Common Design Language によるプログラム設計プロセス

新谷　勝利　　内梨　寿生　　（日本アイ・ビー・エム株式会社）

１．はじめに

今日ソフトウェア開発に求められているのは，使用者に親和性があると共に、単にサポートされる現行機能のみならず、新機能の要求あるいはトラブルに対する変更／保守に対応し易いものを高品質で生産性高く作ることである。

筆者等は上記命題解決の一助にすべく次項に取り組んでいる。
－複雑で大きな対象をモデルで抽象化することに依り取扱い易くする。
－モデルの構成要素として、制御構造、データ構造、パッケージング構造を使用する。
－上記構造を置換則あるいは写像に基づき段階的に詳細化あるいは逆に抽象化して設計の正しさを検討する。
－データの抽象化に対しては有限状態機械の考え方に基づくモデルを使用する。
－モデルの詳細化／抽象化の設計プロセスにおいて、対象を記述する設計用言語，Common Design Language（ＣＤＬ）を使用する。
－上記作業を効率的に実施するためにシンタックス指向エディター等のツール／エイドを使用する。

これらを種々の演習，討議等を通してソフトウェア設計に対する基本的なアプローチの理解、あるいは、"しつけ"として持とうというのが筆者等の取り組みの目的である。　当小論において、筆者等の試行の一端を情報処理学会誌の共通問題に取り組むステップを示すことに依って紹介する。

２．基本的な設計ステップと Common Design Language(CDL)

ソフトウェア開発プロセスを基本的に次のものと考える。

要求仕様－製品仕様－製品設計－製造－保守

筆者等の試行のカバーしているプロセスは、製品仕様－製品設計であり、次に当プロセス内において、どのような設計ステップを踏むか、夫々においてＣＤＬをどの様に使用してゆくかを概説する。

２－１．要求仕様の検討

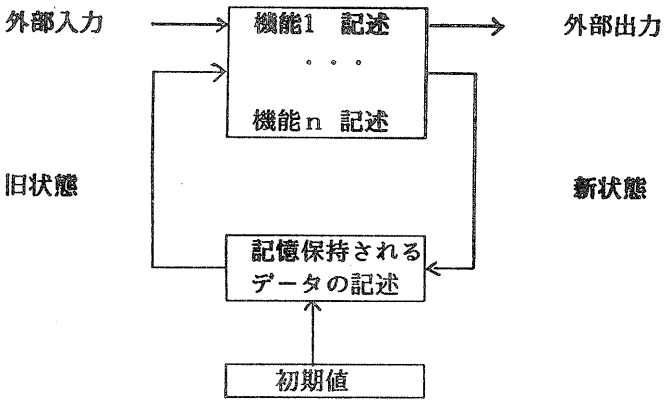
たとえ製品仕様－製品設計をカバーするとしても、製品仕様は、要求仕様に対して外部的に約束するものであるから、要求仕様は充分検討され、正しく製品仕様に反映されなければならない。　検討の過程で、不明の点、製品には反映しない要求等の問題点が発生すれば問題解決者としての設計者は要求者とそれらを検討、確認しないで勝手に解釈、決定するべきではない。　一般的に要求者の意図が製品に反映されるのかどうかの要求者側からの最初の大規模な検証は製品仕様を通してなされる。

当小論においては、要求仕様の検討の方法論に関しては特定しない。　共通問題は、情報処理 V.25 N.9，N.11，V.26　N.5　に説明されているが，問題は自由解釈で良いとされているので、筆者等がどう解釈したかは製品仕様に記述してある。

２－２　製品仕様作成過程

製品仕様は、要求された開発対象をモジュールの集合と考え、１つのモジュールは有限状態機械の考え方に基づいて他のモジュールとはカプセル化に依り分離してモデル化、検討され表現される。次に、有限状態機械の図解表現とＣＤＬ表現を示す。

図解表現



ＣＤＬ表現

```
MODSPEC        モジュール名　(生成パラメーター) ;
  STATE  ・・・  データの記述
    TYPE        モジュール名　=　データ構造記述;
    INVAR       TYPEの補注;
    INIT        初期値;
  TRANS  ・・・  機能の記述
    ENTRY         機能１名　　(パラメーター)
          "機能１の記述" ;
                  ・
                  ・
                  ・
    ENTRY         機能ｎ名　　(パラメーター)
          "機能ｎの記述" ;
END  モジュール名;
```

