異なる複数アクタを持つユースケースが存在する場合、そのユースケースはアクタごとに分割出来ないか、またあるアクタが片方のアクタの代理として振る舞っていないか検証する。さらに前提条件が存在する場合、その前提条件が他のユースケースで実現されているか検証する。

c.運用可能性の検証

最後にすべての記述されてユースケースで業務の運用が可能か検証し、可能と判断されればユースケースの抽出は終了する。

3. 4. ユースケースアクティビティ

ユースケースアクティビティはユースケース毎にその機能を実現するのに必要な業務手順(アクティビティ)に細分化し、実際に行うべき作業を明確化したものである。

アクティビティは入力項目ないし出力項目を基準に分割し記述する。その際システムの内部的な動作は記述しない。これはこの段階ではユースケースのうちどこまでをシステム化するかまだ決定していないためである。

またプロセス内にアクタによる排他的な選択肢が存在すればそれも記述する。

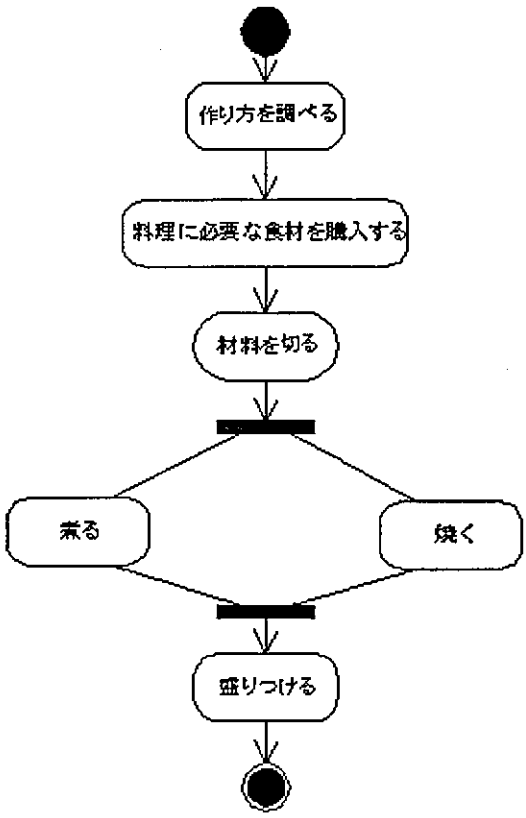

図 3.1.ユースケースアクティビティ図

3. 5. ユースケースアクティビティのチェック

a.アクタから見たアクティビティか検証

アクタ中心にアクティビティが記述出来ているかチェックする。その場合システム的なプロセス表現が含まれていたらこれを排除する必要がある。

b.アクティビティ内の項目の検証

次にアクティビティ内の項目がアクタから見て、一度に決定する項目のみで記述されているかを検討する。一度に決定できないようであれば、アクティビティの分割あるいはユースケースレベルの分割を考える。

c.曖昧な表現の排除

最後にアクティビティ内の記述で形容詞を使った曖昧な表現が使われているならば、そこで使用されている形容詞は必ず名詞を使った具体的な表現に変換し、すべて業務項目を抽出する。

3. 6. ドメイン分析

ユースケース(アクタ)間の依存関係を整理する。依存関係には次の4つがある。

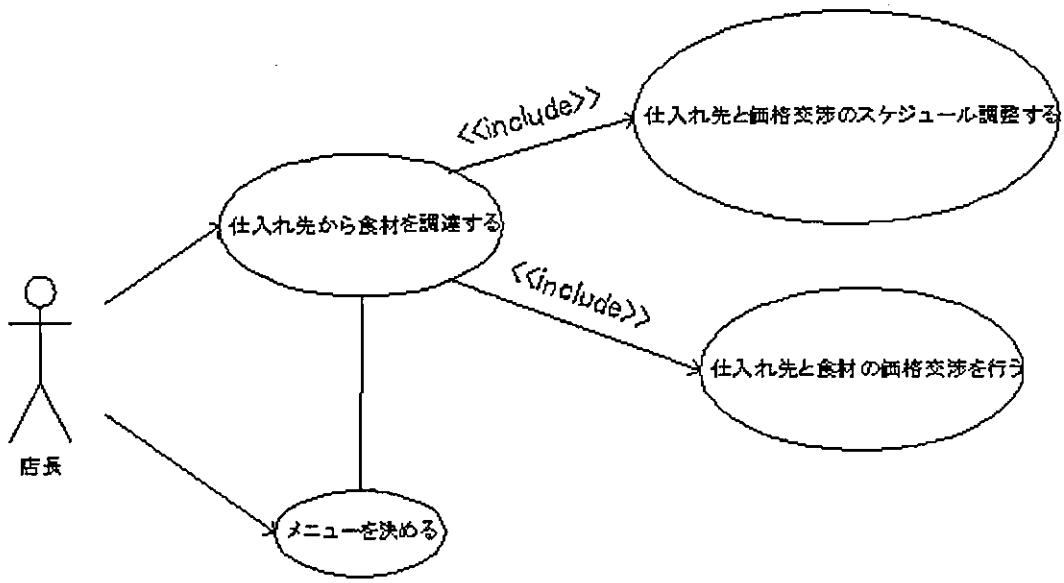

図 3.2.ドメイン分析

a.使用

あるユースケースが他のユースケースを前提としている

b.関連

あるユースケースが他のユースケースに対してメッセージを送っている。

c.包含

あるユースケースがその内部の選択肢や、選択実行プロセスとして他のユースケースを含んでいる。

d.継承

あるユースケースが他のユースケースに特別な制約を付加したもの。

3. 7. ユースケースの利点

ユースケースの利点は、ユーザ・開発者双方から理解可能であり、さらに業務が成立するか検証が可能であると同時に、あるユースケースを更新・追加した場合に影響を受けるユースケースやアクタを簡単に検証出来る点である。さらに変更がユースケース、アクティビティの変更か業務項目の追加なのかといった工数の把握が可能になる点が非常に大きい。

