

アプリケーション・ソフトウェア・システムの 要求仕様決定方法

大槻 繁^{*}

新ソフトウェア構造化モデル研究本部
情報処理振興事業協会(IPA)

ソフトウェアシステムの開発における、インプリメンテーションと要求分析・定義との間の意味や構造上のギャップは無論のこと、さらに、要求分析・定義と組織のミッションや目的の間にもギャップがあるという生産パラダイムを提唱する。これ等2つのギャップを克服するために必要な仕様記述言語の満たすべき要件や、技法の枠組みについて述べる。特に、要求仕様の妥当性の根拠を組織の戦略や目的に求め、その検証のために対象世界モデリングや組織業務のリファレンスモデルを導入している。さらに、情報制御システムの分野で要求されるものの流れや、資源の管理に必要な仕様の構成概念、フロー概念等について論ずる。

Framework for Requirement Definition Techniques of Application Software Systems

Shigeru Otsuki

Laboratory for New Software Architectures
Information-technology Promotion Agency, Japan(IPA)

This paper describes one of the frameworks or research programs for software development methods towards aim or mission of organization. There is not only a great semantic gap between implementation and requirement analysis/definition phase, but also is gap between requirement phase and mission of organization. Every practical method should handle these semantics gaps. To manage this problem we introduce the techniques of real world modeling and referencing modeling for categorize the tasks or missions. We also describe the concepts of dependency and flow structures for the area of information and control system.

^{*} (株)日立製作所より出向 Also with Hitachi, Ltd.
E-Mail:otsuki@sdl.hitachi.co.jp/otsuki@caa.ipa.go.jp

1. はじめに

ソフトウェアの生産技術に関する本論文の問題設定は、

- ・組織がその責務を果たすべく持っている目的に従ったソフトウェアを開発するにはどういった技術が必要であるのか。
- ・組織の知識をソフトウェアの仕掛けを利用して蓄積するにはどういった方法が合理的であるのか。

という2点に集約されている。ここでは、その考え方、ソフトウェア開発技術の満たすべき要件について概説する。特に、従来の要求仕様が技法の中でどのように扱われるべきかについて論じる。

対象としているソフトウェアの種類は、情報制御分野のソフトウェアである。この分野の特徴は、開発時の考慮対象がソフトウェアのみならず、システム全体を構成するハードウェアや外界にまで及ぶことが特徴となっている。

2. 言語／仕様／開発プロセス

2.1 言語の3つの局面

ソフトウェア生産技術に対するアプローチにはさまざまな形態が考えられるが、その成果の殆ど全てに何等かの形の言語が使用されている。

ここで言う言語には、一般のプログラミング言語に加え、仕様記述言語も含む。より一般化して人間どうし、あるいは、人間と計算機とのインタフェースも広い意味での言語と考えられる。

言語を中心として3つの技術上の局面が考えられる。その第1は、対象世界の局面である。これは、ソフトウェアの適用対象であるビジネス、プラント、オフィスなどの対象世界（アプリケーションドメイン）のモデリングに関する技術である。第2は、計算機世界の局面で、言語が計算機上で実行されたり操作されたりするための仕掛けに関する技術である。第3は、人間世界の局面で、人間、あるいは、組織が言語をどのように使うかという技術であり、ソフトウェア生産技術が本来対象として包含すべき局面である。

ソフトウェアの開発過程では、多くの言語を使用するため、上記の三つの局面が有機的に統合化された言語活動を解明し、その行為を支援する技術が必要である。

ソフトウェア開発に関する技術の原理をどの局面に求めるかは重要な問題である。上記の三つの局面についても、人間の管理的側面、さらには、社会的ないし経済的側面からのアプローチもあるであろうし、対象世界の機械や自然界の法則や理論に頼る方法も考えられる。筆者は、ここ四半世紀のソフトウェア工学の技術蓄積が、これ等の調和、協調、あるいは、総合化の視点で行われて来たものと見なしており、以下の節で述べる事項に集約していると考えている。

2.2 仕様

(1) モジュール

独立に記述される仕様のことをモジュールと言う。これは、プログラミング言語の世界ではコンパイルされる単位であり、設計の分野でも複合設計や構造化設計において「モジュール」と呼ばれているものである。

