Vol.2017-IS-141 No.4
2017/8/25

取引処理のための基幹情報システムの状態側面から見た一般モデル

児玉公信[†1]

概要：取引処理を，受注者が依頼者からの注文を受けてその要求を満足するまでの一連の処理とする．取引処理は企業ビジネスの根幹であり，これを支援する情報処理システムはミッションクリティカルな基幹システムである．このようなシステムについて，業種業態を超えて適用可能な情報システムのモデルを提案する．このモデルは，静的側面だけでなく，従来議論されて来なかった機能側面および状態側面を含む．このモデルは，「注文」の状態を，依頼者と受注者との対話状態と，システム内部の制御状態に分離し，さらに注文確約の条件を分離することでよりシンプルになり，システムの施主の要求をより明確にし，設計者の意図を実装者により正確に伝達できる．

キーワード：基幹情報システム，取引処理システム，状態モデル，一般モデル

1. はじめに

1.1 研究の動機

企業（Enterprise）の情報システム，とりわけ取引処理に関わる情報システムは，その企業の発展の歴史とともに成熟してきた基幹システムと言える．しかし，それは市場ニーズの多様化と，情報技術の進歩に伴って急速に陳腐化し，保守費や運用コストを増大し，結果としてアーキテクチャや組織体制も含めたシステム全体の再構築の必要に迫られている．再構築に当たって CIO は，自社の強みを活かすために莫大な費用で個別開発するか，巨大なブラックボックスである ERP 製品に任せるか，あるいは緊急避難的に延命を図るか，難しい選択に迫られている．もし，情報システムの構築と運用が，より安全確実で，拡張性に富み，短期間かつ安価に実施できれば，CIO の悩みはかなり軽くなるだろう．本報告では，取引処理の基幹システムの一般モデルを提示することによって，その実現性を展望する．

1.2 取引処理

ここで取り上げる「取引処理」とは，企業取引において顧客から注文（Request）を受け取ってそれを適切に処理して価値を移動して，その対価を得るまでの一連の処理を指す．注文の対象は商品や製品だけでなく，サービスも含まれるし，注文の内容は販売注文だけでなく，企業内部での購買注文や製造注文，請求注文なども含める．注文の形式は業種を問わない．誰かが，ほかの誰かに，いつまでに，これこれ何をやってくれとお願いする形式になる．

取引処理のための基幹情報システムとは，このような取引処理を支援する更新系の情報処理システム（以下，「取引処理システム」と呼ぶ）をいう．取引処理システムはデータのロスや処理のミスが許されないミッションクリティカルな基幹システムであり，データベース論における「トランザクション処理」を内包しているが，本報告では業務上の取引処理のシステムの設計について議論する．企業情報システムには，このほかに，取引処理を支える資源残高を導出したり作業計画をガントチャートで表したりする取引参照系のシステムと，大量の取引データを集計してビジネスを分析するなどの報告系のシステムがある．

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

2. 一般モデル

ここで言う一般モデルとは，これまでの業務知識やシステム設計の経験を基に，取引対象やビジネスプロセスの違いを超えて，取引のとらえ方，処理のとらえ方を，一般知識や基本的な概念のモデル化を通して再定義することをいう．モデルの表現力にも依存するが，ひとたび一般モデルが定義されれば，さまざまな取引対象や業種において差分適用が可能となり，短時間で一定品質の情報システムの構築が期待できる．

2.1 静的側面の一般モデル

筆者は，2008 年に多品種少量生産のための生産管理システムの一般モデル（CHARM）を発表した[1]．これは生産管理システムの静的側面，すなわち，システムで扱うべき情報とそれらの関係を時間の流れを止めて見た構造を定義したモデル図であった．その後，CHARM は金融業の金融商品の取引のモデル[2]としても適用できたことで，その一般性を確認できた．

しかし，情報システムの実装においては，静的側面だけでは，業務設計者および実装設計者に設計意図を伝えきれなかったという歯がゆい思いがある．

2.2 階層間連携モデル

あるプロジェクトで，非製造業の取引処理システムの全体（受注から，物流計画，作業計画，機器制御，料金計算，請求まで）を設計することになった．いまどき，基幹システムをまるごと設計するチャンスはめったにない．ここでは，取引処理システムを，製造業の経営（Management）システムから制御（Control）システムまでを連携するための参照モデルである IEC 62264-1[3]に倣って，機能階層構造を採ることとした．階層は 4 つあったが，各階層を構成するサブシステム（以下，ドメインと呼ぶ）の静的モデル

にはともに CHARM を採用し，上下のドメイン間の通信をともに Customer-Performer モデル[4]のビジネスプロトコルを採用して連携させた（これらのモデルについては後述する）．

2.3 状態側面のモデル

異なるドメインが同じ一つの静的モデルで扱え，異なるドメイン間連携も同じ一つのプロトコルで扱えた経験は，状態側面も同様に一般化できるという期待をもたらした．状態側面のモデルとは，システムで扱う情報とそれらの関係の時間の経過にともなう変化に対する制約を言う．

これまで実装者に任せていた状態側面の設計を，あらためて注意深くモデル化し，それが一般的であることを確認する．まず，候補となる状態側面のモデルを作成し，ついで製造業とサービス業のお客様のプロジェクトにおいてその一般性を確認した．このモデルによって，取引処理システムの実装は，静的モデルおよび連携プロトコルとともにシステム設計者の設計意図に基づいて作られる．