3. 8. カタログ作成

ユースケースアクティビティから、業務項目、業務ルールに関するカタログを作成する。カタログを作成することで、ユーザ・開発者が同一項目について異なる解釈(ロジックの独自実装)を行うのを防ぐねらいが作成の目的である。

この作業により、開発者が業務上の意味や重要度などにかかわらず、独立かつ同一レベルで扱うことが可能になる。

3. 8. 1. 業務項目の抽出

まずユースケースアクティビティに記述された名詞を抽出するのが最初の仕事である。同一名詞ながらも異なる意味で用いられているものは異なる業務項目名に変更する。逆に異なる名詞で同一の意味を与えられているものは一つに統一する。注意するべき点は業務項目名には階層構造や継承関係を与えない点である。必ず各業務項目はデータ構造としての関係とは独立であることを保証するためである。

業務項目をすべて抽出したら、各業務項目に対して、名称、多重度(単値、複数値)、定義域を持つか持たないか、そして型(整数値、実数値、日付、通番)を決定する。

上記の内容はXMLで記述される。作成されたXMLファイルは実行モデル内部で業務項目メタ情報として利用されると同時に、実行モデルはXMLからHTMLカタログを成果物として自動生成する。

3. 8. 2. 業務ロジックの抽出

他の業務項目からある業務項目を導出しているものを探す。ポイントはユースケースアクティビテ

ィにおいて入力されていないにもかかわらず出力されている項目を探すことである。その場合分析時のもれなのか、業務ルールによって決定している業務項目かを判断しながら作業を進める。

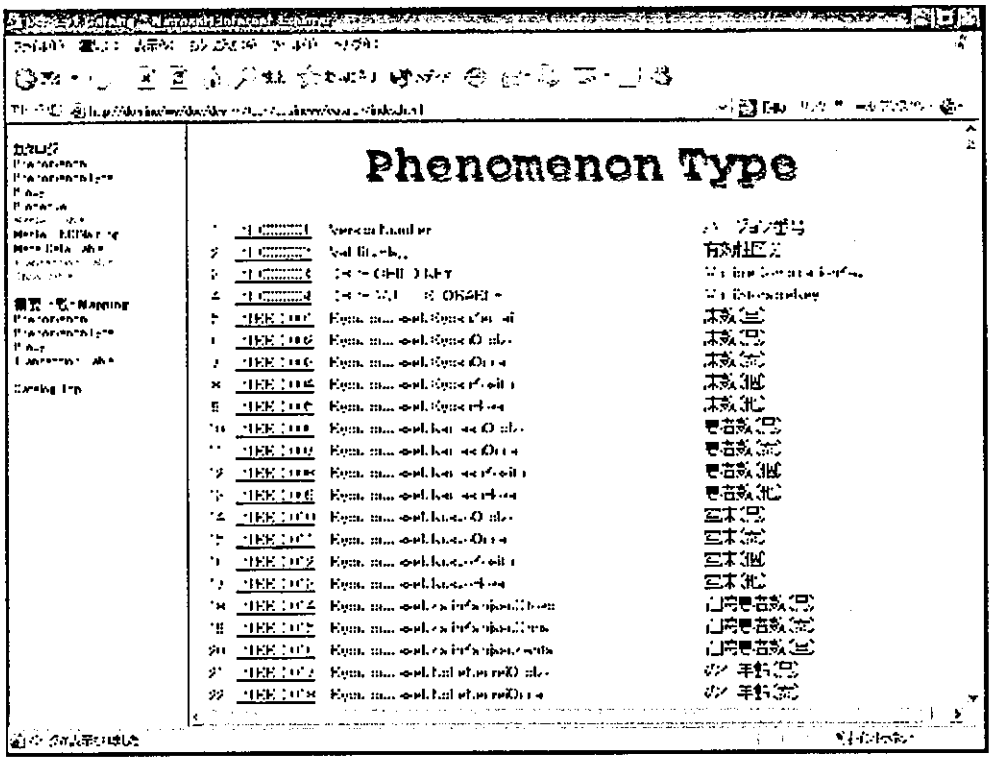

図 3.3.カタログ(現象型)

3. 9. フィージビリティー計画

これから開発しようとしているシステムの目的に照らし合わせ、各ユースケースの優先度を決定し、それに応じてそれぞれのユースケースに割り当て可能なリソースを決定する。この計画はプロジェクトで採用するハードウェア、ミドルウェアの選定、配置に利用される。計画決定に際しては顧客との合意が前提となる。

計画策定上必要な要件は、システムのスループット、スケーラビリティ、信頼性、費用対効果などがあり、費用対効果など算定が必要な項目についてはその算定法も開発者と顧客間で合意が必要になる場合がある。