言葉が意味する通り、モジュールは独立であるという特性が重要であり、これは、ソフトウェア開発において各個人の担当範囲を決定したり、管理の対象にする単位として設定することができる。無論、プログラミングや設計だけでなく、要求分析や、テスト段階の記述の単位としても定義することが出来る。

(2) 抽象化

抽象化とは、ある問題を解決するために、不要な部分を捨象することである。抽象化にきわめて似た概念に一般化がある。その違いは、一般化がいくつかの事柄を、その主要な性質を失わないように、共通の観点のもとにまとめることであるのに対し、抽象化は、中心的側面に着目し、それを目的（問題解決）に関係ない他の側面から切り出すことである。

抽象化には、プログラミング言語論の立場からよく言われるようにデータ抽象、関数抽象、制御抽象の三つが代表的である。ソフトウェア開発での仕様とは、こういったいくつかの抽象化の結果の記述である。計算機科学の分野で蓄積された理論を適用して、効率よく仕様の正しさを検証するためには、抽象化の問題は極めて重要である。

(3) 構成関係

ここで言う構成関係とは、記述ならびにその記述対象の間の関係のことである。ここで扱う関係概念そのものは普遍代数(universal algebra)で言

う単純な代数構造に対応した概念である。

構成関係には、抽象化の見方からは、データ構造、関数構造（呼び出し関係）、制御構造、さらには、モジュールの見方からはモジュール構造（包含関係、使用関係等）が考えられる。

一般的にソフトウェア生産技術で使用される構成関係には、理論的には素朴なものが多く、実用上は直積、直和、列と、若干の階層やネットワーク関係のみで済む。すなわち、実体・関連モデルに基づく仕様蓄積モデルでソフトウェアの成果物の管理をある程度行うことができる。特に、後に論ずる対象世界モデリングや要求定義の工程では、開発プロセス上の手順や手戻りといった事項が問題になるが、これ等も実体・関連モデルに若干の通時的な関係（開発プロセス上の関係）を導入することによって取り扱うことが出来る。

（４） 対象とメタ

対象世界の問題を解くための方法が対象理論であるとするならば、その解く方法自体を対象とした理論をメタ理論と言う。この意味では、ソフトウェア生産技術そのものが対象世界の問題に対してはメタ技術として位置付けられる。

2. 3 生産パラダイム

生産パラダイムという用語を、ここでは、ライフサイクルモデルや生産形態といった意味で使う。実用上よく用いられている代表的なものとしてV字モデルがある。これは、ソフトウェアの実体であるプログラムを抽象化レベルの底辺に据え、これより設計とテストとを順次対応させながら抽象度を上げて開発工程を捉えるものである。

V字モデルでは、プログラミング言語の高級化や、それに対応したテストの概念を説明することができるが、要求定義と設計との間の関係が適切に説明できない。

一方、この要求定義と設計との間の本質的な違いを説明する生産パラダイムが山モデルである。ここでは、要求定義を対象領域の問題を定式化する過程、すなわち、抽象化のレベルを上げる過程と見なし、設計を定式化された問題仕様を計算機の世界に具体化する過程と見なすことによって説明することができる。

我々は、V字モデルや、山モデルを前提としつつ、さらに、目的指向の考え方から図1に示す生産パラダイムを提案する。プログラムという計算機の世界とユーザの要求という世界との間に意味上のギャップ（ルビコン川）があるのと同様に、組織のミッションや目的との間にももう一つのギャップがある。元来、要求仕様の構造はシステム（と環境）／サブシステムの構造に基づくものであるのに対し、目的の構造は目的／手段という因果関係に基づく構造であるためにギャップが生じている。しかも、プログラムが要求仕様に従っていないくはならないのと同様に、要求仕様は組織の目的に沿っていないくはならない。このように考えることによって、要求仕様の妥当性を導き出す根拠を与えることが出来る。

