ソフトウェア工学 70－4
（1989．12．8）

オブジェクト指向による ソフトウェアプロセスの記述と実行について

荻原剛志　井上克郎　鳥居宏次
大阪大学基礎工学部

ソフトウェア開発プロセスを記述，実行するための，オブジェクト指向に基づく記述モデルを提案する．プロダクト（文書やソースプログラムなどの生成物）の記述にオブジェクト指向を導入することで，プロセスおよびプロダクトの記述をクラス継承を使って容易に行うことができ，また，並列した複数の開発作業をオブジェクトの並列動作として記述することが可能である．プロセスの記述モデルは，プロセスの流れの定義，プロダクトの相互関係の定義，プロダクトの定義の３つの部分からなる．それぞれの部分は独立に記述でき，また実行中にオブジェクトを動的に結合させることが可能なため，開発プロセスの記述，実行を容易に行うことができる．

PROCESS DESCRIPTION AND ENACTION MODEL BASED ON OBJECTS

Takeshi OGIHARA, Katsuro INOUE and Koji TORII
Faculty of Engineering Science, Osaka University

An object-oriented model for description and enaction of software process is proposed. Software products such as documents and source programs are represented as objects. Inheritance mechanism helps to define the classes of these objects. Concurrent development works are described as concurrent execution of the objects. The model consists of three definition parts: process flow, relation between the objects, and operations of the objects. These parts can be described independently. The objects are created, related each other, and deleted dynamically during the development works.

1. はじめに

ソフトウェアの開発過程の形式的な記述，および，この記述の実行によるソフトウェア開発支援の試みが盛んになっている．

これまでのソフトウェア開発では，作業の方針や考え方は示されていても，具体的にどのようなツールを用いて，どのような順序で作業を行うかなどについては明示されていないことが多かった．このため開発者は，モジュールの設計，ソースプログラムの作成といった，ソフトウェア開発にとって本質的と考えられる作業以外に，作業進行の管理，ツールの選択，起動，ファイルの管理などを自分自身で行わなければならなかった．

そこで，ソフトウェア開発作業の過程（これをソフトウェアプロセス，あるいは単にプロセスと呼ぶ）を形式的に記述しておき，その記述に基づいて作業を行うことが考えられる．さらに，この記述に基づいて作業の内容に合ったツールを順次起動し，行うべき作業を開発者に指示するようなシステムがあれば，開発者は指示された作業のみに専念でき，一連の作業を効率よく行えるようになるであろう．このようなシステムは開発者の負担の軽減とともに，ソフトウェアの生産性，品質の向上にも効果があると考えられる．

我々はこれまでに，ソフトウェア開発作業に伴うツールの起動，メッセージの表示などを記述，実行するための言語*PDL*(Process Description Language) とそのインタプリタを作成している[1,2]．また，このシステムを利用してさまざまなプロセスの記述，実行を行ってきた[3]．

これまでの我々の試みでは，ツールの起動やウィンドウの操作等のプロセスは記述できるが，ソフトウェア開発に伴うプロダクト（文書やソースプログラムなどの生成物）の管理作業の記述は煩雑になることが分かった．これは個々のプロダクトがさまざまな相互関係を持っていること，開発作業中にそれらの関係が動的に変化してゆくことなどによるものと考えられる．

本稿では，オブジェクト指向の考え方による，プロセスおよびプロダクトの記述モデルを提案する．記述モデルは，プロセスの流れの定義，プロダクトの相互関係の定義，プロダクトの定義の３つの部分からなり，これを Trineモデルと呼ぶ．

以下，2.ではプロセス記述の概要と，プロダクト管理の記述について述べる．3.ではクラス定義の方法とオブジェクトの並列実行について述べる．4.ではオブジェクト指向によるプロセスの記述モデル Trineの提案を行う．5.ではモデルの記述形式や新たな機構の導入について検討する．

2. プロセス記述とプロダクト管理

2.1 プロセスの記述

本稿ではプロセスを，一人のソフトウェア開発者が一台のワークステーション上で行う一連の作業であると考える．チームによる複数の計算機を用いた開発や，発注者との話し合いなどの人的側面については考慮しない．

我々の作成したプロセス記述用言語*PDL*では，実行可能なプロセスの記述は，ツールの起動，ウィンドウの操作，メッセージの表示，開発者からの応答の入力の系列として記述される．また，次の作業に進むことができるか，あるいは作業をやり直すべきか等の条件を判定し，作業の順序を決定することができる．

