ソフトウェアリポジトリにおける意味制約の管理

沢田篤史[†] 満田成紀[††] 鯵坂恒夫[††]

アプリケーションプラットフォームとして機能するソフトウェアリポジトリにおいて，情報の要素間の意味制約を明示的に表現し管理することは，プロダクトの機能性の正確な把握や再利用性の向上などの面から有効であるが，PCTE（Portable Common Tool Environment）をはじめとする既存のリポジトリでは，このような意味制約の取扱いは不十分である．本論文では，ソフトウェアリポジトリのデータモデルを精密に与えるため，構造定義と意味制約を明確に分離して記述する言語 TDL（Topology Definition Language）および CDL（Constraint Definition Language）を提案し，それらによる記述を系統的に管理するリポジトリシステムについて説明する．さらに，システムが提供するプログラミングインタフェースを用いて，意味指向型グラフエディタ UGE（Universal Graph Editor）を構築し，意味制約を管理するリポジトリの有効性を示す．

A Framework for Managing Semantic Constraints in Software Repositories

ATSUSHI SAWADA,[†] NARUKI MITSUDA[††] and TSUNEO AJISAKA[††]

Explicit descriptions and management of semantic constraints among information elements of software are vital for enhancing understandability, maintainability, concreteness and reusability of the products which are stored in and executed on software repositories. However, existing repositories such as PCTE (Portable Common Tool Environment) do not sufficiently handle those semantic constraints. In this paper, we first propose two languages named TDL (Topology Definition Language) and CDL (Constraint Definition Language) for describing respectively the structural definitions and the semantic constraints on a repository's data model. We next explain how we developed a software repository which can systematically manage both TDL and CDL descriptions. Based on the programming interface provided by this repository, we also design a semantic-directed graph editor which we call UGE (Universal Graph Editor) to present effectiveness of our repository system.

1. はじめに

ソフトウェアが扱う情報の要素間には様々な意味制約が存在する．ソフトウェアには，これらの制約に基づいて情報をフィルタリングする論理と，フィルタリングされた情報を計算処理によって加工する論理とがプログラミングされる．これら 2 種類のプログラム論理は本来異質であり，手続き的な計算の論理だけをプログラムとして形成し，宣言的なフィルタリングの論理は，アプリケーションプラットフォームとしてのソフトウェアリポジトリ☆で扱われるべきものである．それにより，意味制約の管理がプログラムコードに埋没させられ保守が困難となっている状況を打開すること，意味制約を開発の早期から考慮する新たな方法論を構築すること，意味制約を再利用の対象とすることなどが可能になると期待できる．

このようなプラットフォームを構築する観点から既存のソフトウェアリポジトリを見ると，国際規格である PCTE（Portable Common Tool Environment）[1),2)] をはじめとして，継承や集約あるいは存在制約など，型に関する限られた意味制約のみがデータモデルに提供されており，意味制約の一貫的な取扱いの面では不十分であるといえる．このような問題に対し，本研究では，リポジトリに格納されるデータ要素間の意味制約をより柔軟かつ明示的に記述する枠組みと，それ

† 京都大学大型計算機センター
Data Processing Center, Kyoto University
†† 和歌山大学システム工学部デザイン情報学科
Department of Design and Information Science, Faculty of Systems Engineering, Wakayama University

☆ ソフトウェアリポジトリは，元来 CASE 環境の情報資源を管理するものとして登場したが，ここでは開発環境だけではなく，情報モデルに従うソフトウェアの動作を保証する実行環境としてもとらえている．

らを系統的に管理するソフトウェアリポジトリを提案する．

以下，2 章では既存のリポジトリにおける制約管理とその問題点について PCTE を例に述べ，3 章では構造定義と意味制約記述のための 2 つの言語 TDL（Topology Definition Language），CDL（Constraint Definition Language）の説明と，それらを管理するリポジトリシステムの実現について述べる．4 章では，システムの適用例として，意味指向型グラフエディタ UGE（Universal Graph Editor）の説明を行う．次いで，5 章でシステムのソフトウェア開発プラットフォームとしての有効性や他の仕様記述方式への適用に関する考察を行った後，6 章で本論文のまとめをする．