2.4 一般モデルの要件

提示されたモデルが十分に一般的であるとは，その領域で扱われる「もの」やサービスの特性，業種，ビジネスプロセスといった特定の状況を越えて，異なる状況にも，要求の変化にも適用できるポテンシャルをもっていることを言う．このために，モデルはある程度抽象的となるが，それを利用する設計者が理解可能なレベルにとどめる．ただし，そのようなポテンシャルの測度は今のところ存在しないため，実践的に評価を重ねていく．

2.5 一般モデルの例

先行研究として，一般モデルの例をいくつか挙げる．

2.5.1 アナリシスパターン

Fowler の「アナリシスパターン」[5]は情報システム設計のための一般モデルの一つと言える．そこでは，経験に基づいてよく練られたオブジェクトモデルが 60 数個紹介されている．紹介されたパタンは情報システムの設計において極めて有効であり，筆者も大いに参考にしている．ただし，それらはモデルの断片であり，多くは静的側面の設計要素である．具体的な情報システムをどのようにモデル化するかについては書かれていない．状態側面については，会計処理のための手続きが一部記述されているだけである．

2.5.2 企業情報システムの一般モデル

Marshall の Enterprise Modeling with UML（邦題：「企業情報システムの一般モデル」）[6]は，システム思考に基づいて企業システム全体のモデル化を試みている点が特徴的である．企業システムは，目的，プロセス，もの（entity），組織のコンポーネントからなり，それぞれに静的な構造をもっているとする．それぞれのモデルについては，本質的な議論がなされているものの，システムとしての実装に至るまでのイメージができない．

2.5.3 生産管理システムの一般モデル

特定目的の情報システムの一般モデルの例として，筆者自身による「少量多品種むけの生産管理システムの一般モデル CHARM」[1]を挙げておく．これは，生産変動が起こりうる生産管理の活動を支援するシステムの静的モデルで，その一部に「アナリシスパターン」の観測パタン，勘定パタン，計画パタンを使用している．観測パタンにより，多仕様の製品を表現し，勘定パタンによって資源の変化量を記述している．勘定パタンは計画パタンを用いて予定と実績を対比して記録するように拡張され，将来の資源の引当可能残（availability）を導出することで，実行可能な作業計画が立てられ，必ずしも計画どおりに実行できない作業の実績とを対で記録する．このモデルは，さまざまな特性をもつ製品の在庫や製造プロセスを記述できるという点で，生産管理システムの静的側面の一般モデルであるとした．

このモデルは，のちに製造業という枠を越えて，金融業と非製造業の取引処理に適用され，取引処理システムの一般モデルを考えるきっかけとなった．

3. 状態のモデル

DeMarco[7]を始め多くの先人によって，情報システムのモデルは，経験的に静的側面，機能的側面，状態側面の 3 つの側面から記述するとよいとされてきた．静的側面のモデル（以下，静的モデルという．他の側面についても同様）については上述したとおりである．機能モデルにはユースケース記述が利用できる．状態モデルとして，制御系システムにおいては状態遷移図がよく利用される[8]．その理由は，システムで扱うべき状態が客観的に観察可能だからと思われる．一方，管理系システムでは，扱うべき状態が明確ではない．しかし，実装者に設計意図を伝えるためには，しっかり考えられた状態遷移図は経験的に有効であり，本報告ではそれを提案する．

3.1 状態遷移図

3.1.1 状態とは

Weinberg[9]によると，状態とはシステムの挙動の観察から推定される変数値の組合せの集合である．ある状態から他の状態への遷移制約は有向グラフで記述される．しかし，これはシステムの状態を言っている．情報システムを設計する局面では，結果として立ち現れるシステムの状態を積極的に設計するのは困難なので，主要ないくつかのオブジェクトに注目して，オブジェクトの状態，発生しうる事象，その際の遷移先，遷移に伴って行うアクションを決める独立事象を組み合わせて，システムの振る舞いを設計する．

3.1.2 状態機械図

状態遷移図の表記法には，かつて Mealy 図と Moore 図があるとされた[10]．前者は遷移の矢印に沿ってイベントとアクションを併記する．後者は遷移の矢印にはイベント

のみを書き，状態にアクションを記述する．Harel[11]は，遷移の矢印上に**イベント[ガード]/アクション**の形式で書くことで，先の二つの図法の別をなくした．さらに下位状態（substate）の記法を提案して，状態遷移をよりシンプルに表示する方法（ステートチャート）を提案した．この図法はのちに UML に取り入れられ，第 1 版ではそのままステートチャートと呼ばれた．第 2 版では状態機械図と改名され，システム全体の状態図を「プロトコル状態機械図」，オブジェクトの状態図を「振る舞い状態機械図」と読んで区別するようになった[12].

