Vol.2012-IS-120 No.3
2012/6/4

企業情報システムための早期アーキテクティングの一方法

児玉公信 †

基幹情報システムをシステムアプローチに基づいて改革的に再構築するための方法を提案する．システムアプローチでは，問題の解決に原因除去を行わない．本方法では，問題が解決された新しいシステムを構想するために業務シナリオを書いて，業務構想書（Concept of Operations）としてまとめる．それに基づいて仕事を設計し，その仕事の遂行を支援する機能を発明してユースケースとして記述する．企業情報システムを 4 つの管理階層と 3 つの業務相でドメインを分割し，その単位でドメインモデルを描く．これを中心とするようにユースケースを配置してサブシステムモジュールとし，その間を疎結合するインタフェースを設計する．こうしてできたユースケース，ドメインモデル，アーキテクチャを整合するよう調整して，要求仕様とする．

A Method for Early Architecting of Enterprise Information Systems

Kiminobu Kodama†

In this paper, a methodology based on the systems approach is proposed to reconstruct basic information systems of an enterprise. In the systems approach, any method to remove the cause to solve systemic problems will not be taken. In this methodology, instead, operation scenarios are written in order to imagine the new system the problems were solved, and those are compiled into a Concept of Operation. Next, works and workflows are created from the operation scenarios, and functions to achieve goals of the works are invented, and then the functions are described by use case format. In parallel with those processes, the conceptual models are built for each domain, which are isolated into four level hierarchy of management object and three aspects of work. Next, some modules are set gather the use cases and the suitable conceptual model, and some interfaces between the modules are designed to have loose coupling. Finally, the use cases, conceptual models, and the architecture are adjusted to be consistent with each other, and these artifacts are used as requirements specification.

1. はじめに

基幹情報システムを再構築するにあたって，企業情報システム全体のアーキテクチャから見直すことが多い．そこでは，今後想定されるビジネス環境の変化に，いかに柔軟に対応できるかが主要な課題となる．コンテクストの違いがあるので，現行資産をどう生かすかについては違いがあるものの，当該システムさえ入れ替わればよいという考えはなく，多くの場合，企業情報システムの構造を再構築することが期待されている．これは，ソフトウェアの製造工程から見ての「超上流」の活動ではない．

本稿では，システムアプローチに基づく企業情報システムのアーキテクチャ再構築の方法論として，経営レベルの要求，業務レベルの要求，機能レベルの要求と，段階的に設計し，要求仕様に至る事例を紹介する．

2. システムアプローチ

まず，システムアプローチについて述べる．

2.1 システム理論

システム理論は，1950 年代に起こった Bertalanffy[1]たちの分解還元主義に対するアンチテーゼから始まった．観察対象を分解してしまった瞬間から，それはもとの観測対象ではなくなってしまう．観察対象に見られる目的的行動は，全体とその要素間の相互作用の結果として観察しなければならない．その主張は，多くの学問分野に影響を与え，さまざまな実りをもたらした．

情報システム学にとってシステム理論は，生きている情報システムを見るための基礎理論である[2]．

2.2 設計における態度

基幹情報システム再構築において，システム理論から導かれる認識および設計の態度をシステムアプローチと呼んでおく．すなわち，システムがもつ問題もシステム自身が創発しているものであり，原因除去による解決は図れないこと．そのため，システムの目標状態をビジョン（幻想）として設定し，現状システムからそこへ向かう最もコストの低い移行経路を取り出すことになる．ただし，ビジョンを達成したとしても，新しいシステムはまた新たな問題をもつ．ビジョンと現状のギャップを，Senge[3]は創造的緊張と呼んでいる．このギャップは解消されることがない．

2.3 情報システムサイクル

情報システムサイクル[4]は，情報システムのシステム要素である主要な利害関係者の相互作用とアーティファクトを，Zachman[5]の分析を参考にして整理したものである．ここに登場する利害関係者は，Zachman が建築プロセスをモデルにしたため，「施主」「設計者」「施工者」「アーキテクト」となっている．

† （株）情報システム総研
Information Systems Institute, Ltd.

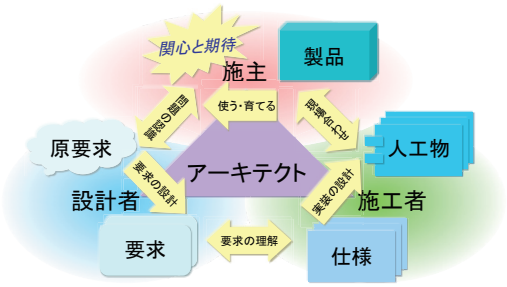

図 1　情報システムサイクル

このサイクルでは，まず施主の「関心と期待」は問題の認識プロセスを経て「原要求」に変わる．このプロセスは施主と設計者との相互作用である．「原要求」は要求の設計を経て「要求（仕様）」に変換される．これは設計者ひとりのプロフェッショナルな仕事である．「要求」は施工者との要求の理解により「（製造）仕様」に変換される．この過程で「要求」が変わってもよい．「仕様」は「もの（人工物）」に変換される．これは実装作業であり，施工者のプロフェッショナルな仕事である．「もの」は施主との間で現場合わせされ，受け入れられた「製品」は，施主が利用者に使わせることで「育てられる」．これは施主のプロフェッショナルな仕事である．これらの相互作用は，おおむね順次的に起こるが，同時並行に進むことも逆流することもある．

