ソフトウェア工学 125−12
（2000. 1. 14）

PCTE を用いた UNIX コマンドデータベースの作成

田中 聡　　権藤 克彦

北陸先端科学技術大学院大学 情報科学研究科

真のソフトウェアデータベースの実現への第一歩として、本研究では対象を UNIX ファイルシステムに絞り、PCTE を用いてコマンドやファイル間の制約・関係の記述を試みる。制約には意図に関する制約や動的情報に関する制約など、計算機上での記述が困難なものが多数存在する。そういった制約のいくつかの例題に対して計算機の支援方法を提案し、完全に実現できない制約をどこまで記述が出来るかを、実際にその実現を試みたうえで検討する。UNIX コマンドにおける制約の実現をいくつか試みることにより、制約は同じような手段で実現できるものに分類出来ることが確認できた。

Implementing a UNIX command database using PCTE

Satoshi Tanaka　　Katsuhiko Gondow

School of Information Science
Japan Advanced Institute of Science and Technology

In this study, we try to implement a UNIX command database using Emeraude PCTE, which gives us information of relations and restrictions. It is a first step to realize useful software databases. But we found it difficult to realize, restrictions including dynamic information or software semantics. For several examples, we propose partial solutions to describe such restrictions. We consider how to cope with restrictions that is not able to realize completely, and we try to realize restrictions. We classify their restrictions based on the experience.

1　はじめに

必要なソフトウェアオブジェクトの制約・関係を正確 かつ形式的に記述したソフトウェアデータベースは現在のところ存在しない。本研究のゴールは ソフトウェアデータベースの構築方法の確立である。

本研究は、材料として UNIX コマンド、道具として PCTE を用い、ソフトウェアデータベースの構築実験を行なう。ソフトウェアデータベースには、形式的でないオブジェクトの処理が不可欠だが、本研究はその問題には触れず、ソフトウェアオブジェクト間の制約・関係や動的情報 (実行時まで決定されない情報) を重視する。具体的には以下の手順で、UNIX コマンドデータベースの作成を行なう。

- 対象となる UNIX コマンドを選択する。

- UNIX コマンドから関係を抽出し、実体・関連・属性で記述したERA図で記述する。
- UNIX コマンドから制約を抽出し、それをプログラムとして実現する。

本研究は以上の手順を踏むことにより、ソフトウェアデータベースの構築方法に有効な手がかりとなる諸性質を抽出することを目標とする。

本論文の流れは以下の通りである。まず、第2章で UNIX コマンドデータベース が必要となるいくつかの例題を挙げ、その実現可能性を検討し、第3章では、本研究で用いる PCTE の概要を紹介し、PCTE がオブジェクトの関連を参照しやすいということを述べる。第4章では、UNIX コマンドがソフトウェアデータベースの持つ諸問題を含んでいるため、実験材料として適切であることを述べた後で、具体的な UNIX コマンド、cat, man, tar に対して、ERA 図の作成を行ない、また制約の抽出を行なう。第5章では、2つの制約について具体的な、制約のプログラミングの方針を述べる。

ソフトウェアオブジェクト間の制約・関係には記述が難しく、その完全な実現は不可能なものが多数存在する。本研究では そういう制約に対して、いくつかの支援方法を示し、それを実際に実現していく過程で、ソフトウェアの制約の記述方法は分類出来ることが確認出来た。

現在、制約の分類は 大きく分けて、位置情報に関する制約、意図に関する制約、動的情報に関する制約の3つが確認出来ている。

2 例題

ここで、問題を具体化するために、いくつかの例題とその実現可能性を検討する。

例題1 コマンド tar のオプション x と 同時に使えないオプションは何か?

オプション c, r, t, u は x と同時には使えない。

この例題の実現は容易である。解答はインストール前に静的に一意に決まり、かつ形式的に表現できるからである。

例題2 環境変数 MANPATH にセットするべきパス名の候補は何か?

この例題の実現はやや難しい。解答はインストール後に一意に決まり、かつ形式的に表現できる。以下は部分的な実現案であるが、いずれも問題がある。

- ほとんどの環境で共通である、/usr/man/や/usr/local/man などが組み込まれているかチェックする。
- .cshrc などで、他のユーザの MANPATH の設定を参照する。
- find コマンドで、マニュアルファイルのある場所を探す。
- 管理者が新しいマニュアルを設置したとき、特定の場所にその位置情報を記述する。

例題3 環境変数 PAGER にセットするべきコマンドは何か?