3.1.3 ペトリネット

状態モデルの表記法としてペトリネット[13]がある．ペトリネットは，プレース，トランジションおよびそれらをつなぐアークから構成される有向 2 部グラフである．このグラフ上で，トランジションの発火によりトークンと呼ぶ制御のインスタンスが移動していく（token passing）ことでシステムの状態を表現するa．状態機械は，ペトリネットの特殊形であり，トランジションにアークがたかだか 1 本入るまたは出て行く場合をいう．トークン移動の考え方を援用して状態機械の表現を解釈できる．

3.2 状態を認識するための文脈

上述のように，制御系システムにおける状態の認識は，物理的状態の観測が裏付けとなっている．一方，管理系システムでは，多くの場合，物理的な状態をもたない．そのかわり，静的モデルで見出されたオブジェクトが，業務処理の達成を支援する機能（ユースケース）[14][15]によって変換されていく．この変換の道筋が，論理的な状態および遷移を決定するための文脈を与える．そこで，状態の検討の前に業務フロー図を記述する．

3.2.1 事務分析図

コンピュータが導入される以前の管理系システムの設計では，担当者間にどのような伝票や帳票を流してどのような処理を行うかを流れ図の形式で記述していた[16]．その処理は伝票内容の転記，集計，一覧などの事務処理が中心であった．そのため，コンピュータを使った初期の情報処理システムは，そのような事務処理を計算機処理に置き換えたもの（EDP）が多かった．

コンピュータの利用が前提となる現在の業務処理の設計においては，単なる事務作業としてではなく，業務の本質をとらえる必要がある．

3.2.2 業務フロー図

業務は，その達成目的に向けて複数の担当者（アクタ）が共同して行う行為の連続である．これを流れ図の形式で記述したのが業務フロー図である．業務フロー図は，現実世界でくり返される一つひとつ異なるケース（業務のインスタンス）を，一定の前提の下で集約して一連の条件つき手順としたものである[17]．図法としては，UML のアクティビティ図または BPMN が使われる．本報告では前者を用いる．

4. 取引処理の一般モデル

取引処理システムの一般モデルを提案する．一般モデルは，静的モデル，業務フロー図の例，ユースケースの例，状態モデルからなる．

4.1 一般モデルの前提

提案する一般モデルの前提となる主要な原則を整理しておく．

4.1.1 対話の原則

業務における対話は，Customer-Performer モデル[4][18]に基づいて行われる（図 1）．Customer（発注者）の相手となるアクタはお願いの言葉，すなわち，売れ（sell），買え（buy），作れ（make），払え（pay）bなどによって決まる．Performer は，Promise した Request を達成する作業を自ら実施してもよいし，成果物に合わせて作業を分解して，自分が Customer のロールとなって，作業ごとに下位の Performer に Request してもよい．

Customer と Performer の間で交わされる具体的な対話のプロトコル（メッセージングのモデル）は図 2 に従う．この図は，オリジナルの Winograd のモデル[19]を，筆者が UML のプロトコル状態機械図に翻訳したものに，実際のプロジェクトでのビジネスプロトコルに合わせて追加（赤字で示した）したものである．実際には，Promise（たとえば，注文請書の発行）をした後にも，取消や変更がたびたび発生する．場合によっては，作業に着手した後にも変更が起こりうる．これを明示するために状態 3′を追加している．これが日本特有の状況かどうかは分からない．

4.1.2 ドメイン分割

取引処理システムは，業務フローのアクタごとにドメイン（実態的にはサブシステム）を分離する．すなわち，モデルの境界を設ける．業務フローの例として示した図 4 および図 5 には，発注者と営業担当者のアクタがあることから，発注者と営業担当者の二つのドメインを想定し，それ

