクティビティ図を書くと図 2-9 のようになる。状態 S1、S2 は web ページで表示されるため、ステレオタイプ<<frontEndView>>を付与している。

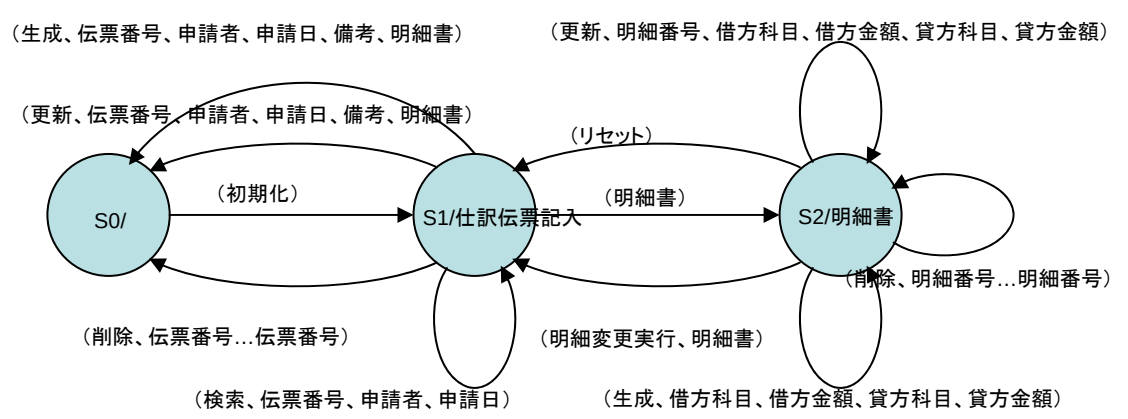

図 2-7　基本的な会計システムの状態遷移図

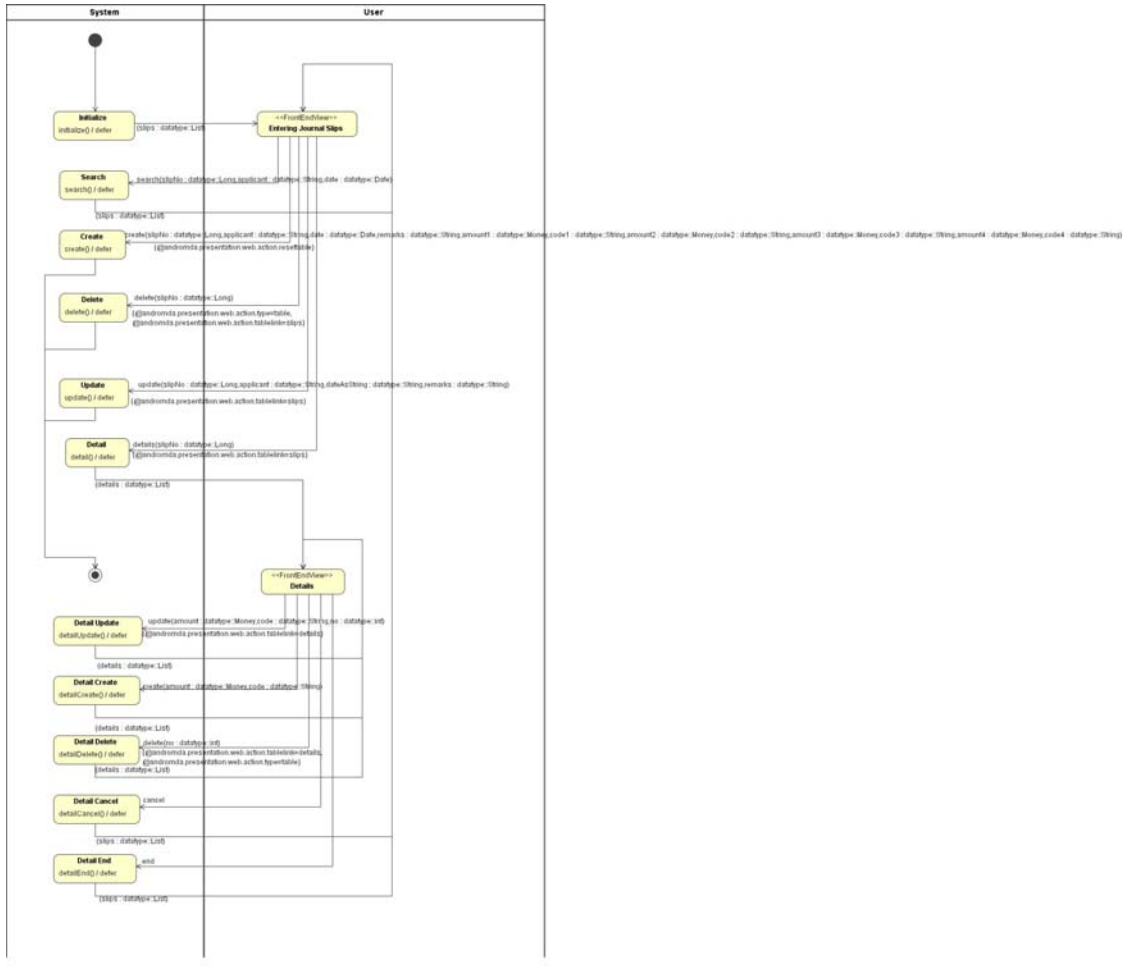

図 2-9　基本的な会計システムのアクティビティ図

状態遷移図での入力を受けての次の状態への遷移 f は、接着関数のところで述べた空間の間での接着である。そこで、各遷移での接着、分離の関係を示すと表 2-1 のようになる。

表 2-1 各遷移での接着、分離の関係

	分離	接着
生成		仕訳空間に、伝票番号、頭書、明細の空間を接着。名称はそれぞれ、伝票番号、頭書、明細書とする。
更新	仕訳空間から、更新された伝票番号、頭書、明細の空間のみを削除	仕訳空間に、更新された伝票番号、頭書、明細の空間のみを接着
削除	仕訳空間から、伝票番号、頭書、明細の空間を削除	
検索		

また、AndroMDA においては Web の部分の表現にユースケース図を用いる。図 2-10 にユースケース図を示す。

図 2-10　基本的な会計システムのユースケース図

ステレオタイプ FrontEndApplication はアプリケーションの開始点を意味する。

ステレオタイプ FrontEndUseCase は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。

AndroMDA においては 1 つのユースケースに対して 1 つのアクティビティ図を指定する仕様になっており、ここでは前に記述した仕訳伝票の操作によるアクティビティ図に対応させている。

2.7. 可視レベル

このレベルは最も具体的なレベルであり、システムはプログラムコードによって表される。

表現レベルで書かれた UML を基に AndroMDA を利用して、プログラムを自動生成する。AndroMDA によって生成されるシステムのフォルダ構成を以下の表に示す。

表 2-2 AnroMDA のフォルダ構成

project-root	
app	最終的な成果物を格納
common	Web、EJB で共有する生成物を格納
core	EJB 側に関する生成物を格納
src	編集対象となるプログラムコードを格納
target	自動生成されるプログラムコードを格納
mda	UML モデル、変換定義などを格納
web service	Web Service に関する生成物を格納
web	Web に関する生成物を格納
src	編集対象となるプログラムコードを格納
target	自動生成されるコードを格納

AndroMDA では、ビジネスロジック部分以外が、自動生成される。ビジネスロジックを記述する Session Beans のファイルは Web 側が「web」、EJB 側が「core」フォルダの「src」フォルダに格納されている。ここでは、仕訳伝票の生成、削除、更新、検索に関する実装を行った。ここで、記述するビジネスロジックは SlipContoroller と SlipService 内に定義した、メソッドの実装である。

ホモトピーレベルで定められた仕訳伝票 *J*、伝票番号リスト *N*、頭書リスト *H*、明細リスト *D* が、データベースのテーブルとして定められ、プログラムではクラスとして表されている。これらのファイルは「web」「core」内の「target」フォルダに自動生成されている。また、システムの使用開始以前の状態では、仕訳伝票と、伝票番号リスト、頭書リスト、明細リストの間は分離している。図 2-11 は使用開始以前の仕訳伝票データベースを JBOSS に付属している DataBaseManager を用いて表示したものである。ここでは伝票番号リストと頭書リストへの接着がそれぞれ SLIP_NUMBER_FK、HEADER_FK、で用意されているが、これらが空のことから、分離していることを見ることができる。また、明細リストとの間の接着は多対多のため、それらの接着を表すテーブルが別途用意されている。

クラスとテーブルの間の写像は、このプログラムの部分には現れないが、自動的に生成されたプログラムの部分で、オブジェクト指向－リレーショナルデータベース写像により実現されている。集合レベルで定められた集合の要素は、データベースのレベルではレコードとして、クラスではインスタンスとして実現されている。

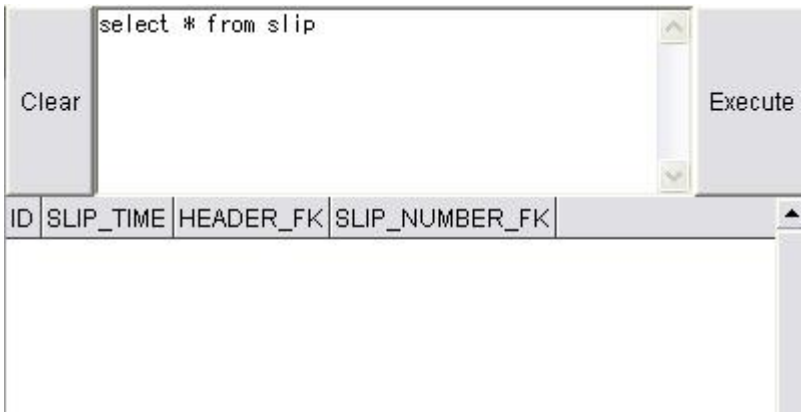

図 2-11　使用開始以前のデータベース

接着空間レベルでの仕訳伝票と、伝票番号リスト、頭書リスト、明細リストの間での接着は、プログラムでは、一対一の関係にある場合には set 関数で、一対多の関係である場合には add 関数で実現される。

例えば、一対一の関係にある仕訳伝票(s)と頭書リスト(h)の場合には、

s.setHeader(h);

のようにプログラミングでき、一対多の関係である仕訳伝票と明細リスト(d)の間では、

s.getDetails().add(d);

と、プログラムを書くことができる。

これは、データベースでは、一方のレコードの指定されたフィールドに他方のレコードの主キーの書き込みを引き起こす。

表現レベルでの UML によるモデリングは、可視レベルではビジネスロジックの部分を除いて自動生成され、テスト、検証の必要性はまったくない。ビジネスロジックの部分のみ、プログラミングする必要があり、この部分に対してのみテスト、検証の必要がある。