あるコマンドが PAGER にセッしてよいコマンドであるかどうか判断することは困難である。なぜなら、PAGER の「長いテキストをユーザにとって読みやすく少しずつ表示する」という意図はプログラミングが難しいためである。

以下に実現案を示すが、いずれも問題がある。

- 特定の場所に意図に沿ったコマンドを列挙して記述しておく。
- ソフトウェアデータベースに記述されている関係を参照し、判断する。例えば、PAGER の場合は標準入力から読み取った データを標準出力に出力しているかを判断する。
- コマンドに プログラムの意図を示す属性 type を与える。例えば type には pager, manual, compiler, printer, browser などがあるとする。

例題4 nemacs 用の .emacs を mule 用の .emacs に変更したい。

これは非常に難しい。この問題は本質的に移植の自動化が必要だからである。変更された関数のリストがあれば、変更が必要な箇所を示すことはできる。

3 PCTE について

3.1 PCTE とは?

PCTE [3] [4] [5] とは Portable Common Tool Environment の略で、さまざまなソフトウェア・ツールを統合的に利用する環境の一つである。

PCTE の仕様は ERA モデル (実体・関連・属性を記述したモデル) にリンク属性など、特定の性質を付加した SDS (Schema Definition Set) により、記述し、また 意味的制約もサポートしている。

本研究では PCTE を実現するソフトウェアの1つ、Emeraude PCTE を用いて UNIX コマンドデータベースの作成を試みる。Emeraude PCTE は esh と呼ばれる Emeraude PCTE 特有のシェルにログインすることによって、オブジェクトの参照、生成を行なう。C 言語やシェルスクリプトを用いて制約をプログラミングでき、プログラミングしたものを ツール と呼ぶ。

3.2 なぜ PCTE か?

PCTE は ERA モデルを拡張した SDS を用いて記述するため、関係を用いてオブジェクトを参照するのに優れている。また、SDS は多重度や存在関係等の制約の記述をサポートしており、サポートしていない制約は C 言語を用いてプログラミングすることにより、ツールとして実現できる。本研究は 制約・関係の記述を重視しているため、以上の利点から PCTE は使用するソフトウェアとして適当なものであると判断した。

3.3 SDS 図

図1は 学校内の関係を示した SDS 図 である。

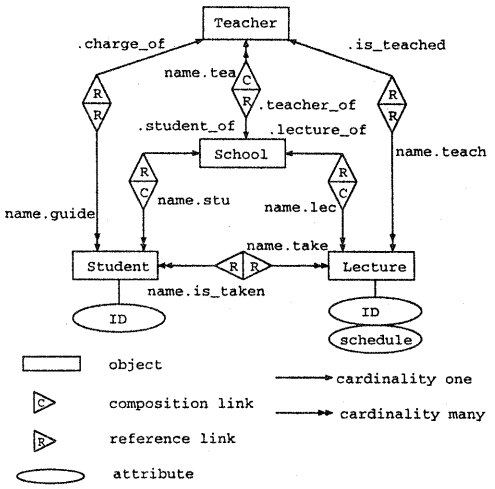

図 1: 学校内の関係を示した SDS 図

SDS 図では、Link には category があり、図中では Composition Link と Reference Link が見られる。Composition Link は対象のオブジェクトを生成できることを意味し、図 1 の例の場合、新しい Student や Teacher や Lecture のオブジェクトを作ろうとした場合、School から Composition Link を用いることによって生成される。また、Reference Link は対象のオブジェクトを参照出来ることを意味している。

以下、オブジェクトとその関連を示す図は、繁雑さを防ぐ意味で SDS 図ではなく、ERA 図を用いる。前で述べたように、SDS 図は ERA 図に特定の性質を付加したものなので、ERA 図で表現できるものは、SDS 図でも表現できる。

4 UNIX コマンドデータベース

4.1 なぜ UNIX コマンドか?

UNIX コマンドは身近であるため、その理解しやすさ故に実験材料として扱いやすい。また、UNIX のファイルシステムはさまざまなオブジェクト間で制約・関係が成り立っており、また引数や環境変数、ファイルデータなど、実行毎に異なる情報も多いが、一般のソフトウェアに比べて比較的簡潔な形になっている。これは UNIX のファイルシステムが制約・関係や動的情報といった、ソフトウェアデータベースが保持している問題の一部を含みつつ、簡単な形になっていることを意味している。また、UNIX コマンドはソースが公開されているので、それを用いてソフトウェアの持つ意味を読みとることが出来る。

