Asakusa Framework
詳細設計ケース・スタディ

改訂履歴

版	日付	内容
1.0	2012/01/17	新規作成

1. 詳細設計ケース・スタディ概要

AsakusaDSL 基本設計のケース・スタディで作成した DAG を元に詳細設計を実際に行う。
ケースは、「小売業の POS データの塊をクレンジング・正規化してカテゴリー別に集計し、同時に確定伝票を作成するフローパート」になっている。
業務系のログデータを集計する典型的な業務バッチ処理の構成フローである。基本的な流れは、レシートデータを束ねた入力データをもとに、各種のマスターチェックを行う。その後、クリーニングされたデータを元に、店別→部門別→カテゴリー別（2 階層）→単品、と、全店→部門別→カテゴリー別（2 階層）→単品の二つの構成で集計データを作成し、同時に店別部門別単品別の売上伝票を作成するフローである。

データフローの設計は、AsakusaDSL 設計手法を参照のこと。
実際の DAG は以下の通りである。

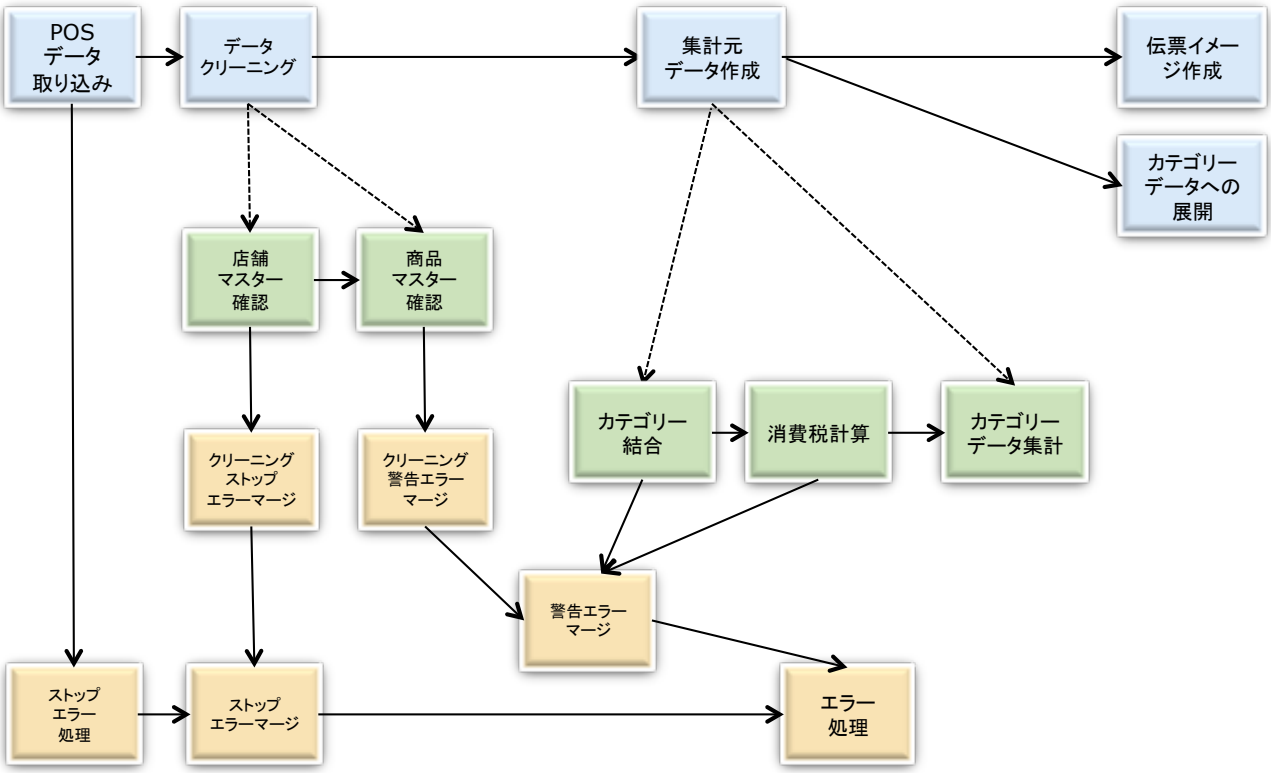

図 1

データフローはできあがっているので、それを元に詳細設計を起こしていく。入力モデルからスタートして、出力モデルへ到達するように導いていく。DAG の各頂点でデータ操作が行われるため、各頂点に適切な演算子の割当設計をしていく。具体的には添付の Excel シート（以下、データフロー設計シートとする）を作成していく手順になる。順に項目を解説していく。

2. 全体の構成

参考のデータフロー設計シート上では、各行が各データフィールドの一連の操作になっている。基本的にそのフィールドに対して操作を行う内容が、対応する Operator のカラムになっている。したがって、各行を順に追っていくことで、内容が一覧できる。

