平成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」
総括研究報告書

(資料6)

MDAによる開発事例とその技術的基盤

目次

1. はじめに 3
2. 実行モデル 4
2. 1. 実行モデルとは 4
2. 2. 基本プロセス 4
3. 分析 5
3. 1. 目的 5
3. 2. ユースケース分析 5
3. 3. ユースケースのチェック 6
3. 4. ユースケースアクティビティ 7
3. 5. ユースケースアクティビティのチェック 8
3. 6. ドメイン分析 8
3. 7. ユースケースの利点 9
3. 8. カタログ作成 9
3. 9. フィージビリティー計画 10
3. 10. テスト計画 10
3. 11. その他の成果物 11
4. モデル設計 11
4. 1. ビジネスパターン 11
4. 2. モデル図 11
4. 3. コンポーネントアクティビティ 11
5. 実装 11
5. 1. 実装インターフェイス 12
5. 2. 手続きによる業務ロジックの実装 13
6. その他 13
7. 実行モデルの利点 16
8. CAFE 16

8. 1. CAFEとは 16
8. 2. 特徴 16
8. 3. モデル図 17
8. 4. 開発規模 17
8. 5. 利点 19
9. 最後に 20
参考文献 20

1. はじめに

多くの企業・団体が自らの業務を効率化または省力化しようと、毎年膨大な数のシステム開発プロジェクトが遂行され稼働を開始し、稼働後もさらなる業務効率化を求めて修正され続けている。しかしこれだけ多くのシステムが稼働しているにも関わらず、納期・品質・顧客満足度の3点をすべて満たすことの出来たシステムは数少ないと言われているのが現状である。

従来システム開発を依頼した顧客のプロジェクトで重視されていた点は完成したシステムの品質であった。品質さえ一定水準以上であれば、納期が少々遅れようとあまり問題とされないのが一般的であった。しかし近年の厳しい経済状況はそのような甘えを許さなくなってきている。厳しい経済状況は顧客の情報化投資予算への圧力となり、システム構築予算の縮小傾向に拍車をかけていることは自明である。これは開発者に対しプロジェクト予算内でシステムを開発させるために納期の厳守という新たな要求を突きつけ、品質でさえあまり保てていなかった開発現場をさらに混乱に追い込んだ。そして見逃せない点は顧客がシステムを単なる業務データの記録媒体としての扱いから顧客自身のビジネスプロセスやビジネスルールを直接扱う戦略ツールとして位置づけ始めたことである。システム上に記録された膨大なデータから未来の事業戦略はどうあるべきかを考察したり、新商品の開発に利用したりと、データの再利用を行う事例が急速に増加した。競争に勝ち抜くためにも絶えず変化が求められる現状にあるため、顧客は開発者に対してシステム上の業務ルールのタイムリーな変更を要求するようになった。追加要求がなくても品質・納期を守ることすら出来ない開発現場にとって頻繁に起こる細かな業務ルール・項目変更は、開発現場を混乱の渦に巻き込んだ。結果上記の 3 点を達成出来ないプロジェクトの増加に拍車がかかり、動かないコンピュータの見本が増産されているのが現状である。

本稿ではシステム開発の品質面を打開するため考案した開発プロセス・プログラミングフレームワークである実行モデルを紹介する。この実行モデルは仕様策定から実装に至るまで、フェーズ毎に目標、成果物、チェック項目を明確に定義したプロセスを通じ開発することで品質の高いシステムを構築し、開発途中で必ず発生するであろう仕様変更に対しても柔軟に対応出来るよう考慮された開発プロセスである。さらに分析・設計結果で作成された成果物からプログラムコードを自動生成する仕組みを備えており、分析結果と実装との間で違いが発生しにくいプロセス・構造になっている。まずこの実行モデルのプロセス、成果物、チェック項目などを解説し、どのような効果が期待できるかを説明する。そしてこの実行モデルを具体的に検証するため、実際にこの実行モデルを用いて開発された、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 CAFE を紹介し、その開発に関する分析を行う。