3. 仕様記述方法

一般に言語の提示は、構文と意味とを与えることによって行われる。しかし、仕様記述言語の場合にはこの構文と意味とを厳密に与える方法を単純に採用することは出来ない。

仕様記述言語定義の難しさは、2.1節で述べたように、計算機上での動作の局面だけでなく、対象世界局面と人間の解釈の局面とを同時に考慮しなくてはならない点にある。しかも、言語自身の意味が時間とともに変わって行くダイナミズムも加わり事態をいっそう複雑にしている。また、仕様が最終的には厳密な構文と意味を持つプロ

gramming言語に結び付けられるとしても、中間的な段階では、未詳細化や未決定の要素も取り扱わなくてはならない。

ここでは、仕様記述言語で、特に、対象世界記述や要求分析・定義段階で用いられる言語（図式）体系の基本概念について概説する。

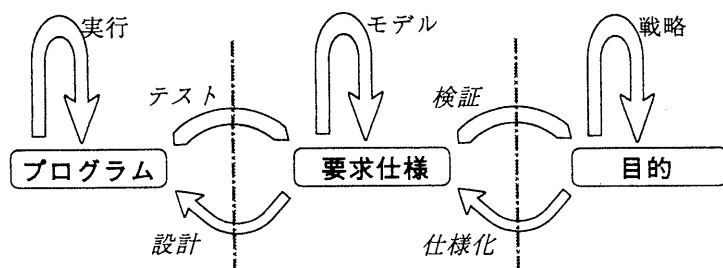


図1 目的指向生産パラダイム

3. 1 ラベル

記述の要素やモジュール等にはラベルが付けられる。これはプログラミング言語において計算上のリソースを指示する機構を持っている識別子に用法が似ている。識別子が単に他との区別のためだけに用いられるのに対し、仕様記述の場合にはこのラベル自体が重要な役割を果たし、その意味付けを直接行っている。ラベルの意味は一般的に日常言語（自然語）に従っている。

対象世界記述に当っては、ラベルを用語（用法）辞書として定義しておくことは意義がある。辞書そのものの整備が対象世界認識の助けになるし、開発者相互、開発者とユーザとのコミュニケーション上も欠かせない。さらに、仕様の確認といった行為の場では、こういった非形式的な情報がなんらかの形でまとめられていると有効であると考えられる。

また、ラベルは範疇によって分類される。範疇とは、設計やプログラミング言語での型、日常言語での品詞、対象世界での概念範疇といったいくつかの局面を総合したものと考えている。また、ラベルは識別子と同様に仕様の包含関係に従った有効範囲を持たせることもできる。

3. 2 フラグメントとコネクション

対象世界記述や要求分析・定義、さらには、上流設計工程では、図式が多用される。図式の表現上の特徴は、四角形や円といった閉領域と、それ等の間を線や矢印で結合する結合子とから構成される。

フラグメントとは後に説明する構成関係の構成要素のことである。言葉の意味の通り仕様の断片であり、構造の構成単位ならびに、構成関係によって構成された複合的仕様を示す。その表現の多くは図式の閉領域によって表わされる。しかし、ある部分はテキスト形式であったり、表形式であったりする可能性もある。一般的に同一の範疇に結び付けられたフラグメントの形状は同一でなくてはならない。無論、一つのフラグメントが複数の形状に対応することもあるが、その範疇が異なれば必ず形状も異ならなくてはならない。

フラグメントは前節で述べた範疇と、それに属するラ

ベル付けがなされる。なお、構成関係や他の関係によって仕様を自明に識別出来る場合にはラベルは省略されることもある。

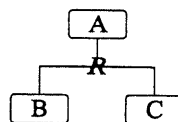
コネクションとは、フラグメント間の関係である。ここでは、3種のコネクションを考えている。第1は次節の構成関係の上位/下位の関係、第2はこの構成関係の下位（子）どうしの間の関係、そして、第3はラベル付けによる関係である。下位どうしの間の関係とは、例えばデータフロー図式での矢印によって表わされるものである。ラベル付けによる関係は、モジュールの詳細化の際に同一のラベルを用いてより詳細な仕様を記述する時の関係や、ステイトチャートのプロセス（状態）間の同期をとるために同一のラベルを用いる場合が相当している。

コネクションもフラグメントと同様に、範疇とそれに属するラベル付けがなされ、図式表現上においても同一範疇同一形状の制約がある。さらに、ラベル省略も可能である。

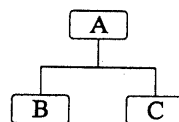
3. 3 構成関係

直積、直和、列という構造は、仕様記述の中でくり返し現れる。この構造は、データ、関数、制御、イベント、状態等さまざまな局面で使用される。さらに、この構造に空間と時間の概念を組み合わせることによって、技法構築の基本概念をより精密に示すことができる。