3. 10. テスト計画

上記でも述べたように、ユースケースは業務を運用する上で必要な機能一覧である。よって、テストケースはユースケースを基に作成される。

各ユースケースを複数のテスト項目に分解する際、分析フェーズで記述された前提条件、終了条

件、他ユースケースとの関連性など含める必要がある。

3. 11. その他の成果物

マスタテーブル、メタデータテーブル、トランザクションテーブルをXMLを使い定義する。これらもまたHTMLに自動的に変換され、ユーザ・開発者双方から特別なツールが無くても検証出来るようにする。

4. モデル設計

4. 1. ビジネスパターン

業務項目に依存しない典型的な振る舞いのことをビジネスパターンと呼ぶ。ビジネスパターンの利点は業務を越えて再利用が可能である点である。つまりある業務に対するモデルは他業務でも再利用出来る場合が多い。たとえば酒屋さんも八百屋さんも商売をしているので、仕入れがあり、その後売り上げが発生し、その売り上げと仕入れの差が利益になるという振る舞いから見た場合同じ振る舞いであると考える。

開発者はパターンの構成名から、その振る舞いを簡単に理解できる利点が発生する。

4. 2. モデル図

ビジネスパターンに基づき、各ドメインごとに各パターン要素に対応したモデルが持つべき業務項目を決定する。同時に実行モデルでは各モデルの取得、更新、削除それぞれの手続きにおいて操作可能な業務項目をカタログ化する。

4. 3. コンポーネントアクティビティ

コンポーネントとはアーキテクチャによる分割単位である。実行モデルでは3層構造をとり、クライアント、ファサード、エンティティに分割される。

ユースケースごとに各コンポーネントごとに割り振られたプロセスや、トランザクションの管理単位や例外について記述する。

5. 実装

5. 1. 実装インターフェイス

実行モデルではシステムの拡張性を高めるために、オブジェクトから業務項目に対応する値を取得・設定する場合にオブジェクト・業務項目によらない利用可能なオブジェクトモデルならびに業務項目にアクセスするためのインターフェイスを提供している。

実行モデルでは共通に利用出来る機能に合わせ、以下のように分類している。

a.観測

データ取得に関するオブジェクトモデルを提供している。具体的には業務項目を表すオブジェクト「現象型」と、現象型に対応する結果値を格納するオブジェクト「観測」を利用して、業務項目によらない値の取得インターフェイス「観測対象」を提供している。

b.操作

データを設定・更新するオブジェクトモデルを提供している。インターフェイスの共通化によって、業務項目の追加に対して、オブジェクトの構造を変換させずに、変更をロジック部分に集中させることが出来る利点がある。

モデルにアクセスし業務項目を取り出すインターフェイスであることから、業務項目に対応した観測のコンテナとも捉えることの出来る観測対象は、内部に現象型-観測の組が格納された表を持ち、問い合わされた現象型に対応した観測を検索する。存在した場合は対応する観測を返し、存在しない場合は「不在」を返す。現象型の導入によって、業務オブジェクトと業務項目が分離される。

利点は業務項目ごとにメソッドを用意する場合と比較して、業務オブジェクトへの問い合わせを動的に行うことが出来、同時に業務オブジェクトが扱う業務項目をデータベースなどで管理することが可能になる。また検索するときに業務項目名が必須となる形態を取っているため、仕様とコードが一致する結果をもたらす。

拡張性についても項目を追加した場合、アプリケーション内部では観測対象単位で取り扱っているため、アプリケーションプログラムを修正する必要がない。つまり同一モデルであればアプリケーションは変更することなく、さらには開発依頼者毎に固有業務項目を持つことも可能である。

欠点は呼び出しのオーバーヘッドが増えることとコンパイル時に型チェックを行うことが出来ないことである。

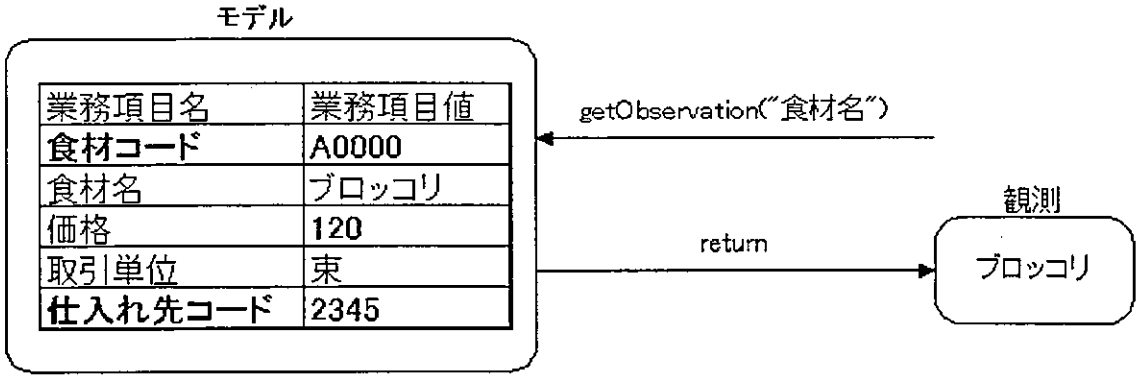

図 5.1.観測・観測対象関係図

5. 2. 手続きによる業務ロジックの実装

業務ロジックとは、ある業務項目の結果が他の業務項目値から動的に決定する場合、その導出方法を指す。つまりモデル内部で業務項目名に対応して必要な業務項目の取得ならびにロジックを実行して、業務項目に対応する値を動的に返す。

利点はアプリケーションから見たときにロジックで決定しているのか値で持っているのか区別する必要がない。またロジックを入れ替えることが可能である。決定ロジックを集中させているため、ロジックを変更することは全体を変更したことを意味する。最後に仕様の記述内容と手続きの内容が一致する。