2. 実行モデル

2. 1. 実行モデルとは

従来のシステム開発では、要件をあいまいにしたまま開発を始めたために、成果物に対する検証が不能になることが多発した。加えて現状の検証すら出来ないため、このまま開発を進めて良いのかの判断すら不可能になることが頻繁に発生した。そのような場合、業務に詳しいエキスパートを開発チームに投入し難局を乗り切ろうとするが、結局問題解決する仕事がそのエキスパートに集中するため、エキスパート自身が開発のボトルネックとかし、開発効率を下げてしまう現象が起きた。

次に実装面を見てみると、業務ロジックの変更や業務データ項目の追加に対して非常に工数がかかるなど、拡張性に関する問題点は相変わらずであった。これは従来の業務ロジックがデータストア(リレーショナルデータベースなど)から業務オブジェクト(データ)をどのように取得・保存するかを記述する部分と、取得した業務データへの処理を記述した部分が分散していたことによるものであり、言い換えればデータストア上のテーブル構造と密接に関係していたためである。

開発上の諸問題を解決しシステムの拡張性を確保する試みとして、オブジェクト指向開発の導入が叫ばれた。結果としてオブジェクト指向開発ではデータストアから得た業務オブジェクトを隠蔽(カプセル化)し、業務オブジェクトを生成するロジック等の実装面では有効に利用された。しかしオブジェクトの再利用については成果を上げることが出来ないのが現実であった。これは業務ロジックならびに業務オブジェクトを記述する一貫した手段が提供されていなかったからである。

以下で紹介する実行モデルは、最新のオブジェクト指向開発法を取り込み、さらに一貫した業務ロジックならびに業務オブジェクトの記述法を提供することで、業務ロジックの変更や業務データ項目の追加に対して柔軟性のなるシステムが構築出来るような手段を開発者に提供する、プロセスならびに開発用フレームワークである。そして実行で開発したシステムが長期間利用可能であり、柔軟性のあるシステムとなることが最終的な目的としている。

2. 2. 基本プロセス

実行モデルでは成果物中心主義で開発を進め、各開発フェーズで成果物に対しルールとチェックリストによる検証を必ず行う。実行モデルでは開発プロセスとして以下のフェーズが定義されている。

a.分析
システムが必要とする機能を抽出し、ユーザ、開発者双方で検証可能かつ、変更影響範囲が特定可能な形式で記述する。
b.モデル設計
ビジネスパターンに基づく振る舞いベースのモデル設計を行う。
c.実装
上記の分析・設計結果を直接利用し実装する。その際業務ロジックの実装が分散しないような仕組みを提供する。

そして各フェーズで作成される成果物は必ずフェーズ関連者(ユーザ、開発者など)のいずれからも理解・検証が可能であり、他フェーズでの成果物との対応関係がトレース可能でなければならないことを求めている。成果物がこれらの条件を満たすよう、一部は UML(Unified Modeling Language)を使用している。

また変更が発生した場合は、速やかに変更が解決可能なフェーズまで戻り、変更点を修正あるいは追加し開発を再開する。これを繰り返し行い顧客の要望に迅速に対応し、求められているシステムを確実に開発していく。

3. 分析

3. 1. 目的

このフェーズでは業務を達成するためにどのような機能が必要かを抽出することが最初の目的である。言い換えればシステムのユーザが何をしようとしているのかを導き出すことが当面の目標である。そして導出された機能一覧からその後の開発に対してどのような方針で開発を進めていくかを決定する。

3. 2. ユースケース分析

ユースケースは、ユーザ(関係者、関係システム)から見たシステムが果たすべき外部機能を表現したものである。典型的な業務フロー(シナリオ)を想定し、そのシナリオに関わる人(アクタ)を見つけ記述する。
通常アクタごとに業務を行う上で必要な機能を見つけ、ユースケースを作成していく。
記述される項目は以下のものである。

a.アクタ　シナリオ関係する人もしくはもの
b.機能　業務を達成するために必要な機能
c.前提条件　シナリオを開始する前に達成しておかなければならない条件
d.終了条件　シナリオが終了する時に達成しているはずの条件