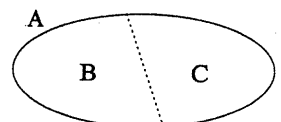
一般的な構成関係は、上位と下位との従属関係のことである。例えば、図2(1)は、BはAに従属しており、また、CもAに従属しており、その従属の仕方が構成（従属）関係Rに従っているという事態を表わしている。より具体的な例をあげれば、日常言語において文が主部と述部から構成されるとか、動詞は自動詞と他動詞とから構成されるといった場合、さらには、図2(3)に示すような表現を使用して、システムAがサブシステムBとサブシステムCとから構成されるといった用法における構成関係のことである。この関係Rは、以下に示すように直積、直和、列に詳細化さ



(1) 一般構成関係



(2) 未決定構成関係



(3) 一般構成関係概念

図2 一般構成関係

れる。しかし、中間段階の仕様では、この関係 R の種類も未決定であいまな場合もありうる、こういった場合には図2(2)のような無印表現も許容する必要がある。

a. 直積

直積構成関係は、図3(1)に示すように、上位のものAは、下位のものB、Cが空間あるいは時間上共存することを示している。直積そのものの集合論上の解釈はCartesian Productである。空間上共存するとは、Aがある時点(期間)で存在することが、BとCとがその時点で存在すると等価ということである(図3(2))。時間上共存するとは、Aが起こるということは、Bが起こってその後Cが起こるということと等価ということである(図3(3))。

空間直積は図に示すように●^sで表わす。オブジェクト指向表記法の全体一部分構造に対応している。また、データ構造では構造体(レコード型)に対応している。

時間直積は図に示すように●^tで表わす。データで言えばストリームに相当しており、Aというデータが到着(あるいは発信)するということは、データBが到着(発信)し、その後データCが到着(発信)するということを表わしている。イベントならばAが起こるということが、Bに引き続きCが起こるということである。

ステイトチャートのAND状態分割は、状態(ステイトチャートで言うプロセス)の空間直積である。関数合成では、 $A(x) = C(B(x)) = B \circ C(x)$ を表わしている時間直積である。

b. 直和

直和構成関係は、図4(1)に示すよう

に、上位のものAは、下位のものB、Cが空間あるいは時間上選択されることを示している。直和そのものの集合論上の解釈はUnionである。空間上選択されるとは、Aがある時点(期間)で存在するということ、BまたはCがその時点で存在すると等価ということである(図4(2))。時間上選択されるとは、Aが起こるということは、Bが起こるかまたはCが起こるということと等価ということである(図3(3))。直和構成関係は、直積構成関係とは異なり、下位の事柄が

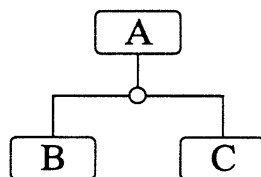
一つしか存在または生起しないので空間と時間との区別をする必要がない場合が多い。

空間直和は図に示すように○^sで表わす。オブジェクト指向表記法では、汎化-特化構造に対応している。また、データ構造では共用体(可変レコード型)に対応しており、Aというデータは、BというデータかまたはCというデータであることを示している。

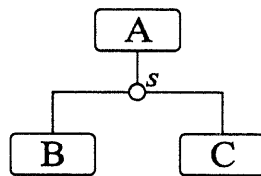
時間直和は図に示すように○^tで表わす。データで言えばストリームに相当しており、Aというデータが到着(あるいは発信)するということは、データBが到着(発信)するかまたはデータCが到着(発信)するということを表わしている。イベントならばAが起こるということが、BまたはCが起こるということである。

c. 列

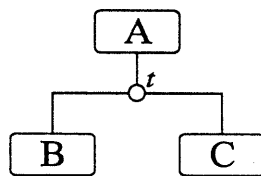
列構成関係は、図5(1)に示すように、上位のものAは、下位のものB空間あるいは時間上並んでいることを示している。列そのものの集合論上の解釈は同じ集合の無限の直積で自然数によって添え字付けられているものである。空間上列をなしているとは、Aがある時点(期間)で存在するということ、Bの並びがその時点で存在すると