これらの理由により、UNIX コマンドはソフトウェアデータベース構築の実験材料の一つとして適切なものであると判断した。

4.2 階層構造

本研究では UNIX コマンドデータベースを作成する際、以下の 3 つの階層に分かれるようにした。

- メタ・クラス図 ··· UNIX コマンド全ての実体型、関連型の定義を示した図
- クラス図 ··· 特定の UNIX コマンドの 実体間の関係を示した図
- インスタンス図 ··· 具体的な実行例に対し、具体的なオブジェクト間の関係を示した図

これを階層構造 [1] と呼ぶ。メタクラス図を導入したことにより、複数のコマンドのクラス図の記述が統一的となり、比較が行ないやすくなり、インスタンスを導入することにより、具体的なオブジェクトを参照し、意図しない実行結果の究明に役立てることが出来る。

4.2.1 メタ・クラス図

UNIX コマンドデータベースの 全ての実体型と関連型を定義し、1 つだけ、作成する。Emeraude PCTE では SDS 図を作成し、メタ・クラス図を表記する。

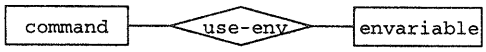

図 2: メタ・クラス図の例

4.2.2 クラス図

UNIX コマンドの静的な関連を記述するもので、UNIX コマンド毎に作成する。メタ・クラス図で定義された 実体型と関連型を用いて作成する。Emeraude PCTE 上では作成した メタ・クラス図を ワーキング スキーマに与え、esh 上でオブジェクトや リンクを作成することによって実現する。

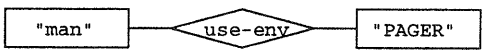

図 3: クラス図の例

4.2.3 インスタンス図

インスタンス図は 具体的な実行例を与えたときに記述することが出来、クラス図で定義した、実体間の関係を用いて作成する。

しかし、全ての場合のインスタンスをソフトウェアデータベースにあらかじめ記述することは出来ないため、それに対応するには、クラス図からインスタンス図の作成の自動化が必要である。しかし、インスタンス図を自動作成するツールを作成するには、コマンドのセマンティクスを詳細に知り、かつそれをプログラムとして表現する必要があるため、現実的ではない。しかし、多数の UNIX コマンドで共通の動作は、インスタンス図の一部を生成するツールとして、ある程度は実現可能であり、かつ有効と我々は考える。

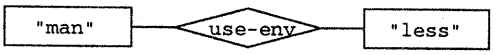

図 4: インスタンス図の例

4.3 メタクラス図の作成

UNIX コマンドデータベースの作成にあたり、まず最初に行なうことはメタクラス図の作成である。メタクラス図の作成は実体型、関連型、属性型 の定義を意味する。

まず、実体型の定義を行なう。

実体型	属性	説明
command	name, type	UNIX コマンド
parameter	name	引数
envariable	name, default	環境変数
file	name	ファイル
data	name, type	内部データ
com	name, type	コマンドが使用するコマンド

図 5: 実体型の定義

次に関連型の定義すると、UNIX コマンドデータベースのメタクラス図が作成できる。メタクラス図を 図 6 に示す。

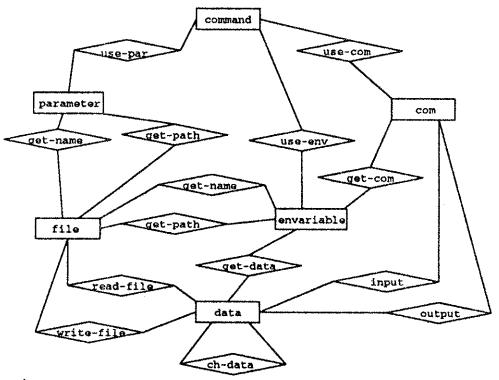

図 6: メタ・クラス図

このメタクラス図は ERA 図で、長方形が 実体型、菱形が関連型、楕円形が属性を表している。

以上の実体型、関連型はこれから紹介する UNIX コマンド、cat, man, tar を元に定義したもので、UNIX コマンドデータベースの作成を続けていく上で、他の UNIX コマンドではこのメタクラス図は不十分である可能性がある。例えば、lpr での プリンタや、rlogin での ホスト など、ネットワーク上の実体は未定義である。しかし、今後さまざまなコマンドのデータベースを作っていく際に、必要に応じてメタクラス図の修正を行なっていくことで、完全なメタクラス図に近付けていくことが出来る。