図１に簡単なＣプログラム開発の作業の流れを示す．このプロセスを*PDL* で記述，実行すれば，自動的に新しいウィンドウが開いてエディタが起動する．ソースプログラムを作成してエディタを終了すると，自動的にコンパイルが始まる．コンパイルで何らかの誤りが発見されれば再びプログラムの編集の段階に戻る．このようにして，プロセス記述に従い，開発作業を支援することができる．

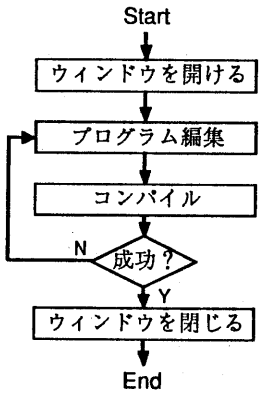

図１　簡単なＣプログラムの開発プロセス

2.2 プロダクト管理の記述

開発作業によって作成される文書，ソースコード，目的プログラムなどをプロダクトと呼ぶ．プロダクトは種類や状態，相互関係，物理的な情報など，さまざまな情報（プロダクトの属性と呼ぶ）を持ち，また，生成，編集，表示などの操作を行うことができる．

図１であげた例では，Ｃのソースプログラムが１つしかない単純なものを考えている．しかし一般には，プログラムは複数のモジュールから成り，いくつものソースファイル，ヘッダファイルから構成されている．このような複数のプロダクトから構成されるソフトウェ

アの開発のためには，それぞれのプロダクトの生成，修正，あるいはバージョンの管理などの作業を支援できなければならない．例えば，あるソースファイルを修正する場合，同時に修正しなければならないソースファイルは何か，作り直さなければならないオブジェクトプログラムは何か，などの情報を利用して必要な部分への修正を効率的に行いたいという要求がある．

しかし，ソフトウェアをどのようなモジュールから構成し，どのようにファイルに分割するかということは開発を開始してから決まる事項であり，また，後から変更されることもある．このため，開発作業中に現れるすべてのプロダクトについての作業や情報をあらかじめ記述しておくことはできない．

また，プロセスの流れを中心としたこれまでの記述方法では，プロダクトの管理についての記述とプロセスの流れの記述が混在してしまい，記述・変更が行ないにくい面があった．ファイルの編集方法やツールの起動方法など，プロダクトの属性に依存する部分をプロセスの流れと切り放して記述しておけばそれぞれの部分の記述・変更が容易になり，再利用性も向上する．

3. クラス定義とオブジェクトの並列実行

3.1 オブジェクトの導入

この節では，開発作業の進行に伴って生成，使用されるさまざまなプロダクトを記述するため，オブジェクトの概念を導入する．

開発作業中に使用されるプロダクトは，その形式，用途，内容などによっていくつかの種類に分けることができる．プロダクトはこれらの種類ごとに，共通に持つ属性，操作についての記述をクラスとして定義する．実際の個々のプロダクトは，このクラスのインスタンスオブジェクトとして動的に生成されるものと見なす．

本稿ではクラス定義の詳細については論じないこととするが，整数，文字列などのデータ型，および配列やリストなどのデータ構造はすでに定義されているものとして，記述中で自由に用いることができるものとする．また，クラスの記述には手続き言語風の記述形式を用いる．

3.2 継承によるクラス定義

新たなクラスの定義を行う時，その定義がすでに存在するクラスの定義を含んでいる場合がある．このような場合，すでに存在しているクラス定義を継承することにより，新たなクラスの定義を簡潔にすることができる[4].

例えば，テキストファイルを表すクラス TEXT があらかじめ定義されており，そのインタフェース（外部から使用できる属性と操作の集合）が図2に示すものであったとする．このとき新たにCのソースプログラムを表すクラス SRC を，クラス TEXT の定義に，compile という操作を付け加えたものとして定義できる（図3）．なお，exec はツールを起動するための関数，self はそのインスタンスオブジェクト自体を表す記法であるとする．

クラスの継承機構を導入することにより，さまざまなプロダクトのクラスをより簡単に分かりやすく記述することができるようになる．

```
class TEXT {
    var name: string;
    ...
    proc create(s: string) ...
    proc delete() ...
    proc edit() ...
    proc view() ...
    ...
}
```

図2　クラス TEXT のインタフェース

```
class SRC : TEXT {
    proc compile(out success: bool;
                 out obj: OBJ) {
        obj.create(self.name);
        exec("cc -c "+self.name);
        success = (status() = 0);
    }
}
```