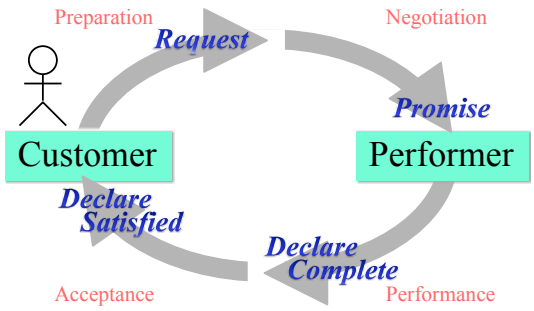

図 1　Customer-Performer モデル

a したがって，状態図は型レベルであることがわかる．

b 要求を命令形の動詞で表現する．

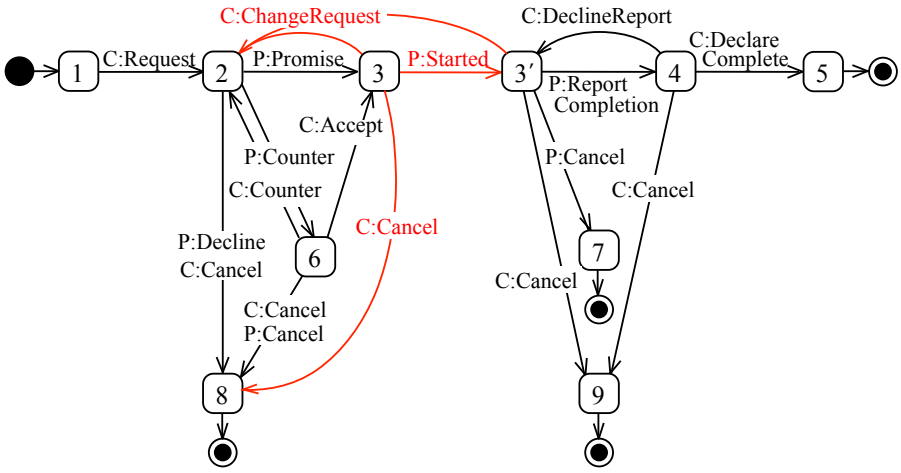

図 2　Customer と Performer 間の対話のプロトコル

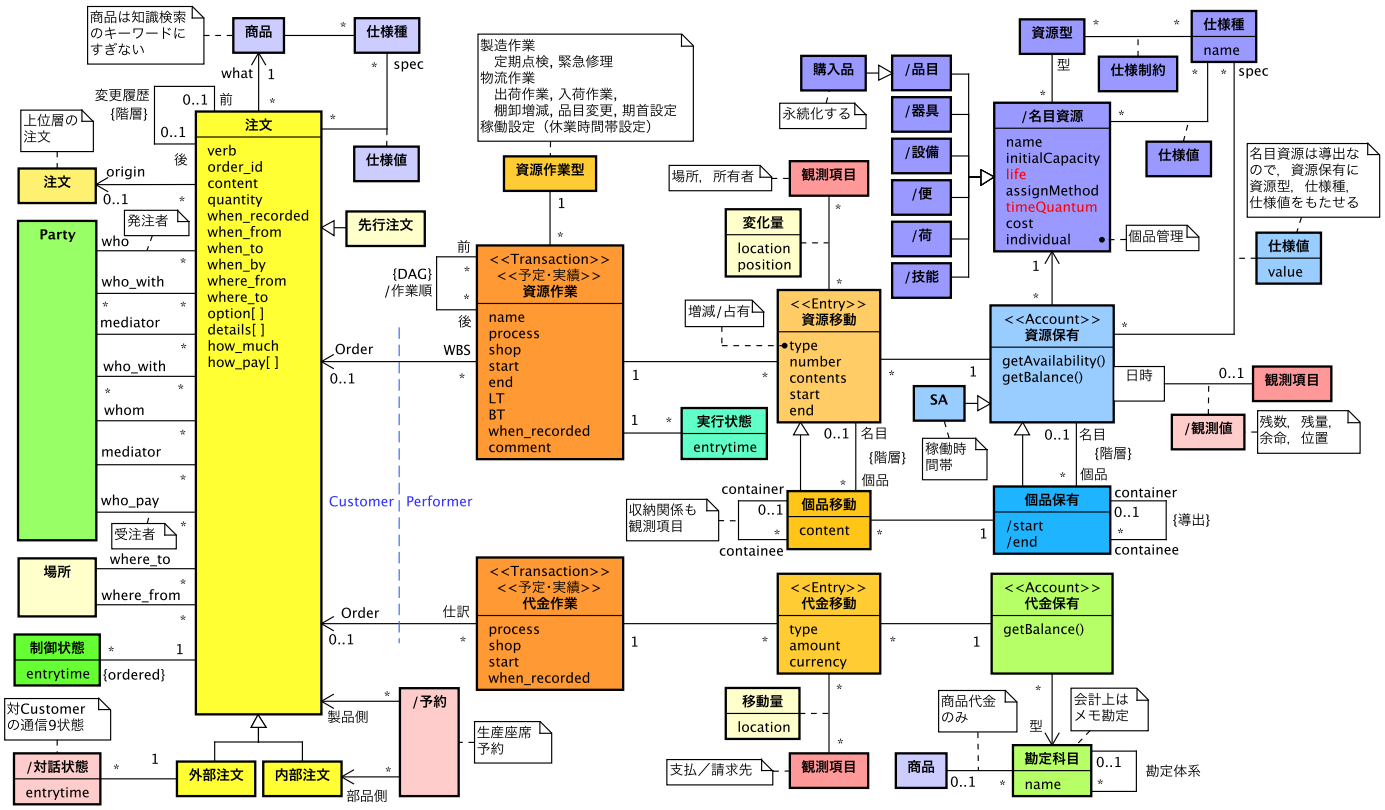

図 3　取引処理システムの一般モデル（静的側面）

ぞれにモデルおよびサブシステムの実装を考える．

取引処理システムでは，営業担当と製造部門（生産管理）との対話はもちろん，製造部門とその下の機器制御（MES）との対話が想定される．また経理部門（販売会計）とその下には金融機関との対話制御（バンキング）との対話が想定される．それぞれにモデルおよびサブシステムの実装を考え，その間を Customer と Performer 間の対話のプロトコルで疎につながることにより，取引処理システム全体が構成される．

ただし，以下では営業担当ドメイン（われわれは Sell ドメインと呼んでいる）についてのモデルを代表として提示する．他のドメインのモデルも基本的には同様だからである．

4.2 静的モデル

一般モデルの静的側面を図 3 に示す．これは CHARM[1] の拡張である．主な拡張点の一つは代金作業のモデル要素 TEA（Transaction-Entry-Account）の追加である．このために，CHARM の「取引」「移動」を「資源作業」「資源移動」に改名した．代金の移動は，資源の移動と直接リンクせず，注文にリンクする．もう一つの拡張点は個品管理のための「個品移動」「個品保有」の追加である．これによって，一つひとつの現物，たとえば部品や製品の現品を識別し，在庫を管理でき，トレーサビリティを確保できる．

