ソフトウェア工学 96－3
（1994．1．20）

PCTE スキーマから導出されるデータ操作プログラムについて

満田 成紀　沢田 篤史
鯵坂 恒夫　松本 吉弘
京都大学工学部

PCTE では，SDS (Schema Definition Set) と呼ばれる拡張 ERA モデルを利用し，ツールが用いるデータモデルを定義することができる．ある特定のツール群のために設計された SDS には，それらが持つ固有の論理が大きく反映される．

本稿では，特定の SDS に基づく典型的なデータ操作のパターンを類形化することによって，それに対応するプログラムを自動的に生成するシステムを提案する．本システムは，記述された SDS から C++ のプログラムを導出する．導出されたプログラムでは SDS に定義されたオブジェクト型に対応する C++ のクラスが提供され，オブジェクト型に適用されている属性型，リンク型に関するデータ操作関数がそのクラスのメソッドとして実現される．本稿では，この導出の手順を簡単な例題を用いて説明する．さらに今後拡張される PCTE と本システムとの対応についての考察も行う．

Data Access Programs Derived from PCTE Schemas

MITSUDA Naruki　SAWADA Atsushi
AJISAKA Tsuneo　MATSUMOTO Yoshihiro
Faculty of Engineering, Kyoto University
Yoshida honmachi, Sakyo-ku, Kyoto 606-01, JAPAN

PCTE uses extended ERA models called SDSs (Schema Definition Sets) to define data models used by tools. An SDS very much reflects logic of tools when it is designed specially for them.

In this paper, we propose the system which automatically generates the typical data access programs from SDSs. The C++ classes are derived from the object types, and the C++ methods are derived from the link types or the attribute types which are applied to the object types. A small example shows how this system generates the data access programs. We also discuss the extension of the system which enables the system to work with the future PCTE.

1 はじめに

統合型 CASE 環境を構成するために，さまざまな CASE ツールを統合する手だてが模索されている．その一解として，PCTE (Portable Common Tool Environment) が開発され，現在は ISO による標準化の途上にある．PCTE とは，意味モデルを用いたオブジェクトベースを中核とした開放型リポジトリのインタフェース標準であり，ツールの可搬性を支援するものである．

PCTE 上で動作するツールの開発は，ツールの扱うデータモデルを表現する SDS (Schema Definision Set) の設計と，そのデータモデルを用いるプログラムの開発とに分かれる．最終的に何らかのプログラミング言語を用いて実現されたツールのプログラム中で，リポジトリに対するデータ操作プログラムは重要な部分を占める．また，これらのデータ操作プログラムは，設計された SDS に適合した形で作成されなければならないため，ツール作成者には SDS と一貫したデータ操作プログラムの開発作業が課せられることになる．

本稿では，SDS からデータ構造およびデータ操作機能を含んだ C++ プログラム部品を自動的に導出するシステムを提案する．このシステムを用いることで，データ操作プログラムをツールの用いる SDS と一貫したものとするために必要な設計者の負担を軽減することが可能となる．以下，第 2 節で PCTE と SDS について簡単に説明した後，第 3 節では SDS から C++ プログラムを自動生成するシステムについて解説する．さらに第 4 節では，近年試みられている PCTE スキーマ自体の拡張と本システムとの対応について考察し，第 5 節で本稿のまとめをする．

2 PCTE, SDS, ツール

PCTE とは，開放型リポジトリのための共通ツールインタフェースの標準である．PCTE

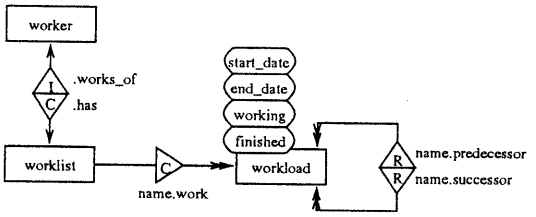

図 1: SDS によるスキーマ記述例

が定めるリポジトリサービスは，CASE ツールの可搬性の保証と高度なデータ統合を実現することを主な目的として設計されている．

PCTE では SDS と呼ばれる拡張 ERA モデルによってリポジトリのスキーマを記述し，これらを異種ツール間で共有することが可能である．各ツールは SDS の順序集合である**作業スキーマ**を通じて，対応するデータの操作を行うよう設計される．