2. ソフトウェアリポジトリに必要な制約管理

ソフトウェアリポジトリには，情報資源を効率良く格納する機能と同時に，設計された情報モデルを忠実に表現し，それに従ったソフトウェアの正しい実行を保証する機能が求められている．このような要求に対応するリポジトリでは，システムが提供するデータモデル部品が系統的に整理され，これを組み合わせて利用できるようになっていることが重要となる．既存の汎用リポジトリでは，制約をいわば汎用かつ比較的粗粒度のモデル部品として提供し，用意されたモデル部品の管理はデータ管理システムが行う．反面，汎用部品で表現できない制約に関してはアプリケーションで管理する必要が生じる．

たとえばPCTEでは，実体関連モデル[3]に一定の性質を付加したデータモデルをSDS（Schema Definition Set）と呼ばれる記法で記述する．PCTE 仕様は基本的なデータ型として「もの」を示す `object` 型やファイルに対応する `file` 型などを定めるが，開発するソフトウェアの目的に応じて設計者が新たな型を定義することも可能であり，それらが1つのSDSとしてまとめられる．**図1** にSDSを用いたデータモデルの例を示す．これは簡単な工程管理のためのデータモデルである．ここには，開始日時と終了日時（`start_date`, `end_date`）を記録でき，前後関係（`predecessor/successor`）のある単位工程（`workload`）のリスト（`worklist`）が作業者（`worker`）に割り当てられる様子が示されている．

長方形が実体の型であり，その間を結ぶ矢印を持つ三角形の要素が関連の型である．実体型間の継承関係は特殊な関連として灰色の太い矢印（矢印の始点が基本型で終点が派生型）で示される．実体型に付随する長円形が属性型で，文字列や数（実数，整数）などの

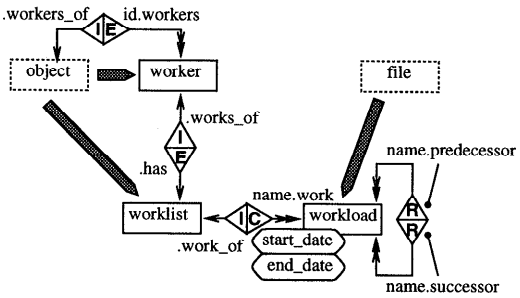


図 1 SDS によるデータモデルの記述例
Fig. 1 An example of SDS data model.

値を格納する要素を示す．

1 つの SDS には，PCTE 仕様が定める基本型や他の SDS が定義する型を取り込み，それらとの継承関係を定めることで新たな実体型を定義し，また実体型間に新たな関連型を定めることができる．図中，破線で表現されている長方形が取り込んだ型であり，実線のものが SDS で新たに定義する型である．

継承関係以外の関連型は基本的に 2 方向の関連の組として表される．関連型に付随する矢印は，その方向に関連のインスタンスが複数存在可能か否かを示すものである．2 重の矢印は一対多の関連を示し，1 重の矢印は一対一の関連を示す．一対多の関連はキー属性を持ち，その値で同じ型を持つ他のインスタンスと識別される．関連型の名前はキー属性型とあわせて，“(キー属性型名).(関連型名)” と示される．たとえば，図 1 の `worklist` と `workload` を結ぶ関連型は，1 つの `worklist` のインスタンスから `workload` の方向へ複数の `work` 型の関連の存在が可能で，それは `name` 属性をキーに識別され，逆方向の関連 `work_of` が一対一であることを示す．

関連の三角形内部の文字（**C**, **E** など）はリンクカテゴリと呼ばれ，それぞれ対応する方向への関連インスタンスに関する以下の意味関係を示す．

- **C**： 構成関係・存在依存性・参照完全性・始点依存性
- **E**： 存在依存性・参照完全性・始点依存性
- **R**： 参照完全性・始点依存性
- **I**： 参照完全性
- **D**： 始点依存性

これらは以下の性質として定義されている．

- 構成関係
  - (1) 関連インスタンスの終点にある実体は，始点にある実体の構成要素である．
  - (2) 構成関係を持つ実体の集合に対する，一括消去，複写などの特別な操作が存在する．

- 存在依存性
  (1) 関連インスタンスの終点として実体を生成できる.
  (2) 関連インスタンスを削除すると終点の実体が削除される.
- 参照完全性
  (1) 関連インスタンスが存在すれば必ず終点の実体が存在する.
  (2) 逆方向にも必ず参照完全性を持つ関連インスタンスが存在する.
- 始点依存性
  (1) 関連インスタンスを始点側から明示的に生成,削除できる.
  (2) この性質を持たない関連インスタンスは,つねに逆向きの関連インスタンスに対する操作に付随する形で操作される.