入力データモデル メンバー	operator1	operator2	operator3	operator4	operator5	出力モデル メンバー
a	R	→	R	→	→	a
b	→	→	→	R	→	b
c	R	R	D			
d	→	→	→	→	→	d
e	→	→	→	→	R	e
f	→	→	U	→	→	f
			C	→	U	g

一番左の最初のカラムがスタートモデルになっており、順に各カラムが対応する演算子になっている。最後のもっとも右のカラムがエンドモデルになり、生成されるデータになっている。

2.1. *入力モデル*

入力されるデータモデルを表す。構成されるフィールドの詳細、それに対応する型が Type のカラムに設定される。付帯情報として、ユニークネスを規定するキーを設定している。参考例ではデータインスタンスのユニークキーは店番号＋日付＋ID＋レシート ID＋行番号になっている。

尚、データインスタンスが、所属する集合での数値を各インスタンスで持っている場合、（参考例の場合でいうとレシートの合計値になるが、これはレシートの合計値であり、レシート ID が同じデータインスタンスでは「すべて」同じ値が入る）集合とインスタンスの関連を附記しておく便利である。参考例では Ref1_Key と Ref1 の関係になる。

モデルのフィールドは小売店舗で買い物をしたときに発行されるレシートデータのイメージであり、フィールドの意味は以下の通りである

・店番号：レシートを発行した店舗のユニークコード
・日付：レシートの発行日
・ID：レシートを一定の単位でまとめたデータの束への ID。通常は締め処理のタイミングで発行される。
・承認番号：締め処理で発行される承認コード
・レシート ID：レシートユニーク ID
・行番号：レシート内部での商品の並び順

以上が ID 系になる。投入データの単位は ID 単位＝締め処理単位であり、処理フローは、複数のデータ（すなわち投入データは複数インスタンス）の一斉投入で実行される。

・JAN コード：商品コード
・商品名：商品名
・売価：単品の売単価
・数量：販売された数量
・金額：単品の販売金額　売価 x 数量

以上は、店番号＋日付＋ID＋レシート ID＋行番号でのユニークな情報。集計の基礎単位になる。

・合計金額：レシート ID 単位での合計金額
・消費税：レシート ID 単位での消費税額の合計単位
・受取金額：レシート ID 単位での入金金額

以上はレシート ID 単位での集計数値であり、データユニークではなく、同じレシート ID には同じデータが入る（レシート ID ユニーク）。典型的な半構造データである。

参考）ここでは、データの効率性をあげるために正規化は最低限にしている。特にトランザクション系のデータでは、必要に応じて冗長的にデータ構造を持つ。ヘッダーの詳細情報を各明細行にアタッチするフラットなデータ構造がこれに当たる。。ただし、あまりに不要なものまで持つと、データ転送時に不必要なオーバーヘッドが発生するので注意を要する。

DMDL 記述名：POS_ITEM

2.2. *出力モデル*

正常モデルは 3 種類になっている。

2.2.1. 売上伝票レイアウト

店別部門別の売上伝票を表す。ヘッダーは、店番・日付で構成される。明細は 6 行とする。
各行の記載は、GTIN コード・商品名・販売数量・販売金額・消費税額の結合文字列とする。
改ページごとにページ番号付与される。イメージデータなので、行数はハードコードになる。

モデル・フィールドは以下の通り
・店番号：店舗コード
・日付：売上日付（レシート日付）
・部門コード：部門コードで集計計上する
・行番号 1～行番号 6：上記の結合文字列
・ページ番号

DMDL 記述名：FORM_LAYOUT

2.2.2. カテゴリー集計レイアウト

単品データとそれに関連する集計データがアタッチされている形式になる。各単品データに冗長的にすべての集計データを保持する。

モデル・フィールドは以下の通り（店舗別・単品別が最終アウトプットのソート順になっている）

・店番号：店舗コード
・日付：レシート日付
・JAN コード：商品 ID
・商品名：レシート品名（商品マスターで結合）
・部門コード：最上位カテゴリーコードになる
・カテゴリーコード 1：中位層カテゴリー
・カテゴリーコード 2：下位層カテゴリー

以下は集計フィールドになる

・カテゴリーカットの集計
・全店計金額：単品の全店舗分の集計の金額
・全店計数量：単品の全店舗分の集計の数量
・全店計消費税：単品の全店舗分の集計の消費税
・全店計客数：単品の全店舗分の集計の客数
（上記はすべてのデータで同じ数字が表示される）

・全店部門計金額：全店の各部門（最上位カテゴリー）の積み上げ金額（以下同様）
・全店部門計数量：全店の各部門の積み上げ数量
・全店部門計消費税：全店の各部門の積み上げ消費税額
・全店部門計客数：全店の各部門の積み上げ客数
（上記は同じ部門に属するすべての単品データで同じ数字が表示される）

・全店カテゴリー1 数量：全店計の特定カテゴリー（中位層カテゴリー）の積み上げ数量
・全店カテゴリー1 金額：全店計の特定カテゴリーの積み上げ金額
・全店カテゴリー1 消費税：全店計の特定カテゴリーの積み上げ消費税額
・全店カテゴリー1 客数：全店計の特定カテゴリーの積み上げ客数
（上記は同じカテゴリー1 に属するすべての単品データで同じ数字が表示される）