2.1 SDS によるスキーマ記述

SDS によるスキーマ記述例を図 1 に示す[1]．

SDS とは，ツールによって操作される各種データに関する型とそれらの間の静的な関係を定義するものである．SDS の要素となるデータ型には，**オブジェクト型**，**リンク型**，**関係型**，**属性型**がある．

長方形で示されているオブジェクト型はツールによる種々の操作の対象となるエンティティを表すデータ型であり，内部の文字列は型名を示す．新たな型は既存の型のサブタイプとして定義される．この継承木の頂点として 'object' と呼ばれるオブジェクト型が与えられているほか，'file'，'dir' など，システムに既存の型が与えられており，これらを SDS に**移入** (import) して用いることもできる．なお，継承関

[1]本稿が対象としている PCTE は PCTE 1.5 [3, 5] であり，ECMA-PCTE [1] とは SDS 記述においてもリンク型や属性型の値などに若干の相違がある．

係は図1には表されていない.

三角形で示されているリンク型は，オブジェクト間に存在し得る有向関係を示すデータの型である．内部の文字はリンクの**カテゴリ**であり，C: Composition は構成複合関係を，R: Reference は参照関係を，I: Implicit は暗示的関係を示す．傍らに示される '.' で区切られた文字列のうち，最右の文字列がリンク型名を表し，左側のものはキー属性型名を表す．キー属性型を指定することで，リンク型は一対多の関係を表すことができる(カージナリティ多: 二重の矢印)．指定しない場合は 一対一の関係を表す(カージナリティ一: 一重の矢印)．例えば，図中 'name.work' で示されるリンク型は，一つの 'worklist' 型オブジェクトの構成要素として複数の 'workload' 型オブジェクトを生成することができ，それらは 'name' 型のキー属性により一意に識別されることを示している．

関係型はオブジェクト型間に定義され，互いに逆方向を持つ二つのリンク型の対で表されるものである．図中では，'.has' + '.works_of' などが定義されている．

長円形で示されている属性型は，オブジェクトあるいはリンクに適用され，文字列，整数，真偽値，日付の値を持つ属性データの型を示す．前述のキー属性型は属性型の特別な場合である．内部の文字列は属性型名を示す．

2.2 ツールと作業スキーマ

PCTE の上で実行されるツールは，作業スキーマと呼ばれる SDS の順序集合を持ち，それを通じてリポジトリ上のデータを操作することができる．言い換えれば，作業スキーマがツールの用いるデータモデルの全体像を規定することになる．

したがって，PCTE 上に構築されるほとんどのツールの内部では，作業スキーマに定義されたデータ型のインスタンスに関する生成，削除，変更，リンクのトラバース，キー検索などのデータ操作が行われていると言える．

一方，PCTE が提供するリポジトリ上のデータ操作関数群は**プリミティブ関数**と呼ばれ，これらはいわば UNIX のシステムコールに対応する汎用の機能を提供するものである．プリミティブ関数は個々の SDS や作業スキーマには全く依存しないため，ツール固有の論理に即した，より専用のデータ操作は，通常これらを組み合わせることによって実現されている．

3 SDS から導出されるデータ操作プログラム

ツールによるリポジトリ上のデータ操作は SDS に適合した形で行われなければならない．通常，データ操作はツール開発者によってプログラム中に表現されるものであるが，特定の SDS に従ってリポジトリに保存されているデータに対して，実際に行うことのできる操作を，その SDS を手がかりに導くことは可能である．

本節では，SDS から C++ のデータ操作プログラムを自動的に導出するシステムの手続きについて説明し，さらに，自動導出システムによって導出されたプログラムを用いることで得られる利点についての考察も行う．