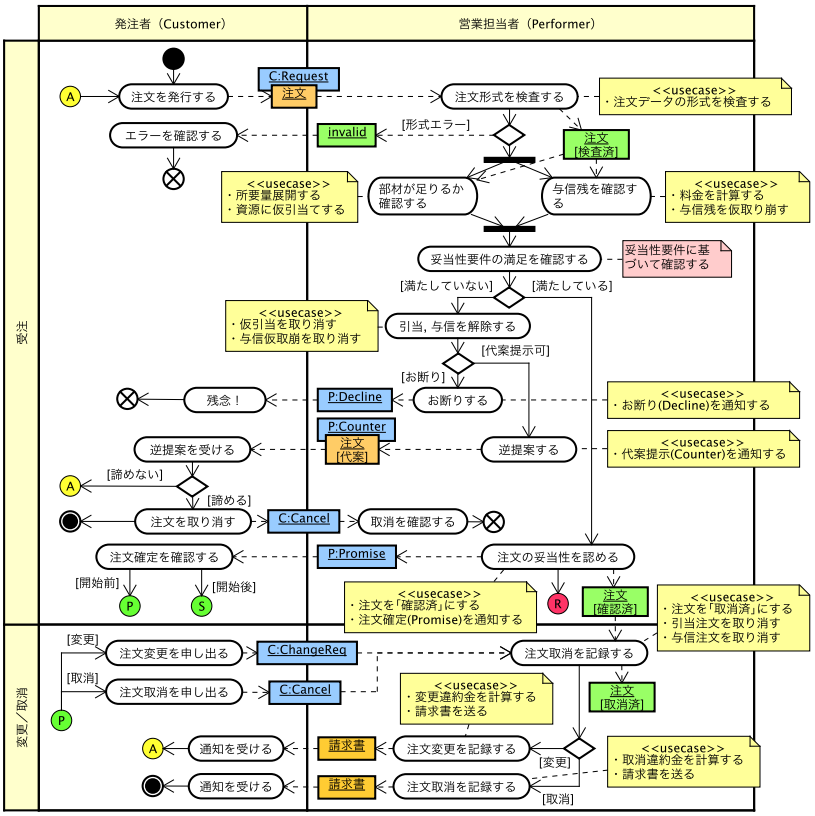

図 4 業務フローの例（受注確定まで）

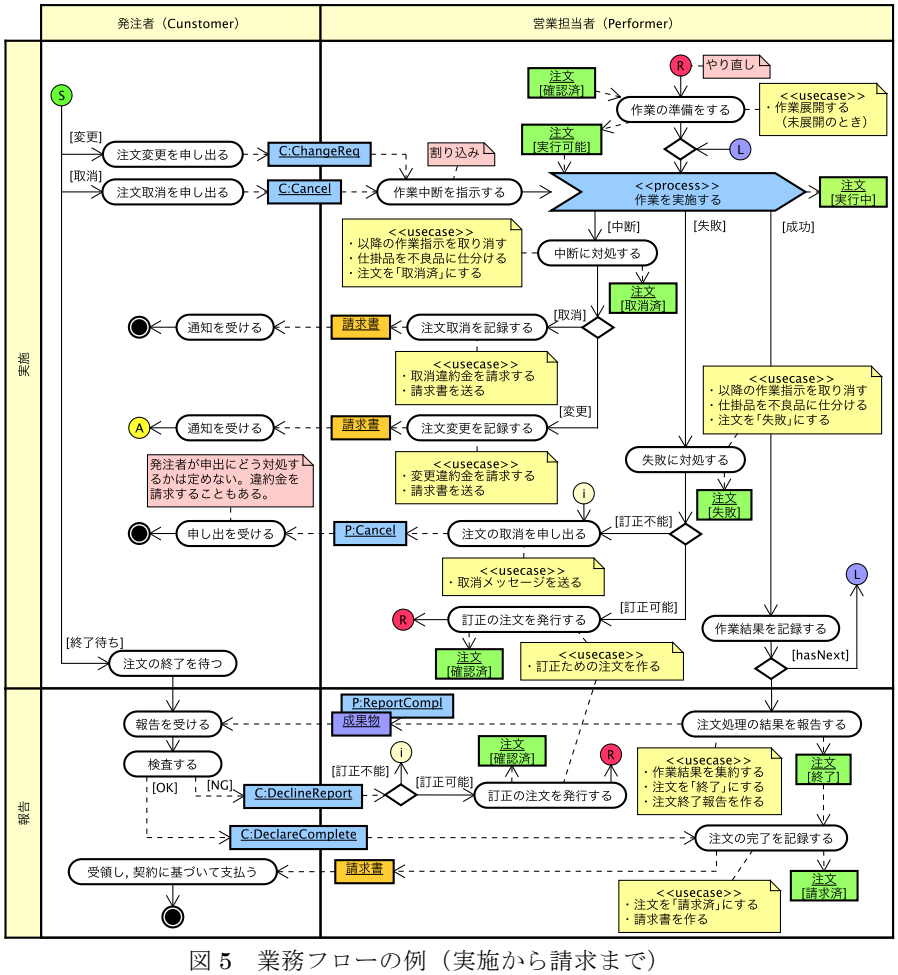

図 5　業務フローの例（実施から請求まで）

ほかに軽微な拡張として，CHARM では「注文」と「生産計画」を分離していたが，図 3 ではそれらを分離せず，「注文」だけをもたせた．これは発注者ドメインと製造者ドメインを分離したために，一つのモデル内に二つの注文をもつ必要がなくなったためである．

状態モデルの設計では，Sell ドメインの「注文」に着目して状態を考える．

4.3 業務フローの例

取引処理における業務フローの例を図 4 および図 5 に示す．図 4 は図 2 の対話プロトコルの状態 1 から 3 までの一群の遷移，図 5 は同じく状態 3′ から 5 までの一群の遷移に対応している．アクタ間のコントロールフローにはオブジェクトを明示し，とくに図 2 のイベント名（C: Request など，Performative[20]と呼ぶ）を追記した．