「生成」に対する Web 側のプログラムは次のようになっている。

```
public final void create(ActionMapping mapping, jp.ac.hosei.ohmori.accounting.web.CreateForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception
    {
        SlipList sl = new SlipList();
        sl.setSlipNo(form.getSlipNo());
        sl.setApplicant(form.getApplicant());
        Calendar cal = Calendar.getInstance();
        DateFormat f1 = new java.text.SimpleDateFormat("yyyy.MM.dd");
        cal.setTimeInMillis(f1.parse(form.getDate()).getTime());
        sl.setDate(cal);
```

```
        sl.setRemarks(form.getRemarks());
        DetailList dl = new DetailList();
        dl.setDebterAmount(form.getDebterAmount());
        dl.setDebterItem(form.getDebterItem());
        dl.setCreditorAmount(form.getCreditorAmount());
        dl.setCreditorItem(form.getCreditorItem());
        List a = new ArrayList();
        a.add(dl);
        sl.setDetails(a);
        Hashtable ht = new Hashtable();
        ht.put("slip",sl);
        ht.put("details",a);
        SlipService ss = this.getSlipService();
        ss.createSlip(ht);
    }
```

また、EJB 側では次のようなプログラムが実行される。

```
    protected void handleCreateSlip(java.util.Hashtable ht)
        throws java.lang.Exception
    {
        createOrUpdate(ht, true);
    }
    private void createOrUpdate(java.util.Hashtable ht, boolean create)
    {
        SlipList sl = (SlipList)ht.get("slip");
        List a = (List)ht.get("details");
        Slip s = new SlipImpl();
        if(create) {
            SlipNumber sn = new SlipNumberImpl();
            sn.setNumber(sl.getSlipNo());
            try {
                getSlipNumberDao().create(sn);
            } catch ( Exception e) {

        System.err.println("jp.ac.hosei.ohmori.accounting.ejbs.SlipService.handleCreateSlip(java.util.Hashtable
```

```
ht) This slip number has been already used!");
                }
                s.setSlipNumber(sn);                                    //Attaching map
            }else{
                s = getSlipDao().findBySlipNo(sl.getSlipNo());
                if(s.getSlipTime() != sl.getSlipTime()) {
                        System.err.println("It has been already changed");
                        return;
                }
                s.getDetails().clear();
            }
            Calendar cal = Calendar.getInstance();
            s.setSlipTime(cal.getTimeInMillis());
            Header h = new HeaderImpl();
            h.setApplicant(sl.getApplicant());
            h.setDate(new Timestamp(sl.getDate().getTimeInMillis()));
            h.setRemarks(sl.getRemarks());
            if(getHeaderDao().findHeader(h.getDate(),h.getApplicant(),h.getRemarks())!=null) {
                h = getHeaderDao().findHeader(h.getDate(),h.getApplicant(),h.getRemarks());
            } else {
                getHeaderDao().create(h);
            }
            s.setHeader(h);                                         //Attaching map
            Iterator it = a.iterator();
            while(it.hasNext()) {
                DetailList dl = (DetailList)it.next();
                Detail d = new DetailImpl();
                d.setDebterAmount(dl.getDebterAmount());
                d.setDebterItem(dl.getDebterItem());
                d.setCreditorAmount(dl.getCreditorAmount());
                d.setCreditorItem(dl.getCreditorItem());

if(getDetailDao().findDetail(d.getDebterAmount(),d.getDebterItem(),d.getCreditorAmount(),d.getCreditorItem())!=n
ull) {
                        d =
getDetailDao().findDetail(d.getDebterAmount(),d.getDebterItem(),d.getCreditorAmount(),d.getCreditorItem());
```

```
            } else {
                getDetailDao().create(d);
            }
            s.getDetails().add(d);                          //Attaching map
        }
        getSlipDao().create(s);
    }
```

実際に出来上がったシステムは図 2-12、図 2-13 である。ステレオタイプ<<frontEndView>>を付与した状態において、web ページが生成されていることが見ることができる。

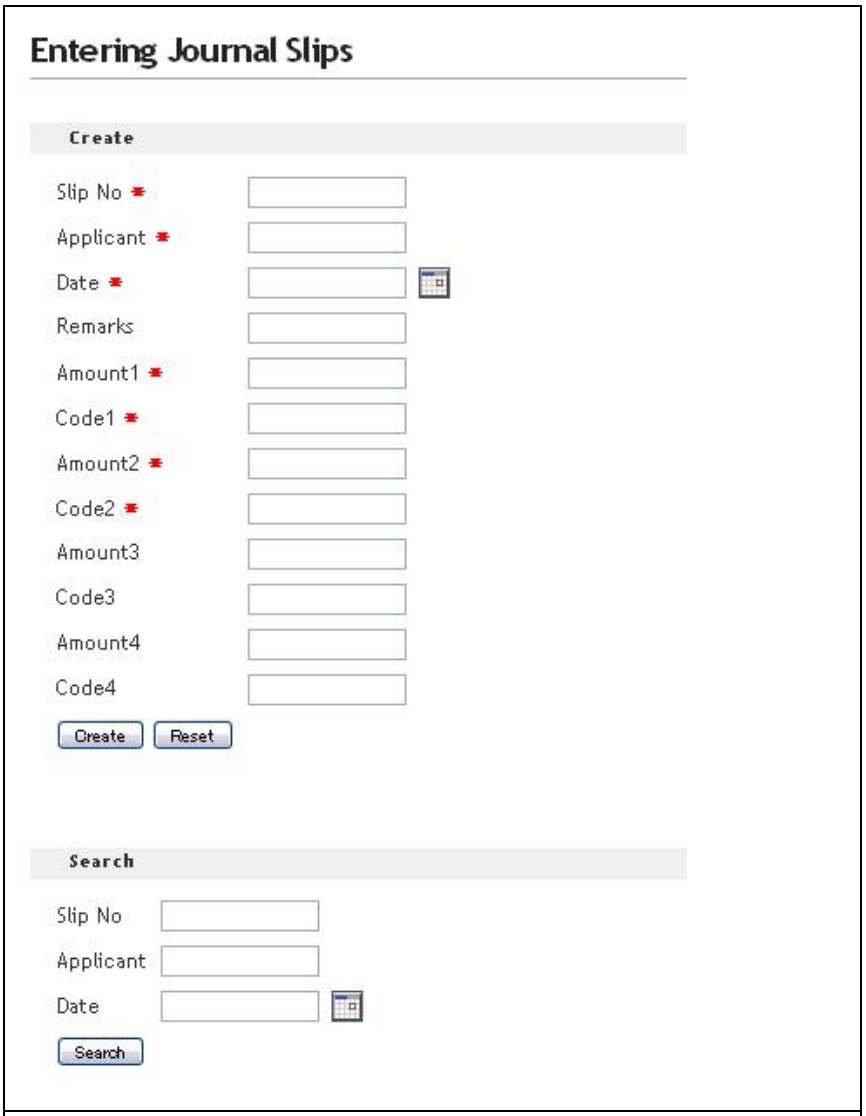

図 2-12　基本的な会計システム

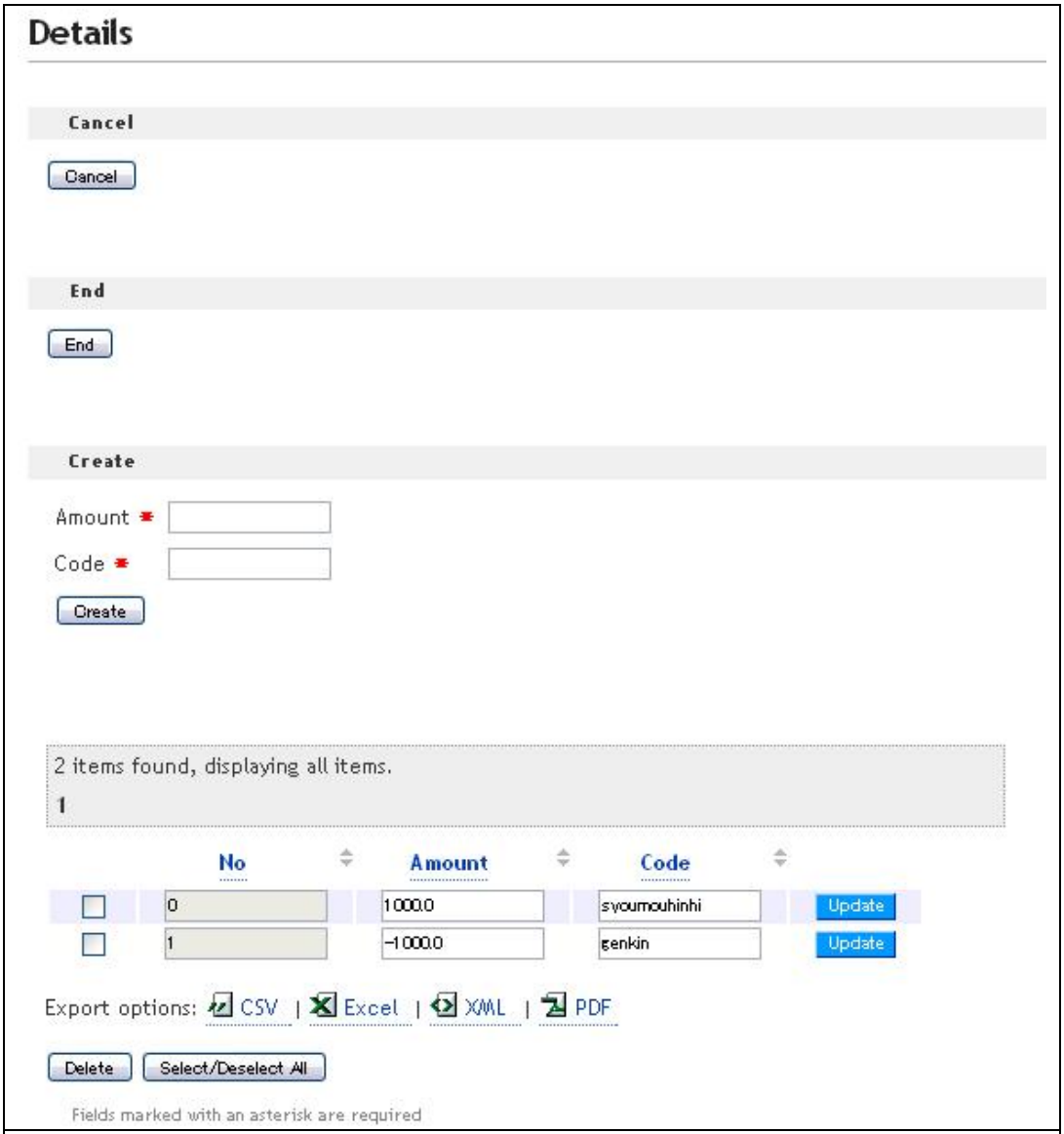

図 2-13　基本的な会計システム

3. 発展させた会計システム

ここまでは、仕訳帳というきわめて基本的な会計システムについて開発を行ってきた。ここでは元帳を利用できるようにし、また、仕訳伝票の承認を行える会計システムを、基本的な会計システムからの概念的発展と捉え、それを、最上位のホモトピーレベルのところで Homotopy Lifting Property (HLP)を用いて概念的発展が持つ不変量を定義し、階層構造を下りながら、モジュールの再利用を行うとともに、モジュールを線形に追加していく。