Senge が言ったように，情報システムにも常に問題があり，創造的緊張は解消しない．ゆえに，情報システムサイクルは，企業が存続する限り回り続ける．

情報システムサイクルは，JIS X0170「システムライフサイクルプロセス」[6)7)]のテクニカルプロセスに対応するが，使わせることによって「育てる」という施主の仕事を明示している点が重要な違いである．

2.4 アーキテクトの役割

情報システムサイクルにおいてアーキテクトは重要な役割を負う．施主の「用」，設計者の「美」，施工者の「強」に対する主張を，制約に合わせて均衡させる役割[8)]である．均衡できなかった場合，三角形はいびつになる．それぞれの主張を抑制しすぎると小さな三角形になり，抑制しないでいると，予算におさまらないほど大きな三角形になってしまう．

アーキテクト（建築家）と設計者の違いが不明確に見えるかもしれないa．実際，対象システムの規模が小さい場合には，二つの役割はよく兼務される．施工者までを兼務することもまれにある．本来の役割は，設計者は業務分析を行い，機能要求を設計する．アーキテクトは情報システムのアーキテクチャや用美強の観点から，制約を与える役割を負うと考える．

企業情報システムの再構築に当たっては，その全体のアーキテクチャを中長期的に考えておく必要がある．このための枠組みの例がエンタープライズアーキテクチャ[10)]である．この中長期的な計画を，現代的な都市計画になぞらえる[11)]こともある．

2.5 情報システムサイクルの推進者

情報システム構築プロジェクトにおいて，プロジェクトマネジャーの役割が重要であり，プロジェクトの成否を決めるとよく言われるが，それは誤りである．プロジェクトマネジャーは本来，実装工程の責任のみを負う．

情報システムサイクルは要求設計という恣意的な工程bを含み，割り切りや政治的判断が必要となる．その推進役は，プログラムマネジャーcと呼ぶべきであり，施主の立場に立つ．しかし，実際のプロジェクトの多くでは，施工者側に立つプロジェクトマネジャーに，問題認識，要求設計，全体システムのアーキテクティングの役割までを負わせようとしている．プロジェクトマネジャーに，職責以上の過大な期待がかかってしまっている．

2.6 情報システムの運用

企業情報システムは，構築よりもそれを運用し，業務を執行することで利得を生むことが本来の目的である．CIO は情報システムが有効に働き続けるよう，中長期的に取りはからう役割を負う．この様子を図 2 に示す．

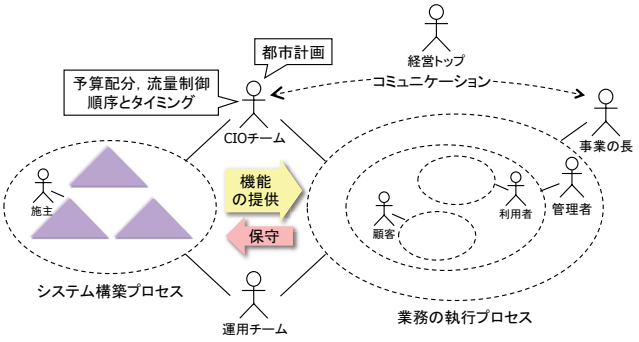

図 2　情報システムの構築と運用

2.7 アーキテクチャ再構築の方法

企業情報システム再構築の方法を一例として示す（図 3）．ここでは，施主が原要求を業務構想書の形で作成し，それに基づいて業務設計を行い，それぞれの業務の実施を支援する機能（ユースケース）を設計する．これと並行して，ドメインの概念モデル，概略のアーキテクチャ設計を行う．ユースケース，概念モデル，アーキテクチャ設計書が施工者に提示する要求仕様となる．

この方法の特徴は，問題を分析して原因除去を目指すのではなく，いきなり目標状態（to-be）を構想し，その実現を図る，いわば“構想主導”で進める点にある．以下に，概要を述べる．

a ITSS に「IT アーキテクト」という職種が定義されている．これは，施工アーキテクチャを設計する役割である．ここでいうアーキテクトは情報システムのアーキテクトである．

b 建築業界では要求設計の工程を，なんと“建築プログラミング”と呼ぶ．

c 建築業界では「コンストラクションマネジャー」と呼ばれる役割がある．施主側に立つならば，この役割がプログラムマネジャーに相当する．実際には，大手ゼネコンがこの役割を負うことが多いようである．