・全店カテゴリー2 数量：全店計の特定カテゴリー（下位層カテゴリー）の積み上げ数量
・全店カテゴリー2 金額：全店計の特定カテゴリー（下位層カテゴリー）の積み上げ金額
・全店カテゴリー2 消費税：全店計の特定カテゴリー（下位層カテゴリー）の積み上げ消費税
・全店カテゴリー2 客数：全店計の特定カテゴリー（下位層カテゴリー）の積み上げ客数
（上記は同じカテゴリー2 に属するすべての単品データで同じ数字が表示される）

・全店 SKU 数量：全店計の該当データの商品の積み上げ数量
・全店 SKU 金額：全店計の該当データの商品の積み上げ金額
・全店 SKU 消費税：全店計の該当データの商品の積み上げ消費税額
・全店 SKU 客数：全店計の該当データの商品の積み上げ客数
（上記は同じ GTIN 単品データで同じ数字が表示される）

・ロケーション別カットの集計
・店舗計金額：特定店舗の販売金額
・店舗計数量：特定店舗の販売数量
・店舗計消費税：特定店舗の受取消費税額
・店舗計客数：特定店舗の客数総計
（上記は同じ店舗でのデータであれば、同じ数字が表示される）

・店別部門別数量：特定店舗の特定部門（最上位カテゴリー）での積み上げ数量
・店別部門別金額：特定店舗の特定部門での積み上げ金額
・店別部門別消費税：特定店舗の特定部門での積み上げ消費税額
・店別部門別客数：特定店舗の特定部門での積み上げ客数
（上記は同じ店舗での同じ部門に属する単品のデータであれば、同じ数字が表示される）

・店舗カテゴリー1 数量：特定店舗の特定カテゴリー（中位層カテゴリー）の積み上げ数量
・店舗カテゴリー1 金額：特定店舗の特定カテゴリー（中位層カテゴリー）の積み上げ金額
・店舗カテゴリー1 消費税：特定店舗の特定カテゴリー（中位層カテゴリー）の積み上げ消費税額
・店舗カテゴリー1 客数：特定店舗の特定カテゴリー（中位層カテゴリー）の積み上げ客数
（上記は同じ店舗での同じカテゴリー1 に属する単品のデータであれば、同じ数字が表示される）

・店舗カテゴリー2 数量：特定店舗の特定カテゴリー（下位層カテゴリー）の積み上げ数量
・店舗カテゴリー2 金額：特定店舗の特定カテゴリー（下位層カテゴリー）の積み上げ金額
・店舗カテゴリー2 消費税：特定店舗の特定カテゴリー（下位層カテゴリー）の積み上げ消費税額
・店舗カテゴリー2 客数：特定店舗の特定カテゴリー（下位層カテゴリー）の積み上げ客数
（上記は同じ店舗での同じカテゴリー2 に属する単品のデータであれば、同じ数字が表示される）

・店舗 SKU 数量：特定店舗での該当データの商品の積み上げ数量
・店舗 SKU 金額：特定店舗での該当データの商品の積み上げ金額
・店舗 SKU 消費税：特定店舗での該当データの商品の積み上げ消費税額
・店舗 SKU 客数：特定店舗での該当データの商品の積み上げ客数

（上記はユニークデータ）

DMDL 記述名：SUMMARY

エラーデータモデルは 3 カテゴリーで、合計 5 種類

1:ストップエラー
当該データインスタンスの処理を停止するエラー。
データフォーマットのエラーと、店番エラー
エラーデータのフォーマットは、入力データに対して、エラー種別を添付した形式になる。

DMDL 記述名：ERROR_POS_ITEM

2:警告エラー1
データエラーではあるが、処理を続行するエラー処理。
商品マスター結合エラーとカテゴリーマスター結合エラー
エラーデータのフォーマットは、入力データに対して、エラー種別を添付した形式になる。

DMDL 記述名：ERROR_POS_ITEM

3:警告エラー2
データエラーが推定される軽微なエラー
消費税の不整合エラーデータ
エラーデータのフォーマットは、計算された消費税（レシート ID でグルーピングした単品から逆算した消費税額）、レシート上の消費税額、および、店番号・日付・レシート ID・合計金額・受取金額が記述されている。