3.1. ホモトピーレベル

3.1.1. 元帳を利用できる会計システム

会計システムには、会計科目が用意されており、通常は、勘定科目ごとに、元帳(ledger)を作成する。

そこで、HLP を用いて、仕訳帳による簡単な会計システムから、元帳を参照することのできる会計システムへの概念上の進歩を考えることにする。

基本的な会計システムでは、底空間は仕訳伝票の枠組みであった。概念上の進歩を加えるときも底空間は変えない。これは会計システムでの大切な不変量である。しかし、次のようにしたい。円の周りを一周ではなく二周すると元に戻る。最初の一周では、これまでと同じで、仕訳伝票が見えるようにする。次の一周では、元帳が見えるようにする。二周で元に戻るものとしてメビウスの帯があるが、図 3-1 のように、この上にファイバーを立てることにする。即ち、次の周に対しては、仕訳伝票と元帳のファイバーをたて、仕訳伝票の明細書から元帳への記帳が行えるようにする。元帳への記帳の際には、それぞれ勘定科目の合計額、とそれぞれの明細を記載するため、元帳明細の空間も準備する。

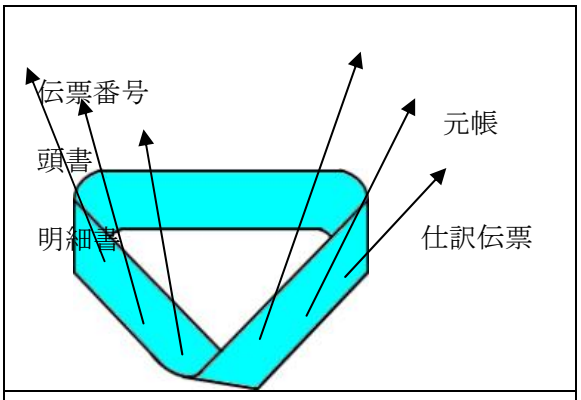

図 3-1　メビウスの帯

HLP は不変量を保存しての動的な変化を示す。図 3-2 に示すように、基本的な会計システムを開始点 I_0 とし、次の変化 I_1 を元帳が追加された会計システムとすると、HLP となる。L は元帳が作る空間、LD は元帳明細が作る空間とすると、X は $B\times(S\times H\times D\times L\times LD)$,である。また、$X\times I_0 = B\times(S\times H\times D)$. メビウスの帯で一周目を X^+、二周目を X^-とすると、

$$X\times I_1 = (X^+ \cup X^-)\times I_1 = (X^+\times I_1) \cup (X^-\times I_1),$$

$$X^+\times I_1 = B\times(S\times H\times D),$$

$$X^-\times I_1 = B\times(D\times L\times LD).$$

これより、$X\times I_0$ から $X\times I_1$ へは離散集合での連続的な変化である。

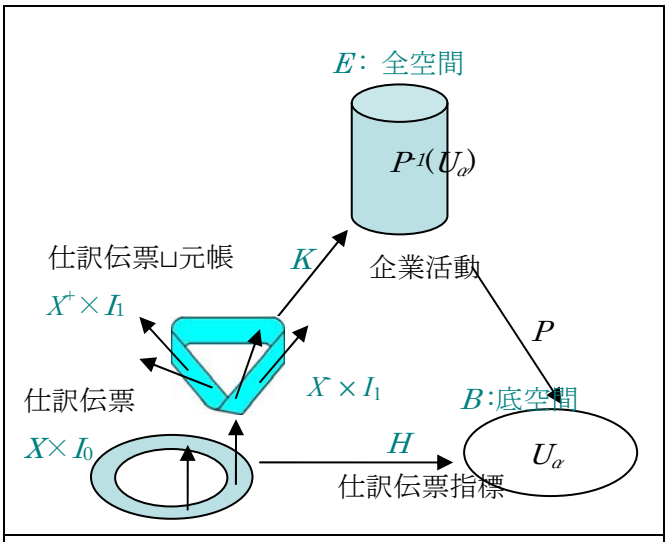

図 3-2　発展させた会計システム

元帳追加に伴い、明細と元帳の両方に勘定科目が現れている。これは両方を同じに保たなければならない。勘定科目のリストの空間を別途用意することが考えられる。また、申請者も通常は社員であり、社員のリストの空間を用いたほうがよいことがわかる。そこで、勘定リストの空間を A とし、社員リストの空間を E とすると、

X は $B \times (N \times H \times D \times L \times LD)$

$$X \times I_0 = B \times (S \times H \times D \times A \times E),$$

$$X \times I_1 = (X^+ \times X^-) \times I_1 = (X^+ \times I_1) \times (X^- \times I_1),$$

$$X^+ \times I_1 = B \times (S \times H \times D \times A \times E),$$

$$X^- \times I_1 = B \times (D \times L \times DL \times A).$$

3.1.2. 仕訳伝票の承認を行える会計システム

通常、業務では申請書類だけでは済まされることはない。申請された書類を承認することが必要になる。よって、これまでの仕訳伝票には承認欄も必要になる。

通常の会計システムからの概念の拡張と捉えると、やはり、HLP で記述することが出来る。今度は円の周りを３周すると元に戻るものを考える。メビウスの帯は紙をねじってくっつけたようなものだったが、ここでは三角柱をねじってくっつけたものを考える。１周目で仕訳伝票の申請を、２周目でその承認を、３周目で元帳の参照をして、元に戻るということにすると、図 3-3 のようになる。さらに、$X \times I_2$ は次のようになる。

$$X \times I_2 = B \times (S \times H \times D \times A \times E) \cup B \times (S \times H \times D \times A \times E) \cup B \times (D \times L \times DL \times A).$$

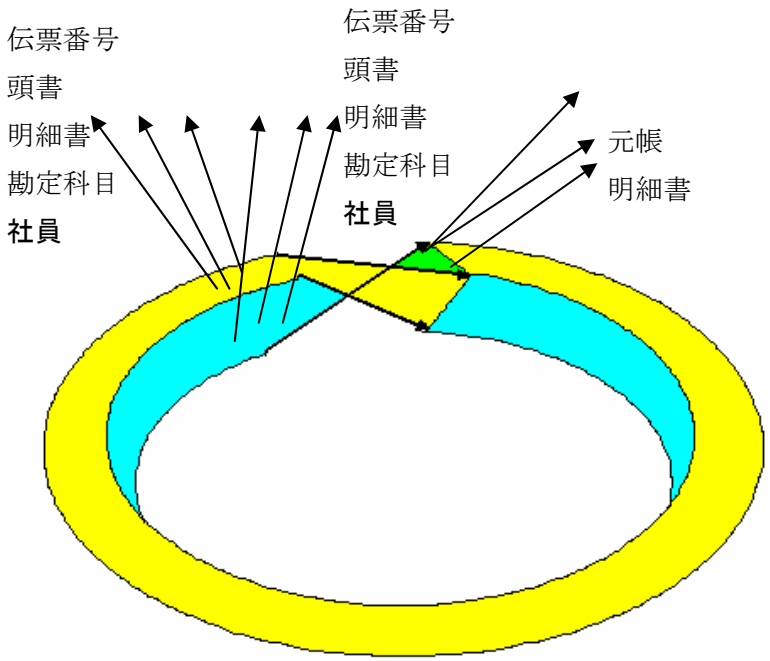

図 3-3 承認機能の追加

3.1.3. ログイン機能の追加

この会計システムは、申請者、承認者、会計士、管理者、の利用するユーザ毎に機能が分けられている。トップページから、それぞれの権限を持ったユーザがログインすることで、各々のユーザにあったページへ遷移する必要がある。ユーザのログインにおいては元帳追加の際に分離しておいた、社員空間を見ればよい。

今度は、円の周りを４周すると元に戻るものを考える。１周目でユーザのログイン、２周目で仕訳伝票の申請を、３周目でその承認を、４周目で元帳を参照して、元に戻るということにする。$X \times I_3$ は次のようになる。

$$X \times I_3 = B \times (E) \cup B \times (S \times H \times D \times A \times E) \cup B \times (\mathrm{S} \times H \times D \times A \times E) \cup B \times (D \times L \times DL \times A) .$$

3.2. 集合論レベル

3.2.1. 元帳を利用できる会計システム

勘定科目リストの空間と、社員リストの空間を新しく用意したので、定義する。

勘定科目は木構造をなしており、一つの勘定科目には、上下に勘定科目が存在する場合があり、上に存在している勘定科目を親の科目、下に存在している勘定科目を子の科目という。親の勘定科目はあっても一つである。子の勘定科目は複数個存在する場合がある。簡単な例では、資産の下には、現金や、預金などの勘定科目があり、預金の下には当座預金や普通預金などの勘定科目がある。従って、勘定科目は次のように表せる。

$a = (an, ac, ap) \in A,$

ここで *an* は勘定科目名、*ac* は勘定科目コード、*ap* は親勘定科目$\in A$ である。

同様に、社員は次のように表せる。

$e = (en, ec, ed, ep, epw, eco) \in E,$

ここで *en* は名前、*ec* は社員コード、*ed* は所属部門、*ep* は役職である。システムに対して、利用する社員ごとの振る舞いが異なるので、*epw はログイン用パスワード*、*eco はシステム権限*も加えた。

なお、社員のリストと勘定科目のリストは、通常は、マスターファイルと呼ばれ、会計システム以外の企業の基幹システムでも共通して用いられる。

この変更に伴い、頭書 h_i と明細 d_i も以下のように再定義される。

$h_i = (t_i, e_i, r_i) \in H,$

$t_i \in \text{Date}, e_i \in E, r_i \in \text{String}$

$d_i = (da_i, a_i) \in H,$

$da_i \in \text{Money}, a_i \in A$

元帳 *L* は、勘定科目ごとの元帳 $L_i \in L$ のリストである。L_i は勘定科目 a_i、合計金額 t_i、未処理伝票のリスト JY_i、処理済伝票のリスト JA_i、元帳明 LDS_i で構成される。以下のように表せる。

$L_i = \{a_i, t_i, JY_i, JA_i, LDS_i\}$

$a_i \in A, t_i \in \text{Money}, JY_i \subset J, JA_i \subset J, LDS_i \subset LD$

元帳明細 *LD* は伝票番号 *s* と金額 *da* で構成され、以下のように表せる。

$LD_i = \{s_i, da_i\}$

$s_i \in \mathbb{N}, da_i \in \text{Money}$

3.1.2. 仕訳伝票の承認を行える会計システム

承認機能の追加によって、頭書には承認日付 *at*、承認者 *an*、承認認否 *ac* を加えたので、以下のように表せる。

$H = \{h_1, h_2, \ldots, h_s\},$

$H_i = (t_i, eap_i, at_i, eac_i, ac_i, r_i) \in H,$

$t, at \in$ Date, $eap, eac \in$ E, $ac \in$ Boolean, $r \in$ String

3.3. 位相空間レベル

底空間 B は、

$$B = \{\phi, j_1, j_2, j_3, \ldots, j_n, (j_1, j_2), (j_1, j_3), \ldots, (j_{n-1}, j_n), \ldots\ldots, (j_1, j_2, j_3, \ldots, j_n)\}$$

で仕訳伝票の指標を表し、これまでと変わらない。

元帳追加におけるファイバーF は

$$F = S \times H \times D \times DL \times A \times E \times L.$$

である。

また、承認機能追加におけるファイバーF は

$$F = S \times H \times D \times A \times E$$

で表せる。

ログイン機能追加におけるファイバーF は

$$F = E$$

で表せる。

3.4. 接着空間レベル

基本的な会計システムに元帳を追加したときに、仕訳伝票と元帳の間で動的な結びつきが発生する。即ち、接着関数 f によって、次のようになる。

$$J_f = J \sqcup L / \sim$$
$$= J \sqcup_f L$$
$$= J \sqcup L / (l_j \sim f(y) \mid \exists l_j \in L, \forall y \in J_0).$$

元帳と元帳明細の間にも接着関数が定義される。

$$DL_f = DL \sqcup L / \sim$$
$$= DL \sqcup_f L$$
$$= DL \sqcup L / (l_j \sim f(y) \mid \exists l_j \in L, \forall y \in DL_0).$$

勘定科目と社員の間を分離したために、ここにも新たな接着関数が必要となる。

$$A_f = A \sqcup D / \sim$$
$$= A \sqcup_f D$$
$$= A \sqcup D/ (d_j \sim f(y) \mid \exists d_j \in D, \forall y \in A_0).$$

$$E_f = E \sqcup H / \sim$$
$$= E \sqcup_f H$$

$= E \sqcup \mathrm{H}/ (h_\mathrm{j} \sim f(y) \mid \exists h_\mathrm{j} \in H, \forall y \in E_0)$.