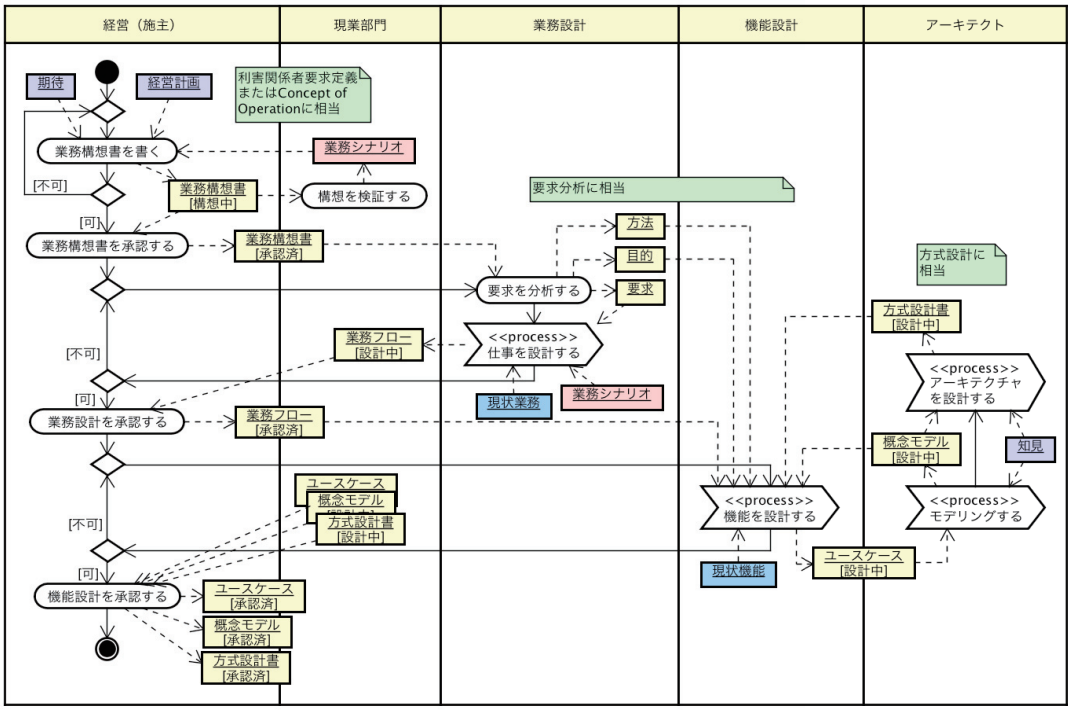

図 3　問題認識から要求設計までの業務フロー

3. 原要求を得る

原要求（proto-requirements）とは筆者の造語で，施主のビジネスレベルでの要求を指す．システムライフサイクルプロセスでは利害関係者要求定義と呼ばれ，その記述形式の一つとして，IEEE-Std 1362 Concept of Operations がある．原要求を得るための方法は次のとおり．

3.1 態度の形成

施主による業務（システム）の理想状態を構想するための手法に，ソフトシステム方法論（SSM），概念データモデル設計（CDM）などがある．

SSM では，施主が主催する何回かのワークショップをとおして，accommodation された理想システムの基本定義（根底定義，Root Definition）を得る．ここから概念活動モデルを導出し，これと現状システムの活動との比較を行って，理想システムに至る活動計画を原要求とする．

CDM はデータ中心の業務の整理手法である．現状の業務に拘泥せず，データとそのライフサイクルに着目して，あるべき業務に対する気づきを促す．静的モデリングに Crow’s Foot に似た図法を使っているので誤解されるが，実質的にデータフロー図であることに留意する．施主が主催し，現業部門の担当者が参加する何回かのワークショップをとおして，活動計画を導き，原要求とする．

ともに，問題は認識しても，原因を追求しそれを除去する議論に持ち込まない仕掛けになっている．

3.2 業務シナリオ

原因除去アプローチは，現状システムの変更なので，変更後のシステムを具体的にイメージすることができる．システムアプローチでは，これまでに経験したことのないシステムをイメージアップして，業務設計を行うことになる．このための方法が業務シナリオである．

業務シナリオは，業務のあり方をストーリ形式で表現したものである．細部にわたって具体的にイメージするために，天地人（日付，場所，ペルソナ）を具体的に定めて，あたかも実際に理想の業務が行われている様子を，対話を伴う行動シミュレーションによって，科白，独白，ト書きを使って生き生きと描き出す．

現業部門の巻き込みを兼ねて，業務シナリオの作成を部門の担当者にお願いすることになる．シナリオが局所的な記述にならないように，前もってプロット会議を開いて全体の流れを決めて分担する．担当者は最初のうちは書き渋っても，やがて止まらなくなるほど書いてしまうことが多い．実は，楽しい作業なのである．そのため，ストーリが思わぬ展開を示し，プロットの再調整が必要となることもある．

3.3 業務構想書

業務シナリオによって描き出された理想システムを，IEEE-Std 1362 Concept of Operations に準拠して整理する．これを業務構想書と呼ぶ．業務シナリオは業務構想書の一部となる．

ここに記述される業務単位（業務モード）は，業務シナリオをモデル化し，一般化したものに当たる．つまり，業務シナリオは業務単位のインスタンスである．この過程で，シナリオの内容を見直す必要が生じることもある．そのと

きは，もう一度記述し直して，実行可能性を検証する．

成文化された業務構想書は，施主によってレビューされ，権限者によって承認されて，システム再構築プログラムの憲法となる．

4. 要求分析

業務設計は，主に情報システム部門が担当する．ここでは，業務構想書を受け取って，そこに書かれている曖昧な要求事項を分析する．これを要求関連図と呼ぶ．SysML[12]の要求図を用いる．

要求図では，要求どうしを，包含，複製，導出，満足，検証，洗練，追跡などの多様な依存関係で規定するが，要求関連図では厳密な依存関係を取り出す必要はない．要求事項を，目的，要求，方法の構造として分類し，その構造を明らかにする．これによって，新システムの目的を明確化し，目的に対して，記述された要求が妥当であること，要求の過不足を確認し，要求から方法を分離して，方法は例示に過ぎないことを確認するのが主眼である．