DMDL 記述名：ERROR_CONSUMPTION_TAX

3. 演算子の各カラムの内容

各演算子を順に解説していく。設計手法で作成された DAG を再掲する。

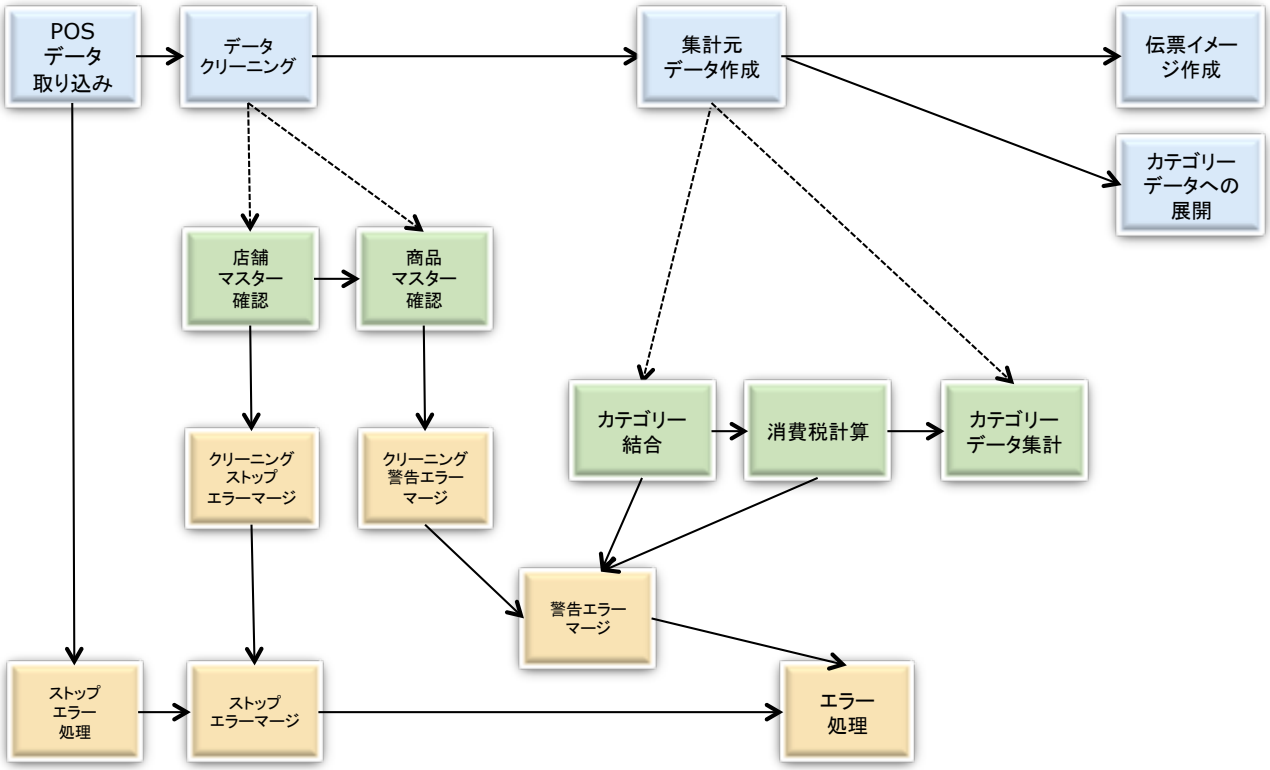

図 2

DAG の各頂点の処理を CRUD 操作に分解して、個別の演算子として参考のデータフロー設計シートに展開することが演算子の詳細設計になる。**尚、データフロー設計シートはデータ操作の記述に注力しており、具体的な業務ロジックは記述していない。**

以下にデータフロー設計シートの解説を記述する。まず、演算子の記述項目を順に解説する。エクセルの各カラムの内容は以下の通りになっている

・Operator
DAG の各ノードに対応する。演算子の名前を記述している。また、その演算子が操作するフィールドに対して行われる CRUD 操作を記述する。記述内容は各演算子によって異なっている。

・X-Ref
DAG の各エッジに対応する。特にデータフローが分岐する場合のクロス・リファレンスを記述している。具体的には、branch や merge の処理の場合に、分岐元のデータモデルのフィールドと分岐先のデータモデルのフィールドの参照を記述している。

・Constrain
各フィールドについての制限を記述している。処理の事前条件を設定する。表明を行うためのフィールド。

・Edge
DAG の各エッジに対応する。演算子の出力の表示。どのように演算子が受け取ったフィールドを出力するのかを記述している。記述例は以下の通り。

Mov 値をコピーする
Branch 値を分岐処理した先に流す
Merge 他のフィールドとマージする
Del 値を削除する
生成する場合は、フィールド名を記述する。

これらは組み合わせて利用することも可能である。例えばコピーして、同時に条件をつけて、他のフールド分岐させるときは MovAndBranch、また XOR の場合は、MovXorBranch のように利用する。

・Type
演算子の出力でフィールドを生成したときに、そのフィールドの型を記述している。
基本的に primitive を想定している。(ComplexType は現時点の Asakusa でサポートされていないが、今後サポートされる予定である。)

・Ref
補足事項を記述する特に、構造データを渡すような場合で、特定のフィールドが別の特定のフィールドに紐づいているような場合に、その関連を記述する。

4. 演算子解説

参考の DAG を各演算子に割り当てた内容を順に解説する。

4.1. *POS データ取込チェック*

データをチェックして、異常があるものは検出して別のフローに流す。問題があるデータは別フローに流す。分離するので演算子は Branch を選択する。

各フィールドを読み込んで(Read)設定された制約について検査を行う。本ケース・スタディでは、一部定数はハードコードしている(将来、定数表がされる予定。) 日付チェックのようにコンテキスト依存のものは、BatchContext から有効な値を取得させる実装を利用する。