6. その他

a.権限管理

実行モデルではユースケースごとに権限を設定できる機能を用意している。ユースケースで記述されたアクタにユースケースの機能を実行できるよう設定する。

b.J2EE

実行モデルの基盤技術に Java 2 Platform, Enterprise Edition (J2EE)を利用している。J2EE では、多層のエンタープライズアプリケーションを開発するための標準規格を定義している。J2EE は、標準化されたコンポーネントをベースにし、それらのコンポーネントにサービス一式を提供し、アプリケーションの動作を細部にわたって自動的に処理することにより、開発者は複雑なプログラミングが必要なくなり、エンタープライズアプリケーションを簡略化しながら構築出来る。

c.3 層アーキテクチャ

実行モデルの基本構造は 3 層アーキテクチャである。クライアントアプリケーションとデータストア

から業務ロジックを独立させる。具体的には

・プレゼンテーション層(クライアント)

ユーザに対する入出力、およびユーザのリクエストの発行を担当。

・業務ロジック層(ファサード)

クライアントからのリクエストの受信、結果の送信、トランザクション管理、エンティティに対する操作を担当。

・データストレージ層(エンティティ)

モデル内容の保持、ならびに業務ルールの実行を担当。

d.拡張性

実行モデルにはない拡張機能を利用したい場合でも実行モデルの枠組みを利用して拡張機能を利用することが簡単に出来る。例えば 8 章でも説明するが、CAFE では薬剤禁忌チェックのために知識処理エンジンを使用しているが、実行モデルの仕組みをそのまま利用して知識処理エンジンに接続している。

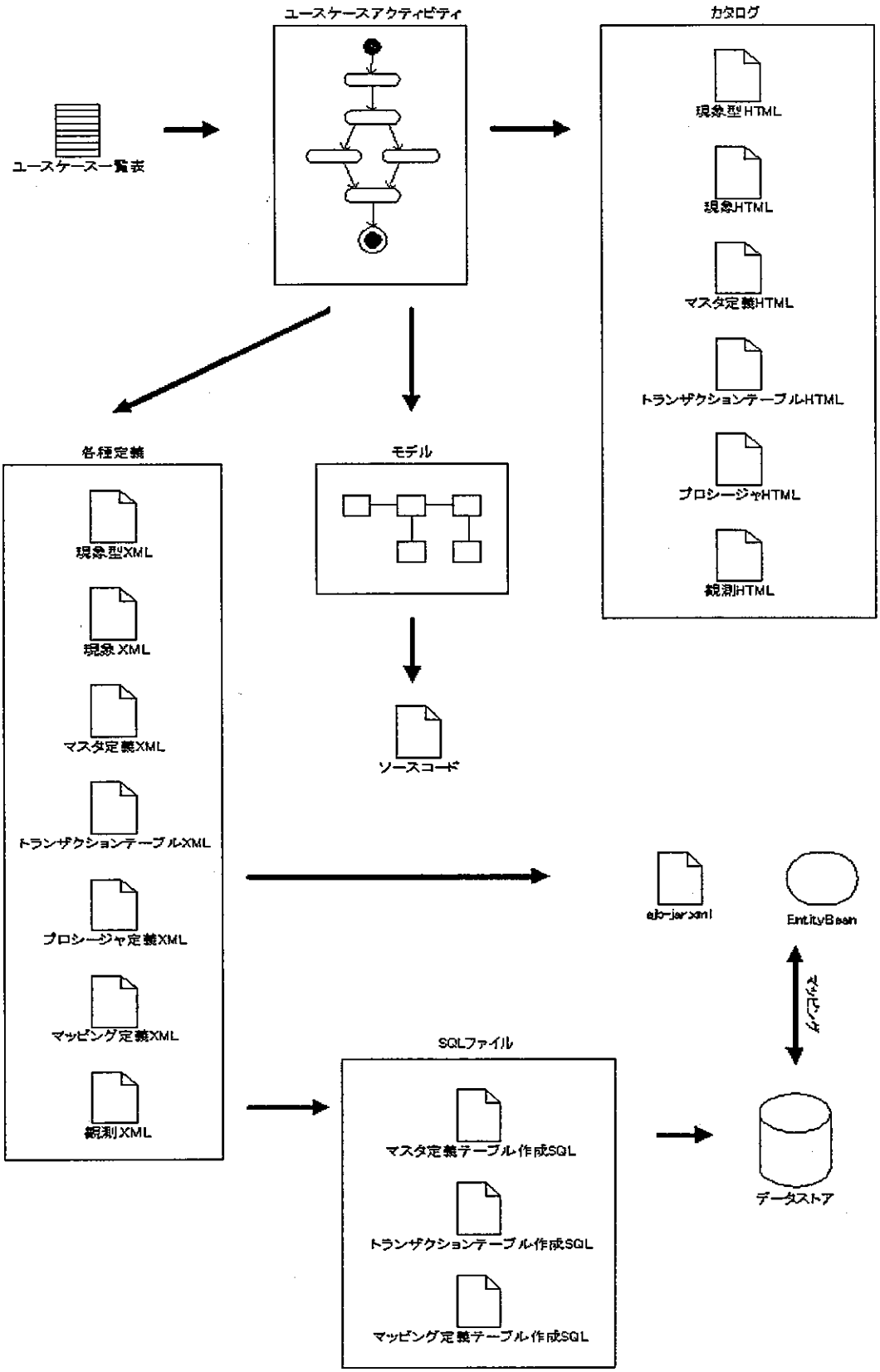

図 6.1.実行モデルフロー図

7. 実行モデルの利点

a.実行モデルの利点は各フェーズの作業内容は独立している点である。
業務項目・モデル(ビジネスパターン)・アーキテクチャはそれぞれ独立しており、再利用可能である。ビジネスパターンはそれがどの項目を扱うか、2 層、3 層などどのようなアーキテクチャ上で動作するのかについて依存していない。またアーキテクチャがどのようなモデルを扱うか関係ない。同様に業務項目はどのモデルで利用されるかは関係ない。