図3　クラス SRC の定義

3.3 オブジェクトの並列実行

ソフトウェアの開発作業では，いくつかの作業を並列に行いたいことがある．例えば，要求仕様書を表示しながらモジュール設計を行ったり，関連するいくつかのソースプログラムを別々のウィンドウで並行して編集したりということが頻繁に起こる．上で述べたように，それぞれのプロダクトがひとつのオブジェクトとして表現されている場合，このことは，複数のオブジェクトを並列に実行することにあたる．

複数のオブジェクトの並列実行についてはさまざまな方法が提案されているが，本稿では具体的な実現方法については言及しない．ただし，以下の記述例では obj をオブジェクト，op() を操作とした時，

obj.op()　で操作を逐次的に起動すること，

obj<-op()　で操作を現在の作業と並列に起動することを表すとする[5]．逐次的な起動の場合，起動した操作の動作が終了するまで次の動作に移れないが，並列に起動した場合には，起動した操作と現在の作業とを並列して行うことができる．ひとつのオブジェクトが一度に実行できる操作は1つだけであると仮定する．

*PDL*によるプロセス記述の経験から，ソフトウェア開発の場合，実際に並列して行われる作業の数は多くても4つくらいまでである．これは開発者があまり多

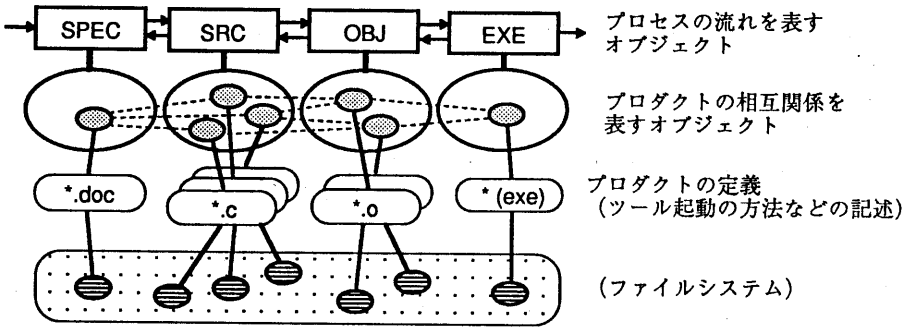

図4　プロセス、プロダクトの記述モデル

数の作業を同時に処理できないためであり，逆に，作業の数をあまり増やすとかえって作業が行ないにくくなる．並列して実行されるオブジェクトの数を必要以上に増やさないために，現在実行されているオブジェクト数などの情報を実行中に動的に利用して，オブジェクトの動作を制御するなどのことが考えられる．

4. プロセス記述のモデル

4.1 記述モデルの概要

ここでは開発プロセスを記述するための，オブジェクトに基づいたTrineモデルを提案する．この記述モデルは，独立に記述できる3つの定義部分からなる．記述されたプロセスは，図4のような構成を持つ．

基本的なプロセスの流れは，図の上段の，オブジェクト間の操作の流れとして表される．それぞれのオブジェクトは，開発作業によって作成されるプロダクトの集合を表している．この部分ではプロダクトの集合への操作と，集合の間の関係が記述される．

図中段はプロダクトの集合の要素と，それらの間の相互関係を表すオブジェクトである．オブジェクトの集合への操作が個々のプロダクトにどのように適用されるか，また，個々のプロダクトの間にどのような相互関係があるのかを記述する部分である．

図の一番下はファイルシステムであり，その上に，属性，操作を持ったオブジェクトとして個々のプロダクトが記述される．個々のプロダクトに対する操作がどのように実現されるかを記述する部分である．

以下では，プロセスの流れの記述方法，プロダクトの相互関係の記述方法について述べ，次にこのモデルを用いて開発プロセスを記述する手順について述べる．

4.2 プロセスの流れの記述

プロセスの流れは，開発中に作成されるさまざまなプロダクトを表すオブジェクトとその間の操作の起動によって表現することができる．

例えば，Cプログラムを開発する簡単なプロセスは図4上段のようなプロダクト間の関係として表すことができる．まず仕様(SPEC)からソースプログラム(SRC)に対してプログラム生成の操作が起動され，ソースプログラムが作成できたらオブジェクトプログラム(OBJ)に対して生成の操作（コンパイル）が起動される．コンパイルが成功すれば，実行プログラム(EXE)に対して生成の操作（リンク）が起動されるが，失敗すればソースプログラムに対して編集の操作が起動される．