図 4 に要求関連図の例を示す．この例のように，業務構想書に明示されない目的や要求は多い．方法にしても，短絡した結論を導いていることも多い．暗黙のコンテクストの共有があるためである．これをあえて明示し，要求の妥当性を確認したうえで優先順位を決める．対象システムが小さいときは，方法だけでなく，機能要求までが記述されることもある．取り出された方法は，業務設計によってより良い代替案に置き換え可能であることを施主と共有する．

要求関連図は，施主の業務構想書に対する設計側の回答書である．施主の承認を受けるために，何度かの往復が必要であり，業務構想書の手直しも発生しうる．

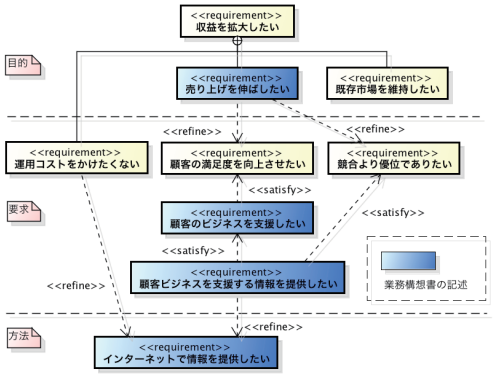

図 4　要求関連図の例

5. 仕事の定義

要求関連図の要求に基づいて，業務設計を行う．

5.1 仕事，機能，手段の定義

「もの・こと」分析[13]によれば，仕事とは「行わなければならないこと」を「体や頭を使って」行うことである．つまり，人の作業であって，機械やプログラムが行うことではない．また，行わなくてもよいこと（ムダ）は仕事ではない．仕事は，「人」が意図を持って，仕事の対象の始めの状態を終わりの状態に変えようとする目的的行為である．

状態を変えるプロセスを基本変換と呼ぶ．基本変換を実現するものが手段である．可能な手段はいくつかあり，仕事に対する制約条件に基づいて選択される．仕事の対象の型は，「もの」「人」「情報」のどれかであり，型は変換によっても変えることができない．これを「次元の一致」と呼ぶ．本方法はこの立場に立つ．特に，基本変換を機能と呼ぶ．業務設計では，この意味での仕事を設計する．機能を仕事と取り違えてはいけない．

5.2 業務フロー図の作成

業務を仕事の時間的連続でとらえる．これを，いわゆる流れ図として記述するために，UML のアクティビティ図やその拡張であるプロセス図[14]を用いる．

業務構想書に添付された業務シナリオを見て，「人」が行う情報の変換行為に着目して，仕事を取り出し，それに命名し，アクションとして流れ図に並べる．業務シナリオが書かれていない部分も想定して，業務単位を完結させる．ここで，仕事名は「○○を××する」という形式の文とする．省略されているこの文の主語は「アクタ」である．ただし，この段階では，仕事の内容は手順レベルで設計されておらず，枠ができている状態である．

プログラム流れ図とは異なり，現実の業務は規定の流れ通りに行われないこともよくあるし，想定外のことも起こりうる．すべての事象を書き尽くすことはできない．書くことができるのは，可能な流れのうちのわずかであり，そこには多くの前提や見なしがある．業務フローには，このような前提や見なし事項を注釈の形で書き加えていく．業務フロー図は，フローの厳密な規定ではなく，仕事を認識するためのワークシートである．

5.3 Customer-Performer の関係

業務フロー図を書く際の重要な観点がある．ある仕事が自分一人で遂行できない場合に，それを誰かに手伝ってもらいたい．そこには，仕事を依頼する人（Customer）とそれを引き受けて実行する人（Performer）の関係がある．両者の間では特徴的なプロトコルが交わされる．それが，提案，合意，実行，満足の流れ（図 5）である[15]．これを Customer-Performer 関係と呼ぶ．

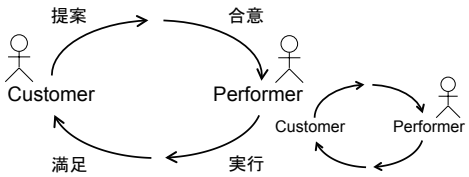

図 5　Customer-Performer 関係

Customer が依頼しようとする内容が Performer に拒否されることがある．Performer が合意した以上は実行する責任（accountability）が生ずる．実行結果は Customer に報告さ

れ，それが満足できなければ Performer にやり直しをしじする．提案，合意，実行，満足の行為は，それぞれさらに下位の Performer に依頼することができ，依頼の分岐と連鎖が発生する．

この性質を基に，業務フロー図の表記ルールを次のように定める．

5.3.1 顧客満足の原則

業務フローは必ず，それを開始したアクタに戻って完結すること．フローは Customer の満足をもって完結する．途中キャンセルなどを除いて，フローが他のアクタで終わってはならない．

5.3.2 緩やかな提案

Winograd[16]は A(Customer)と B(Performer)の 2 者間のやり取りで起こりうる状態遷移を整理している．これは 9 つの状態からなるモデルである．しかし，日本では提案の拘束が緩やかであり，合意の前に変更されることがよくある．たとえば，内示と呼ばれる状態である．この状況に対応するため，オリジナルの状態機械図に対し，合意成立の前に「確定なしの提案（Request without confirmation）」と「合意なしの受け入れ（Accept）」後の 2 状態を追加した．それが図 6 である．追加分を赤字で示している．