b.各フェーズの成果物を直接利用
各フェーズの成果物を直接利用出来る点も長所である。設計で作成し XML で記述したモデルを同じ内容のものを再度作成することなく、全フェーズで作成された成果物を再利用可能である。

8. CAFE

8. 1. CAFE とは

CAFE(Clinical Assisting Front End system for order entry)は、現在筑波大学附属病院で稼働中のオーダエントリシステムである。従来システムでは医師によってオーダが入力され、関係部門にオーダを配信するだけのシステムであった。また、従来システムではオーダに対する実績結果はシステムの能力上一定期間後すべて破棄されていた。

CAFE では全実績は入力されたオーダと関連づけて記録を行うよう、一元管理出来るよう設計されている。この仕組みが患者に対する安全で効率的な医療を提供出来ると考えられている。

8. 2. 特徴

a.すべてのオーダ・スケジュールを電子化。
b.看護師、技士によるオーダ指示受け、ならびに実施後の実績入力など、リアルタイム性を実現。
c.ハンディ端末による患者医療データ確認やオーダ実績入力のオンラインサポート。
d.医師が入力したオーダの妥当性チェックのための知識処理エンジン(DSP)を導入。
医師が入力したオーダ妥当性のチェックを行うため、なうデータ研究所が開発した知識処理エンジン(DSP)を利用して、オーダ検証を入力時リアルタイムで行っている。
業務項目 Validation を用意し、それに対する手続きとして接続。Validation の有無を確認する。知識処理エンジンに接続していない場合、結果として Validation が存在していないのと同じにな

る。上記により処方の薬剤禁忌だけでなく、あらゆる業務項目について知識処理エンジンとシームレスに結合が可能である。業務アプリケーションとしてメンテナンス、テストが可能である。

8. 3. モデル図

CAFEで使用されたモデルを下に示す

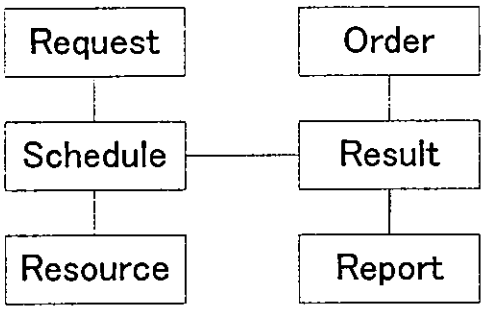

図8.1.CAFEモデル図

Order	医師が医師自身あるいは他者に対して指示出ししたもの
Request	医師が他者に対して依頼したもの
Schedule	Requestに対する予定とどのResourceを使用するか示したもの
Resource	Scheduleが引き当てる資源
Report	申込またはオーダ者以外の他者による判断内容

8. 4. 開発規模

CAFEが対象としたドメインは以下のとおりである。

一般指示オーダ	病理検査結果
注射オーダ	病理部
処方オーダ	病理部マスター管理
病棟看護	病理検査看護
薬剤部	内視鏡検査オーダ
服薬指導オーダ	内視鏡検査結果
リハビリオーダ	内視鏡検査部
リハビリテーション部門	内視鏡検査部マスター管理
透析オーダ	内視鏡検査枠
透析部	病名オーダ
給食オーダ	患者属性計算マスタ管理
栄養管理部門	処方オーダ支援マスタ管理
入院申込オーダ	処方オーダ支援
検体検査オーダ	処方オーダ検証マスタ管理
検体検査看護	処方オーダ検証
検体検査結果	検体検査要約マスタ管理
放射線オーダ	検体検査オーダ検証マスタ管理
放射線検査結果	検体検査異常値警告マスタ管理
放射部	検体検査オーダ検証
放射線部マスター管理	検体検査要約
放射線検査枠	放射線オーダ検証マスタ管理
機能検査オーダ	検体検査要約マスタ管理
機能検査結果	検体検査異常値警告
機能検査部	条件による患者検索マスタ管理
機能検査部マスタ管理	放射線オーダ検証
指導・文書予定及び実績	患者属性計算
外来診療行為	注射オーダ支援マスタ管理
外来再診予約	条件による患者検索
外来手術申込	注射オーダ支援
外来手術申込マスタ管理	注射オーダ検証マスタ管理
ホスピタルサービスオーダ	内視鏡検査オーダ検証マスタ管理
モニターオーダ	注射オーダ検証
処置オーダ	機能検査オーダ検証マスタ管理
看護部門入院処置	内視鏡検査オーダ検証
看護部門ホスピタルサービス	機能検査オーダ検証
看護部門ホスピタルサービスマスタ管	ハンディー般
看護部門モニター	ハンディ検体検査
予約センター	ハンディ看護部門ホスピタルサービス
部門外来枠管理マスタ	ハンディ注射オーダ
病棟移動会議	ハンディ看護部門入院処置
重症病棟・ICU入室申込オーダ	ハンディ看護部門モニター
退院予定	ハンディ病棟看護
病棟移動実績	権限マスター管理
手術・麻酔申込オーダ	メッセージボード
外泊予定	病床管理
病棟移動申込オーダ	