また、勘定科目は親を持つので、自分自身に接着する必要がある。

$A_\mathrm{f} = A \sqcup A / \sim$

$= A \sqcup_\mathrm{f} A$

$= A \sqcup A/ (a_\mathrm{j} \sim f(y) \mid \exists a_\mathrm{j} \in A, \forall y \in A_0)$.

3.5. セル空間レベル

3.5.1. 元帳を利用できる会計システム

勘定科目空間 A_i、社員空間 E_i、元帳空間 L_i、元帳明細空間 LD_i は、それぞれ、$\sqcup_i B_{ai}^{\ 2}$、$\sqcup_i B_{ei}^{\ 6}$、$\sqcup_i B_{li}^{\ 1}$、$\sqcup_i B_{ldi}^{\ 2}$ で表される。

接着空間のところで見たように、これらは接着関数で関連付けられており、次のように表すことが出来る。

$f_1 : \partial^1 \mathcal{B}_{sj}^{\ 1} \to \partial^1 \mathcal{B}_j^{\ 1}$, or $\partial^1 \mathcal{B}_j^{\ 1} \sqcup_{f1} \mathcal{B}_{sj}^{\ 1} / \sim$.

$f_2 : \partial^2 \mathcal{B}_{hj}^{\ 2} \to \partial^1 \mathcal{B}_j^{\ 1}$, or $\partial^1 \mathcal{B}_j^{\ 1} \sqcup_{f2} \mathcal{B}_{hj}^{\ 2} / \sim$.

$f_3 : \sqcup_j \partial^1 \mathcal{B}_{djj}^{\ 1} \to \partial^1 \mathcal{B}_j^{\ 1}$, or $\partial^1 \mathcal{B}_j^{\ 1} \sqcup_{f3} (\sqcup_j \mathcal{B}_{djj}^{\ 1}) / \sim$.

$f_4 : \partial^1 \mathcal{B}_{lj}^{\ 1} \to \partial^1 \mathcal{B}_j^{\ 1}$, or $\partial^1 \mathcal{B}_j^{\ 1} \sqcup_{f4} \mathcal{B}_{lj}^{\ 1} / \sim$.

$f_5 : \sqcup_j \partial^2 \mathcal{B}_{ldl}^{\ 2} \to \mathcal{B}_l^{\ 1}$, or $\mathcal{B}_l^{\ 1} \sqcup_{f5} (\sqcup_j \mathcal{B}_{ldl}^{\ 2}) / \sim$.

$f_6 : \partial^6 \mathcal{B}_{eh}^{\ 6} \to \mathcal{B}_h^{\ 2}$, or $\mathcal{B}_h^{\ 2} \sqcup_{f6} \mathcal{B}_{eh}^{\ 6} / \sim$.

$f_7 : \partial^2 \mathcal{B}_{adj}^{\ 2} \to \mathcal{B}_{dj}^{\ 1}$, or $\mathcal{B}_{dj}^{\ 1} \sqcup_{f6} \mathcal{B}_{adj}^{\ 2} / \sim$.

勘定科目空間 A_i、社員空間 E_i、元帳空間 L_i、元帳明細空間 LD_i を新しく追加したので、定義する。スケルトン X_{account} は以下のように表せる。

$X_{\mathrm{account}}^{\ 0} = \{e_{aid}^{\ 0}{}_1, \ldots, e_{aid}^{\ 0}{}_k, e_{name}^{\ 0}{}_1, \ldots, e_{name}^{\ 0}{}_k, e_{code}^{\ 0}{}_1, \ldots, e_{code}^{\ 0}{}_k\}$

$X_{account}^{\ 1} = \{X_{aid}^{\ 0}, e_{aidname}^{\ 1}{}_1, \ldots, e_{aidname}^{\ 1}{}_k, e_{namecode}^{\ 1}{}_1, \ldots, e_{namecode}^{\ 1}{}_k, e_{codeaid}^{\ 1}{}_1, \ldots, e_{codeaid}^{\ 1}{}_k \mid f_1 : \partial e_{aidname}^{\ 1}{}_i \to e_{aid}^{\ 0}{}_i$, $f_2 : \partial e_{aidname}^{\ 1}{}_i \to e_{name}^{\ 0}{}_i$, $f_3 : \partial e_{namecode}^{\ 1}{}_i \to e_{name}^{\ 0}{}_i$, $f_4 : \partial e_{namecode}^{\ 1}{}_i \to e_{code}^{\ 0}{}_i$, $f_5 : \partial e_{codeaid}^{\ 1}{}_i \to e_{code}^{\ 0}{}_i$, $f_6 : \partial e_{codeaid}^{\ 1}{}_i \to e_{aid}^{\ 0}{}_i\}$.

$X_{account}^{\ 2} = \{X_{account}^{\ 1}, e_{account}^{\ 2}{}_1, \ldots, e_{account}^{\ 2}{}_k \mid f_1 : \partial e_{account}^{\ 2}{}_i \to e_{aidname}^{\ 1}{}_i$, $f_2 : \partial e_{account}^{\ 2}{}_i \to e_{namecode}^{\ 1}{}_i$, $f_3 : \partial e_{account}^{\ 2}{}_i \to e_{codeaid}^{\ 1}{}_i\}$.

$X_{account}^{\ 3} = \{X_{account}^{\ 2}, e_{accountaccount}^{\ 3}{}_1, \ldots, e_{accountaccount}^{\ 3}{}_k \mid f_1 : \partial e_{account}^{\ 3}{}_i \to e_{account}^{\ 2}{}_i\}$.

これを利用して、スケルトン X_{detail} は以下のように得られる。

$X_{detail}^{\ 4} = \{X_{detail}^{\ 1}, e_{detail}^{\ 4}{}_1, \ldots, e_{detail}^{\ 4}{}_k \mid f_1 : \partial^3 e_{detail}^{\ 4}{}_i \to e_{detail}^{\ 1}{}_i$, $f_2 : \partial e_{detal}^{\ 4}{}_i \to e_{account}^{\ 3}{}_i\}$.

スケルトン X_{employee} は以下のように表せる。

$X_{employee}{}^{0} = \{e_{eid}{}^{0}{}_{1}, \ldots, e_{eid}{}^{0}{}_{k}, e_{name}{}^{0}{}_{1}, \ldots, e_{name}{}^{0}{}_{k}, e_{code}{}^{0}{}_{1}, \ldots, e_{code}{}^{0}{}_{k}, e_{affiliation}{}^{0}{}_{1}, \ldots, e_{affiliation}{}^{0}{}_{k}, e_{appointment}{}^{0}{}_{1}, \ldots, e_{appointment}{}^{0}{}_{k}, e_{pass}{}^{0}{}_{1}, \ldots, e_{pass}{}^{0}{}_{k}, e_{commission}{}^{0}{}_{1}, \ldots, e_{commission}{}^{0}{}_{k}\}$

$X_{employee}{}^{6} = \{X_{emploee}{}^{5}, e_{employee}{}^{6}{}_{1}, \ldots, e_{employee}{}^{6}{}_{k} \mid$

$f_{1}: \partial e_{employee}{}^{6}{}_{i} \rightarrow e_{eidnamecodeaffiliationappointmentpass}{}^{5}{}_{i}$,

$f_{2}: \partial e_{employee}{}^{6}{}_{i} \rightarrow e_{namecodeaffiliationappointmentpasscommission}{}^{5}{}_{i}$,

$f_{3}: \partial e_{employee}{}^{6}{}_{i} \rightarrow e_{codeaffiliationappointmentpasscommissioneid}{}^{5}{}_{i}$,

$f_{4}: \partial e_{employee}{}^{6}{}_{i} \rightarrow e_{affiliationappointmentpasscommissioneidname}{}^{5}{}_{i}$,

$f_{5}: \partial e_{employee}{}^{6}{}_{i} \rightarrow e_{appointmentpasscommissioneidnamecode}{}^{5}{}_{i}$,

$f_{6}: \partial e_{employee}{}^{6}{}_{i} \rightarrow e_{passcommissioneidnamecodeaffiliation}{}^{5}{}_{i}$,

$f_{7}: \partial e_{employee}{}^{6}{}_{i} \rightarrow e_{commissioneidnamecodeaffiliationappointment}{}^{5}{}_{i}\}$.

同様に頭書き X_{header} を表す。

$X_{header}{}^{8} = \{X_{header}{}^{2}, e_{header}{}^{8}{}_{1}, \ldots, e_{header}{}^{8}{}_{s} \mid f_{1}: \partial^{6} e_{header}{}^{8}{}_{i} \rightarrow e_{header}{}^{2}{}_{i}, f_{2}: \partial^{2} e_{header}{}^{8}{}_{i} \rightarrow e_{applicant}{}^{6}{}_{i}\}$.