(1)直和関係

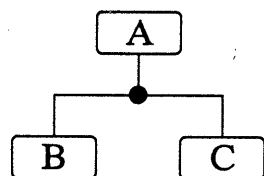


(2)空間直和

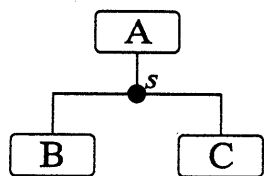


(3)時間直和

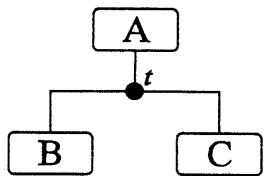
図4 直積構成関係



(1)直積関係



(2)空間直積



(3)時間直積

図3 直積構成関係

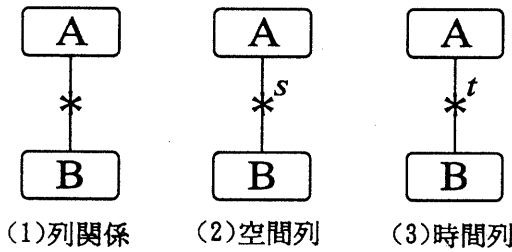


図5 列構成関係

等価ということである(図5(2))。時間上列をなしているとは、Aが起こるということは、Bの並びが次々に起こるということと等価ということである(図5(3))。

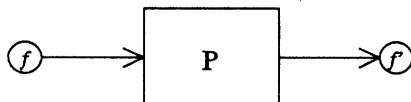
空間列は図に示すように $*^s$ で表わす。データ構造ではリストや配列に対応しており、Aというデータは、Bというデータの並びである。

時間列は図に示すように $*^t$ で表わす。ストリームに相当しており、Aというデータが到着(あるいは発信)するということは、データBが次々に到着(発信)するということを表わしている。イベントならばAが起こるということが、Bが任意回起こるということである。

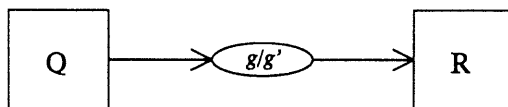
3. 4 フロー

構成関係が対象の上位/下位という縦の関係を仕様化しているのに対し、フローというのは、主として横の関係に関する抽象概念である。従来のデータフローの考え方や、S.Shlaerのオブジェクト指向開発技法でのバッチの概念、さらには、対象世界のものの流れ(移動)等を統一的に扱うことを意図している。

仕様中で別々の箇所では認識されたものを同一のものとして扱う抽象化をフローと呼ぶ。図6(1)はフローの概念を表わしたものである。 f と f' とは異なる所ではあるが、これを同一のものがPによって変形(変換)されたとみなすことができる。



(1)フローの概念



(2)アスペクト変換

図6 フローの概念

場合がある。データフローモデルでは f や f' はデータで、Pは関数(処理)ということになり、これによって計算による因果関係を表わしている。

同一のものという認識方法を導入すると、ものが扱われる時々の方によって、見方を変える必要が出てくる。筆者はこれをアスペクト変換と呼んでいる。図6(2)では、Qからの扱われ方が g という見方で行われ、同じものをRからは g' という見方で行われることを示している。このような g と g' 、あるいは、 f と f' といったアスペクト変換は3.3節の構成関係と深く関連しており、データ構造の視点からは構造不一致の概念として整理されている。

いくつかの代表的なアスペクト変換の例を上げると、下記ようになる。

- a. 時間/空間変換：時間的構成関係を同構造の時間構成関係に、あるいは、その逆に変換することである。
- b. 合成/分解変換：構成関係の下位のものを扱う見方から、下位構造を捨象して上位のものとして一括してものを扱う見方への変換を合成変換、その逆を分解変換という。組み立てラインの部品から合成部品への流れは合成変換の例である。
- c. 順序変換：同種の構成関係で下位のものの順序が入れ代わる場合のことを言う。これは、データ構造ではソートを含む場合に必要な仕様化である。
- d. 境界変換：構成関係において下位の構成要素は同じで、順序も保存されているが、そのまとめ方が異なる場合である。
- e. 脈絡変換：意味のない下位構造のものから、いくつかの意味の視点から対応するものを合成する変換を脈絡変換と言う。