この状態機械図を参考にすることで，仕事の意味を明確に整理し，抽出漏れを防ぐことにつながる．

5.3.3 営業活動

Customer-Performer 関係は，他者に対する仕事の依頼のモデルである．たとえば，商品の注文から納入までは表現できる．納入に対する支払いも，満足に暗黙に含まれている．この関係は，Customer が最初に提案をしなければ始まらない．しかし，たとえば営業やマーケティングの活動は，顧客から依頼されるものではない．顧客に対して注文を促す働きかけである．この場合の仕事は社内活動であり，営業担当自身が Customer とある業務フローができる．

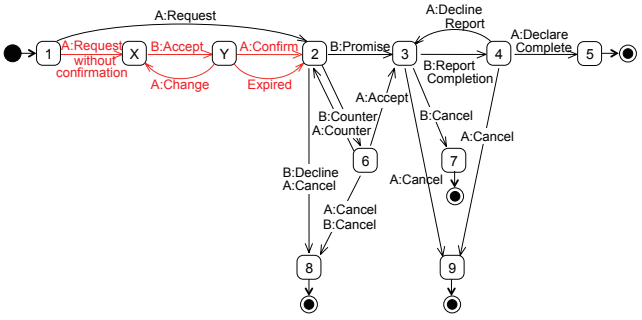

図 6　日本における 2 者間の交渉プロトコル

5.4 ワークフロー自動化とのずれ

Customer-Performer 関係に基づく業務フローと，ワークフロー自動化でいう処理の流れは必ずしも一致しない．たとえば，①注文を担当者が受け付けて，②上司の承認を得た後，③担当者が出荷指示をする，という仕事の流れがあったとする．業務フローの原則から，②の上司承認の結果は担当者に戻される．この上司承認は付加価値を生まない無駄な仕事である．しかし内部統制上必要な仕事であるので，その仕事にできるだけコストをかけないようにする．仕事の意味を考慮した業務フローの設計は，このように無駄に対する気づきを与えてくれる．

一方，ワークフロー自動化では，②の上司承認後は直ちに自動出荷指示を行うようにフローを組むだろう．担当者は受付機械にすぎず，上司承認が無駄かもしれないという検討の観点がなくなってしまう．

業務フローの設計は，ワークフロー自動化の設計とは全く別の作業である．ワークフロー自動化は，機能設計後に，機能の実装方法の一つとして考えるのがよい．

6. 機能の定義

次に，施主に承認された業務フローを基に，機能設計者（主に情報システム部門）が機能設計を行う．機能とは，上で述べたようにアクタの仕事を可能にする基本変換のことである．仕事の対象である情報の始めの状態を終わりの状態に変える操作である．ただし，ここでは手段を規定しない．機能を定義する前に手段を考えると，機能定義が不純になってしまうからである．手段の規定は実装の直前で，施工者の立場で行う．

6.1 機能の発明

必要な機能は論理的に導かれるというよりは，むしろ発明と言える．設計者の能力が問われる作業である．

業務フロー図の仕事ごとに，機能を発明し，注釈記号にステレオタイプ《usecase》を付けて，機能名を記述していく[17]．図 7 に図書館で本を借りる例を示す．

機能名は「○○を××する」という形式の文とする．仕事名と似ているが，省略されている主語は「情報システム」である．したがって，機能名の動詞は，「記録する」「表示する」「集計する」など決まり切ったものになる．

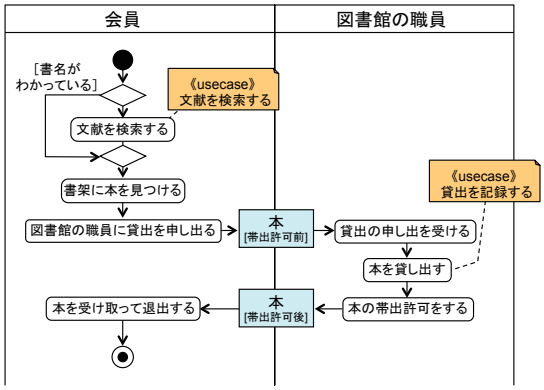

図 7　業務フローと機能定義

6.2 ユースケース記述

機能定義は，手段を設計しないシンプルなユースケース記述[17]を用いる．そのサンプルを図 8 に示す．性能やユー

ザビリティに関わる要求は備考に記述する．

基本系列では，基本変換を手順化し，アクタとシステムの一回のやり取りが，安定した中間状態を作るように書く．ここで扱われる情報は，ドメインモデルの概念名や属性名と一致させ，不足がある場合はドメインモデルに反映していく．この段階でドメインモデルができていない場合は，ユースケース記述と並行してドメインモデリングを行い，記述内容と突き合わせる．