各フィールドの確認は以下の制約による。ゼロ売価は許容する。

店番号:	0<<9999
日付:	コンテキストによる外部依存
ID:	0<
承認番号	存在確認
レシート ID	0<
行番号	0<
JAN コード	14 桁
商品名	存在確認
売価	0<=
数量	0<=
金額	0<=
合計金額	0<=
消費税額	0<=
受取金額	0<=

出力エッジは、分岐(エラー処理)するか、そのまま続行するかの排他になる。(エクセル上は MovXorBranch という記述になっている。)

エラー処理
エラー処理にトラップされて、Branch している。(エクセル上は改行して、下の行に別のフローを記述している。)ここでは、エラー種別を新規に Create する。元データにエラー種別を追記するフィールドを追加するので、エラー処理の演算子は Extend を利用する。その他フィールドはコピーされて生成される。データフロー設計シートでは、各フィー

ルドに Mov が記述され、フィールドと型が記述されている。Branch の出力先と出力フィールドの関連は X-Ref で記述する。

このエラーはストップエラーになる。

Flow クラス
PosImportFlowPart

演算子クラス
PosCheckOperator

入力クラス　PosItem
出力クラス　PosItem, ErrorPosItem

4.2. *店舗マスターチェック*

店番号をマスターと照合して、異常があるものは検出してエラー処理とする。マスターの有効期限を確認して、有効なマスターのみを結合に利用する。これにより有効でない店番、たとえば新店のケースでまだオープンしていないような場合は処理をガードすることができる。

演算子は MasterCheck を選択し、selection を利用して有効な店舗マスターのみ利用し、店番号を結合して、存在チェックを行う。selection の有効範囲は BatchContext を参照することで選択する。結合するマスターデータは Store で、参照フィールドは storeCode になる。データフロー設計シートでは、結合キーは店番号のフィールドになるので、キーとして指定して Read を記述している。制約は Existence を記述する。店番号で結合できなかったデータはエラーデータとして分離する。

エラー処理
エラーデータは、エラー処理に分離する。店番号でのエラーは継続処理ができないので、エラーはストップエラーとして処理する。エラーになったデータ・インスタンスは、処理フローを異常系にスイッチする。

処理を分岐させた後は、エラーコードを Create して当該データに付与する。演算子は Extend を利用する。尚、入力されたデータタイプは POS データ取り込み処理で処理されたデータタイプであり、PosItem クラスになる。これを拡張してエラーコードをアタッチするので、このエラー処理で生成されるエラーデータの型は、POS データ取り込み処理で生成されるエラーデータと同じ型になり、ErrorPosItem になる。
設計上のデータフロー設計シートでの記述は MovXorBranch(処理を続行するか、ブランチさせるかが排他)になる。

Flow クラス
CleaningFlowPart
このフローパートでは、後続の商品マスターチェックまで行っている。

演算子クラス
CleaningOperator

入力クラス PosItem, Store,
出力クラス PosItem, ErrorPosItem

4.3. *商品マスターチェック*

JAN コード（GTIN）を参照して、有効な商品コードかどうか確認する。店舗単位で利用してる商品マスターに違いがあるため、商品コードの結合は店番号と商品コードの複合で結合する。商品コードが確認できる場合は、商品名を商品

マスターから上書きして、レシート表示名から商品マスター名に更新する。利用する演算子は MasterJoinUpdate を利用する。有効マスターの選択は、店舗マスターチェックと同様に selection 演算子を援用する。

エラー処理
エラーデータについては、ストップエラーとはしない。基本設計にあるように、商品マスターの登録ミスの可能性もあるので、処理は続行する。データ・インスタンス上には GTIN と商品名はあるので、これで処理を進める。店番号もチェック済みなので、処理が続行可能。合計金額の集計を優先し、マスター不整合についてエラーデータを作成するにとどめる。(マスター自体の不存在は、後段のカテゴリー結合で失敗する可能性があるが、「その他カテゴリー」で救済して処理をする。)したがって、エラー処理は、処理続行と分岐になる。データフロー設計シート上は、MovAndBranch という記述になっている。エラーデータはデータ・コピーを行って、異常系処理を行うと同時に、そのまま次の処理を続行させる。エラー・データにはエラー種別を付加する(Create)ので、Extend を利用する。エラーデータの型は店番エラーと同じ型になる。

Flow クラス
CleaningFlowPart
店番号チェック->商品マスターチェックのフローを処理している。商品マスターチェックについては、エラーデータを分流したあとで、正常処理を続行させるために、一度エラーデータを成功したデータに confluent して合流させて、正常フローに復帰させる。