よって、仕訳帳は以下のようになる。

$X_{\text{journal}}{}^{0} = \{e_{jid}{}^{0}{}_{1}, \ldots, e_{jid}{}^{0}{}_{i}, e_{time}{}^{0}{}_{1}, \ldots, e_{time}{}^{0}{}_{i}\}$

$X_{journal}{}^{1} = \{X_{journal}{}^{0}, e_{journal}{}^{1}{}_{1}, \ldots, e_{journal}{}^{1}{}_{i} \mid f_{1}: \partial e_{journal}{}^{1}{}_{i} \rightarrow e_{jid}{}^{0}{}_{i}, f_{2}: \partial e_{journal}{}^{1}{}_{i} \rightarrow e_{time}{}^{0}{}_{i}\}$

$X_{journal}{}^{2} = \{X_{journal}{}^{1}, e_{journalnumber}{}^{2}{}_{1}, \ldots, e_{journalnumber}{}^{2}{}_{i} \mid f_{1}: \partial e_{journalnumber}{}^{2}{}_{i} \rightarrow e_{journal}{}^{1}{}_{i}, f_{2}: \partial e_{journalnumber}{}^{2}{}_{i} \rightarrow e_{number}{}^{1}{}_{i}\}$

$X_{journal}{}^{10} = \{X_{journal}{}^{2}, e_{journalheader}{}^{10}{}_{1}, \ldots, e_{journalheader}{}^{10}{}_{i} \mid f_{1}: \partial^{8} e_{journalheader}{}^{10}{}_{i} \rightarrow e_{journalnumber}{}^{2}{}_{i}, f_{2}: \partial^{2} e_{journalheader}{}^{10}{}_{i} \rightarrow e_{header}{}^{8}{}_{i}\}$

$X_{journal}{}^{18} = \{X_{journal}{}^{10}, e_{journal}{}^{18}{}_{1}, \ldots, e_{journal}{}^{18}{}_{i} \mid f_{1}: \partial^{8} e_{journal}{}^{18}{}_{i} \rightarrow e_{journalheader}{}^{10}{}_{i}, f_{2}: \partial^{10} e_{journal}{}^{18}{}_{i} \rightarrow e_{detail}{}^{8}{}_{i}\}$

スケルトン $X_{ldetail}$ は以下のように表せる。

$X_{ldetail}{}^{0} = \{e_{ldid}{}^{0}{}_{1}, \ldots, e_{ldid}{}^{0}{}_{k}, e_{number}{}^{0}{}_{1}, \ldots, e_{number}{}^{0}{}_{k}, e_{amount}{}^{0}{}_{1}, \ldots, e_{amount}{}^{0}{}_{k}\}$

$X_{ldetail}{}^{2} = \{X_{ldetail}{}^{1}, e_{ldetail}{}^{2}{}_{1}, \ldots, e_{ldetail}{}^{2}{}_{k} \mid f_{1}: \partial e_{ldetail}{}^{2}{}_{i} \rightarrow e_{ldidnumber}{}^{1}{}_{i}, f_{2}: \partial e_{ldetail}{}^{2}{}_{i} \rightarrow e_{numberamount}{}^{1}{}_{i}, f_{3}: \partial e_{ldetaill}{}^{2}{}_{i} \rightarrow e_{amountldid}{}^{1}{}_{i}\}$.

これらを用いてスケルトン X_{ledger} は以下のように表せる。

$X_{ledger}{}^{0} = \{e_{lid}{}^{0}{}_{1}, \ldots, e_{lid}{}^{0}{}_{k}, e_{total}{}^{0}{}_{1}, \ldots, e_{total}{}^{0}{}_{k}\}$

$$X_{ledgerl}^{1} = \{ X_{ledger}^{0}, e_{ledger}^{1}{}_{1}, \ldots, e_{ledger}^{1}{}_{k} \mid f_{1:} \partial e_{ledger}^{1}{}_{i} \to e_{lid}^{0}{}_{i} \ , \ f_{2:} \partial e_{ledger}^{1}{}_{i} \to e_{total}^{0}{}_{i} \}.$$

ここに元帳明細、未処理仕訳伝票、処理済仕訳伝票を接着する。

$$X_{ledger}^{3} = \{ X_{ledger}^{1}, e_{ledgerldetail}^{3}{}_{1}, \ldots , e_{ledgerldetail}^{3}{}_{k} \mid f_{1:} \partial^{2} e_{ledgerldetail}^{3}{}_{i} \to e_{ledger}^{1}{}_{i} , \ f_{2:} \partial e_{ledger}^{3}{}_{i} \to e_{ldetail}^{2}{}_{i} \}$$

$$X_{ledger}^{21} = \{ X_{ledger}^{3}, e_{ledgerlunprocessed}^{21}{}_{1}, \ldots , e_{ledgerlunprocessed}^{21}{}_{k} \mid f_{1:} \partial^{18} e_{ledgerlunprocessed}^{21}{}_{i} \to e_{ledger}^{3}{}_{i} , \ f_{2:} \partial^{3} e_{ledgerunprocessed}^{21}{}_{i} \to e_{unprocessed}^{18}{}_{i} \}$$

$$X_{ledger}^{39} = \{ X_{ledger}^{21}, e_{ledger}^{39}{}_{1}, \ldots , e_{ledger}^{39}{}_{k} \mid f_{1:} \partial^{18} e_{ledger}^{39}{}_{i} \to e_{ledger}^{21}{}_{i} , \ f_{2:} \partial^{21} e_{ledger}^{39}{}_{i} \to e_{processed}^{18}{}_{i} \}$$

3.5. 仕訳伝票の承認を行える会計システム

ここでは頭書に承認日付、承認者、承認が加えられているので、定義する。

$$f_2 : \partial^{4} \mathcal{B}_{hj}{}^{4} \to \mathcal{B}_{j}{}^{1}, \text{ or } \mathcal{B}_{j}{}^{1} \sqcup_{f2} \mathcal{B}_{hj}{}^{4} /\sim.$$

$$f_6 : \partial^{6} \mathcal{B}_{eh}{}^{6} \to \mathcal{B}_{h}{}^{4}, \text{ or } \mathcal{B}_{h}{}^{4} \sqcup_{f6} \mathcal{B}_{eh}{}^{6} /\sim.$$

$$X_{header}^{10} = \{X_{header}^{4}, e_{headerapplicant}^{10}{}_{1}, \ldots , e_{headerapplicant}^{10}{}_{s} \mid f_{1:} \partial^{6} e_{headerapplicant}^{10}{}_{i} \to e_{header}^{4}{}_{i} , \ f_{2:} \partial^{4} e_{headerapplicant}^{10}{}_{i} \to e_{applicant}^{6}{}_{i} \}.$$

$$X_{header}^{16} = \{X_{header}^{10}, e_{header}^{16}{}_{1}, \ldots , e_{header}^{16}{}_{s} \mid f_{1:} \partial^{6} e_{header}^{16}{}_{i} \to e_{header}^{10}{}_{i} , \ f_{2:} \partial^{4} e_{header}^{16}{}_{i} \to e_{acceptant}^{6}{}_{i} \}.$$

よって、仕訳伝票と元帳も以下のようになる。

$$X_{journal}^{26} = \{ X_{journal}^{18}, e_{journal}^{26}{}_{1}, \ldots , e_{journal}^{26}{}_{i} \mid f_{1:} \partial^{8} e_{journal}^{26}{}_{i} \to e_{journal}^{18}{}_{i} , \ f_{2:} \partial^{18} e_{journal}^{26}{}_{i} \to e_{detail}^{8}{}_{i} \}$$

$$X_{ledger}^{55} = \{ X_{ledger}^{29}, e_{ledger}^{55}{}_{1}, \ldots , e_{ledger}^{55}{}_{k} \mid f_{1:} \partial^{26} e_{ledger}^{55}{}_{i} \to e_{ledger}^{29}{}_{i} , \ f_{2:} \partial^{29} e_{ledger}^{55}{}_{i} \to e_{processed}^{26}{}_{i} \}$$

3.6. 表現レベル

元帳の利用を可能にし、仕訳伝票の承認を行えるようにした会計システムの、クラス図を書くと図 3-5 のようになる。

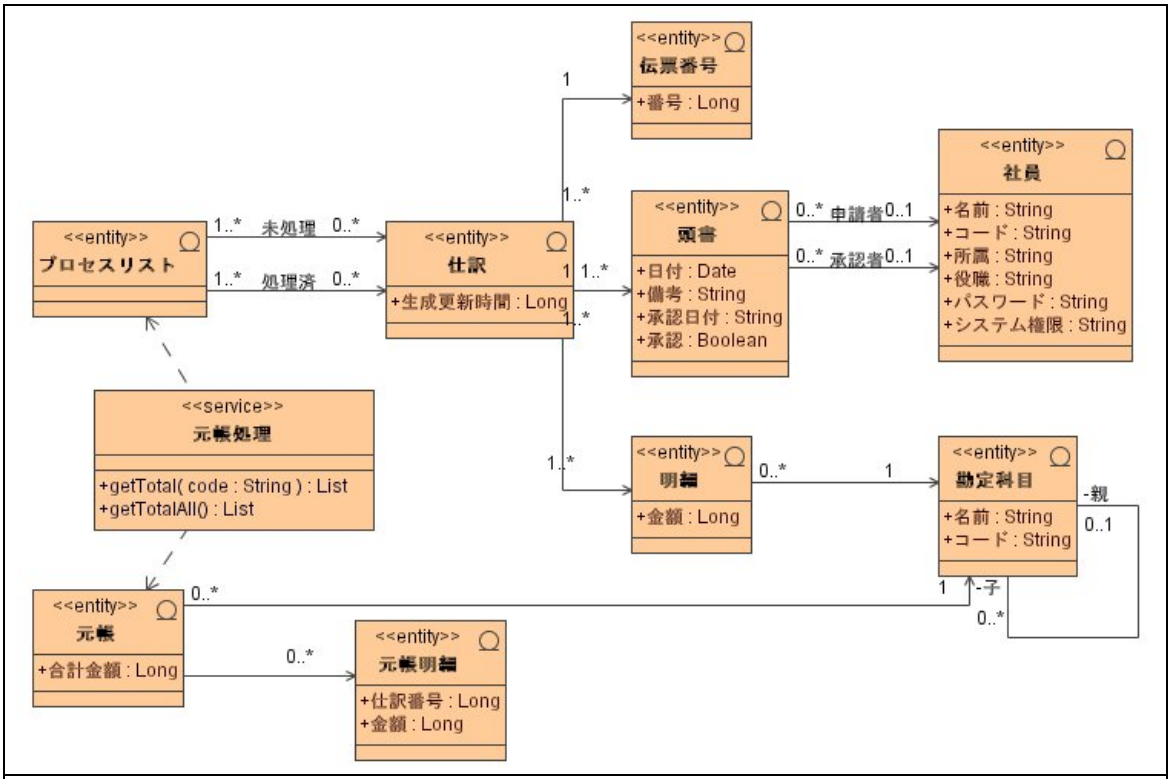

図 3-5　発展させた会計システムのクラス図

ここでは元帳と、仕訳の間に未処理仕訳伝票と、処理済仕訳伝票が接着関数で対応付けられている。

発展させた会計システムでは、元帳への加算が必要となるが、ここでは、仕訳伝票が入力され、その仕訳伝票が承認されたときに元帳までの記帳を終えるのではなく、仕訳伝票のデータベースへの登録までに留めておく。インターアクティブなシステムでは応答時間が求められるので、入力が発生したときの処理は最小限にとどめておきたい。登録で留めておくというのは、このための配慮である。その代わり、別途、元帳への加算処理が必要となるが、これは、元帳が必要となったときに行うことになる。そのため、ここでは仕訳伝票の承認後にプロセスリストに未処理明細として加えておく。