たとえば,図1の worker と worklist の間の関連型は,has の方向の関連(**E** カテゴリ)が存在依存性,参照完全性,始点依存性を持ち,works_of の方向の関連(**I** カテゴリ)が参照完全性のみを持つ.つまり,2つの実体を結ぶ関連インスタンスを生成する際には,

- has 方向の関連インスタンスが,worker を始点として生成され(始点依存性),
- その終点に worklist のインスタンスが存在しなければならず(参照完全性),存在しない場合には生成でき(存在依存性),
- 逆方向には works_of の関連(始点依存性なし)インスタンスが付随して生成される(参照完全性)

という一連の意味関係とそれに基づく操作が定められることになる.

SDS では,関連が実現する性質がカテゴリという一定の枠に基づいて定められるため,設計するデータモデルに冗長な制約が含まれる可能性が生じる.たとえば,図1で **E** カテゴリを持つ関連(has など)の作る意味関係は **C** カテゴリでも実現できる.つまり,**C** カテゴリを持つ関連(work)によって実現される意味関係が仕様の持つ意味を忠実に反映したものか否かが逆に不明確となる☆.

アプリケーションの仕様によっては,workload 間の前後関係が循環してはならない,あるいは workload の start_date は end_date より前であるなどの意味関係が考えられるが,このような制約をリンクカテゴリでは表現できない.PCTE には SDS で表現できない意味制約を明示的に記述・管理する手段が用意されていないため,作業の前後関係の循環性に関してはそれを表す関係インスタンスを生成する際に,日付の前後関係については属性値を設定する際にそれぞれ検査する必要が生じる.つまり,制約の充足検査により情報をフィルタリングする論理をアプリケーションプログラムに埋め込まざるをえなくなる.

以上述べたとおり,制約に一定の枠がはめられ柔軟な記述が困難であることと,システムが提供する比較的粗粒度の制約部品以外の意味制約を管理できないことが,PCTE の問題点である.この問題に対して,本研究では,実体と関連の接続関係などを定める構造定義と,それ以外の意味制約とを明確に分離し,構造のみを管理するリポジトリシステムの上に,より柔軟で精密な意味制約記述の管理を行うシステムを構築することを試みる.これにより,アプリケーションプログラムに課せられた制約管理の負担を軽減できる.また,これまでプログラムに埋没させられてきた意味制約を明示的な記述とすることで,ソフトウェアの保守が容易となり,制約記述を考慮した新たな分析設計方法論の構築や,制約記述を対象とした再利用の実現も考えられる.

3. 意味制約を管理するソフトウェアリポジトリの構築

前章で述べたように,ソフトウェアリポジトリにおいて柔軟な制約管理を実現するためには,構造定義と意味制約を明確に記述できる枠組みを構築する必要がある.リポジトリスキーマとして表現される情報構造記述をソフトウェアの設計情報としてとらえると,そこに構造と制約という意味的な分類を導入し,それぞれを分離した形で管理することは,設計情報を新たな基準のもとにモジュール化して取り扱うことであり,再利用性やトレース性の向上につながる.

本研究では情報構造記述の基礎として Chen[3] により提唱された実体関連モデルを採用する.既存のソフトウェア設計方法論の多くで実体関連モデルが導入されており,リポジトリがこのモデルをそのまま管理することは,プロダクトの意味の忠実な表現と管理につながると考えられる.

本章では,実体関連モデルの構造を実体集合と関連集合,値集合(属性)の接続関係ととらえ,これらの関係を記述する言語を TDL として設計する.次いで,TDL による構造定義に対する意味制約の記述言語 CDL を設計する.CDL は,初等的な集合論の意味

☆ PCTE の旧仕様である PCTE 1.5[4] に基づく Emeraude PCTE[5),6)] では,構成関係と存在依存性の区別が存在しない(**C** カテゴリしか定められていない)ため,さらに設計判断が不明確となる.

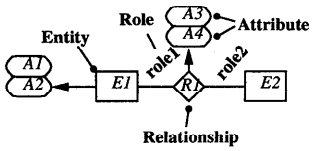


図 2 実体関連モデルの例 (1)
Fig. 2 Example E-R model (1).