２－３．製品仕様の具体化

製品仕様は要求仕様側に近い外部仕様のみを意図した抽象的なものと、製品設計側に近い内部仕様を意識した、より具体度の高いものがある。設計を進めてゆく過程においてこの間の記述をＣＤＬでは次の様に行なう。

```
MODDES      モジュール名（生成パラメーター）；
 STATE
  TYPE      モジュール名 ＝ REC
              コンパイラーのサポートするレベルのデータ構造；
              （別のMODSPECを参照する抽象的なデータ構造；）
            END；
  INVAR     TYPEの補注；
  INIT      初期値；
  MAPPING   MODSPEC ：＝ MODDESのデータ空間記述；
 TRANS
  ENTRY     機能１名（パラメーター）
              "上述データ構造を考慮した機能１の記述"；
                  ・
                  ・
                  ・
  ENTRY     機能ｎ名（パラメーター）
              "上述データ構造を考慮した機能ｎの記述；
END モジュール名；
```

２－４．製品設計

製品設計は、それに依って対象とする計算機上でプログラムとして実現されるが、対象機械、使用言語、組込み機能のレベル、部品再使用のレベル等に依って、実現の形態が異なり、ＣＤＬは夫々に対応する次のようなパッケージング構造を提供する。

MODDES, PROC, FUNC, SEGMENT, PROG

２－５．仕様から設計への変換時の正しさの検証

ＣＤＬはそれに依る記述テキスト中に仕様と設計の双方を同時にサポートしたり、仕様と設計を関連づけるキー・ワードを提供する。

制御構造の段階的詳細化時

```
第一段階          第二段階

f；              DO  "f"；
                   a；
                   b；
                 END；
```

第二段階の"ｆ"は単なるコメントではなく、前段階のアサインメント・ステートメントであり、 ａ，と ｂの抽象表現である。 即ち、"・・・"が仕様でその展開を設計ということができる。 任意の段階の記述テキストをシンタックス・チェックするツールは"機能記述"のみならず仕様と設計双方の間の関係をもチェックしリストする。

モジュールの具体化時

ＭＯＤＳＰＥＣは抽象度のより高いレベルのデータ構造を中心とした記述であり、
モジュールの仕様といえ、ＭＯＤＤＥＳは具体的に実現されうるデータ構造を中心とした
記述であり、モジュールの設計といえる。　両者はそのデータ構造記述が異なっても
対象とするデータ空間は同じでなければならないという考えのもとにそれを確認する
キー・ワードが準備されている。設計をＭＯＤＤＥＳで記述する時にその確認をする。

```
MAPPING      MODSPECのデータ空間  :=  MODDESのデータ空間;
```

３．データの抽象化

前章で概説したように筆者等の設計への基本的アプローチは、抽象化データ・タイプである。
よって対象のモデル化の第一ステップは、データ構造の検討にある。

３－１．データ構造の検討およびその記述

１）在庫の有無の観点より（品名別）
品名－数量

２）入庫の観点より（コンテナ別）
コンテナ１－（品名１、数量）
－（品名２、数量）
コンテナ２－（品名２、数量）
－（品名３、数量）

３）品名別とコンテナ別を共にサポートするために

品名１－コンテナ１－数量１
コンテナ２－数量２

品名２－コンテナ１－数量３
コンテナ３－数量４

上記観点は次の二種類にまとめられ、ＣＤＬで以下のように記述される。

コンテナ毎のアクセスを考慮　（上記２に対応）

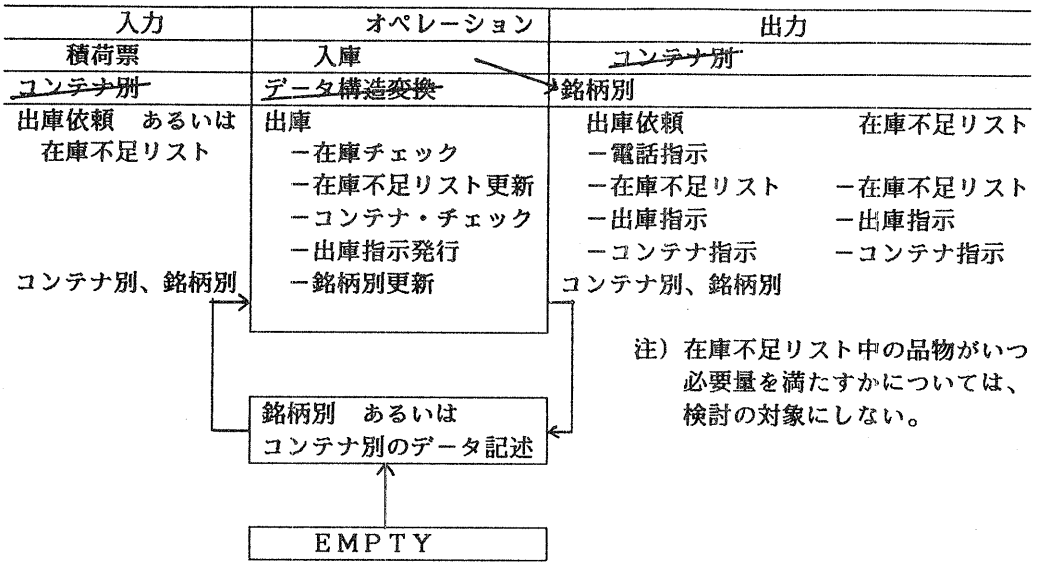
```
TYPE  コンテナ別  =  MAP  OF（C#、コンテナ内容）;
TYPE  コンテナ内容  =  SET  OF  REC
                          品名;
                          数量;
                        END;
```