3. 5 プロセス

プロセスの概念は制御システム、特に、リアルタイムやリアクティブシステムと呼ばれる分野の仕様化では主要な役割を果たす。無論、設計概念さらには計算機のOSの仕掛けとしてのプロセスとの関係にまで最終的には帰着させなくてはならないが、ここでは、対象世界モデリングや要求分析・定義段階でのプロセス概念について簡単に述べる。

J.RumbaughのOMTや、D.Harelのステイト

チャート等で扱っているふるまいの仕様化は、リアルタイムやリアクティブシステムの必要な側面を記述している。ふるまいの仕様化は、あるイベントが起こった時、そのイベントに起因してどういった結果としてのイベントがどのように起こるかを記述することである。このイベントの因果関係が主要な位置を占めることは、対象世界（制御システム）の特性でもある。

ふるまいは、単純な逐次処理や状態遷移モデルでは記述することはできない。なぜなら、そこには本質的に並行性と非決定性の問題を含んでいるからである。ステイトチャートのAND分解も、一見状態遷移モデルの等価な拡張（すなわち記述量を減らすための仕掛け）に見えるが、実際には、並行プロセスモデル（この場合はブロードキャストイングプロセスモデル）へ表現能力は上がっている。また、JSDのプロセスネットワークはCSPモデルそのものである。

5. 対象世界モデリング

前提としている生産パラダイムでは、2節で述べたように山モデルの見方によって、抽象化のプロセス（山登り）と、具体化のプロセス（山下り）とに大きく分割される。前者に含まれるのが、対象世界モデリングと要求分析・定義であり、後者に含まれるのがいくつかの設計とプログラミングである。大局を言えば、前者の世界は対象世界の構造を、後者の世界は計算機の構造を反映したものでなくてはならない。しかるに、対象世界の問題設定は、本来、計算機とは関係ない世界である。すなわち、計算機を使用しなくても解決出来るかもしれないし、使用しても解けないかもしれない。従って、対象世界の記述を、計算機の構造をたとえ抽象化したとしても、これを含む概念で仕様化を進めるのは設計上の意志決定を同時に行っていることになってしまう。

そこで、ソフトウェアシステムの記述の前に、当該組織の持つ情報や業務そのものをモデル化し、しかる後に、ソフトウェアシステムに対する要求分析・定義をこれをもとに行うというアプローチが妥当であると考えることが出来る。

対象世界のモデリングは開発工程上もコストが余分にかかるが、頻繁なバージョンアップや環境変動、システムの拡張といった場面を考慮すると全体のライフサイクルコストは著しく低減出来るものと考えられる。

当該組織の知識のモデル化は、その組織として、業種といった若干広い意味で捉えるとシステムインテグレーションをビジネスとする場合には極めて有効である。

6. 要求分析・定義

要求分析・定義段階ではじめて、ソフトウェアシステムを対象とした仕様化が行われる。対象世界モデリングが人間社会や自然界までも含んだ意味でのシステムの仕様化であったのに対して、ここでは、ソフトウェア（計算機システム）が対象になる。

ここではそのソフトウェアシステムを、導入する組織の責務に適合し、かつ、システム構築の早期段階でその要求の妥当性を検証・確認出来る仕掛けを提供し、さらに、対象世界の時々刻々と変わる状況に迅速に対応したシステム要求とそれを受けた迅速なソフトウェアの保守とを行えるようにすることを目的としている。

要求分析・定義では、まず、範囲限定化とインタフェース定義とを行わなくてはならない。範囲限定化とは、対象世界モデル記述の中から、当該システムとして（制御の）対象とする範囲を記述することである。これによって、対象世界モデル記述は大きくシステムと環境とに分割される。これ等二者の間の関係を定義するのがインタフェース定義である。

一口に範囲限定といっても、これだけでは判断の基準がないため仕様化は難しい。既存のシステムの保守を対象としている場合には、そのシステムそのものの記述が対象世界モデル記述の中に含まれており、それとの差分を考えればよいのでさほど困難は伴わないであろうが、新規開発や大きな変更を伴う場合には、当該分野のソフトウェアシステム構築のリファレンスモデルを設定する必要がある。ここで言うリファレンスモデルとはソフトウェアシステムに対する要求項目の抽出順序やそれ等の間の関係を規定するモデルのことである。