ここでは、Session Beans のクラス元帳処理を用意し、勘定科目名を得て、その元帳の内容を返すメソッド getTotal()を考えることにする。

また、ここでのユースケース図は図 3-6 のようになる。

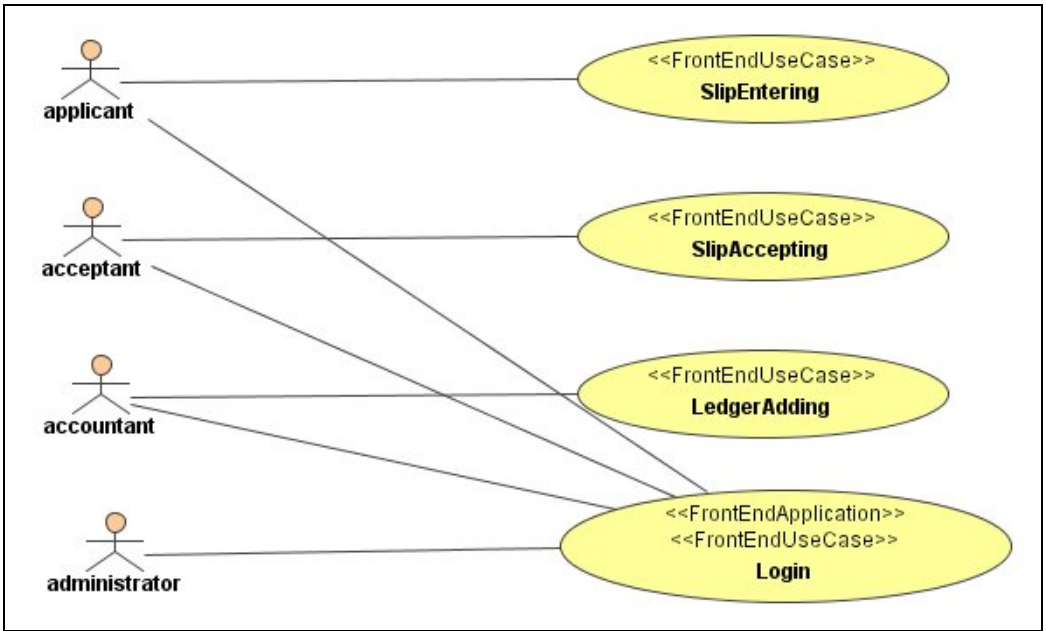

図 3-6　発展させた会計システムのユースケース図

元帳を参照する際の状態図は図 3-6 のようになる。また、この状態遷移に基づいてアクティビティ図を書くと図 3-7 のようになる。

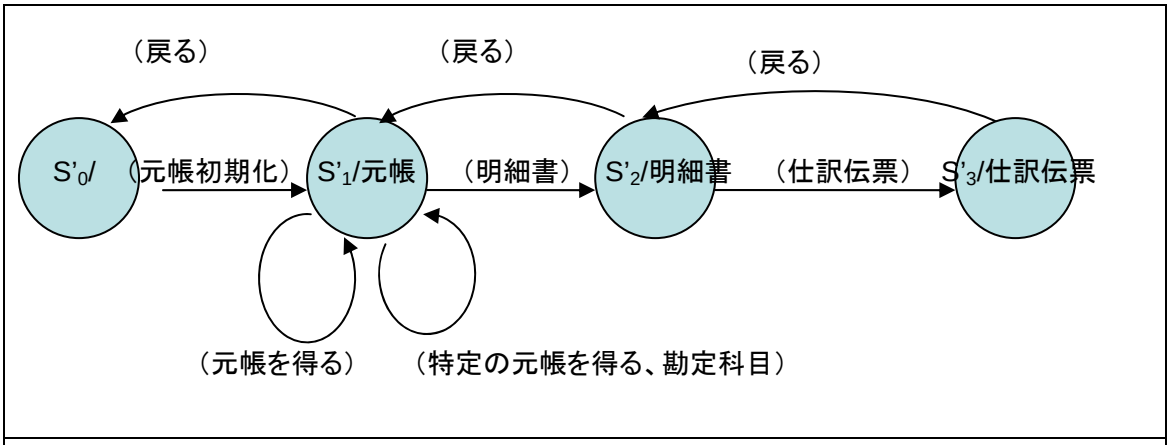

図 3-7 元帳参照の状態遷移

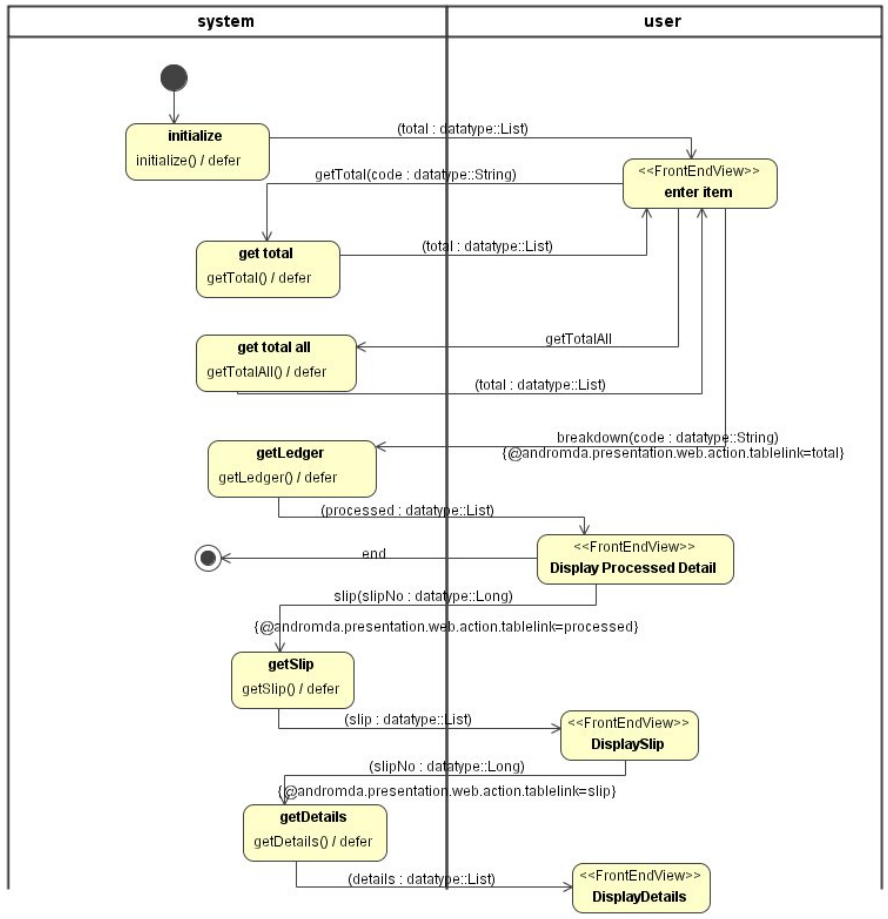

図 3-8 元帳参照のアクティビティ図

状態Ｓ’$_1$での元帳を得た場合、未処理仕訳伝票に接着された明細の空間と元帳の空間の間で接着・分離が行われる。この関係は表 3-1 に示すようになる。

表 3-1

	分離	接着
元帳を得る		該当する元帳空間に、未処理仕訳伝票の明細の空間を接着。

仕訳伝票承認の仕様を定義する状態図、図 3-9 のようになる。仕訳伝票承認では、申請のときに必要であった四つの空間に加えて、元帳の空間が必要となる。また、この状態遷移に基づいてアクティビティ図を書くと図 3-10 のようになる。

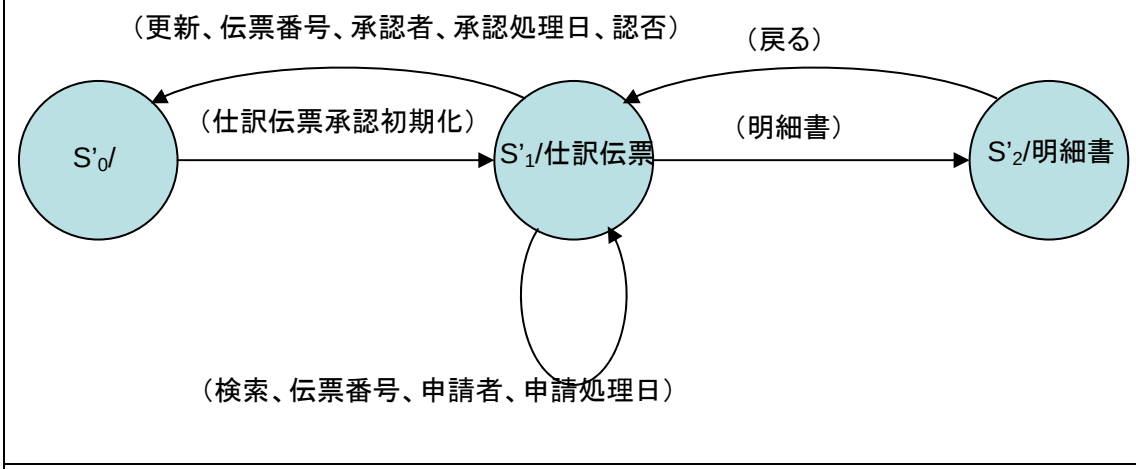

図 3-9 仕訳伝票承認の状態遷移

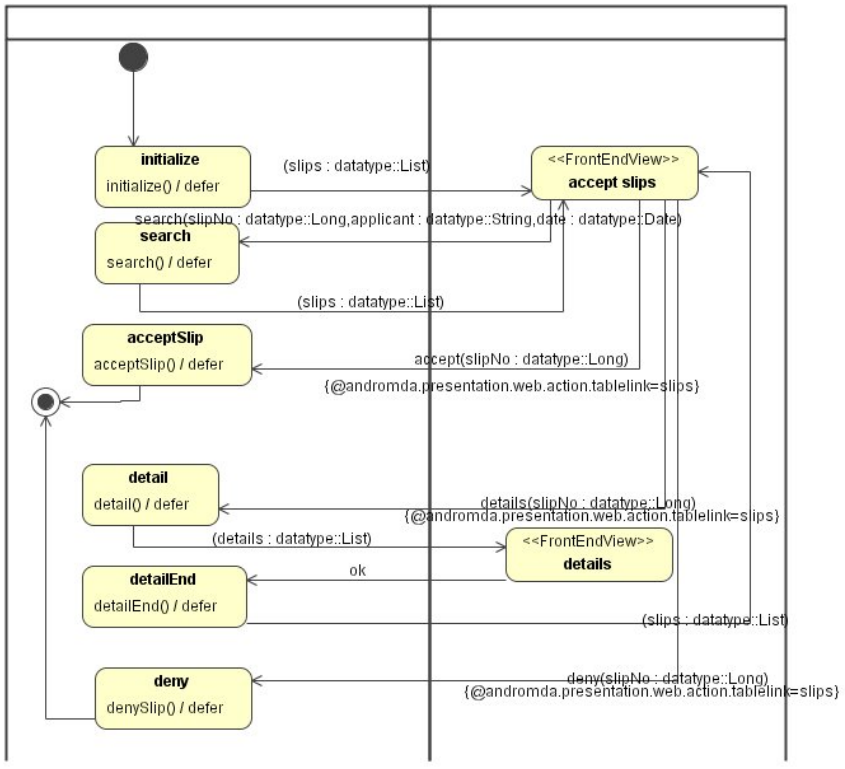

図 3-10　仕訳伝票承認のアクティビティ図

状態Ｓ'$_1$での更新で承認された場合は、元帳の作成を引き起こすので、仕訳伝票で作られた空間と元帳の空間の間で接着・分離が行われる。この関係は表 3-2 に示すようになる。