演算子クラス
CleaningOperator

入力クラス PosItem, Item
出力クラス PosItem, ErrorPosItem

4.4. *TX のコミット*

ここでいったんトランザクションを終了して、クリーンデータを保存する。実装上はジョブフローを明示的に分離して TX を終了させる。以降のフローで処理が失敗しても、クレンジングの結果は保存されているので、自動的にロールバックさせることが可能になる。(運用上は WindGate と ThunderGate とで取り扱いが異なることに注意すること)ジョブフローはこれ以前のものが CleaningJobFlow となり、これ以降が SummerizeJobFlow になる。

4.5. *カテゴリーコード結合*

商品 ID(GTIN)をカテゴリーマスターに結合して、単品コードで必要なカテゴリーコードを付加する。付加するコードは、部門コード、上位カテゴリーコード、下位カテゴリーコードになる。

論理的には、部門・上位カテゴリー・下位カテゴリーの順でツリー構造を形成しており、最下層が単品にデータになっている。各カテゴリーコードはカテゴリーツリー内部ではユニークである。ただし現在の Asakusa では構造データを持つ事ができないので、フラットな構造になっている。通常であれば、構造データをフラットに引き直す処理が必要になる。このチュートリアルでは省略する。(RDBMS で処理をすると多層結合になるので、Asakusa 側での処理が基本になる。)

カテゴリーマスターとの結合は、演算子は MasterJoin を利用する。単品データにフラットに、そのエンティティが所属する上位エンティティを結合する。DMDL で結合モデルを定義する。(結合モデル＝JOIN_CATEGORY) 尚、プロパティのマッピングを行うので、結合後モデルを追加的に定義する。(JOINED_POS_ITEM)

エラー処理
適切なカテゴリーが見つからない場合のエラーは、商品マスター同様に、ストップエラーとせず、警告エラーとして処理をする。エラー・データはデータ・コピーを行って、異常系処理を行うと同時に、そのまま次の処理を続行させる。エラー

種別を付加するので、Extend を利用する。尚、結合すべきデータが見つからない場合は、定数で処理する。部門＝”その他”、カテゴリーコード 1＝”その他” カテゴリーコード 2＝”その他”の形で処理を続行する。その他のカテゴリーコードは Integer.MAX_VALUE とする。通常の Update 演算子で処理する。

Flow クラス
PrepareSummaryFlowPart
(フロークラスとしては、後段の消費税の計算も後続的に行っている。)カテゴリーマスターで結合して失敗したものに、デフォルトの値を設定する演算子を適用して、その後正常系に復帰させる合流処理(confluent 演算子を利用)を行い、そのまま後段の消費税計算のフローに流し込んでいる。

演算子クラス
CategoryJoinOperator

入力クラス：PosItem, Category
出力クラス:JoinedPosItem, ErrorPosItem,

4.6. *ジャーナルビルドと消費税の税剥がしと差額割り付け*

レシート単位での消費税を単品に割り振る処理。レシート総額の消費税を単品に割り当てる。外税で一括表示されている消費税額を単品に振り当てる処理。レシート単位で集計して、単品に一定のロジックで割り振る。まずレシート単位での単品をユニオンする。集合のキーは店番号＋日付＋ID＋レシート ID になる。複数キーでのユニオンは CoGoupe 演算子を用いる。集計した結果を消費税額のフィールドを Create してセットする。

計算には各単品毎の数量・売価・金額が必要であるので、これらのフィールドを Read する。この結果、レシート単位での合計値はいらないので、合計金額・消費税額・受取金額のフィールドは全部 Delete する。

計算処理は、単品税込みの金額を 105%で割り戻して、端数切り捨てで消費税額を計算。レシート単位で集計する。
単品販売金額は、販売金額-消費税額とする。
差額は最も金額が大きい単品に割り付ける。ただし、差額が、単品数量 x1 円を越える場合はレポートする。

エラー処理
消費税に不整合があった場合、すなわち単品積み上げと合計消費税額が大幅に合わない場合はエラーデータを作成し計算税額とレシート税額を表示する。通常は発生しないが、すべてのレシートデータをユニオンすることに失敗していれば発生する。エラーデータの作成は、エラー種別・レシート表示消費税と合計計算消費税を表示して、その他必要な項目を転記、エラー検出に有用な情報を String で設定する。CoGoup の出力の一つをエラー出力に割り当てる。

Flow クラス
PrepareSummaryFlowPart
入力クラス:JoinedPosItem
出力クラス：PreparedPosItem, ErrorConsumptionTax

演算子クラス
ConsumptionTaxOperator

4.7. *カテゴリー集計処理*

カテゴリーマスターの大きさが巨大でないことを利用して、各カテゴリーのコードによる集計キーで並列集計を行う。

演算処理は、通常行われる積み上げ計算を行わない。各カテゴリー階層の各カテゴリーで所属単品を並列で積み上げる方式による。したがって、理論上は本来では必要ない計算を行う。例えばカテゴリーA の所属が、カテゴリーB とカテゴリーC であり、それぞれに単品 B 群と単品 C 群が所属している場合、カテゴリーA の計算は、カテゴリーB とカテゴリーC の計算結果の和で足りる。そうではなくて、カテゴリーA の計算は、単品 B 群と単品 C 群の総計という計算を行う。これにより、カテゴリーA・B・C の計算を同時に行うことができる。力技による並列計算になる。