注文はその要求を達成するための作業に分解され，必要な資源が引き当てられ，順序計画が立てられて実行される．作業中であっても，Customer からの取消と変更の申し入れがありうる．この場合は違約金が発生する．Customer は，実施結果を評価してやり直しを求めることがある．

4.4 ユースケース

業務フローで設計した行為（アクション）の目標を達成するために必要と想定されるユースケースを業務フロー図に追記した．ステレオタイプ<<usecase>>をつけたノートシンボルの内容がそれである．業務フロー図にあるとおり，これらのユースケースのアクタはすべて営業担当者である．これ以外に，取引処理に関わらない，運用管理についてのユースケースが存在するが，状態モデルを考える際には必要ない．また，本報告では，ユースケース記述までは立ち入らない．

4.5 状態モデル

発注者と営業担当間で交わされる対話の進行によって，Sell ドメインの「注文」の状態が移り変わっていくことは自明に思える．たとえば，Promise 前の注文について請求をしないように，あるいは注文された作業を二度実行しないように，対話のプロトコルや作業の進度に応じて「注文」がたどってきた道筋を記憶しておく必要があることは直感的に分かる．問題は，注文にどのような状態を設定して，いつどの状態に遷移するかを決定することである．

4.5.1 対話状態と制御状態の分離

まず，Customer との対話によって何らかの状態が変わることは明らかである．ただ，これは「注文」の状態とは限らない．Customer との対話プロトコルの状態ととらえるべきである．たとえば，Customer に Promise を送ったとすると，対話の状態は，確かに図 2 でいう 3 の状態に遷移する．しかし，Promise を送るまでにはさまざまな内部処理がある．具体的には，注文データの形式チェックがあり，それが妥当であれば，作業展開をして，資源（時間軸上の有効在庫や設備など）を引き当てて，納期を確定させる一方で，代金計算をし与信チェックをするか，前入金が必要であれば請求して入金を待つという処理をする．これらの処理がすべて完了状態になってはじめて Promise が出せる．ビジネスによっては資源の引き当てが確認できなくても，あるいは入金が確認されなくても，Promise する場合もある．このように状況が混乱して見えるのは，状態変数は一つしかないと解釈して，状態が複合しているはととらないためである．状態の分離のために，Promise できるための条件を別に取り出して考えてみる．

4.5.2 確約条件の確認

Promise は Request に対する履行の確約である．確約をもって「注文」と呼ぶ契約が成立する．確約するためには，一定の条件が満足できていること（状態）を確認しなければならない．たとえば，出荷時点で在庫が引き当てられることを保証しなければならないし，金額が大きい場合には権限者の承認を得なければならないだろう．代金が与信残高を超えてはならないし，前金あるいは必要な内金が入金されていなければならないなどの条件の組合せが考えられる．ここで，与信確認と入金確認はどちらか一方でいいし，ビジネスによっては在庫の確認がなくても Promise してよいということもあるだろう．

よって，確約条件はビジネスのあり方によって選択されるものである。これを抽象化して，必要な下位状態が満足されていることを確認する状態（確認中）を設ける．

4.5.3 着手条件の確認

もう一つ，作業が開始できるために一定の条件が満足していること（状態）を確認しなければならない．これを着手条件と呼ぶことにする．たとえば，着手条件として必要な資源が引当済であることが要求されるとする．ところが，確約条件でも引当済であることが要求されていいて，それが満足されているならば，着手条件はすでに満たされている．逆に，着手条件として代金が入金済であることが要求さ，確約条件にはそれがないという場合も想定できなくはない．先に Promise しておいて，入金を待って着手するというケースである．

よって，着手条件は作業手順に対応して選択するものであり，必要な下位状態が満足されていることを確認する状態（準備中）を別に設ける．ただし，確約条件と準備条件は相補的な関係にある．

4.5.4 制御状態のモデル

確約条件の確認中と着手条件の準備中という状態を分離できたら，あとは業務フローに沿って状態を想定できる．そのうえで，対話プロトコルの状態との関係を遷移条件と対応づけて整理した．その結果が図 6 である．

このモデルを見ると，「実行可能」から「実行中」に遷移する expired を除いて，遷移のトリガになるのは外部イベント，すなわち Customer からのメッセージであることが分かる．ガードがあるのは処理の結果に基づく遷移であり，