品名に依るアクセスでコンテナ配分をも考慮　（上記３に対応）

```
TYPE  銘柄別  =  MAP  OF（品名、コンテナ配分）;
TYPE  コンテナ配分  =  SET  OF  REC
                          C#;
                          数量;
                        END;
```

どちらのデータ構造を選択すべきであろうか、次に検討する。

3－2．データ・タイプ化手順



注）在庫不足リスト中の品物がいつ必要量を満たすかについては、検討の対象にしない。

ＣＤＬはstrong typingの言語であるから異なるタイプの２つの構造を扱う時には変換が必要となる。　上図の／が示すように銘柄別タイプに統一すれば対応できることがわかる。

4．モジュール記述

ＣＤＬにおいては、ユーザーの処理系を含む抽象化データ・タイプをモジュールと呼び、ＭＯＤＳＰＥＣとしてパッケージ記述する。

```
MODSPEC  銘柄別；
 DEF
  TYPE   コンテナ配分 ＝ SET OF REC
                            C＃；
                            数量；
                          END；
 STATE
  TYPE   銘柄別 ＝ MAP OF（品名、コンテナ配分）；
  INVAR  TRUE；
  INIT   EMPTY；
 TRANS
  ENTRY  入庫（IN  積荷票、OUT  銘柄別）
    "積荷票の内容で銘柄別を更新する"；
  ENTRY  出庫（IN      出庫依頼、
              INOUT   銘柄別、  在庫不足リスト
              OUT     出庫出力：＜電話指示，コンテナ指示＞，出庫指示書）
    "出庫すべき品物が，銘柄別に充分あれば，コンテナ・チェックの後に出庫指示書を
     出力し更に出庫入力が在庫不足リストならば在庫不足リストを更新する。
     品物が充分なくて出庫入力が出庫依頼ならば，不足の旨電話し，在庫不足リスト
     を更新する。"
END  銘柄別；
```

５．モジュール記述の具体化

５－１．　銘柄別の具体化

抽象化の程度をよりプログラムに近いレベルまでおろし、銘柄別のサイズは主記憶装置の範囲内とすれば、ＡＲＲＡＹ構造が考えられる。

```
TYPE　銘柄別 ＝ ＲＥＣ
                  ＡＲＲＡＹ（品名｜｜Ｃ＃）ＯＦ 数量；
                  品名：ＳＴＲＩＮＧ；
                  Ｃ＃：ＳＴＲＩＮＧ；
                ＥＮＤ；
```

５－２．　ＭＯＤＤＥＳパッケージ記述

ＭＯＤＳＰＥＣにおけるＭＡＰ構造をＡＲＲＡＹ構造に具体化した時の記述はＭＯＤＤＥＳで行う。

```
ＭＯＤＤＥＳ　　銘柄別；
 ＳＴＡＴＥ
  ＴＹＰＥ　　銘柄別 ＝ ＲＥＣ
                  ＡＲＲＡＹ(品名｜｜Ｃ＃)ＯＦ　数量；
                  品名：ＳＴＲＩＮＧ；
                  Ｃ＃：ＳＴＲＩＮＧ；
                ＥＮＤ；
  ＩＮＶＡＲ　　ＴＲＵＥ；
  ＩＮＩＴ　　品名　：＝ ＮＵＬＬ　ＡＮＤ　Ｃ＃：＝　ＮＵＬＬ；
  ＭＡＰＰＩＮＧ 銘柄別　：＝　｛(品名，(Ｃ＃，数量)) ｜ {品名} ｘ {Ｃ＃} の行列は
                                各々配分された数量を持つ｝；
 ＴＲＡＮＳ
  ＥＮＴＲＹ　入庫（ＩＮ　積荷票，ＯＵＴ　銘柄別）
    "積荷票を読み品名毎に（品名｜｜Ｃ＃）の引数に対応する数のエントリーを追加する"；
  ＥＮＴＲＹ　出庫（ＩＮ　　出庫依頼，
                ＩＮＯＵＴ　銘柄別，在庫不足リスト
                ＯＵＴ　　出庫出力：＜電話指示，，コンテナ指示＞，出庫指示書）
    "出庫すべき品物が、銘柄別に充分あれば、コンテナチェックの後に出庫指示書を出力し、
     更に出庫入力が在庫不足リストならば在庫不足リストを更新する。
     品物が充分なくて出庫入力が出庫依頼ならば，不足の旨電話し，在庫不足リストを
     を更新する。"
ＥＮＤ　銘柄別；
```