3.1 システムの概要と導出手順

本システムは，ツールが用いる作業スキーマに含まれる複数の SDS を入力とし，それらに従ったデータ操作プログラムを C++ のソースコードとしてツール作成者に提供する．

提供される C++ プログラムにおけるクラスは PCTE のオブジェクト型に対応し，クラスの持つメンバ関数は SDS を構成するオブジェクト型，リンク型，属性型をもとに導出される．これらのメンバ関数は入力された SDS に特化されたものとなり，実際には引数を固定したプリミティブ関数の呼び出しとなる．

SDS に定義されたデータ型と C++ のプログラムとの関係を図2に示す．以下，SDS に定義されたデータ型から，それぞれどのような C++

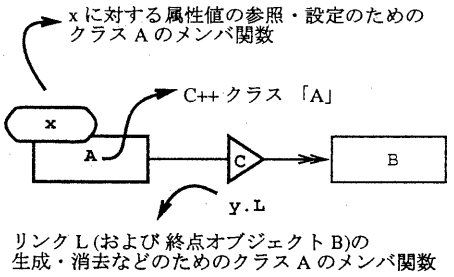

図 2: データ型と C++ プログラム

プログラムが導出されるかを説明する．例として図1の SDS を用い，データ型に対する C++ クラス名あるいはメンバ関数名を示す．

オブジェクト型

オブジェクト型はデータ操作の対象となるものであり，C++ のクラスが導出される．(図 2 にはオブジェクト型 A からクラス A が導出されることが示されている．) これによって SDS によって設計されたデータモデルが，そのまま C++ のプログラム中に表現されることになる．オブジェクト型間の継承関係は C++ のクラス継承によって表現される．クラス名はオブジェクト型名から決定される．

例) オブジェクト型 `workload` から，C++ のクラス `workload` が導出される．

リンク型

リンク型からは，その始点オブジェクト型に対応するクラスのメンバ関数が導出される (図 2のリンク型 L)．メンバ関数は，リンクの始点オブジェクトから終点オブジェクトに対して行われるデータ操作 (生成，削除，参照) に相当するものとなる．メンバ関数名はリンク型名から決定される．

表1に示すように，リンク型のカテゴリやカージナリティによって導出されるメンバ関数が異なる．

例) リンク型 `predecessor` から，クラス `workload` のメンバ関数
`predecessor(char *name)`(個別参照用) が導出される．

属性型

属性型からは，その属性の値を設定，参照するメンバ関数が導出される (図 2の属性型 x)．オブジェクト型に適用された属性型 (オブジェクト属性型) からは，対応するクラスのメンバ関数が導出され，リンク型に適用された属性型 (リンク属性型) からは，リンクの始点となるオブジェクトの型に対するクラスのメンバ関数が導出される．リンク属性型の場合は個々のリンクを識別するために，メンバ関数の引数として，リンクのキー属性，あるいは終点オブジェクトの識別子を指定する必要がある．また，リンクのキー属性値を参照することはできるが，再設定はできない．メンバ関数名は属性型名から決定される．

例) オブジェクト属性型 `start_date` から，クラス `workload` のメンバ関数
`get_start_date()`(参照用) が導出される．

複数のスキーマを用いるツールへの対応

PCTE ツールは作業スキーマ内の複数の SDS を同時に利用する事ができる．このようなツールに有用なデータ操作プログラムを導出する際には，各 SDS で定義されたデータ型の同一性が問題となる．すなわち，図 3に示すように，データ型が一方から一方へ移入され，同一のオブジェクト型あるいはリンク型が複数の SDS によって定義されている場合，それらの定義に対応して導出されるプログラムは一つの C++ クラスとしてまとめられた形で提供されなければならない．