ユースケース名：本の貸出を記録する
アクタ：図書館の職員
目的：本を紛失したときに追跡したい。
事前条件：貸出した事実が記録されていない。
事後条件：貸出した事実が記録されている。
基本系列：
①アクタがこのユースケースを起動する。
②システムは利用者と貸出そうとする本の識別を要求する。
③アクタはそれらの識別を提示する。
④システムはその利用者が貸出上限を越えていないことを確認して，その本が本日貸出されたことを記録する。
代替系列：
①基本系列④で利用者が貸出上限を超えている場合は貸出しを拒否する。
備考：
①利用者の識別は利用者カードで行う。
②本の識別は本に埋め込んだICタグで行う。
③利用者の本人確認は，利用者カードの写真によってアクタが行う。
シナリオ：
①利用者である児玉公信は，9月14日に「UMLモデリングの本質」を借りに来た。しかし，その本は現在貸出中だったので，他の図書館を紹介した。

図 8　ユースケース記述のサンプル

6.3 仕事を変える機能

本来，仕事を設計してから，それを支援する機能を考えるが，よい機能が発明されると仕事自体が変わることがある．その場合は，仕事を変更し，業務フローを変更する．

7. ドメインモデリング

ドメインモデルは機能の内容に基づいて作成するが，アーキテクトはドメインモデルの経験や蓄積があり，まったくの白紙からモデルを書くことはない．むしろ，他のドメインモデルとの比較によって，新しいドメインモデルの特徴が見いだされ，より良いモデルが構築できる．ここでは，要求変更に柔軟に対応できるような概念モデルを得るための手法を述べる．

7.1 管理階層

ドメインモデルを一枚岩で書くべきではない．企業システムは，いくつかの管理階層に自己組織化されている．情報が同じ名称だからといって，同じ概念であるとは限らない．たとえば，「製品」という情報が，製造現場では一つひとつの現物としての製品を指すのに対し，経営者は製品群を指しているかもしれない．意味の違いは視点の違いに起因する．

管理階層は，管理の主体と管理対象のライフサイクルに基づいている．図 9 は管理階層分けの例である．ここでは，顧客，案件，執行，制御の 4 つの管理階層に分けた [18]．管理対象はそれぞれ，利益，注文，製品，ワーク（製造行為の対象）であり，互いに関連しながらも，異なるライフサイクルをもっている．このように分離することで，階層ごとのモデルはよりシンプルになる．

7.2 相別化

さらに管理階層を業務の相に分ける [19]．知識相は，実行計画を立案するための基礎情報をもつ．たとえば，執行階層においては，製品の作り方を述べ，案件階層においては商品の成り立ちを述べる．これらは，本質的にはルールであり，複数の型からなる構造をもっている．

計画・実行相は，直下の管理階層の計画・実績相に指示を与え，その実績を受け取る．この相では，現在と未来の情報が扱われる．上位層が Customer，下位層が Performer の関係になっている．

報告相は，計画・実績の成績を見るために，過去のデータを扱う．原価管理や会計報告を行う．報告相の都合のために，計画・実績相に不純な情報を付け加えてはならない．

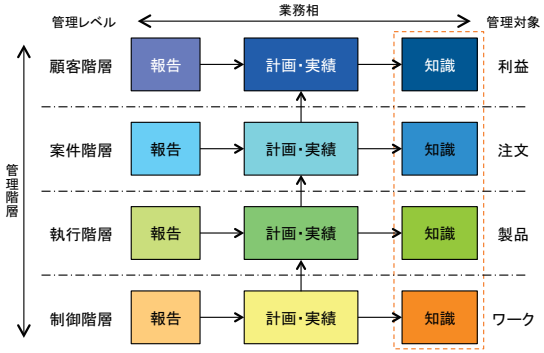

図 9　管理階層と業務相

7.3 階層別モデル間の連携

階層および相ごとに作成したドメインモデルは，単独では基幹システムの機能要求を満足しない．これらのモデル間を連携させることで，全体システムが機能する．モデル間の参照関係は，最小で一方向とする．図 10 は計画・実績相の各階層のモデルが，一つの型をとおして一方向に参照している例である．

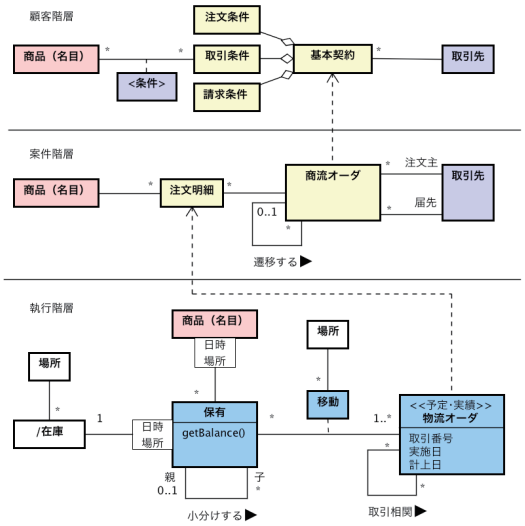

図 10　階層別モデル間の連携

8. アーキテクチャの設計

以上の設計内容を統合して，全体システムの論理構造を設計する．

8.1 階層の実現

管理階層と業務相で切り離されたドメインモデルを，そのままドメインモジュールとして実装する．それぞれを中心に，関連するユースケースを配置する．これによって凝集度の高いモジュールが形成できる．モジュール間を一方向の参照インタフェースによって，疎に連携させる．こうして，システム要素が独立して動作しながら，大局的に連携して動作する基盤環境を確立する．これを各モジュールの実装に先行して提供できれば，継続的結合テストが可能となり，モジュールごとの分離並行開発が可能になる．