に基づいて，実体，関連，属性の各集合間の意味関係を記述するための体系である．さらに，TDL と CDL 記述を管理することのできるソフトウェアリポジトリシステムの構築について説明する．

3.1 構造定義記述言語 TDL

TDL 記述は，1 つの実体関連モデルに対する実体集合と関連集合，およびそれらの接続関係，付与される属性☆の宣言からなる．

実体集合の宣言では，以下に示すように，実体集合名，属性名（もしあれば）が定義される．また，関連集合の宣言では，関連集合名，役割☆☆名とそれに対応する実体集合名，属性名が定義される．

たとえば，**図 2** に示される実体関連モデル（スキーマ名 "sample"）は TDL で次のように記述される．

```
schema sample {
    ent E1 { attr A1, A2; }
    ent E2;
    rel R1 {
       role { role1 E1; role2 E2; }
       attr A3, A4;
    }
}
```

予約語 `schema` の後にスキーマ名が宣言され，続く 2 行では予約語 `ent` の後に実体集合 `E1`（および属性 `A1`, `A2`）と `E2` が宣言されている．続く予約語 `rel` によって宣言される関連集合 `R1` では，関連の役割（予約語 `role`）の名前 `role1` および `role2` を担う実体の集合名がそれぞれ `E1`, `E2` であり，属性 `A3`, `A4` を持つことが示される．

本論文では以下，実体と関連が連結されているとき，実体から関連が流出する，あるいは関連が実体に流入するという．

3.2 意味制約記述言語 CDL

CDL の目的は，TDL によって静的な構造が記述されたデータモデルにおいて，実体と関連のインスタンスや属性の値に関する局所的な意味関係を宣言的に記述することである．

☆ TDL および CDL において，属性は内部構造を持たないと仮定する．

☆☆ Chen[3] の実体関連モデルにおける role に対応する概念で，関連の方向性を示すものである．

CDL では記述された制約の間の矛盾を許している．これは，特にソフトウェア開発の初期段階やプロトタイピングにおける，制約記述の試行錯誤的な更新を可能とするに十分な柔軟性を確保するためである．矛盾する制約記述の実行時の取扱いや検査に関しては，システム構築の節（3.7 節）で後述する．

CDL 記述では，対象となるスキーマ名（TDL 記述）が指定され，TDL で定義された実体，関連，属性に対する制約記述が列挙される．個々の制約記述は，

- 制約名
- ターゲット記述
- 制約式

という 3 つの要素からなる．

制約名は制約記述の識別名である．ターゲット記述には，どの実体集合あるいは関連集合が制約記述の対象となるかを指定する．続く制約式には，論理演算子や四則演算子などを用いてターゲットに対する制約の詳細を記述する．

ターゲット記述には**図 3** に示す 5 種類が存在する．たとえば，"`ent_rel` 実体集合名 (役割名)，関連集合名;" という記述では，指定された名前を持つ実体集合と関連集合との，指定された役割名における接続関係が制約記述の対象となることが示される．それぞれの名前とモデル要素との対応は，図中の矢印によって示され，制約式では，それらのモデル要素を傍らに示した `target1` などの予約語によって参照する．

これらの制約単位が単に羅列される場合には，それぞれのターゲットに関する制約が局所的に成立するよう宣言されているものと解釈するが，`condition` という予約語を設け，和，積，否定の論理演算子で制約名を組み合わせ，より広範囲なデータに対する成立条件の記述も可能とする．

本章では以下，実体集合，関連集合，属性値の間に付与されうる意味的な関係を集合論に基づいて分類し，それぞれに対する CDL の言語要素を設計する．集合論での基本的な意味として，集合要素の数や値域，射の性質，包含関係，構成関係などがあげられるが，これらの意味関係が，実体関連モデルを構成する集合の組合せに対して適用可能か否かを調べ，可能なものに対する言語要素の設計を行う．

3.3 要素数と値域に関する制約

要素数に関する制約には，ある実体や関連のインスタンス数を制限するものが考えられる．本研究で提案する枠組みでは，属性はつねに実体あるいは関連に付随して存在するものと扱うため，この制約には当てはまらない．CDL には，現存するインスタンス数を記