また SDS 間でデータ型の移入が行われる際に，SDS 毎に別名を付けることがある．これ

表 1: リンク型から導出されるメンバ関数

カテゴリ	導出されるメンバ関数	カージナリティ	メンバ関数に必要な引き数
C(複合)	リンクと同時に終点オブジェクトを生成	一対一	なし
		一対多	リンクのキー属性
	リンクと同時に終点オブジェクトを削除	一対一	なし
		一対多	なし (全終点オブジェクトを一括削除する場合) リンクのキー属性 or 終点オブジェクトの識別子 (個々に削除する場合)
	終点オブジェクトの参照	一対一	なし
		一対多	なし (全終点オブジェクトを一括参照する場合) リンクのキー属性 (個々に参照する場合)
R(参照)	終点オブジェクトとの間にリンクを生成	一対一 一対多	終点オブジェクトの識別子
	終点オブジェクトとの間のリンクを削除	一対一	なし
		一対多	なし (全リンクを一括削除する場合) リンクのキー属性 or 終点オブジェクトの識別子 (個々に削除する場合)
	終点オブジェクトの参照	一対一	なし
		一対多	なし (全終点オブジェクトを一括参照する場合) リンクのキー属性 (個々に参照する場合)
I(暗示)	終点オブジェクトの参照	一対一	なし
		一対多	なし (全終点オブジェクトを一括参照する場合) リンクのキー属性 (個々に参照する場合)

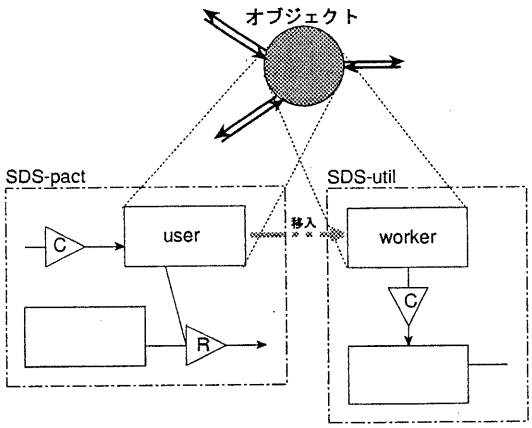

図 3: 複数の SDS による型付け

は，導出されるクラスやメンバ関数に対して別名を用意することで対処できる．

一つの SDS 中ではデータ型の名前は唯一であるが，複数の SDS を用いる場合には，異なるデータ型の名前が衝突することを考慮しなければならない．PCTE ではデータ型名の前に SDS 名が付加された，完全形 (long form) をデータ型名の識別に用いている．SDS 名を伴わない短形 (short form) によるデータ型名が衝突する場合には，作業スキーマ内での SDS の順序によってこれに対処している．本システムでも同様に，導出されるクラス名やメンバ関数名の頭に SDS 名を付け，その別名として短形を用意してこの問題に対処している．

```
// objects
class util_worklist : public util_object {
public:
  // for composition links
  int create_work(char *name);
  int delete_work();
  //....
  // for implicit links
  util_worker& works_of();

  // for link attributes
  char* get_name_of_work(util_worker& workload);
};

// short forms
typedef util_worklist worklist;

// membership functions
util_worker&
util_worklist::works_of(){
  struct linkstat list[1];

  chrefino(ivol, ino, ".");
  lslinks(".", "works_of", NULL, 1, list);
  return util_worker::find(list[0].l_ivol,
   list[0].l_ino);
}
```

図 4: C++ コードの導出例 (一部)

3.2 プログラムの導出例

導出例として図 1の SDS から導出される C++ のソースコードの一部を図 4に示す[2]. ここでの SDS 名は "util" としている.

3.3 導出されたプログラムの利用

SDS から導出される C++ プログラムには, ツール固有のクラス階層とリポジトリ操作を行うメンバ関数が含まれている. ツール開発者は導出されたクラスやメンバ関数を使用することにより, SDS に適応したデータ操作を行うことができる.

著者らが作成したツール (ボウリングスコア集計ツール) の場合, 約 1200 行のソースコードの内, 約 500 行が集計方法などのアルゴリズムを表現するためのものであった. その他の部分はリポジトリ上のデータ操作に対応し, 本システムを用いれば自動的に導出されることになる.