4.4 UNIX コマンドのクラス図の例

4.4.1 cat

cat コマンドの ERA 図を 図 7 に示す。

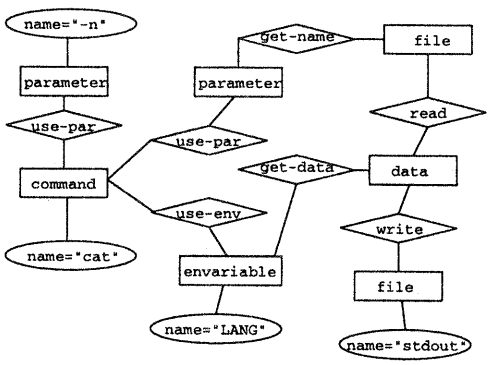

図 7: cat の ERA 図の一部

しかし、図 7 では以下の制約は表現できていない。

- ファイルのデータはテキストファイルでなければならない。
- 環境変数 LANG にはターミナルがサポートしている言語のコードをセットする。

4.4.2 man

man のコマンドの ERA 図を 図 8 に示す。

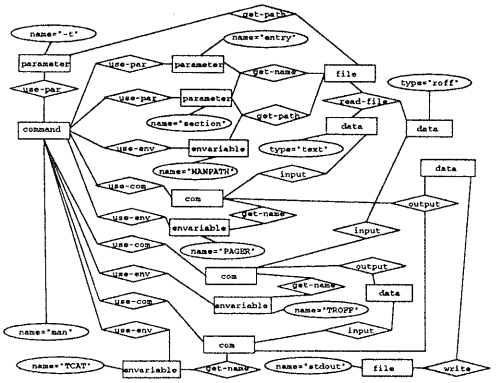

図 8: man の ERA 図の一部

しかし、図 8 では以下の制約は表現できていない。

- 環境変数 MANPATH は マニュアルのファイルが格納されているディレクトリのパス名をセットする。
- 環境変数 PAGER には テキストファイルの中身を表示するコマンド名をセットする。
- 環境変数 TROFF は roff 形式のデータを処理する UNIX コマンド名をセットする。
- 環境変数 TCAT には roff 形式のデータを環境変数 TROFF の値で示されているコマンドで処理されたデータを適切な形でユーザに表示するコマンドをセットする。(TROFF が nroff なら less, more など、TROFF が psroff なら lpr など)
- 引数 entry には MANPATH の中に存在するマニュアルの名前が入る。

4.4.3 tar

tar のコマンドの ERA 図を 図 9 に示す。

しかし、図 9 では以下の制約は表現できていない。

- オプション x と c は同時につけられない。
- オプション z がつけば、type="taz" のデータを扱い、つかなければ type="tar" 型のデータを扱う。

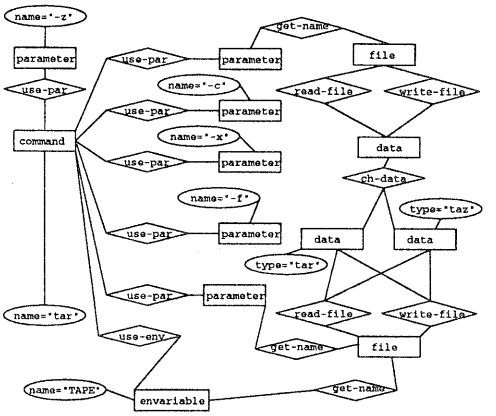

図 9: tar の ERA 図の一部

- オプション x がつけば、type="tar" もしくは type="taz " のデータを入力し、オプション c がつけば 出力する。
- オプション f がついた時だけ type="tar" もしくは type="taz" のデータを示すファイル名を引数にとり、f がつかない時は 環境変数 TAPE が呼び出される。

5 制約の実現

Emeraude PCTE を用いる際、SDS でサポートされていない制約の実現はプログラミングにより行なう。

4 節で、cat, man, tar の制約をいくつか紹介したが、制約の記述を追求していく意味で、これらの制約の実現を行なう必要がある。しかし、これらには完全な実現が不可能なものが多数含まれている。本研究の目的であるソフトウェアデータベースの確立のためには、そのような制約の実現を実際に試みたうえで、どの程度記述できるかを検討し、ソフトウェアデータベースにおける計算機の支援のあり方を考察する必要

種類	例	(部分的な) 解決方法
位置情報に関する制約	環境変数 MANPATH の設定 環境変数 TEXINPUTS の設定 環境変数 PATH の設定 gcc コマンドの オプション -L や -I	管理者が位置情報を記述 find により検索 他のユーザの .cshrc を検索
プログラムの意図に関する制約	環境変数 PAGER の設定 環境変数 TROFF の設定 環境変数 TCAT の設定	適切なもののリストを作成する コマンドの関係の記述を参照 意図に関する型を導入する
動的情報に関する制約	環境変数 TCAT の設定 tar のオプション z や Z	インスタンス図の作成