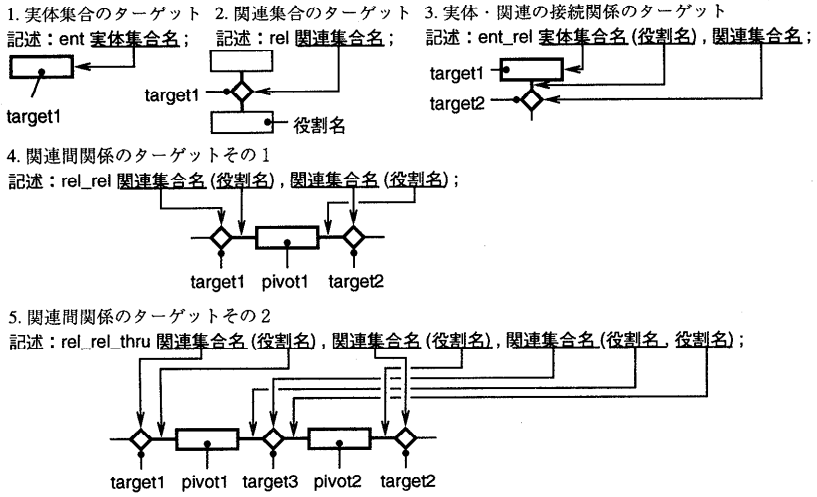


図 3 ターゲット記述
Fig. 3 CDL target definitions.

述するために，num という予約語を設ける．たとえば，図 2 において，実体集合 E1 のインスタンス数の上限を 10 個とする制約は，

```
constraint C1 {
    ent E1;
    num(target1) <= 10;
}
```

と表される．予約語 constraint の後に，制約名 C1 が宣言され，ターゲット記述で，実体集合 E1 に対する制約であることが指定される．制約式で，図 3 のパターン 1 に示すように，E1 は予約語 target1 で参照される．

一方，値域に関する制約は属性のみに与えることができる．属性は実体や関連のインスタンスとともに存在するため，CDL はこの制約を実体または関連に対するものとして記述する．たとえば，次の C2 は，図 2 の実体属性 A1 の値域が正整数であることを示す．

```
constraint C2 {
    ent E1;
    domain(target1.A1, int)
        && target1.A1 > 0;
}
```

domain(target1.A1, int) という記述は，第 1 引数の属性値の種類が，第 2 引数で示されたものであることを示す．ここで，“target1. 属性名” という記法によりターゲットの持つ属性が指定される．値の種類を示す予約語としては，PCTE の提供する値型にならい，int（整数）のほかに，char（文字），float（実数），boolean（真理値），string（文字列），date（日付）を用意する．

3.4 集合間の射の性質に関する制約

この制約は，実体，関連，属性値の各集合について，それらの対応関係の性質を記述するものである．具体的には，ある集合の 1 つの要素に対応する別の集合の要素の数に関する性質（全射，単射，全域性，多価性など）によって表される制約と，要素の持つ値の間に具体的に成立する等式や不等式によって表される制約が考えられる．

射の性質に関する制約には，それが言及する集合の種類の組合せによって，以下に述べる 9 種類の制約を考えることができる．

3.4.1 実体から実体への射に関する制約

実体と実体の対応関係は，それを連結する関連によって表現される．実体間に何らかの意味的な関係が存在すれば，それは関連によってモデル化されるべきであり，CDL は関連が存在しない実体の間には意味制約も存在しないものとして扱う．関連が作る射の性質は，後述の実体から関連への射および関連から実体への射に関する制約を組み合わせることで記述可能であるため，CDL には実体間の射を直接記述する言語要素を設けない．

3.4.2 実体から関連への射に関する制約

これは，1 つの実体に対応して流出する関連の数に関する制約である．たとえば，次の C3 は，図 2 において，1 つの E1 のインスタンスから流出できる R1 のインスタンス数の上限が 10 であることを示す．

```
constraint C3 {
    ent_rel E1(role1), R1;
    num(map(target1, target2)) <= 10;
}
```

図3のパターン3に相当するターゲット記述では，関連 R1 とその役割名 role1 に相当する実体 E1 が制約の対象であることが示される．実体に役割名を付与するのは，1つの実体集合に同じ名前を持つ関連集合が流入する場合，どの役割名で流入する関連を制約の対象とするかを区別するためである．制約式には，実体集合と関連集合の対応関係を表すために予約語 map を設ける．また，前述の予約語 num により，対応関係の数を表現している．

3.4.3 実体から属性への射に関する制約

属性は内部構造を持たないという仮定により，1つの属性には1つの値が付与されうるものであり，実体から1つの属性への射の数は上限が1となる．実体には必ず属性値が付与されることに言及する全域性制約は，射の数が必ず1であることを，予約語 map および num を含む式を用いて記述すればよい．CDL ではこの制約を該当する実体に対する制約として取り扱う．