キーの軸は二種類作成する。一つは店舗→部門→カテゴリー1→カテゴリー2→SKU のカテゴリーツリー軸のものであり、もう一つは全店→部門→カテゴリー1→カテゴリー2→SKU のカテゴリーツリー軸のものである。前者は店舗オペレーション視点の集計であり、後者は商品カテゴリー管理軸の視点である。

カテゴリーデータの結合は既に完了しているので、集計キーで集計するだけである。よって演算子は Fold を利用する。例えば、店舗→部門→カテゴリー1→カテゴリー2→SKU のカテゴリーツリー軸であるカテゴリー2 での集計を行う場合は店舗+カテゴリー2 でキーを生成し、数量・金額・消費税額・客数を取得して、集計元の数量・金額・消費税・客数へ畳み込みを行う。ただし、客数については、ユニーク・レシート ID でカウントする。このため、Fold 時点に order として receiptId を指定する。receiptId が切り替わるタイミングでユニーク数が把握できるため、これを利用して客数をカウントする。

集計はそれぞれ、
{数量・金額・消費税・客数} X
{
・全店計(日付のみ)
・全店部門計(日付＋部門コード)
・全店カテゴリー1 計(日付＋カテゴリー1 のコード)
・全店カテゴリー2 計(日付＋カテゴリー2 のコード)
・全店 SKU 計(日付＋GTIN)

・店舗計(日付＋店舗コード)
・店舗別部門計(日付＋店舗コード＋部門コード)
・店舗別カテゴリー1 計(日付＋店舗コード＋カテゴリー1 のコード)
・店舗別カテゴリー2 計(日付＋店舗コード＋カテゴリー2 のコード)
・店舗別 SKU(日付＋店舗コード＋GTIN)
}
で行う。(内は集計のキー)

また、Fold の集計結果では各集計のキーと GTIN と商品名以外のフィールドは原則として、破壊されているので注意すること。したがって、集計結果を各単品に振る場合は、結合する必要がある。

Flow クラス
SummarizeFlowPart
まず POS データのフィールドに客数を集計するフィールドが存在しないので、Create してデータモデルを拡張しておく、SummarizeFlowPart の中で extend をコールしてデータモデルを変更する。拡張した集計用のデータを受け取って、集計演算子で処理をして、各アウトプットごとに出力を集計キーごとに分けている。出力は 10 通りになる。
すなわち、{全店、個店}X{総計、単品、部門別、カテゴリー1、カテゴリー2}の積になる。

演算子クラス
SummarizeOperator
数量・金額・消費税の積み上げ計算に加えて、客数のカウントをしている。これはユニーク・レシート ID のカウントで行っている。演算子はすべて Fold を利用している。ただし、Fold の適用がない場合は、カウントがゼロになるので、初期値として客数については 1 をセットしてる。

入力クラス：PreparedPosItem
出力クラス：Summarized

4.8. *カテゴリーデータの展開処理*

カテゴリー集計の結果を単品に結合処理を行い必要なアウトプットを生成する。カテゴリー集計での分散集計の結果は、それぞれ各数量・金額・消費税・客数のフィールドにセットされ、10 種類のデータで生成されている。それをしかるべきキーで結合する。10 種類のデータ（型はすべて同じ）を受け取って、単一のカテゴリー集計レイアウトにコンバートしたコンテナーに、必要なキーで結合処理をしていく。基本的にカテゴリーごとに順次、結合していく。

Flow クラス
FlattenSummaryFlowPart
10 種類の集計結果を受け取り、集計データに結合処理をして、カテゴリー集計レイアウトのモデルを生成する。結合戦略は以下の通りになる