また, プログラミングの途上で SDS に問題が生じた場合には, SDS の修正を経て再びデータ操作プログラムが導出される. データ操作プログラムが再導出された場合でも, SDS の変更に影響を受けない部分については以前に記述したアルゴリズムの再利用が可能である. このため, データ操作からは独立したアルゴリズムを表現するプログラムは導出コードとは別のファイルで管理されることが望ましい.

3.4 データ操作機能の拡張

リポジトリ上のデータ操作は, PCTE が提供するプリミティブ関数を使用することで実現される. ここで, プリミティブ関数の提供する機能を一つの汎用クラス (クラス PCTEobject) に集中することが考えられる. 導出される各ツール固有クラスは, 図 5のように汎用クラスを継承することでプリミティブな機能を利用することが可能となる. この形のクラス設計は C++ バインディングでも採用されている [2].

こうすることで, プリミティブな機能の拡張を容易に行うことができる. つまり, プリミティブな機能の拡張は, 汎用クラス (PCTEobject) のみの修正によって実現され, SDS から導出されるコードは何の変更も必要としないのである. プリミティブな機能の拡張は, 次節に述べるような SDS の拡張から要求される場合や, 機能の実現方法に独自の戦略 (バッファリング等) を導入する場合などが考えられる.

[2]PCTE の C++ バインディングが草案の状態であるため, プリミティブ関数として C バインディング [3] で規定されているものを用いている.

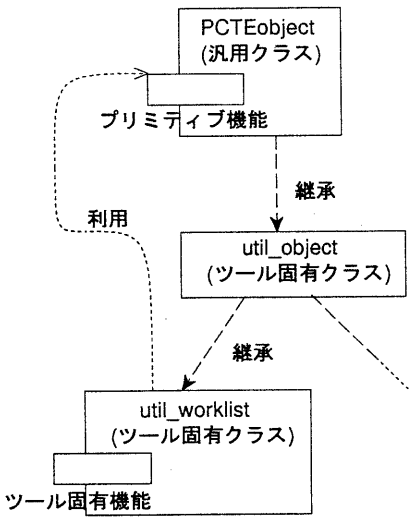

図 5: プリミティブ機能の集中

4 PCTE の拡張との対応

現在，PCTE の拡張に関する研究開発が様々な形で行われている．その中には，SDS 自身を定義する metasds に新たな要素を導入することにより，スキーマの概念自体を拡張するものも含まれている．

本節では，metasds を拡張することで PCTE の機能拡張を図っているもののうち，オブジェクト指向 PCTE (OOPCTE) [6] と制約管理ユーティリティ [7] に関して，それぞれ本システムとの関連を概観する．

4.1 OOPCTE

OOPCTE では，PCTE のオブジェクトにメソッドを定義し，それらが PCTE 上で動作するツールから呼び出されることを可能とするために，metasds が拡張されている [6]．オペレータ型と呼ばれるデータ型が新たに導入され，オブジェクト型と関連付けられている．オペレータ型のインスタンスは，関連付けられたオブジェクトが持つメソッドに相当し，前提条件，帰結条件，メソッド記述，引数を記述することができる．前提条件，帰結条件，およびメソッド記述には PCTE ツール (実行ファイル) を関連付けることが可能で，実際のメソッド呼び出しではオペレータに関連付けられているツールが起動されることになる．

OOPCTE では，オブジェクトに付随するメソッドが SDS 設計者によって明示的に定義されなければならない．本システムを用いることで，これらのメソッドのうちデータ操作に関する部分の自動的な導出が可能となる．この際，OOPCTE ではメソッド呼び出しがツール呼び出しによって実現されているのに対し，本システムではデータアクセスに対応するクラスのメンバ関数のみが導出されるという点が問題となる．これに対しては，導出されたメンバ関数を同様の機能を提供するツールの形にコンパイルした後，メソッドと関連付けることで解決が可能である．