3.4.4 関連から実体への射に関する制約

関連はそれぞれの役割を担う実体1つずつに対して接続されうるので，関連から実体への射に関する制約は，上述の実体から属性への射に関する制約と同様に記述可能である．CDL ではこの制約を関連の持つ性質として取り扱う．

3.4.5 関連から関連への射に関する制約

関連の間の制約に関して考慮しなければならないのは，制約の対象となるインスタンスの指定方法についてである．実体と関連，実体と属性，関連と属性に関する制約では，モデル上で直接連結されているインスタンスが対象となる．しかしながら，関連集合の間にはそれらの対応を示すデータ要素（関連間関連）が存在しないため，任意の関連の間で制約の対象を指定できない．このため，制約式に含めることのできる関連集合の範囲を限定しなければならない．

CDL は，関連間の制約として取り扱うことのできる範囲を，

- 同一の実体集合から流出する異なる関連集合間
- 第3の関連集合で連結された2つの実体集合から流出する関連集合間

としており，それぞれは，図3のパターン4，5に対応する．それ以外の場合，すなわち第4，第5の関連によって多段に連結された実体から流出する関連の間の制約を対象とすることも考えられるが，制約の対象となる実体および関連のインスタンス数が莫大になりうるため，制約検査のための処理が現実的ではなくなる．多段に連結された実体から流出する関連を対象とする場合には，該当する実体集合間に新たな関連集合を導入することで，図3のパターン5として対応できる．

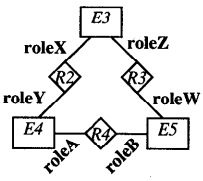


図4 実体関連モデルの例（2）
Fig. 4 Example E-R model (2).

関連から関連への射に関する制約では，1つの関連と別の関連との対応関係の性質を予約語 map を用いて記述する．たとえば，図4の実体関連モデルに対して，次の制約記述 C4 は，同一の実体 E3 から，それぞれ役割名 roleX，roleZ で流出する関連 R2，R3 の間に，R2 のインスタンスが実体から流出すれば同時に R3 のインスタンスも対応して流出しなければならないという制約を示す．

```
constraint C4 {
    rel_rel R2(roleX), R3(roleZ);
    num(map(target1, target2)) > 0;
}
```

図3のパターン4に相当するターゲット記述では，関連 R2 の役割名 roleX 側と，関連 R3 の役割名 roleZ 側が，ともに同一の実体 E3 と連結されている場合，制約の対象となることが示される．

CDL はまた，関連間の射に対して特に map3 という要素を定める．たとえば，図4のモデルに対する次の制約記述 C5 は，C4 で言及している関連間の対応関係が別の関連 R4 の役割 roleA から roleB で実現される射（すなわち実体 E4 と E5 の関連）によって反映されるという制約である．

```
constraint C5 {
    rel_rel R2(roleX), R3(roleZ);
    num(map3(target1, target2,
        R4(roleA, roleB))) > 0;
}
```

ここでは，前述の制約 C4 同一のターゲット記述により，実体 E3 から流出する R2（役割 roleX 側）と R3（役割 roleZ 側）が制約の対象となるが，map3 は，R2 から R3 への対応関係が，第3引数の（役割 roleA から roleB へ向けた）関連 R4 により反映されることを意味する．すなわちこの制約は，R2，R3，R4 の関連によってループが構成されるという意味関係を表現する．map3 の制約は，PCTE を用いたプログラム開発の経験上，SDS 上で循環する3つ（あるいは4つ）の関連型に対応するインスタンスの循環性に関する制約を検査する場合が多くあり，CDL の言語要素として

付加的に必要と判断したものである．

3.4.6 関連から属性への射に関する制約

この制約では，3.4.4 項で述べた実体から属性の射の場合と同様に，関連には必ず属性値が付与されることに言及する全域性制約を記述できる．CDL は，これらの制約を該当する関連に対するものとして取り扱う．

3.4.7 属性から実体への射に関する制約

属性値と実体集合との対応関係において，1 つの値に対する実体の数を制限するものである．1 つの属性値がたかだか n 個（n：自然数）の実体にしか付与されない，という形式で与えられる．特に $n=1$ のとき，属性値によって実体が一意に識別可能となる．逆に，属性の値域が有限集合（真理値）である場合には，1 つの属性値が少くとも n 個の実体に付与される，という制約を考えることができる．CDL は，この制約を該当する実体に対するものとして取り扱う．