表 3-2		
	分離	接着
更新(承認のとき)		該当する元帳の空間に仕訳伝票空間を接着。名称は未処理仕訳伝票とする
検索		

最終的に、session beans でのクラス等を追加したクラス図は図 3-10 のようになる。

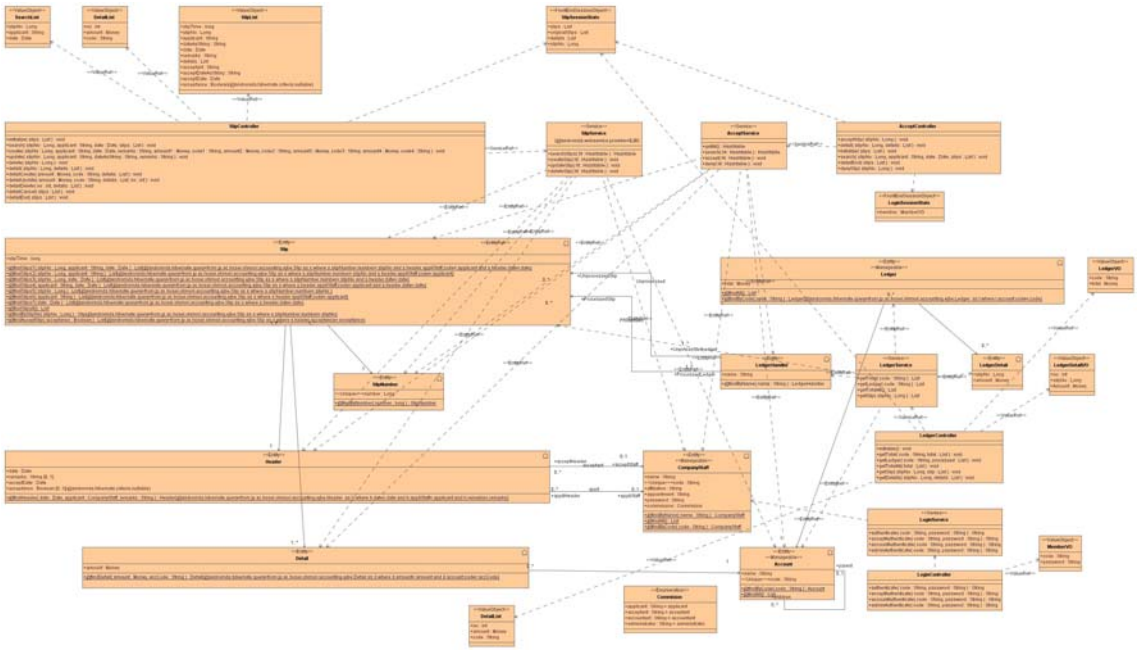

図 3-10 会計システムのクラス図

追加したクラスは以下のようになる。

- LedgerController、AcceptController、LoginController
  元帳、承認機能の Web 側の操作を定義するクラス。
- LedgerService、AcceptService、LoginService
  元帳、承認機能の EJB 側の操作を定義するクラス。
  ステレオタイプ<<Service>>を付与する。
- LedgerVO、LedgerDetailVO、MemberVO
  データ転送用のクラス。
  ステレオタイプ<<ValueObject>>が付与する。
- LoginSessionState
  ログイン情報を保持するクラス。
  ステレオタイプ<<FrontEndSessionObject>>

- Commission
  システム権限の要素を全て列挙したクラス。

3.7. 可視レベル

表現レベルで書かれた UML を基に AndroMDA を利用して､プログラムを自動生成する。上位のレベルで定められている不変量を維持しているため、新しく追加した部分についてのビジネスロジックの記述のみですむ。元帳追加の機能強化により発生する作業は、基本的な会計システムで作られたプログラム・モジュールを再利用することで実現できるため、元帳を得るためのメソッド getTotal(code)、getTotalAll()に伴うビジネスロジックの部分だけのプログラム開発のみでよい。機能の追加による必要な開発工数僅少に限られる。

このため、バグの発生率は低く、モジュールの線形な追加となり、従来の開発法が有していた欠点を回避している。

Web 側から、あるいは、Web service の側から Session Bean“元帳処理”に対して、勘定科目コード code の元帳を渡すように要求があったとする。元帳処理のインスタンスでは、getTotal(code)のメソッドの処理に移る。このメソッドは、まず、プロセスリストに保存してある未処理仕訳伝票を得て、そこに接続されている明細リストを得る。次に元帳から findByCode(code)のメソッドを用いて、code という勘定科目コードの元帳のインスタンスを得る。明細リストの金額をこのインスタンスの合計に加える。その明細リストが接続されていた仕訳伝票番号と明細の金額を、元帳明細に保存し、未処理仕訳伝票を処理済仕訳伝票とする。最後に、処理済明細書の一覧と合計を要求元に返す。

ここでの getTotal (code)メソッドは以下のように記述した。

- WEB 側

```
public final void getTotal(ActionMapping mapping, jp.ac.hosei.ohmori.accounting.webl.GetTotalForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception
    {
        List c = (List)this.getLedgerService().getTotal(form.getCode());
        form.setTotal(c);
    }
```

- EJB 側

```
protected java.util.List handleGetTotal(java.lang.String code)
        throws java.lang.Exception
    {
```

```
        preparing();
        List ss = new ArrayList();
        LedgerVO ll = new LedgerVO();
        ll.setCode(code);
        Ledger l = getLedgerDao().findByCode(code);
        ll.setTotal(l.getTotal());
        ss.add(ll);
        return ss;
    }


    private void preparing () throws java.lang.Exception
    {
        LedgerHandler lh = getLedgerHandlerDao().findByName("handler");
        if(lh.getUnprocessedSlip()==null) return;
        Collection c = (Collection)lh.getUnprocessedSlip();
        Collection ps = null;
        if(lh.getProcessedSlip()!=null) ps = lh.getProcessedSlip();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            Slip s = (Slip)it.next();
            Collection cc = (Collection)s.getDetails();
            Iterator itc = cc.iterator();
            while(itc.hasNext()) {
                Detail d = (Detail)itc.next();
                Account acc = d.getAccount();

                Ledger l = getLedgerDao().findByCode(acc.getCode());
                Collection lds = (Collection)l.getLedgerDetails();
                LedgerDetail ld = getLedgerDetailDao().create(s.getSlipNumber().getNumber(),
d.getAmount());
                lds.add(ld);
                l.setTotal(new Double(d.getAmount().doubleValue() + l.getTotal().doubleValue()));

                Account acc1=acc;
                while(acc1.getParent()!=null){
```

```
                    Account acc2=acc1.getParent();
                    Ledger ll = getLedgerDao().findByCode(acc2.getCode());
                    ll.setTotal(new Double(d.getAmount().doubleValue() +
ll.getTotal().doubleValue()));
                    acc1=acc2;
                    }
                }
                ps.add(s);
            }
            c.clear();
            lh.setProcessedSlip(ps);
        }
```

全ての元帳を得る際に使うメソッド getTotalAll()においても、同様の処理がなされている。

承認機能追加においても、承認を行うメソッド accept()と、承認拒否を行うメソッド deny()についての記述をするだけでよい。

また、ホモトピーレベルにおいて、社員と勘定科目の空間の分離を行った。これによる変更は接着空間で定義されたものを基に、わずかですむ。仕訳伝票登録における、申請者の変更を例に挙げる。分離前では、

h.setApplicant(sl.getApplicant());

のように記述していた。ここでは、h が頭書のインスタンス、sl.getApplivant()がフォームに入力された社員コードである。変更後では、

h.setAppliStaff(getCompanyStaffDao().findByCode(sl.getApplicant()));

のように記述することになる。これまでは頭書きの要素として存在していたものを、社員空間に接着しているだけである。つまり、これまで作り上げた機能に対しては、特に変更を加えるわけでなく、これにおけるバグの発生率は非常に低いといえる。