表 8.2.CAFE 全ドメイン一覧

CAFE でオーダリングについてユースケース分析、ならびにカタログ作成を行った結果、以下の各種成果物が作成された。

総ドメイン数	93
ユースケース数	1060
現象型	1966
現象	477

マスタテーブル	190
トランザクションテーブル	261

表 8.3.ユースケース数、現象型・現象数、RDB テーブル数

分析終了後、平均 20 人の開発者が約 2 年間開発に関わり、以下の実装物を生成した。

エンティティビーン数	104
クラス数	11694

表 8.4.エンティティビーン数、クラス数

8. 5. 利点

a.クライアント・サーバ間のインフラに世界的な標準技術である J2EE を使用している。この技術によりクライアントはサーバと通信できる場所であればどこでも動作する。例えば院外での利用や在宅医療に対してクライアントを立ち上げる場合でも、クライアントアプリケーションに対して特別な変更は必要とされない。

b.規模などに応じてアプリケーションサーバ、データベース製品を選択可能。例えばデータストア部分を、小規模病院なら PostgreSQL(R)、大規模病院ならば Oracle(R)というように、ユーザの規模・予算に応じて変更することが可能である。

c.現在アプレットベースのクライアントであるが、これらを Web ベースに変更することも可能である。このときサーバ側プログラムを変更する必要はない。実際 CAFE ではオーダリングをアプレットベース、ハンディ端末を Web ベースで開発している。

d.共通モデル・共通フレームワークの採用により、各病院で扱うモデル・プロセスを独自に選択することが可能である。例えば各病院で固有の項目、固有のマスタの追加が可能である。独自追加があってもモデルは同様であるため、複数の病院のデータを同時に意識せずに利用することも可能である。

e.3 層アーキテクチャを採用することで、標準的な画面、プロセスに基づく内容をパッケージ的に提供可能である。病院ごとのノウハウを組み入れた画面を独自に持つことが可能だけでなく、既存の画面と組み合わせることも可能である。さらに病院に合わせてプロセスを変更することも可能で、病院毎に固有項目、固有ルールを組み込むことが出来る。扱うオーダも選択可能であるが、オーダを追加しても全体の構造は変化しない。

9. 最後に

本稿では開発プロセス管理手法・開発フレームワークである実行モデルを紹介し、その事例として現在稼働中のオーダエントリシステム CAFE を挙げた。

実行モデルだけではシステム開発を完全に成功に導くことは出来ないものの、属人的でない部分で、プロジェクトが行うべき手順ならびにチェックリストを設定することで、開発リスクならびに変更に対して素早い対応が取れるよう考えている。

CBPF は現在筑波大学付属病院で稼働中の医事オーダエントリシステム CAFE の開発に使用され、現在稼働中である。この実例により実行モデルの有効性が検証された。

今後の課題は、XML ベースの通信プロトコルをサポートすることで、HL7 など XML ベースプロトコルとの連携を柔軟に行える仕組みを提供すること、ならびにまだ開発者に依存せざるを得ない業務ロジック実装部分でさらなる抽象化の仕組みを提供することである。

参考文献

アナリシスパターン　マーチン・ファウラー著　ピアソン・エデュケーション刊

平成 15 年度～16 年度厚生労働科学研究
「標準的電子カルテシステムのアーキテクチャ(フレームワーク)に関する研究」
総合研究報告書

(資料 7)

情報技術選択における技術参照モデルと非技術的側面

目次

1. 概要

電子カルテを実装する為には、各種の情報技術の中から適切なものを選択しなければならない。情報技術の選択を行う為にはそれらが適切に分類されている必要がある。例えば、ある機能を実装する際に用いる情報を選択する際に、製品と規格を選択枝とするのは誤りであるが、複雑な情報技術についてはそのような事が頻繁に起こっている。情報技術について適切な分類を行うことは困難であるばかりではなく、分類方法についてのコンセンサスが得られなければならない。可能であれば、分類方法／分類結果を新たに考え出すよりは、既に存在しコンセンサスが得られているものを再利用することが望ましい。

情報技術の分類方法／分類結果として、米国連邦政府の電子政府化政策の過程で開発された連邦エンタープライズ・アーキテクチャ Federal Enterprise Architecture: FEA）の中で技術参照モデル（Technical Reference Model: TRM）が策定されており、これを情報技術の分類の基準とするのが望ましいことがわかった。今後の課題として、FEA/TRM を再利用して電子カルテの技術参照モデルを開発する。

情報技術の選択を行う上では、選択対象としての情報技術自体の分析／評価だけでなく、非技術的側面として情報技術を取り巻く環境についても検討しなければならない。非技術的側面とは情報技術自身ではなく、情報技術や情報システムが開発され活用されるための環境であり、組織の構造と運営、人間の心理、経済や法律などの側面である。とりわけ、電子カルテが医療機関に導入される過程がスムーズでなければ、電子カルテの開発自体に成功したとしても、プロジェクトとしては失敗である。そこで現状における電子カルテやオーダー・エントリー・システムのような大規模な情報システムの導入過程の問題について分析を行い、対策としてエンタープライズ・アーキテクチャ等の導入を提案する。