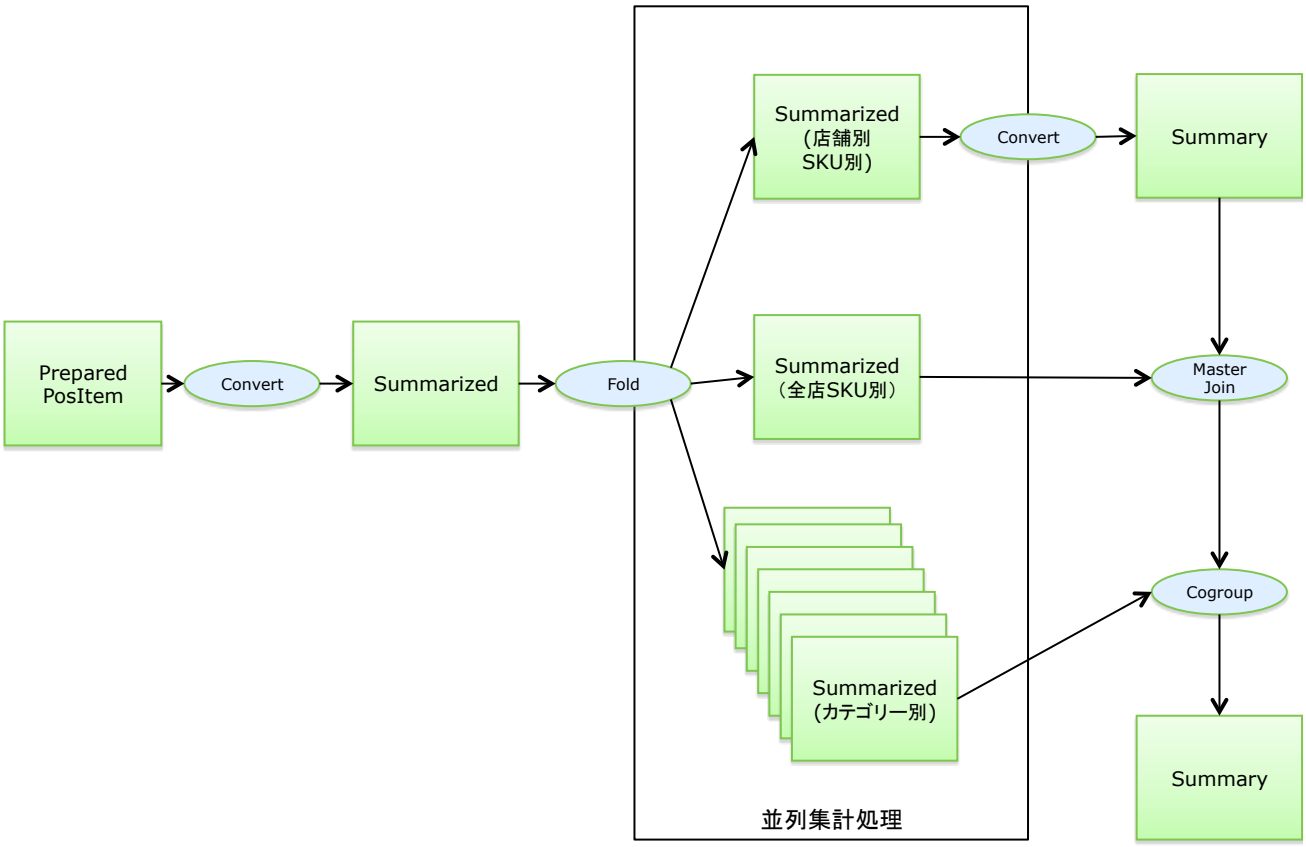

図 3

1.出力の基本ソートが店舗別 SKU 別になっているので、生成された集計データ（すべて同じ型）の中で、この集計に相当する結線の出力を軸にする。convert して集計フィールドを作成する。

2.まず全店 SKU 別の集計を結合する
これは普通に join するので、MasterJoinUpdate を利用している。日付と GTIN がキーになる。

3.次に順に、カテゴリーごとの集計結果を結合していく。このときに、全店計と個店計を一度に結合処理をする。全店計は「一つ」しかデータインスタンスがないことを利用して、値を直接取得して、すべての生成目標の店舗別 SKU 別

データインスタンスに全店計のデータを付与していく。また、個店計は、店舗番号で order を指定しておき、店舗別 SKU 別データインスタンスの店舗番号が変わるタイミングで、マッチングさせて集計データを結合する。

まず SKU 合計で、キーは日付のみ
次に部門計で、キーは日付＋部門コード
次にカテゴリー1 で、キーは日付＋カテゴリー1 のコード
次にカテゴリー2 で、キーは日付＋カテゴリー2 のコード

以上の流れで、店舗別 SKU 別のデータに集計データを結合していく。

演算子クラス
FlattenSummaryOperator
（MasterJoinUpdate で単純結合ができる）全店単品計を除いて、すべて CoGroup で処理する。

入力クラス：Summarized
出力クラス：Summary

4.9. *伝票レイアウト作成*

カテゴリー集計処理の結果のうち、店舗別・SKU 別の集計結果を入力とする。店舗・部門でグループ化する。次に GTIN 単位で order をとって、行明細データを作成する。明細行は、GTIN コード・商品名・販売数量・販売金額・消費税額の結合文字列となる。6 行単位で改ページされる。フローパート＝演算子なので、ジョブフローから直接演算子を結線する。

Flow クラス
ジョブフローから直接演算子を呼び出す。

演算子クラス
FormLayoutOperator

入力クラス:Summarized
出力クラス:FormLayout

4.10. *エラー処理の合流処理*

エラーデータを合流させる。POS データ取り込みエラーデータと店舗マスター結合処理エラーデータと商品マスターエラーをマージする。以上が CleaningJobFlow のエラーとしてレポートされる。また、カテゴリー結合エラーと消費税エラーはそのまま出力し、SummerizeJobFlow でエラーとして報告される。