このような関係を記述するために，プロダクトの集合を表すオブジェクトは

- 入力となるプロダクト集合
- 出力となるプロダクト集合
- 操作（生成，編集など）対象となるプロダクト集合

を表すための属性と，操作の定義を持てばよい．

図4の中の，オブジェクトプログラムの部分の操作の流れをクラスとして記述したものを図5に示す．ただし操作の対象は，次節で述べる相互関係を表すオブジェクトである．

```
class T_OBJ {
    var src: T_SRC;
    var exe: T_EXE;
    var obj: S_OBJ;
    proc create() {
        if obj.compile()
            exe<-create()
        else
            src<-edit();
    }
    proc edit() {
        src<-edit();
    }
}
```

図5　プロセスの流れの定義

4.3 プロダクトの相互関係の記述

2.2 でも述べたように，開発作業を効率的に支援するためにはさまざまなプロダクト間の関係についての情報を利用することが必要である．具体的な個々の関係は開発作業中に動的に変化してゆくため，あらかじめ記述中に記述しておくことはできないが，あるクラスのプロダクトがどのような種類の相互関係を持つのかということは前もって記述できる．

また，複数のプロダクトを集合的に扱う場合，個々のプロダクトを具体的にいくつ生成するのか，どのような相互関係を持つのかといったことにも，開発作業中に動的に対応しなければならない．

Trineモデルでは，プロダクト間の相互関係と，相互関係に基づく操作を記述するオブジェクト，およびそれらを集合として扱うためのオブジェクトを用いて，複数のプロダクトへの操作を記述する．

```
class R_SRC {
    var spec: R_SPEC;
    var obj: R_OBJ;
    var src: SRC;
    var old: bool;
    ...
    proc edit() {
        src.edit();
        obj.old = TRUE;
    }
    ...
}
```

図6　相互関係の定義

```
class S_SRC {
    var src: list of R_SRC;
    proc edit() {
        var p: list of R_SRC;
        p = src;
        while ( p != NULL ) {
            if ( p.element.old )
                p.element.edit();
            p = p.next;
        }
    }
}
```

図7　相互関係の集合の定義

図6は図3のクラス **SRC** の相互関係を定義するクラス **R_SRC** である．ソースプログラムが編集されると，それから作成されるオブジェクトプログラムも新たに作成しなおさなければならない．そこで，ソースプログラムの編集が行われた場合，対応するオブジェクトプログラムの相互関係を表すオブジェクトの **old** という属性を真にする．次にオブジェクトプログラムを作成する時には，**old** が真になっているプロダクトについてのみコンパイルを行えばよい．

図7は **R_SRC** のインスタンスオブジェクトの集合を定義するクラス **S_SRC** である．ここでは，**edit** という操作が起動された時，リストとして表現されているオブジェクトのうち，**old** が真のものについて次々に **edit** 操作を適用してゆくものとして記述した．

4.4 プロセス記述の手順

このモデルによるプロセスの記述は，次のような手順で行うことができる．

1) 開発作業の中で用いられるプロダクトを列挙し，どのプロダクトからどのプロダクトが作成されるかを調べる．それに従って，プロセスの流れを決定する．この段階では，ソースプログラムなど，いくつかのプロダクトの集合はひとつのまとまりと見なす．
2) プロダクトの集合に対する操作が，個々のプロダクトにどのように適用されるかを決定する．
3) 個々のプロダクトがどのような相互関係を持っているかを調べる．また，属性，操作との関係も調べる．
4) プロダクトの属性，操作を，どのようなファイル，ツールなどで実現するか決定する．
5) プロセスの流れ，プロダクトの相互関係，プロダクトへの操作について，クラスの定義を行う．

5. 議論

5.1 形式的な意味付けについて

開発プロセスの記述は，はじめに抽象度の高い記述を作成し，これを順次詳細化するという方法で作成することが多い．詳細化の各段階では，新たに得た記述が前段階の記述を正しく詳細化しているかどうかを検証する必要がある．この時，さまざまな詳細化のレベルの記述が，何らかの同一の形式的な枠組みの中で意味付けられていると検証を容易に行うことができる．

我々はプロセス記述言語*PDL*を，代数的仕様記述言語 ASL[6]の意味付けに基づく関数型言語として設計した．ASL は同一の形式的意味の枠組みのなかで，さまざまな抽象化レベルでの記述，および詳細化を簡明に行うことができる．*PDL*はこの ASL の性質を継承しており，開発プロセスの記述を段階的詳細化によって容易に作成することができる．