3. 3. ユースケースのチェック

業務を達成するのに必要なユースケース一覧が作成出来たら、以下の点で全ユースケースに対してチェックをかける。

a.ユースケースの妥当性の検証

ユースケースをアクタを主語にした一文にしてみて意味が通じるかを検証する。アクタが必ず存在しているか、また同一の業務レベルで記述されているか、また途中でアクタ、時間、場所が変更されず単独で利用することが可能か中心に検証する。

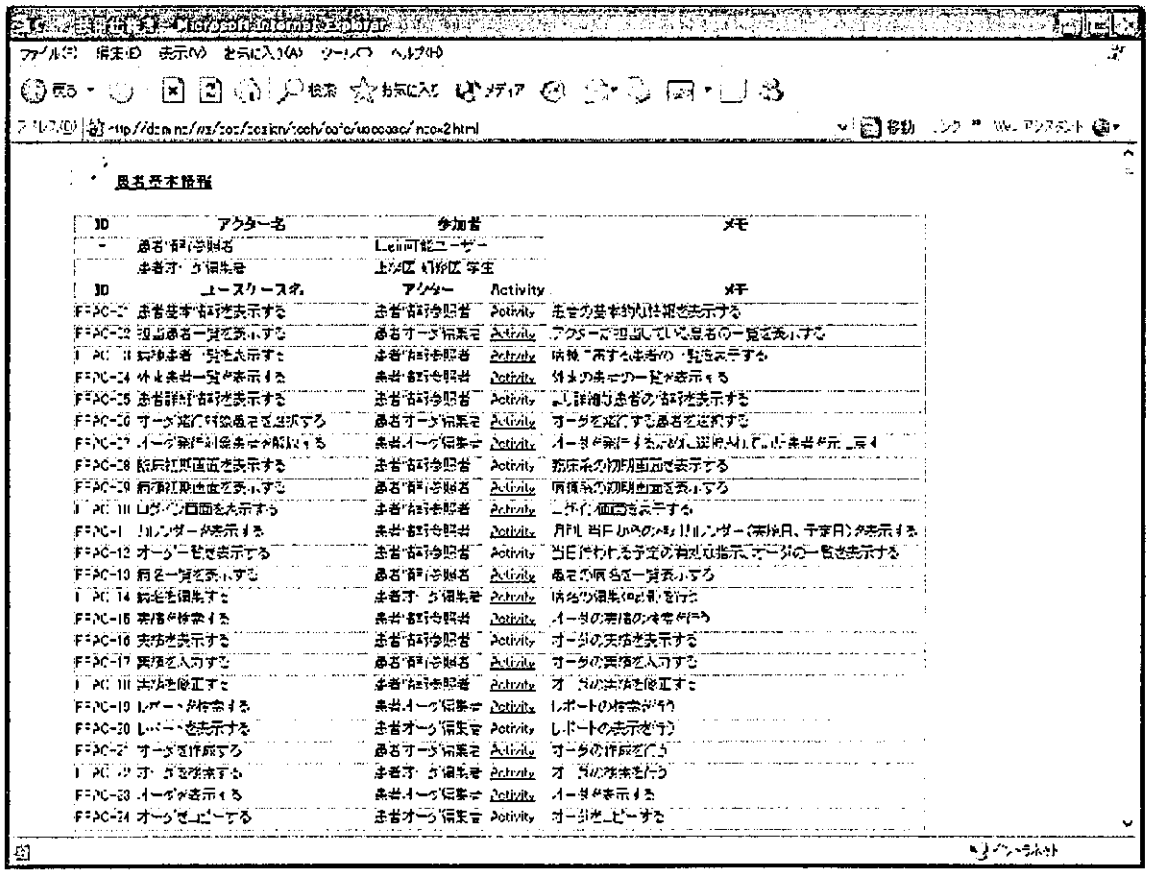

図 2.1.ユースケース一覧

b.ユースケースの混在可能性の検証

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。