従来，実体と属性の対応関係は，属性値の一意性などのように実体側から見た属性の性質ととらえられてきた．この制約は，属性の側から対応する実体の数を制御するという観点から特徴づけられる．

3.4.8 属性から関連への射に関する制約

3.4.7 項で述べた属性から実体への射の場合と同様に，属性値の一意性などを表現できる．CDL ではこの制約を該当する関連に対するものとして取り扱う．

3.4.9 属性から属性への射に関する制約

属性の値の間の具体的な関係を表す等式や不等式によって示される制約である．式によって性質を規定される属性の現れ方には，次の場合が考えられる．

- 同一の実体に付与される属性値間の制約
  → 属性が付与されている実体に対する制約
- 同一の関連に付与される属性値間の制約
  → 属性が付与されている関連に対する制約
- 連結された実体と関連に付与される属性値間の制約
  → 実体と関連の連結関係に対する制約
- 関連の各役割を担う実体に付与される属性値間の制約
  → 当該する関連に対する制約
- 異なる関連の間で，各関連に付与される属性値間の制約
  → 当該する関連間の制約

3.5 集合の包含関係

集合の包含関係は，実体間，関連間，属性間に考えることができる．実体間の包含関係は，一般的なデータモデルで型の継承関係として扱われているものに相当する．実体間の包含関係は，それを反映する関連に対する制約として扱われる．たとえば，図 2 において，R1 の関連により結ばれる，E1，E2 の間の E1 ⊃ E2 なる関係を表す場合には，E1 は E2 のスーパタイプと見なすことができる．この制約 C6 を，

```
constraint C6 {
    rel R1;
    include(role1, role2);
}
```

と記述する．予約語 include は，役割名で示される 2 つの実体集合間に包含関係があることを示す．役割を担う実体集合間に包含関係がある場合，スーパタイプの実体集合が持つ属性などの性質はサブタイプにも利用可能となる．この制約が付与される場合，E1 の持つ A1 および A2 を E2 の属性と見なすことができる．

関連の間の包含関係も同様に記述でき，3.4.5 項で述べた関連間の制約として扱われる．また，属性の間の包含関係は，3.4.9 項に述べた属性間の射の関係と同様に，属性の間に具体的な数式が実現する包含関係として記述可能である．

3.6 集合の構成関係

TDL および CDL は，属性が内部構造を持たないと仮定している．また，ソフトウェア開発方法論には，関連自体が別の副関連によって構成されるといった複合構造を持つようなモデルは存在しないため，CDL でも関連間に構成関係の記述をサポートしていない．属性や関連集合に対する構成関係が必要な場合には，該当する集合を実体集合としてモデル化し，それらの間に関連や制約を付加することで可能となる．

実体間の構成関係の制約を表すために，CDL には予約語 compose を設ける．CDL では構成関係の操作上の意味を，

- 構成要素側の実体は構成関係とは独立して生成・消去できる
- 構成主を複写・消去するとき，構成要素があれば同時に操作される
- 構成関係が定義されていても，それが構成主に対する構成要素の存在依存性を含意しない，すなわち，構成要素の実体は構成主と独立して存在できる

ととらえている．この制約も，包含関係と同様，それを反映して実体の間に作られている関連に対する制約として扱われる．

3.7 リポジトリシステムのアーキテクチャ

これまでに設計した TDL と CDL は，リポジトリが提供するデータモデルをそれぞれ構造定義と意味制約に明確に分離して記述することのできる枠組みである．これらの記述を管理するために構築したリポジト

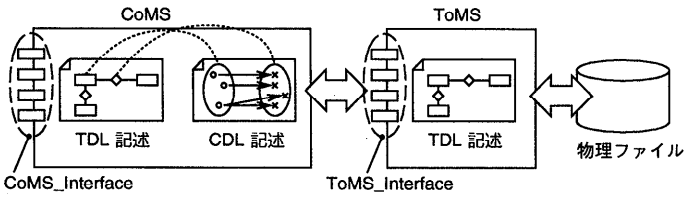


図 5 制約管理システムのアーキテクチャ
Fig. 5 The architecture of the constraint management system.

リシステムは,

- ToMS (Topology Management System)
  TDL で記述された構造定義を管理するシステム
- CoMS (Constraint Management System)
  CDL で記述された意味制約を管理するシステム

という 2 つの独立したサブシステムからなり, 図 5 に示すような構成を持つ.