2. 情報技術選択と技術参照モデル

2.1. 選択と分類

電子カルテを実装する為には、各種の情報技術の中から適切なものを選択しなければならない。選択は候補の一覧の中からの選択となるが、候補に洩れや混入があると適切な選択は行えない。候補の一覧の作成を行うには、対象分野についての分類方法を定義し、それに従い実在するものを分類しなければならない。適切な分類がなされていないと、本来、比較／選択を行うべきでない候補間での選択を行ってしまう可能性がある。特に、企業が提示する資料では誤った比較などがしばしば見受けられる。例えば、あるアプリケーションの宣伝的仕様書に Microsoft 社の MSXML と W3C による DOM を比較し、MSXML の方が優れており、そのアプリケーションは MSXML を採用しているので優れた機能を持

っている、という旨の記述があったとする。この例は、情報技術に精通した者であれば誤った比較であると即座にわかる。DOM、MSXML は XML 文書を処理するソフトウェアに関するものであるが、DOM は各種の実装が共通に採用すべき「仕様」について記述した文書であり、MSXML は DOM を大幅に拡張して実装した「製品」である。しかし、情報技術に精通していないものが仕様策定や製品の評価などを行う場合は、惑わされてしまうであろう。

従って、情報技術に関する適切な分類方法と分類結果（以下「分類」と略す）が必要である。 また、分類が複数存在しそれらが競合しているような状態はさらなる混乱を来す事になる。 よって、少なくとも特定の分野においては、その分野における目的を達成する為に設計された共通の分類を持たなければならない。当然ながら、この分類は広くコンセンサスを得られたものでなければならない。この分類は、電子カルテのような情報システムの調達において、調達側は要求仕様書において採用を希望する情報技術を指定するため、開発側は自らの製品／サービスが採用した情報技術を提示するため共通に参照する重要な情報となる。

2.2. 分類＝モデル

分類が単に分類結果だけを列挙するだけでは、分類の利用者が正しく理解できず誤用や悪用されるおそれがあり、コンセンサスも得られがたいであろう。分類には、背景、方法、根拠等について明確で分かりやすい説明が必要である。また、分類を十分に厳密性のあるものとするためには、分類を系統だった構造をもつ情報の集まりとして定義する必要がある。これらを達成することは即ち分類をモデルとして開発する事になる。

情報技術を分類するモデルを開発するという作業は、特定の業務や情報システムのモデルを開発するのとは全く異なったより複雑なものであり、情報技術を分類する為の新たなモデルの開発には膨大な時間と労力がかかることが容易に予想できる。また、モデルのコンセンサスの獲得にも膨大な時間と労力が必要とことが予想できる。現実的には行政や学会による承認も必要となるであろうし、情報技術は国際的なものであるためそれは国際的でなければならない。何であれ新たなモデルを情報技術の世界に不用意に導入することを主張することは、NIH （Not Invented Here） 症候群の誹りを免れないであろう。よって、既にコンセンサスが得られているモデルが存在するのであればそれを再利用するのが望ましい。調査の結果、既にそのようなモデルが存在することがわかった。米国連邦政府の電子政府化政策の中で定められた EFA/TRM がそれである。

2.3. エンタープライズ・アーキテクチャと技術参照モデル

2.3.1. Zachman Framework

エンタープライズ・アーキテクチャ (Enterprise Architecture, EA) は、1987 年に IBM Systems Journal に掲載された John. F. Zachman の論文「A Framework for Information Systems Architecture」[1]が発端とされている。アーキテクチャ（建築術、建築学、建築様式、建物、構造、構成）という言葉を情報技術や情報システムの中で使うことは Zachman 以外にもあるが、それらはどちらかと言えば「建築様式」或は「構造」という対象物に対して使われる事が多い。一方、Zachman はこの論文の中で、建築物が作られる際にどのような図面や書類が作成され、それを誰が作成し誰がそれを参照するかに注目した。建築術に対比して、アーキテクチャという言葉を情報システムの開発過程に対して与えた。

Zachman は、個人の邸宅建築の過程を例にとり、建築主、建築家、建築業者、施工業者等など建築に関係する人々がそれぞれの立場による見方（perspective）に沿って各種の図面や帳票を作成するが、それらは目的や形式が異なる（unique） ものであって、工程に従って単純に各種の仕様を詳細化するのではないとしている。同じようなことは軍用航空機の開発でも行われていることを示し、複雑な工学的製品の構築ではこの考え方が一般的であることを示した。この（建築術という意味での）アーキテクチャは情報システムの開発にも適用できることを指摘した。

次に Zachman は、それぞれの立場による見方で作成される図面や帳票自体も 1 種類ではなく、英語に於ける疑問詞（What, How, Where, Who, When, Why）に答えるため、それぞれについての記述（types of description）が必要であることを指摘した。見方と記述の二つを軸とした Zachman Framework Matrix と呼ばれる表形式の枠組（Framework）を示し、枠を埋める個々の図面や帳票を記述することが情報システムを開発する上で有効であることを示した。

この枠組について理解する上で重要な点は、枠組自体はこのような表形式をとり、行としては立場による見方、欄としては対象の様相をとるという事だけを提示しているのであって、実際に行、欄としてどのようなものを取るかについては規定しておらず、枠の中に入る記述の仕方や構成要素についても何ら指定していないということである。これは、従来の情報システムの開発技法である段階的詳細化や処理の流れやデータの構造や流れなどの一つの様相を中心に据えそれ以外の様相を付属的なものと考える各種の手法と一線を画すものであった。