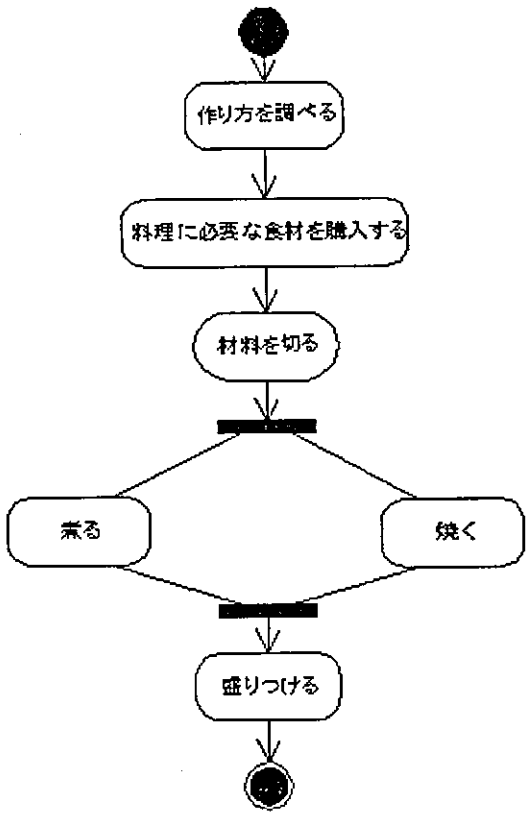

図 3.1.ユースケースアクティビティ図

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。

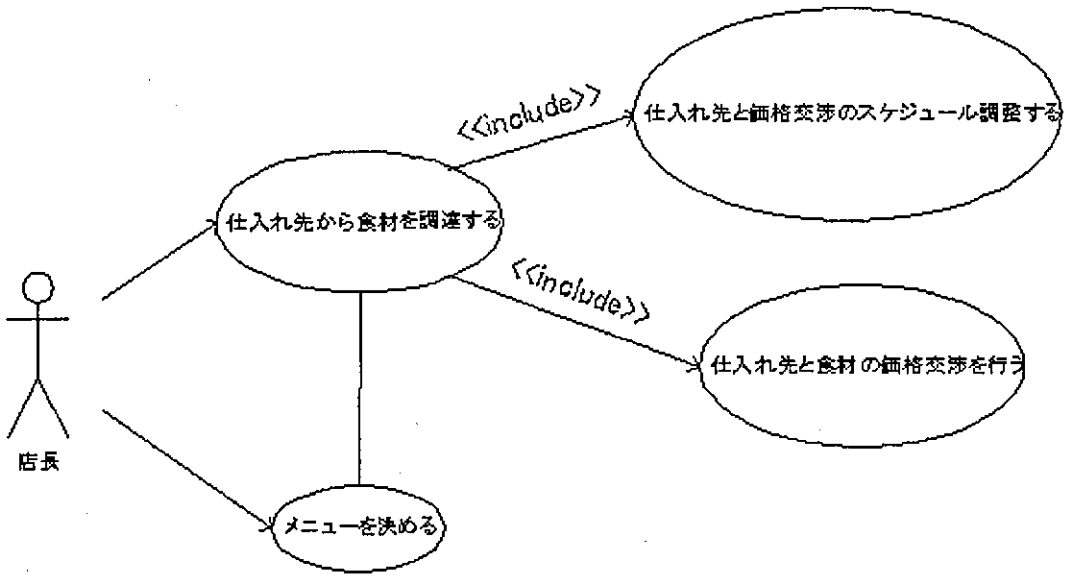

図 3.2.ドメイン分析

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。

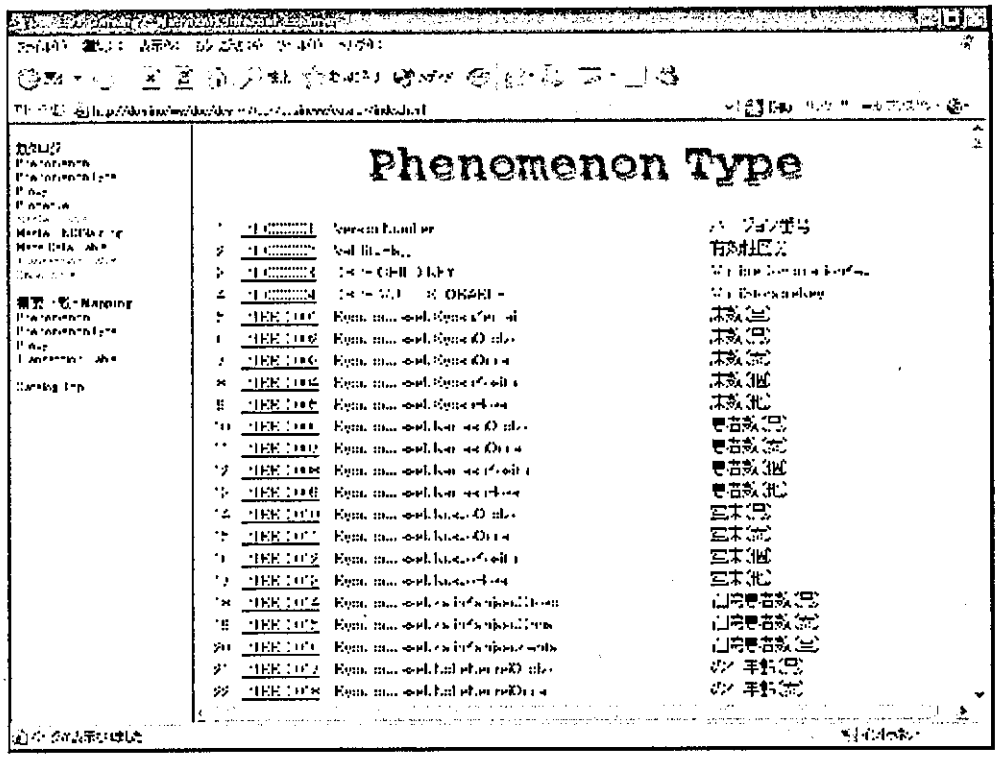

図 3.3.カタログ(現象型)