階層別モジュールとモジュール間インタフェース設計の例を図 11 に示す．

8.2 インタフェースの設計と実装

モジュール間の参照を，非同期メッセージによるデータ結合[20]のインタフェースとし，トランザクションをモジュール内に留める一方，トランザクションリカバリに配慮する．メッセージは XML などでエンコードし，状態遷移は上述した Winograd の 2 者間の交渉プロトコルに準ずる．

一方向の参照関係を徹底するために，たとえば，図 11 中の getXOR（eXecution Order Result）インタフェースは，執行モジュールがインタフェースを提供し，案件モジュールから呼び出すようにするが，これには MVC パタン[21]のシーケンスを内蔵し，逆方向の参照である update インタフェースを隠蔽する．

インタフェースの実装に当たっては，モジュールの実装言語が異なっていても通信可能となるよう，ESB（Enterprise Service Bus）製品上に実装する．ただし，トランザクションの性格上，通信の高速性を確保しなければならない場合には，本来の設計を尊重しつつ，リモートメソッドコールを使用するなどの実装を工夫する．レガシーシステムとの通信にはバッチ転送などに退行させる．

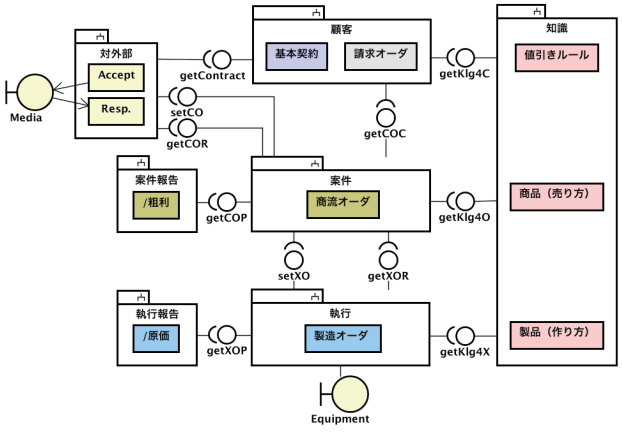

図 11　システム間インタフェースの定義

8.3 外部複雑性の分離

企業システムは，必ず顧客のシステムと連携する．顧客も自社の情報システムをもっており，そのビジネス，意味論，モデル，表現，識別が異なっている．日付や住所でさえ，さまざまなフォーマットをもつ．商品や取引形態がますます多様化する中で，顧客に価値を提供するために，受注側は，顧客ごとに異なるメッセージとプロトコルを受け入れ，かつ応答せざるを得ないd．このような過剰なバリエーションを外部複雑性と呼ぶ．

外部複雑性を絶対にシステムの内部に侵入させない．顧客ごとのバリエーションは，図 11 の対外部モジュールのように，それを受け入れるモジュールで内部標準形式にデコードし，システム内で処理した後，応答するモジュールで再び顧客ごとの外部形式にエンコードする．

顧客ごとのバリエーションの条件は，基本契約の条項で述べられなければならない．従来，このような顧客ごとの契約条項への対応は，プログラム内に顧客個別のロジックとして組み込むことが多かったが，本アーキテクチャでは基本契約型の属性値，すなわちデータとして変換ロジックを記述（外部化）し，それを実行時に評価して適用する．

8.4 共通基本情報のサービス化

共通基本情報とは，取引先，商品，基本契約など，これまで「マスタ（ファイル）」と呼ばれていたデータのことである．マスタは実装形式であり，モデル上では存在する「もの」である．しかし，取引先は資本関係に基づく構造や内部の組織構造をもち，商品は組合せ商品などの構造やルール表現をもつ．さらには変更履歴をもつこともある．こうした構造やバージョンの扱いも，従来は複数のプログラム内のロジックで処理されていた．

本アーキテクチャでは，強い構造で関連する基本情報の型をモジュール化し，外部に対して問合せサービスを提供する．そのインタフェースは，Publisher-Subscriber[21]あるいはそれに準ずるプロトコルで実装される．共通基本情報は頻繁に参照されるので，Subscriber 側が必要に応じて内容をキャッシュすることも許す．この場合，更新同期や版管理の責任は Subscriber 側にある．

8.5 システム管理機能

システム管理機能とは，情報システムの運転および運用管理に関わる機能である．たとえばユーザアカウント管理，アクセス権限管理，ジョブスケジューリング，データおよびプログラムのバックアップ／リストア，運転統計などの機能である．これらは業務機能から独立させ，連携する場合でも疎結合するように設計する．

輻輳管理や容量管理など，計算機システムの維持に関わる機能は，クラウドサービスを利用することで，大幅に肩代わりさせられる．

d EDI（電子商取引）はこれを共通化することで，受発注者間のプロトコル負荷の均等化を狙っていた．

9. 施工者への提案要請

9.1 要求仕様の 3 点セット

こうして発注者側がモジュールごとに作成するユースケース記述，ドメインモデル，および基本方式設計書の 3 点セットは，施工者に提示する要求仕様となる．これらは相互に整合している必要があり，それぞれの設計が進むにつれて，他の内容の連鎖的変更を促す．変更が安定すれば要求設計は終了し，施工者に引き渡すことができる．この 3 点セットは，共通フレーム 2007[22]でいうソフトウェア要件定義の入力となり，施工者に対する提案要請（RFP）に添付する．発注の最小単位はモジュールである．