2.3.2. エンタープライズ・アーキテクチャ

Zachman によって示された枠組は後に Sowa によって企業組織（エンタープライズ）にも適用が可能であることが示された[2]。この枠組を情報システムに適用する場合と組織に適用する場合の違いは、情報システムに適用する場合は、通常新たなシステムを構築する過程に適用するので目標とする表による枠組が一つだけ作られるが、組織に適用する場合は、現在の状態（baseline, as-is）から将来の状態（target, to-be）への組織の再構築する過程に適用することになるので、二つの表による枠組が作られることになる。エンタープライズ・アーキテクチャは、二つの表による枠組を持つことが一つの特徴である。

組織に適用する場合はそれに伴う情報システムの導入或は更新が意図される。そもそもこの枠組自体が情報システムの開発の為に考案されたという背景もあるが、それ以上に重要なポイントは情報システムの導入を、従来のような情報システム部門主導の活動ではなく組織全体の活動としてとらえようという所にある。これは、従来型の情報システム導入が、情報システム部門とシステムを利用する業務部門による個々のプロジェクトとして行われて来たため、重複した機能を持ちながら相互運用性の無いシステムが組織内に多数導入されてしまったため、組織全体の業務の改革／効率化を妨げるばかりか、情報技術の進歩を享受できなくなったという失敗への反省と考えられる。また、従来の情報システムの開発では実装に用いられる情報技術の選択は設計工程の下流と位置付けられてきたが、エンタープライズ・アーキテクチャでは業務分析やシステム設計と並行で進められるものであり、特に新たな情報技術の導入がビジネスをどう変えるかということが注目される。よって情報技術の選択は個々の情報システム開発者の独断で行われるのではなく、組織全体の戦略の一貫として行われることになる。

2.3.3. Federal Enterprise Architecture (FEA)

米国連邦政府 (Federal Government) における電子政府化に向けた情報システムの整備において、省庁毎の縦割りによる情報システム構築の弊害が指摘され、横断的な情報システムな構築を可能にする方法が必要とされた。各省庁の情報責任者 (CIO) による評議会 (CIO Council) は、それぞれの組織（エンタープライズ）の既存の情報資産を破棄して新たな情報システムを構築するのではなく、横断的、段階的、継続的な進化の過程を可能とする枠組 (Federal Enterprise Architecture Framework, FEAF) を開発した[3]。FEAF は、 Zachman の枠組を包含するより上位の全く異なった形の枠組である。FEAF は現在の業務／情報システムのアーキテクチャから目標とするアーキテクチャへの移行の為の枠組を示している。そしてそのアーキテクチャを記述する方法として Zachman の枠組を用いている。

FEAF に従って各省庁（エンタープライズ）は独自の EA の進化を行うわけであるが、Zachman の枠組中の記述がエンタープライズ毎に全く異なっていては FEAF が事実上機

能しなくなってしまう。そこで、概念や用語について共通に参照されるべきモデルが必要となる。 FEAPMO （Federal Enterprise Architecture Program Management Office）は、参照モデル（Reference Model）として、５つの参照モデルの開発を行っている。

2.3.4. Technical Reference Model(TRM)

FEAPMO が定めた５つの参照モデルのうち、情報技術に関する各種の規格、仕様、技術について分類を行ったものが技術参照モデル（Technical Reference Model, TRM）[4]である。 技術参照モデルは、電子政府実現の上で省庁と市民（G2C)、省庁間（G2G)、省庁と企業（G2B）の間の情報のやり取りを支える為に用いられる各種の情報技術を階層的に分類している。分類された各情報技術については簡単な説明と情報元への URL が記載されており一種のカタログのようになっている。この様な資料を完全なものにする事は膨大な労力を必要とする。実際、現時点の技術参照モデルに列挙されている情報技術は完全に網羅されたものではなく説明も簡略なものである。しかし、このような資料が公的に示されることの意義は、内容が完全でないことのマイナスを補って余り大きい。

計算機が汎用機しかなかった時代、情報技術者は計算機に関わるハードウェア／ソフトウェアについて全般的な知識を持つことはそれほど困難ではなかった。しかし、今日では多様化した情報技術について全般的な知識を維持するのは困難であり、自己の専門以外の分野についての理解が十分では無い技術者が多くなっている。実務として情報システムの開発を担当する技術者は、対象とする業務分野あるいは個々の顧客の要求についての理解を深める事に忙しく、情報技術の急速な進歩に追従できないのが実情である。情報技術の進歩に関する概観を得るために商用雑誌や安易な解説書等を情報元にする事は、商業出版物の持つ選択の恣意性を考慮すると、参考とするには問題はないが参照されるべきものではない。少なくとも情報技術者間での議論において参照される資料が公的なものでないのは問題である。技術参照モデルに限らず FEAPMO では各参照モデルについての開発／更新を継続しており、公的な機関が公的なプロセスの中でこういった参照資料を提示することは重要である。

2.4. 今後の課題

FEA/TRM は米国における電子政府化政策の一貫として開発されたものであり米国連邦政府の業務に基づいて導出されたものである。よって、FEA/TRM は情報技術一般という意味で電子カルテに流用できる部分があるが、日本の医療機関や医療情報システム産業の性質との整合性は検証されていない。そこで、本研究において業務分析から導出されるエンタープライズ・アーキテクチャと FEA の違いを検討し、FEA/TRM を基準に電子カルテに適合した技術参照モデルを開発する必要がある。