ただし，本システムが導出するメンバ関数はデータ操作の度に頻繁に呼び出されるものであるため，前述の解決法は効率上必ずしも望ましいものではない．特に細粒度オブジェクトに付加されるメソッドなどの場合には，本システムにより導出されたメンバ関数をメソッド呼び出しの際に起動できるような拡張が OOPCTE の側にも必要であると考えられる．

4.2 制約管理システム

PCTE リポジトリ上に存在するデータインスタンス間の意味的な制約を記述し，記述された制約の管理を行うユーティリティが提案されている [7]．このユーティリティは，一つの SDS について，そこで定義されているデータ型のインスタンスの間に成立すべき関係を制約として記述し，その充足が保証された形でリポジトリのデータを操作する手段をユーザに提供している．

ここでは記述された制約をデータ型に付随さ

せることができるよう，metasds が拡張されている．記述された制約に対して，リポジトリのデータ操作を行うための関数群がライブラリとして自動生成される．これらの関数はそれぞれプリミティブ関数と対応したもので，記述された制約が満たされる範囲においては同等の機能を持つが，制約が充足されない場合には要求されたデータの変更を行わない．

制約管理ユーティリティによって生成されたライブラリ関数を object 型に付加されるプリミティブなデータ操作関数と位置付け，これらを利用するデータ操作関数も同時に導出することで，本システムでも記述された制約の充足が保証されたデータ操作関数を提供することが可能となると考えられる．

5 おわりに

本稿では，PCTE ツールが持つデータモデルを規定する SDS から，データ操作を行う C++ プログラム部品を自動的に導出するシステムについてその概要を紹介した．

ツールの扱うデータモデルを定めるということは，ツールによって為されるデータ操作の種類が規定されることに他ならない．本システムによって導出されたデータ操作プログラムを用いることで，SDS と一貫したデータ操作のプログラミングにかかる負担が軽減される．すなわち，ツール開発者は SDS 設計の後，本システムによって導出された C++ プログラムにデータ操作以外のアルゴリズムを実現するプログラムを加えることで，ツールを完成させることができる．

本システムは，PCTE をソフトウェアパッケージとした代表的な製品である Emeraude V12.4 [8] の上に現在開発中である．Emeraude V12.4 では C++ による開発環境は提供されていないため，導出されたプログラムを PCTE 上で実働化するためには，一旦 UNIX 上でコンパイルし，PCTE 上へ移動させる等の手続きが必要となる．

今後の発展研究としては，ECMA-PCTE への対応，近く定められるであろう C++ バインディングへの対応，現在考案されている SDS の拡張への対応が挙げられる．

参考文献

[1] European Computer Manufacturers Association: *Portable Common Tool Environment (PCTE) Abstract Specification*, Standard ECMA-149, (1990).

[2] European Computer Manufacturers Association: *Portable Common Tool Environment (PCTE) C++ Programming Language Binding*, Draft version 2, (1993).

[3] Commissions of the European Communities: *PCTE: A Basis for a Portable Common Tool Environment*, Functional Specification, Version 1.5, Vol. 1 / C, (1989).

[4] Wakeman, L., Jowett, J.: *PCTE The Standard for Open Repositories*, Prentice Hall, (1993).

[5] 鰺坂 恒夫, 沢田 篤史, 満田 成紀: “ソフトウェア評論: Emeraude PCTE,” コンピュータソフトウェア, Vol. 10, No. 2, (1993), pp. 65–77.

[6] Charoy, F.: “An Object-Oriented layer on PCTE,” *Proc. of the PCTE '93 Conference*, Paris, (1993), pp. 379–389.

[7] Sawada, A., Mitsuda, N., Ajisaka, T., Matsumoto, Y.: “Utilities for Avoiding Constraints Violation in Creating and Updating the Data in PCTE,” *Proc. of the PCTE '93 Conference*, Paris, (1993), pp. 93–113.

[8] GIE Emeraude: The Emeraude Environment, (1992).