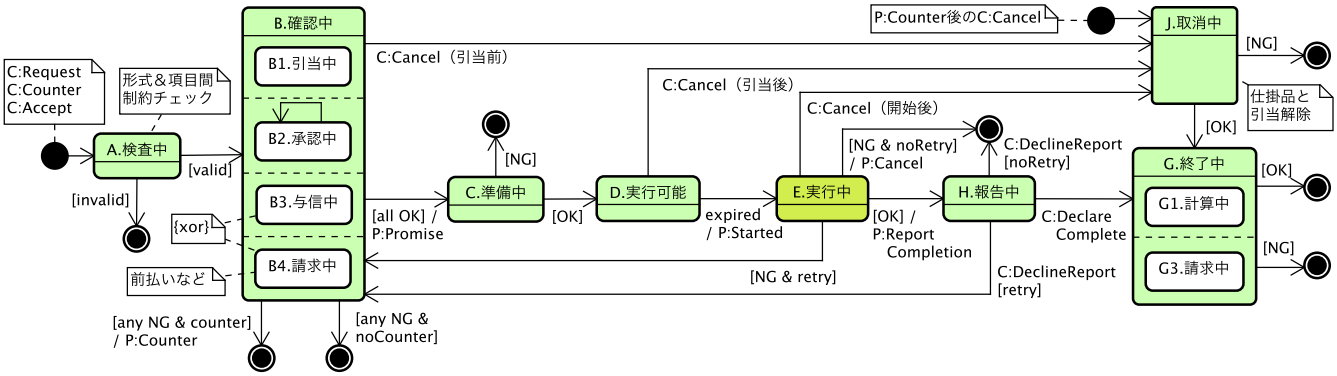

図 6　取引処理における「注文」の振る舞い状態機械図

アクションは外部イベントの発生，すなわち Customer へのメッセージ送信という形に整理できた．

4.5.5 着手イベントの通知

expired は，スケジュールされた作業が実施期日を迎えたときのイベントであり，期日（と時刻）が到来後ただちに着手され，同時にその旨を Customer に通知する．とくに expired が必要な状況は，「実行可能」から「実行中」までに時間がある場合である．たとえば，3 か月前に予約してあった注文の作業がスケジュールされて実行可能状態で待っていて，期日が来たので開始するという具合である．着手後は，営業担当者は変更要求が来たとしても，すでに着手したという理由で断ることができる（実際は違約金の交渉になる）．expired のイベントは自動発火でも可能であるが，オペレータによる作業開始の手動イベントがシンプルな実装になるであろう．

なお，着手通知は，下位ドメインへの作業の委譲が連鎖している場合の進捗状況を確認するために有効である．

4.5.6 状態図の状態とオブジェクトの状態

状態図の状態名と業務フロー図に書いた「注文」オブジェクトの状態名が，前者では「○○中」，後者では「○○済」となっており一致していない．このことは，筆者は状態図では処理を状態と見なしていることを示唆する．たとえば「確認中」は，注文内容の確認処理を実施している状況であり，厳密には状態とは言いにくい．処理中を状態とみるのは，ペトリネットのプレースとの連想が働いているのかも知れない．一方，業務フローでは，確認が終わった注文オブジェクトには「確認済」という状態名を与えている．

例外は「実行可能」であり，これが本来の状態というべきかも知れない．本報告では，状態モデルを記述する目的は実装者に対する設計意図の伝達であるとしているので，処理中を状態と見なすことは許されると考えている．

以上の考察を基に，図 3 の静的モデルにおける「注文」の状態変化を，図 6 の状態機械図で定義した．

4.5.7 状態の静的表現

こうして見出された状態を実装するために，「注文」の属性に「制御状態」を，履歴を保持する型によって表現することにした．最新の状態だけをもつのではなく，履歴を保持することとしたのは，もともとの静的モデルがいつの時点でもその時点の状況を再現することができるように設計されているためである．

対話プロトコルの状態は，制御状態から導出できるので「/対話状態」とした．

4.6 一般性についての考察

図 6 の状態機械図が，取引処理において十分に一般的かどうかを考察する．

これまで，生産管理システムをはじめ，取引処理のシステムをプロジェクトのたびに設計，実装してきた．静的モデルの一般化は 2005 年にはできていたが，機能モデルや状態モデルを一般化することは考えていなかった．業種業態に大きく依存すると考えていたからである．上に述べたように，ある非製造業の取引処理システムの全体を設計する際に，製造業の静的モデルと階層化アーキテクチャを取り入れた．このとき，顧客と営業担当ドメイン間の対話だけでなく，階層間の対話も Customer- Performer の対話モデルで一般化できることに気がついた．

その後，いくつかのプロジェクトで取引処理システムの設計を行うときには，積極的に Customer-Performer モデルに準拠するようにした．注文内容は階層ごとに異なるのだが，対話プロトコルは揺るぎなかった．そこでドメイン内の注文処理についても業務フローを作成し，機能を想定しながら，注文の状態モデルを設計してみた．その成果が図 6 である．

提案した状態モデルは，経験的には，さまざまな業種業態で適用可能であると言える．その一般性も，対話のモデルに強く依存している．したがって，対話のモデルが十分に一般的であるならば，提案したモデルも一般的である可能性が高い．対話のモデルは，現実のビジネス用に一部拡張したが，本質的には変わらない．提案したモデルは，今後，実践を重ねて改訂し，より一般化していきたい．

5. おわりに

最後に，一般モデルの総括と今後の展望を述べる．

5.1 リアクティブな取引処理システム

状態モデルを書きながら，ここで提案した取引処理システムは，マクロには，データベースを中心とし，外部から与えられた注文に対するリアクティブ（即時反応系）のシステムであると気づいた. ものづくりにおける「一個作り」や「一個流し」の意味するところもこのようなリアクティブ性を示唆する．これと対局にあるのが MRP である．この生産管理方式は見込みの大量生産に対応している．今後の生産システムが，顧客要望の多様性に応じた個別受注生産に向かうとすれば，このようなリアクティブシステムの設計は有用であると思われる．