制御システムのソフトウェアのリファレンスモデルは、例えば、図7上部に示すように、制御の考えを外界（環境）の状態をトラッキングし、その情報を基に外界をコントロールし、状態を適宜ロギング（報告）するということであり、これを組み合わせることによって要求項目を分析することができる。また、販売システムの場合には、

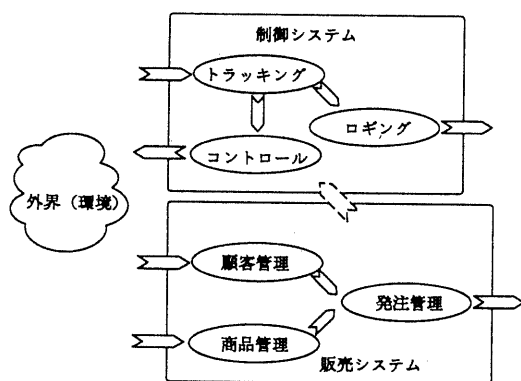


図7 リファレンスモデル例

図7下部に示すように顧客、商品（製品）、受注といった枠組みで要求を整理することができるであろう。リファレンスモデルを旨く決めるということが当該分野のソフトウェアの構成方法としてのノウハウを決定する上で重要な鍵を握るものと思われる。

リファレンスモデルの役割を生産パラダイムに則して述べるならば、これは要求仕様と組織の目的との間のギャップを扱うために、より抽象度の高い共通の基盤を与えている。

7. おわりに

本論文で扱った事項を対象として、基本技術としての意義についていくつかのトピックスに触れておく。

(1) 再利用と部品化の視点：

再利用と部品化の問題意識はソフトウェア工学発祥以来の永遠の課題と言っても過言ではない。近年でもこの視点からオブジェクト指向技術や形式的仕様記述を見直そうという動きも活発である。また、この問題は技術的な観点のみではなく管理上の仕掛けも重要であるということもよく言われている。

本報告で述べた技法では、対象世界モデルを基にソフトウェアの体系化を図るという点では、原理的に部品体系構築上の技術として使用することができる。特に、ユーザインタフェースやデータベースといった汎用のパッケージではなく、アプリケーションドメインのノウハウを集積した部品の体系化には対象世界モデルは欠かせないものになるものと思われる。

生産パラダイム上のプログラムの世界での再利用の仕掛けはデータベース、ユーザインタフェース、ネットワーク等のパッケージやプラットフォームのアーキテクチャである。要求仕様の世界では、システムやサブシステムの構造（リファレンスモデル）になるであろう。目的については、組織の目的自身を導出する技術は経営論の問題であるため一概には言えないが、対象世界記述がインデックスとして機能するものと考えられる。

(2) 情報系技法との融合：

ビジネスアプリケーションや情報系のデータベースシステムの開発では、組織の持つ情報分析を中心に行う方法論が一般的である。ここでの枠組みでは、サブシステムの分割によって、それが情報系ならば情報中心の分析方法（例えば、実体・関連モデルを使用した情報分析を行うことやインフォメーション・エンジニアリングの適用等）、制御系ならば制御やふるまい中心の分析方法を行い、それ等の間のインタフェースを規定することによって全体のシステムの要求分析・定義を行うことが出来る。

(3) オブジェクト指向開発技術との関り：

もともと本技法は全ての開発工程を細かく規定するものではなく、各アプリケーションで確立しているよい部分はそれとして残して、全体としてのライフサイクルコストの観点からのいくつかの必要な技術を提供しようというアプローチを採っている。特に、標記のオブジェクト指向開発技術は有効な面が多いので考慮に値する。設計以降については、順次オブジェクト指向分析、設計、プログラミングの技術と融合させることが出来る。

分析と設計工程で、従来の関数型主体の見方に「加え」、状態の抽象化という意味でのオブジェクトの概念を持ち込むことも有効である。

【謝辞】

本研究は、産業科学技術研究開発制度「新ソフトウェア構造化モデルの研究開発」の一環として情報処理振興事業協会(IPA)が新エネルギー・産業技術総合開発機構から委託をうけて実施したものである。