図 10: 制約の分類例

がある。

以下 UNIX コマンドにおける制約の 2 つの例を挙げ、具体的な実現の方針を示した上で、実際にツールの作成を試みる。

なお、以下の制約実現の例は 必要な参照情報は全て記述済みであると仮定している。

5.1 tar コマンドのオプションに関する制約

tar の排他的なオプションを判断するツールを実現した。このツールは次の手順で判断を行なう。

1. 実行例からオプションを抽出する。
2. オプションが存在するか、データベース上を検索する。
3. ツール内部のオプション一覧を検索し、さらに同時に指定できないオプションが存在していないか、判断する。

実際に実現したツールの実行結果を以下に示す。

```
esh$
esh$ op_chk.tool tar xvf foo.tar
option x is supported.
option v is supported.
option f is supported.
esh$
esh$ op_chk.tool tar xvfQ foo.tar
option x is supported.
option v is supported.
option f is supported.
option Q is not supported.
esh$
esh$ op_chk.tool tar xvfc foo.tar
option x is supported.
option v is supported.
option f is supported.
option c is supported.
but c and x is not supported at the same time.
esh$
```

5.2 man コマンドの環境変数 PAGER に関する制約

PAGER に適切である UNIX コマンドを答えるツールを部分的に実現した。次のいずれかであれば、適切なコマンドだと判断する。

- PAGER にセットする値として適切なコマンドのリストの記述を参照する。
- command 型の属性 type を参照し、pager であるか判断する。
- コマンド内の関係を参照し、標準入力から得られたデータを標準出力に出力しているかを判断する。

しかし、この PAGER に関する制約の実現は十分でない。例えばエディタは不適切だと判断されてしまう。以下に引数として入力されたコマンド

が pager 型である判断し、結果を返すツールの実行結果を示す。

```
esh$
esh$ pager_check1.tool less
less is guessed as right value for PAGER.
esh$
esh$ pager_check1.tool cat
cat is guessed as right value for PAGER.
esh$
esh$ pager_check1.tool man
man is guessed as wrong value for PAGER.
PAGER expects the type "pager".
esh$
esh$ pager_check1.tool hello
command hello is not supported.
esh$
```

6 結論

UNIX コマンドには 計算機上で 制約の記述が難しいものが存在する。本研究では、いくつかのコマンドを調べた結果、図 10 の分類を行なった。

- 位置情報に関する制約 ··· MANPATH 等のパス名に関する制約。サイトの変化等により、適切なパス名が変化するという、難しさがある。しかし、インストール後に一意に決まり、かつ形式的に表現できる。
- プログラムの意図に関する制約 ··· PAGER 等の、要求するコマンドに関する制約。要求の意図はプログラミングが難しいという難しさがある。
- 動的情報に関する制約 ··· ファイルデータや 環境変数などに対する 実行されるまで分からないものに対する制約。具体的なオブジェクトに関する制約なので、インスタンス図を参照する必要があるという、難しさがある。インスタンス図に対する問題点は 4.2.3 節 で説明した。

以上のように、実現の方法の種類により制約を分類することにより、ソフトウェアデータベースにおける制約の細分化を行なうことが出来る。制約の細分化を行なうことで、ソフトウェアデータベースの制約という大きな問題を小さな問題に分割して考えることが出来る。小さな問題に対しを一つ一つ、取り組んでいくことで、ソフトウェアデータベースの構築方法の糸口を得ることが期待できる。

ソフトウェアデータベースの制約の細分化という点で本研究を進めて行くことは有効である。

参考文献

[1] 矢上 克之 : UNIX ファイルの依存解析 , 東京工業大学卒業論文 , 1997

[2] 今井 久夫 : ER モデルによる UNIX コマンドデータベースの作成 , 東京工業大学卒業論文 , 1998

[3] PCTE : A Basis for a Portable Common Tool Environment FUNCTIONAL SPECIFICATION , 1988

[4] ECMA : PORTABLE COMMON TOOL ENVIRONMENT (PCTE) C PROGRAMMING LANGUAGE BINDING , 1995

[5] 鯵坂 恒夫, 沢田 篤史, 満田 成紀 : Emeraude PCTE , コンピュータソフトウェア vol.10 ソフトウェア評論 , 1993