要求仕様は，設計者が納得できれば，施工者の提案によって部分的に調整することもできる．

施工者の作業範囲は，システムライフサイクルプロセスでいう実装プロセスである．すなわち，結合，検証，以降，妥当性確認は，施工者の支援を得るかもしれないが，発注側の責任作業となる．

9.2 施工方法

要求仕様の内容は新しいが，既存のソフトウェアを捨てて，すべて新しくするかどうかまでを規定しているわけではない．施工者が，要求仕様を満たすうえで，既存のプログラムやデータを生かした方が，品質（仕様を満足していることおよび欠陥がないこと），コスト，納期から見て有利であると判断すれば，既存の資産を使うことを提案すればよい．この場合は，既存の資産をよく知っていることが条件であり，施工チームが社内にいて，日頃から保守や拡張を行っている場合に可能である．

パッケージ製品を使ってもよい．パッケージに不足する機能は，設計者と要求仕様の調整を試みて，折り合わなければ追加開発になる．このとき，保守性を確保するために，パッケージをカスタマイズするのではなく，データ結合するようにパッケージの外部に作ること．

移行方法も開発プロセスも，施工者の提案による．

9.3 機能規模の計測

ユースケースとドメインモデルからファンクションポイント（FP）を計測できる [23]．提案要請の際に，設計者が FP を計測しておき，提案書に FP 値の記載もお願いしておくことで，施工者側が理解した機能規模と比較できる．施工者側との FP 値の差の原因を確認することで，要求設計の弱点，施工者側の不安要素が見えてくる．

念のために述べておくが，FP 値は機能規模の指標であり，非機能要求は含まれない．また開発規模（工数や金額）とも直接連動しない．むしろ，大きな機能を，工夫してより少ない金額で作ることが望ましい．

10. おわりに

基幹情報システムの再構築における原要求の生成から要求仕様作成までの方法を一例として示した．しかし，その作業内容以上に重要なことがある．施主，設計者，アーキテクト，そしてプログラムマネジャーと CIO が，それぞれに役割を果たさなければならないことである．特に，施主の責任はこれまでよりはるかに重い．

今日，情報システムの停滞はビジネスの停滞につながる．企業情報システムが，将来にわたって変化し続けるためには，その情報システムサイクルを支えるシステムが持続可能でなければならず，そのためにこそ，システムは組織と人を再生産できる構造を内包するよう設計しなければならない．

参考文献

1) von Bertalanffy: The General System Theory, George Braziller (1968) 長野ほか訳：一般システム理論，みすず書房（1973）
2) 児玉公信：リッチピクチャと因果ループ図の補完的使用について，情報システムと社会環境研究会 IS106-6，2008/12/2（2008）
3) Senge, P.: The Fifth Discipline，Currency Doubleday (1990) 守部訳：最強組織の法則，徳間書店（1995）
4) 児玉公信：情報システムサイクルと原要求の記述，日本情報経営学会誌, Vol. 28(2), pp.77-87（2007）
5) Zachman, J. A.: A Framework for information systems architecture, IBM SYSTEMS JOURNAL, Vol. 26(3), pp.276-285 (1987)
6) JIS X0170:2004（ISO/IEC 15288:2002）システムライフサイクルプロセス
7) ISO/IEC 15288:2008 System life cycle processes
8) Vitruvius 著，森田慶一訳：ウィトルーウィウス建築書，東海大学出版会（1979）
9) JIS X 0160: 1996 及び追補 1: 2007 ソフトウェアライフサイクルプロセス
10) The Open Group: TOGAF version 9, http://www.opengroup.or.jp/togaf.html
11) 南波幸雄：企業情報システムアーキテクチャ，翔泳社, pp.86-106 (2009)
12) OMG Systems Modeling Language (SysML) Version 1.3, http://www.omg.org/spec/SysML/1.3/
13) 中村善太郎：シンプルな仕事の構想法，日刊工業新聞（1992）
14) Eriksson and Penker 著，本位田ほか（監訳）：UML によるビジネスモデリング，ソフトバンク（2002）
15) Medina-Mora, et al: The Action Workflow Approach to Workflow Management Technology, Proc. of CSCW 92, pp.1-10 (1992)
16) Winograd, T.: A Language/Action Perspective on the Design of Cooperative Work, HUMAN-COMPUTER INTERACTION, Vol.3, pp.3-30 (1987-1988)
17) 児玉公信：ユースケースの使い方に関する提案，情報システムと社会環境研究会報告 IS105-7, 2008/8/29
18) IEC 62264-1:2003, Enterprise - Control System Integration Part 1: Models and Terminology.
19) 児玉公信：計画･実行システムの一般モデル―生産管理システムと金融業務システムの共通性―，情報システムと社会環境研究会報告 IS109-4, 2009/9/14
20) Page-Jones, M.著，久保ほか訳：構造化システム設計への実践的ガイド[原書第 2 版]，近代化学社 pp.58-80（1991）
21) Buschmann, F., et al: Pattern-Oriented Software Architecture - A System of Patterns, John Wiley (1996). 金澤ほか訳：ソフトウェアアーキテクチャ,トッパン (1999)
22) 情報処理推進機構：共通フレーム 2007（第 2 版），オーム社, 2009
23) 児玉公信：実践ファンクションポイント法（改訂版），日本能率協会マネジメントセンター，2006