クラスの記述にも同様な意味付けを導入することにより，段階的詳細化による記述の作成，検証を容易にできると期待される．また，クラス定義を部品として再利用する場合の正しさの検証等を行うことも考えられる．

しかし，ASL は手続き的言語の意味での変数を持たないため，上で述べたようなクラスの定義に，そのまま代数的仕様記述の意味付けをすることはできない．また，継承を伴うクラス定義の意味付けなどについても検討する必要がある．

5.2 プロダクトサーバの導入

上で提案したモデルでは，プロダクトに対して具体的なファイル操作を行う部分はクラス内部に定義として記述しておくことになっている．プロセス記述の実行システムがファイル操作について何の支援も行わない場合，これらの操作の定義は，ＯＳ（あるいはハードウェア）の提供する機能を直接利用して記述しなければならない．このためには，ＯＳに関する豊富な知識が必要であり，また，記述自体も複雑で冗長なものになりやすいと考えられる．

これに対し，プロダクトに対する基本的な操作についてはシステムがある程度の機能を提供し，記述はこれらの機能を組み合せるだけで定義できるようにする方法が考えられる．システム内でプロダクトの操作管理を行う部分をプロダクトサーバと呼ぶ．プロダクトサーバの考え方を導入することにより，プロダクトの操作を容易に記述できると期待される．

しかし，プロダクトサーバ自体の必要性も含め，どのようなレベルのどのような機能を提供すべきかについて，十分な検討はなされていない．今後，さまざまなプロセスを記述して経験を蓄積する必要がある．

5.3 チームによる開発への拡張

本稿ではプロセスを，一人のソフトウェア開発者が行う作業に限定して考えた．しかし，各オブジェクトの並列動作を，協調して働くチーム内部の開発者であると見なせば，本稿のモデルをチームによる開発に拡張することができると考えられる．

この場合，各作業者の間のメッセージのやりとりや，分散した資源やプロダクトの管理をどのように行うかということが問題となる．さらに，各作業者をどの作業に配置するのか，全体の作業の管理・運営はどのように進めるのか，などといった人的側面に注目しなければならない．

6. おわりに

本稿では，オブジェクト指向に基づくプロセス記述モデルTrineの提案を行った．Trineでは開発プロセスをプロダクト間の相互関係と操作の起動によって記述する．並列した複数の開発作業をオブジェクトの並列動作として記述することが可能であり，また，実行中にオブジェクトを動的に結合させることにより，それぞれの開発作業に適した支援を柔軟に行うことができる．

現在，Trineモデルに基づくプロセス記述が可能な言語，およびその実行処理系を設計中である．

複数のプロダクトを対象とした開発作業を支援するシステムは，プロセスの実行の状況や，次に行うべき作業，作業の対象となるプロダクトについての情報を整理された形で提供し，開発者の要求に適切に応えなければならない．このためには，使いやすいインタフェースを持った実行処理系が欠かせないものであると考えられる．開発作業支援のためにどのような機能がインタフェースに求められているのかという点についても，今後検討すべき課題である．

参考文献

[1] K.Inoue, T.Ogihara, T.Kikuno and K.Torii: "A Formal Adaptation Method for Process Descriptions", Proc. of 11th International Conference on Software Engineering, pp.145-153 (1989).

[2] 荻原，井上，鳥居："ソフトウェア開発を支援するツール起動自動制御システム"，電子情報通信学会論文誌(D-I), J72-D-I, 10, pp.742-749 (1989).

[3] 稲田，荻原，井上，鳥居："ソフトウェア開発過程の形式化とその詳細化による支援システムの作成 — ＪＳＤを例として—"，電子情報通信学会論文誌(D-I), J72-D-I, 12（予定）.

[4] O.-J.Dahl and K.Nygaard: "Class and subclass declarations", IFIP Working Conference on Simmulation Programming Languages (1967), North-Holland(1968).

[5] 米澤，柴山，Briot，本田，高田："オブジェクト指向に基づく並列情報処理モデル ABCM/1 とその記述言語 ABCL/1"，コンピュータソフトウェア，3, 3, pp.9-23 (1986).

[6] 東野，関，谷口："代数的仕様から関数型プログラムの導出とその実行"，情報処理，29, 8, pp.881-896 (1988).