5.2 お節介な取引処理

Winograd のオリジナルの対話のプロトコルには，Promise 後の取消（Cancel）や変更要求（Change Request）がなかった．これを加えざるを得なかったことは，実際の取引では，Customer に対する Promise の拘束力が緩いことを意味する．実際に，長い内示期間に対して着手寸前での Customer からの確定，取消，変更がある．本来は，Promise 後の取消，変更があった場合は，状況に応じて，資源引当（予約）による期間損失に見合う違約金を取るべきであり，そのような機能も用意してある．

もう一つ，今回，オリジナルにはない P:Started を追加した．これは，実行期日の到来による Performer からの着手済の通知イベントである．このイベントは，着手後の注文の取消，変更を牽制する目的を持つ．着手後の取消，変更の違約金は着手前のそれよりも大きい．着手後は，材料の手配や部品の消費など, 直接的な損害を生むからである．しかるに，なぜオリジナルの対話プロトコルに着手イベントがないのか（そもそも，状態 3′がなかった）．このことは，Customer は Performer が ReportCompletion してくるまでは，一切 Performer に干渉しないことを意味する．これは Promise 後の Performer への揺るぎない信頼を意味しているのだろうか．Winograd が考えたワークフローのレベルでは，このような事態を考える必要がなかっただけなのか，それとも日本の商習慣が異なるのか，いつか確認しておきたい．

5.3 よりよい情報システムの構築に向けて

あるユーザ企業のプロジェクトで，お客様とモデリングを介したシステム設計の過程をオブザーブしているベンダがふと漏らした言葉がある．「15 年前にこのモデルがあったら, 自分たちはもっと良いシステムが作れた」. 彼は,「業務フローと状態遷移図のいい見本がないんですよね」とも言った．筆者も 15 年前は静的側面の一般モデルはあったが，業務フロー図も状態遷移図もできていなかった．

本報告で，取引処理システムの静的側面，機能的側面，状態側面の一般モデルを提示した．これが, 最初の「見本」になって，広く活用され，よりよい情報システムが構築され，モデルが改良されていくことを期待する．なお，このモデルの妥当性を検証するために，これらのモデルに基づく取引処理システムの自社製品を開発し，現在 2 社で PoC（概念実証）プロジェクトを実施中であることも付け加えておく．このフィードバックも受けて，さらによい一般モデルを追求していく．

参考文献

[1] 児玉公信, 水野忠則,「少量多品種型生産管理システムの一般モデル CHARM の提案」, 情報処理学会論文誌, Vol. 49, No.2, pp.902- 909, 2008.
[2] 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―」, 情報処理学会研究会報告, IS109-4, 2009/9/14.
[3] IEC 62264-1:2013, Enterprise - Control System Integration Part 1: Models and Terminology.
[4] Medina-Mora, Winograd, Flores, and Flores, "The Action Workflow Approach to Workflow Management Technology," Proc. of CSCW 92, pp1-10, 1992.
[5] Fowler, M., "Analysis Patterns: Reusable Object Models," Addison-Wesley, 1997. 邦訳）児玉公信ほか訳,「アナリシスパターン」, ピアソンエデュケーション, 1998.
[6] Marshall, C., "Enterprise Modeling with UML: Designing Successful Software Through Business Analysis," Addison-Wesley Longman, 2000. 邦訳）児玉公信監訳,「企業情報システムの一般モデル―UML によるビジネス分析と情報システムの設計」, ピアソンエデュケーション, 2001.
[7] DeMarco, T., "Controlling Software Projects," Yourdon Press, 1982.
[8] 松井聡一，小泉寿男ほか,「仕様の状態遷移図記述による組み込みソフトウェア開発手法」, 電気学会論文誌 C, Vol. 118(2), pp.232-241, 1998.
[9] Weinberg, G. M., "An Introduction to General Systems Thinking," Wiley, 1975. 邦訳）松田武彦監訳,「一般システム思考入門」, 紀伊國屋書店, 1979.
[10] 渡辺政彦,「状態遷移ベースのソフトウェア開発環境の現状と動向」, 計測と制御, Vol.41(2), pp.117-121, 2002.
[11] Harel, D., et al, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", Proc. 10th Int. Conf. on Software Engineering, 1988, pp. 396-406. 邦訳）児玉公信監訳,「ソフトウェアエンジニアリング論文集 80s」, 翔泳社, pp.221-243, 2006.
[12] 児玉公信,「UML モデリング入門」, 日経 BP, pp.164-174, 2008.
[13] 青山幹夫ほか,「ペトリネットの理論と実践」, 朝倉書店, 1995.
[14] 鯉沼 章,「フローチャートによる事務分析」, 日本工業新聞社, 1965.
[15] 児玉公信,「UML モデリングの本質（第２版）」, 日経 BP, pp.35-45, 2011.
[16] 児玉公信,「ユースケースの使い方に関する提案」, 情報処理学会研究会報告, IS105-7, 2008/8/29.
[17] 児玉公信,「企業情報システムのための早期アーキテクティングの一方法」, 情報処理学会研究会報告, IS120-3, 2012/6/4.
[18] Action Technologies, Inc., "Business Interaction Model," http://www.actiontech.com/BPM/BIM.cfm（2017.7.22）
[19] Winograd, T., "A Language/Action Perspective on the Design of Cooperative Work," HUMAN-COMPUTER INTERACTION, Vol.3, pp.3-30 (1987-1988).
[20] Finin, T. et al: “Specification of the KQML Agent-Communication Language (DRAFT),” http:www.csee.umbc.edu/csee/research/kqml/papers/kqmlspec.ps, 1993（2017.7.22）.