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。

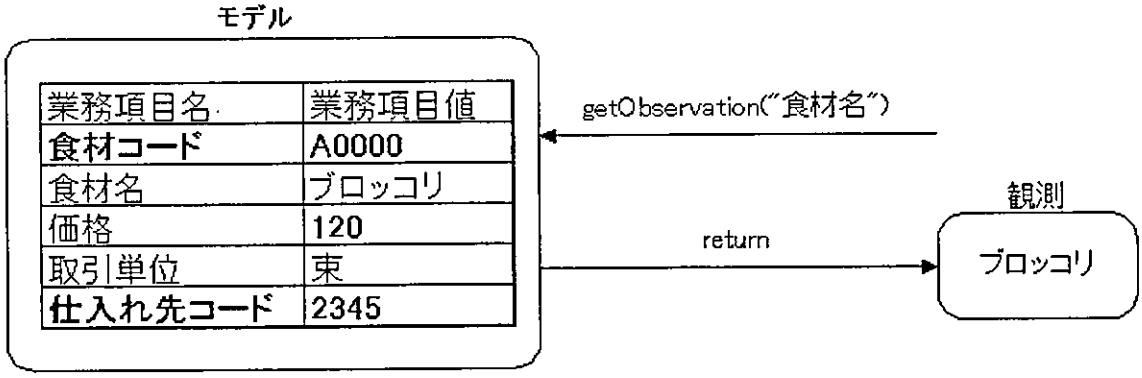

図 5.1.観測・観測対象関係図

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。

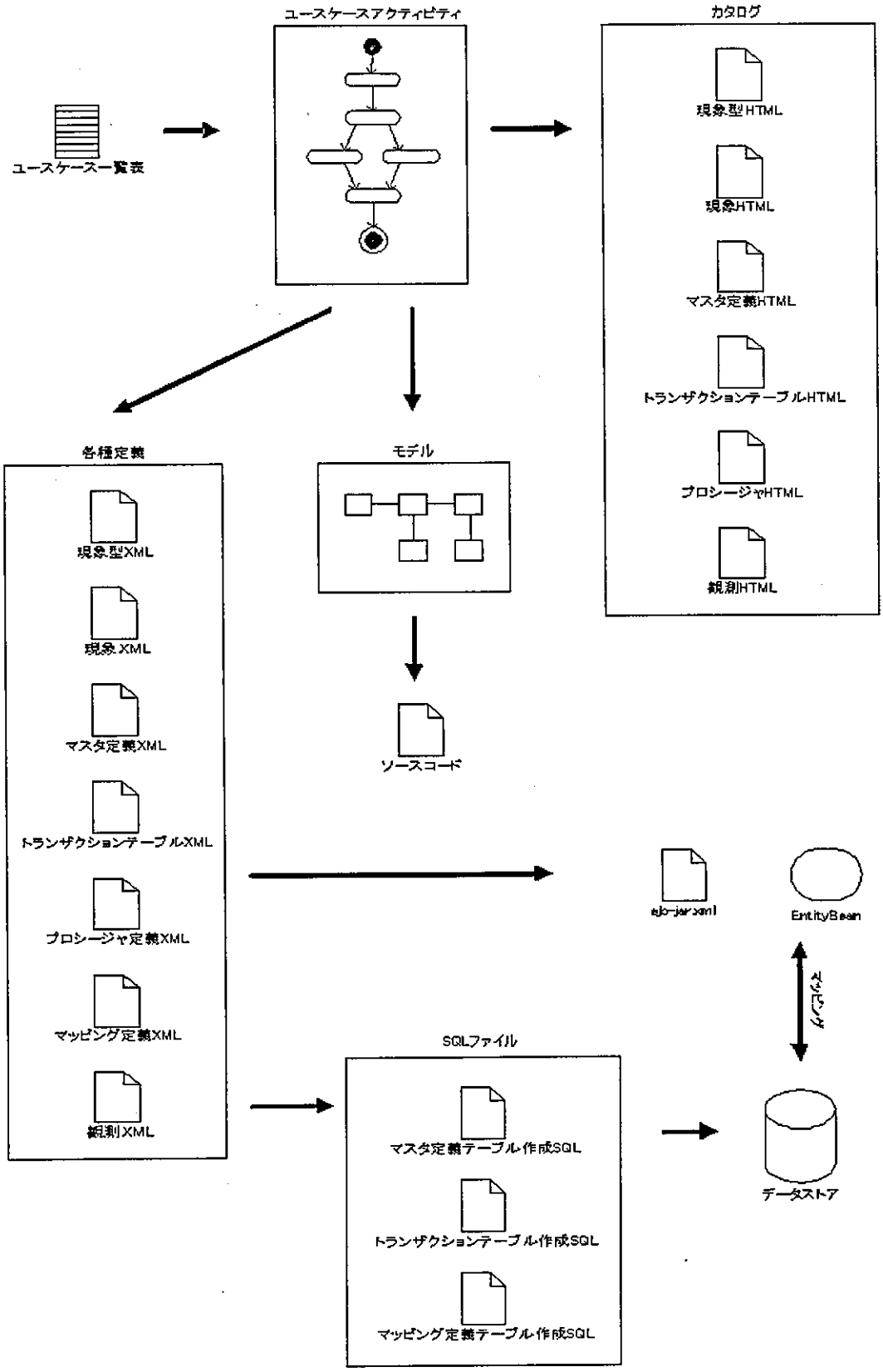

図 6.1.実行モデルフロー図

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

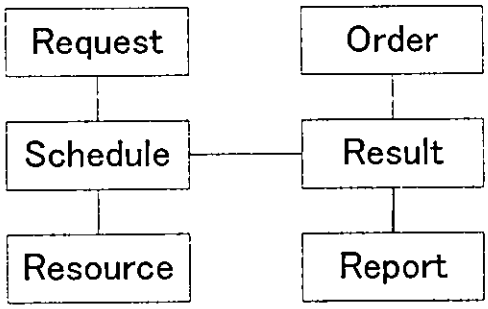

図 8.1.CAFE モデル図

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