



Title	プロダクト間の関係記述に基づく開発支援系生成の試み
Author(s)	松永, 泰明; 岡田, 世志彦; 飯田, 元 他
Citation	電子情報通信学会秋季大会講演論文集. 1991, 1991, p. 40-40
Version Type	VoR
URL	https://hdl.handle.net/11094/51096
rights	copyright©1991 IEICE
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

D-40 プロダクト間の関係記述に基づく開発支援系生成の試み

Generation of a Development Environment from Product Relations

松永泰明 岡田世志彦 飯田元 井上克郎 鳥居宏次

Yasuaki MATSUNAGA, Yoshihiko OKADA, Hajimu IIDA, Katsuro INOUE, and Koji TORII

大阪大学 基礎工学部 情報工学科

Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University

1. まえがき

ソフトウェアの開発過程を形式的に記述し、それを基に開発手法の評価や開発支援を行うとする試みがなされている。開発過程を記述する場合、基本となるモデルに応じて、さまざまな形式化が行われる。例えば、開発過程を基本作業の系列としてとらえた記述として、我々のグループは従来より関数型言語 PDL (Process Description Language) とそのインタープリタを開発し、これを用いた記述を試みてきた^[2]。このようなプロセス指向の記述とは対称的に、開発過程において生成される生産物(プロダクト)の性質と相互関係に着目したプロダクト指向の記述が考えられる^[1]。

本研究では、このような記述を PDL の関数に変換するトランスレータを作成した。変換されたスクリプトをインタープリタによって実行することにより開発支援システムが構築可能となる。

2. プロダクト間の関係記述

このトランスレータの入力となるプロダクト間の関係記述は、プロダクトのクラス分け及び実際のプロダクトへの対応の2つの記述から構成される(図1参照)。

プロダクトのクラス分けの記述は、プロダクトを分類し、各クラスに固有の作業(操作)を定めたものである。例えばC言語によるソフトウェア開発でのプロダクトは、ソースファイル、ヘッダファイル、オブジェクトファイル、実行ファイル等に分類され、ソースファイルのクラスでは、エディット、コンパイル等の作業が定義される。またこれらの基本的なクラスを組み合わせて新たにクラスとして定義でき、そのクラスに対する作業を定めることでクラス間の関係が定義できる。例えば、図2のようにソースファイルをコンパイルしてオブジェクトファイルとなる関係はモジュールのクラスとして定義され、モジュールをリンクして実行ファイルになるという関係は、プログラムのクラスとして定義される。

実際のプロダクトへの対応の記述は、現実のソフトウェア開発で扱うプロダクトがどのクラスに属するかを宣言したものである。例えばあるソフトウェア開発でソースファイルのクラスに属するファイルには main.c と func.c がある、というような宣言である。

この記述においては、プロセス指向の場合とは異なり作業系列の実行順は指定しない。各作業に付加した前提条件が成立する作業の内の1つを作業者が選択し実行する。また、開発過程において扱うプロダクトの数等に変更があった場合、容易に変更が可能である。

3. 変換後の PDL スクリプト

変換の結果トランスレータによって出力される PDL スクリプトは、次の2つの部分から構成される。

- ・各プロダクトの持つ作業を表す関数群(作業関数と呼ぶ)。
- ・各作業の実行順序を制御する関数(制御関数と呼ぶ)。

作業関数は、入力であるプロダクト間の関係記述を変換することによって生成される関数で、この関数が評価されることによってエディットやコンパイルなどといった基本作業が行われる。制御関数は、トランスレータが上記の関数群を組み合わせることで生成される関数で、作業関数の評価順序を制御する。

4. 変換方法

作業関数は、プロダクトのクラスで定義された作業を実際のプロダクトごとに対応づけ、個々の独立した1つの関数とすることによって生成することができる。例えば、ソースファイルにエディットの作業が定義されており、ソースファイルに main.c と func.c がある場合、main.c および func.c に対するエディットを行う関数をそれぞれ生成する。例えば図1の記述より図3の作業関数が生成される。

制御関数は、各作業関数の実行順序を制御するものである。プロダクト間の関係記述には、個々のプロダクトに対する作業及びその作業を行うための条件(前提条件-例えば、ソースファイルがなければコンパイルできない、オブジェクトファイルがなければリンクできない等の条件)は記述されているが、開発過程全体の作業系列について定義していない。トランスレータは、行う作業をユーザが決定できるように次のような機能をもつ制御関数を生成する。

- ・作業の選択は、階層的なメニューの形式で実現する。
- ・メニューの選択肢は、前提条件によって制限する。

これにより、あるプロダクトに対し定義されている作業がメニューとしてすべて表示され、実行することのできる作業だけが白く表示され、選択することができる。多くの作業は灰色で表示され選択することができない。このようにして図1を変換した PDL スクリプトの実行例が図4である。ここでは、メニューの中から、func.c についての作業が選択され、func.c に対する作業が更にメニューとして表示される。この時前提条件が満たされていないため compile, view は、選択不可能となっている。

5. あとがき

このトランスレータによって、プロダクト間の関係記述を開発支援システムに変換することが可能となった。今後の課題としては、プロダクト中心の記述とプロセス中心の記述の併用の手法などが考えられる。

参考文献

- [1] 西村, 飯田, 荻原, 新田, 井上, 鳥居: “あるプロダクト間関係の記述とプロセス記述言語 PDL への変換”, 情報処理研報, SE-77-17, pp.101-106(1991-2).
- [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”, 信学論(D-1), J-72-D-1, 10, pp.742-749(1989-10).

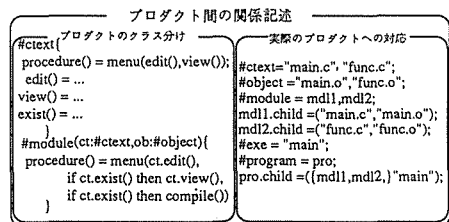


図1 プロダクト間の関係の記述(部分)

```
main_procedure(S) == menubrand("menuselect main.c",{
  ["edit",true,edit("main.c")],
  ["view",main_exists(S),view("main.c")],
  ["compile",main_exists(S),compile("main.c")],
  ["exit",true,S],S);

func_procedure(S) == menubrand("menuselect func.c",{
  ["edit",true,edit("func.c")],
  ["view",func_exists(S),view("func.c")],
  ["compile",func_exists(S),compile("func.c")],
  ["exit",true,S],S);
```

図2 変換された PDL スクリプト

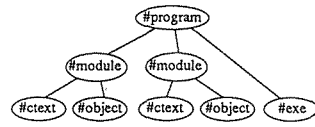


図3 クラス間の関係

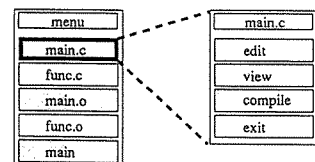


図4 実行時におけるメニューの例