出来上がった会計システムは図 3-12、図 3-13、図 3-14、図 3-15 のようになる。

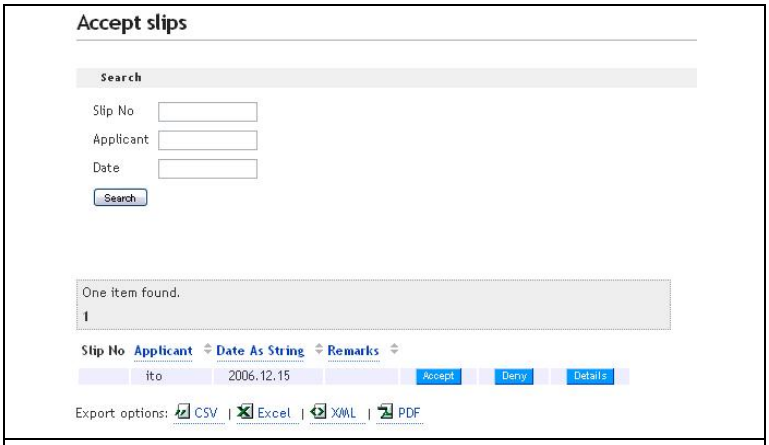

図 3-12 承認画面

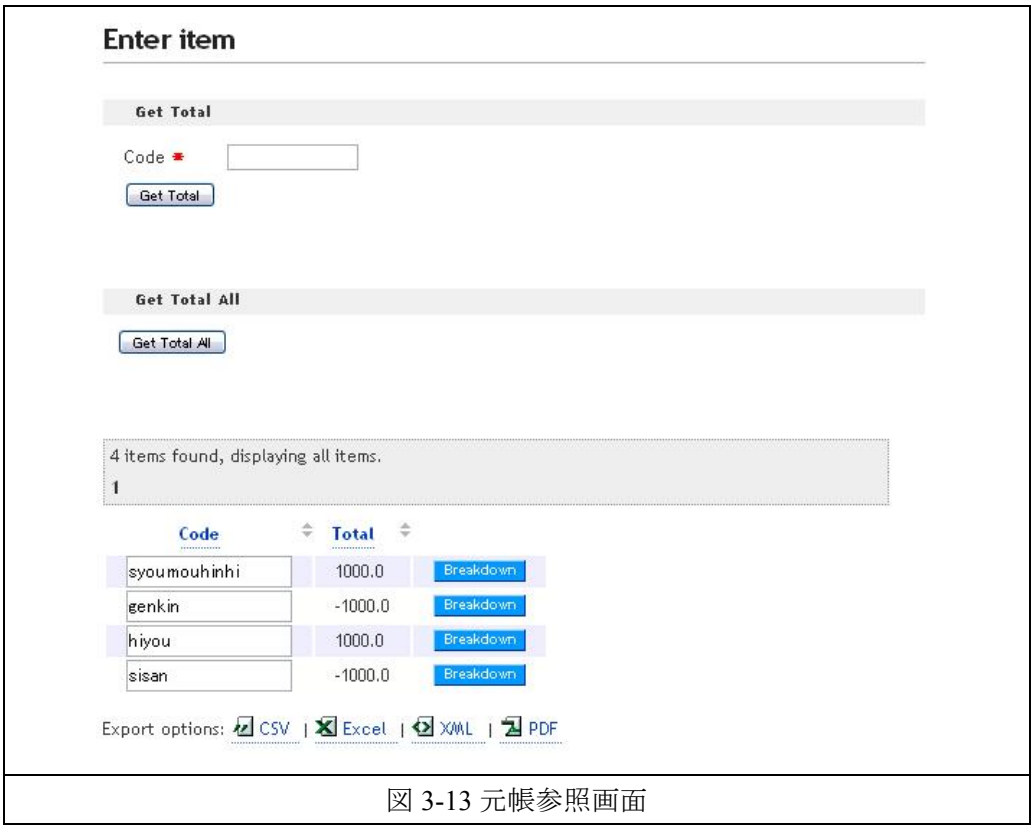

図 3-13 元帳参照画面

図 3-14 元帳の詳細画面

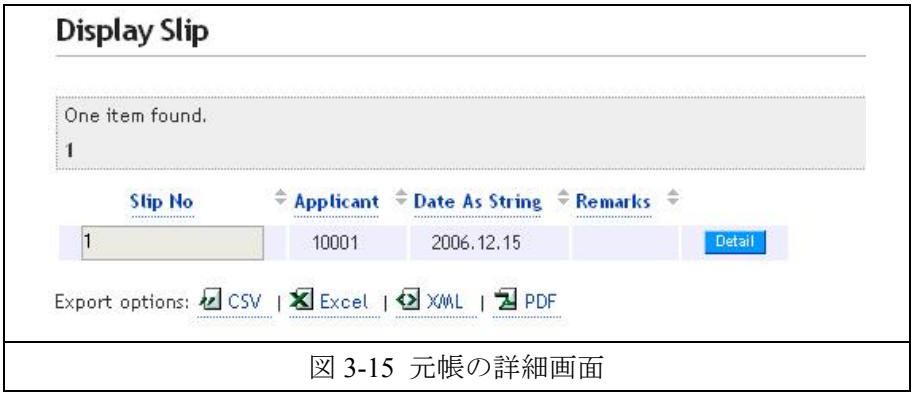

図 3-15 元帳の詳細画面

また、このシステムにおいて管理者は、勘定科目、社員、元帳に対しての初期化、管理を行う。そのための web ページは、それぞれのクラスにステレオタイプ<<manageable>>を付与することで自動生成される。図 3-16, 図 3-17, 図 3-18 のようになる。

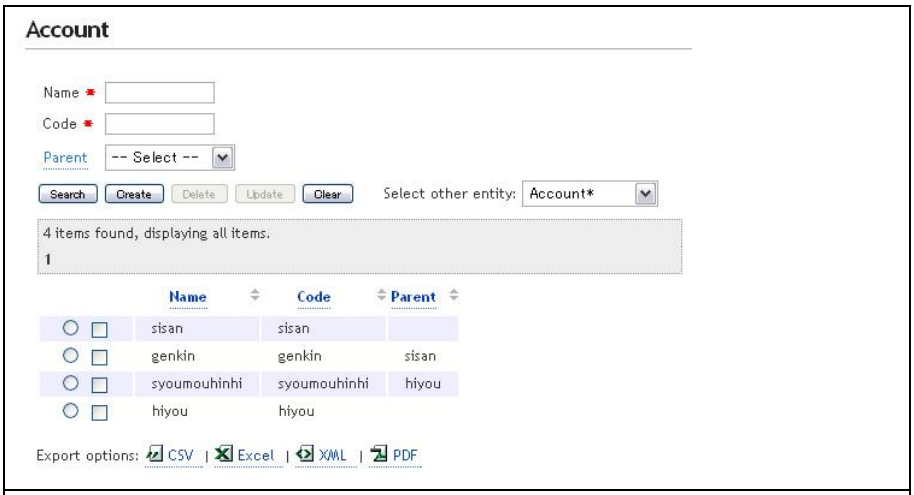

図 3-16 勘定科目の管理画面

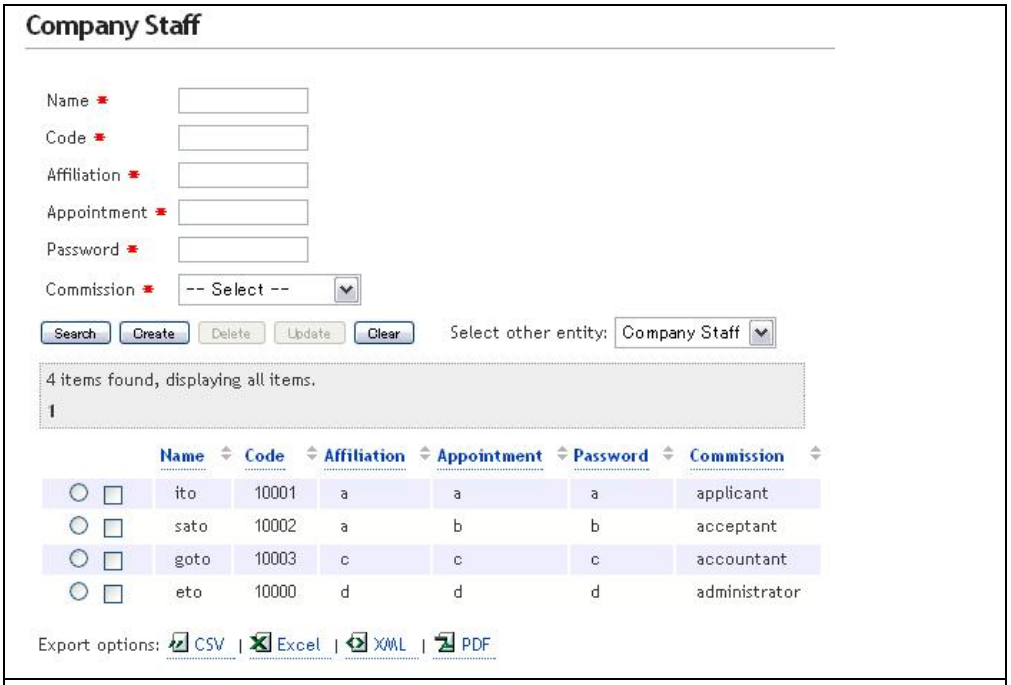

図 3-17 社員の管理画面

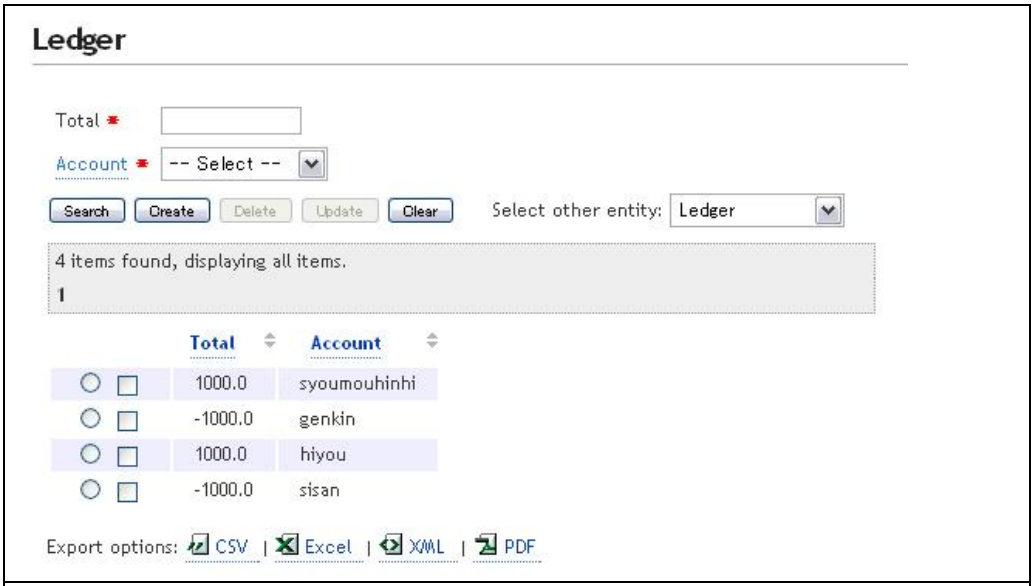

図 3-18 元帳の管理画面

4. 結果

以下に作成した UML 図からから生成されたファイル数を示す（表 4-1）。

表 4-1			
	自動	半自動	合計
WEB	174	4	178
EJB	101	13	114
共通	7		7
合計	282	17	299

自動的に生成されたファイル数と半自動的に生成されたファイル数を用い、単純にプログラムの自動化率を導き出すと 94.3%になる。

自動化されたプログラムにおいては、新たにモジュールを追加することによっても、バグはなく、デバッグの必要はなかった。

半自動で生成されたプログラムにおいて、ビジネスロジックの部分は、プログラムを記述した。この部分においてはデバッグが必要となったが、これらは論理的な構造によるバグ（表 4-2）と、データの参照ミスなどのプログラム誤記によるバグであり、その数は 10 程度であった。

表 4-2 開発時のバグ	
基本的な会計システム。	
バグ	**デバッグ内容**
頭書データベースに書き込み、更新する際に同じ内容であっても、新たにデータを書き込んでいたために、テーブルがすぐに大きくなってしまう。	同じデータの際は、新たにデータを作らないようにした。
承認機能の追加	
バグ	**デバッグ内容**
承認が何度でもできてしまい、元帳に承認した回数分の金額が足されてしまう。	既に承認に値が入っている場合は、承認画面に表示しないようにし、値の変更もできないようにした。

5. 結論

基本的な会計システムを、モジュールを追加しながら発展させた会計システムを開発していったが、それぞれの段階で、ビジネスロジックの記述における、少量のバグの発生のみで順調に開発することができた。

これまでのソフトウェア開発手法においては、モデリングの段階での曖昧さがバグの発生を招いていた。IMAH を利用することで、この段階での情報の整理が正しく行えるようになる。正しいモデリングが行われれば、AndroMDA により約 9 割の信頼性のあるコードを自動生成することができるので、ほとんどのバグを減らすことができるだろう。また、AndroMDA はオープンソースで開発されており、その開発ペースは非常に迅速である。現に 3.0 以前のバージョンでは、クラス図のみでの開発で、Web 側のプログラムの自動化率はあまり高くなかったのだが、現在のバージョンではアクティビティ図を使用することによって、Web 側の自動化率を高めている。このペースで開発が進めば、今回バグが起きたビジネスロジックのプログラム記述においても、自動生成することによって、より人為的なバグを削減することができるようになるだろう。このことによって、より大規模なソフトウェアの開発が容易になり、より情報化社会の発展が期待できる。

参考文献

[1]Ohmori K, Kunii T L, “An Incrementally Modular Abstraction Hierarchy for Linear Software DevelopmentMethodology”. CW 2006. IEEE Computer Society, 2006, pp216-223.

[2]Kunii T L, Ohmori K, Cyberworlds: Architecture and Modeling by an Incrementally Modular Abstraction Hierarchy. The Visual Computer, Springer-Verlag, 2006, 22(12): pp949-964.

[3]Philippe Kruchten, “The Rational Unified Process An Introduction Second Edition”, Pearson Education Japan, Mar. 2001.

[4]David S. Frankel, “Model Driven Architecture – Applying MDA to Enterprise Computing”, SIBaccess Co Ltd., Nov. 2003.

[5] http://www.andromda.org/

[6] http://pantodon.shinshu-u.ac.jp/topology/fibrations/index.html#indexpa1.html