６．処理系の詳細化

ＭＯＤＳＰＥＣ／ＭＯＤＥＳの記述から共通問題は銘柄別というＡＲＲＡＹ構造のデータ構造で表現でき、入庫と出庫という処理系からなる抽象化データ・タイプと考えることができる。処理系は従来からの構造化プログラミングおよび段階的詳細化の方法で抽象レベルをプログラミングのレベルまで下げてゆくことにする。

６－１．入庫の詳細化

ここで前章ＭＯＤＤＥＳのＥＮＴＲＹ入庫の機能を入庫処理とまとめる。

```
ＬＥＴ　入庫処理　ＢＥ　ＥＮＴＲＹ入庫の機能記述；
```

第一段階　　第二段階

```
入庫処理；　　ＤＯ"入庫処理"；
                積荷票を読む；
                内蔵品名、数量"欄の処理；
              ＥＮＤ；
```

第三段階

```
ＤＯ　"入庫処理"；
  積荷票を読む；
  "内蔵品名、数量"欄の処理"
    ＤＯ　ＷＨＩＬＥ　内蔵品名　¬＝　ＮＵＬＬ
      ＡＲＲＡＹ（品名｜｜Ｃ#）ＯＦ　数量　のレコード・エントリー準備；
      ＡＲＲＡＹの更新；
    ＥＮＤ；
ＥＮＤ；
```

６－２．出庫の詳細化

ここで前章ＭＯＤＤＥＳのＥＮＴＲＹ出庫の機能を出庫処理とまとめる。

```
ＬＥＴ　出庫処理　ＢＥ
   ＤＯＭＡＩＮ：　出庫依頼、在庫不足リスト、銘柄別
   ＲＡＮＧＥ　：　出庫出力、　出庫指示書，　銘柄別，　在庫不足リスト
   ＲＵＬＥ　　：　ＭＯＤＳＰＥＣのＥＮＴＲＹ出庫の機能記述；
```

詳細化時における考慮点は次の事項である。

- モジュール化
- call by value
- グローバル・データの極小化
- 共通ルーチンの識別

これらを考慮しながら、次に段階的詳細化のステップを示す。

第一段階

```
出庫処理；
```

第二段階

```
"出庫処理"

IF 入力 ＝ 出庫依頼

THEN;
  "出庫依頼処理";
ELSE;
  "在庫不足リスト処理";
END;
```

第三段階

```
"出庫処理"

IF 入力 ＝ 出庫依頼

THEN "出庫依頼処理";
  IF 在庫充分

    THEN "在庫充分時処理";
      コンテナ・チェツク；
      出庫指示書発行；
      銘柄別の更新;
    ELSE "在庫不足時処理";
      電話指示;
      在庫不足リスト更新；
    END;
  ELSE "在庫不足リスト処理";
    IF 在庫充分

    THEN "在庫充分時処理";
      コンテナ・チェック；
      出庫指示書発行；
      銘柄別の更新；
      在庫不足リスト更新；
    END;
  END；
```

7．おわりに

当試行で意図していることは、要求仕様から製品設計に至る過程において、思考過程を記述できる共通の設計言語とアプローチ法を持つことに依って開発チーム内およびその関係者との間に誤解の発生するのを極少化しようということにある。更に、これらの試行を通じてソフトウエア工学に対する関心を助長してきていると考えられる。

CDLの採用は設計の実行可能性ということからプロトタイピングへの可能性を持つし、有限状態機械モデルに依るユーザー定義の抽象化データ・タイプはソフトウェアの部品化、再使用の可能性を大きくするものである。

おわりに本小論をまとめるにあたり貴重なご意見をいただいた久保未沙氏をはじめ諸氏に感謝します。