ToMS は, TDL で記述された構造記述のみを管理し, UNIX ファイルシステムのインタフェース上でデータの永続性を保証するオブジェクトベースの機能を提供する. ToMS は以下の 4 種類のリポジトリ操作をプログラミングインタフェースとして提供する.

- 操作の基点を得る操作 (pivot operations)
  集合名, 属性名および値, 役割名の情報に基づく, 実体および関連の検索
- 実体に対する操作 (entity operations)
  生成, 消去, 複写, 属性値の書き込み, 属性値の読み出し, 連結する関連の列挙
- 関連に対する操作 (relationship operations)
  生成, 消去, 実体との連結, 実体との連結の切断, 属性値の書き込み, 属性値の読み出し, 連結する実体の列挙
- スキーマに対する操作 (schema operations)
  集合名, 接続関係に基づく, 実体集合, 関連集合, 属性名の検索

CoMS は, データ間の射や要素数, 属性値間の関係など, CDL 記述を管理するいわば意味制約管理のためのミドルウェアである. ToMS と同様のプログラミングインタフェースを提供し, 実際のデータ操作は ToMS の操作を呼ぶことで, ToMS が管理するオブジェクトベース上に意味制約管理を実現している. CoMS では, TDL 記述とそれに対応する CDL 記述を解析し, TDL で定義されたデータ集合と CDL で定義された制約記述との対応付けを行い, それを管理している. あるリポジトリ操作が要求されると, その操作の対象となるデータ集合に対応付けられた CDL 記述が検査される. 対応する制約がすべて満たされれば要求された操作を実行する. 満たされない制約が存在する場合には, 操作は行わずに充足されなかった制約名が返される. アプリケーションからは, ToMS, CoMS がそれぞれに提供するインタフェースを利用できる. これによって, アプリケーションにおいて一時的に CDL 記述を満たさない操作が必要になるような場合にも対応可能となる.

CoMS では, リポジトリ操作の結果として局所的に記述された制約が満たされるか否かを検査する. リポジトリ上の全インスタンスが大域的につねにすべての制約を満たしているか否かという厳密な一貫性に関しては, 個々の制約記述を 3.2 節で触れた予約語 `condition` を用いて組み合わせて記述する. `condition` 記述の充足は, 組み合わされた制約記述のうちの 1 つにでも影響する操作が行われる際に検査される. たとえば, 複数の制約記述を論理積の演算子を用いて連結することで, 関連する任意の操作において, 各制約がすべて成立すべきであることが宣言される. 制約記述の組合せやデバッグに関しては, リポジトリの状態と制約記述との双方をユーザとの対話に基づいて変更するツールによって, 今後対応していく方針である.

すでに 3.2 節で述べたように, CDL で記述された複数の制約は互いに矛盾する可能性がある. このような矛盾のうち, TDL による構造定義との矛盾や静的に充足不可能な制約式は, 制約記述の解析時に検出し報告することができる. しかしながら, 実行時の値やインスタンスの状況によって起こりうる矛盾も存在する. これらに関しても上と同様に, 制約記述のデバッガなどのツールで対応する.

3.8 CDL による意味制約の記述例

2 章では, 図 1 を例に, PCTE データモデルを用いて記述できない以下の意味制約について指摘した.

(1) `workload` 間の前後関係が循環してはならない.
(2) `workload` の `start_date` は `end_date` より前である.

図の SDS のうち, 新たに定義された実体型とその間の関連型に対する TDL 記述は以下のとおりである.

```
schema planning {
    ent worker;
```

```
    ent worklist;
    ent workload {
        attr start_date, end_date;
    }
    rel wlist {
        role {
            works_of worker;
            has worklist;
        }
    }
    rel wload {
        role {
            work_of worklist;
            work workload;
        }
        attr name;
    }
    rel pred_succ {
        role {
            predecessor workload;
            successor workload;
        }
        attr pname, sname;
    }
}
```

SDSの実体型に対してそれぞれ，worker，worklist，workloadが定義される．関連型に対しては，2方向それぞれに対応する役割名を持つ関連集合が定義されている．たとえば，実体型workloadの間の関連型predecessor，successorには，TDL記述の関連pred_succが定義する2つの役割が対応する．このTDL記述に対するCDLでは，上の(1)，(2)の意味制約を以下の2つの属性値の大小関係に関する制約(start_end，non_loop)で記述可能である．

```
constraint start_end {
    ent workload;
    target1.start_date
        < target1.end_date;
}
constraint non_loop {
    rel pred_succ;
    predecessor.end_date
        <= successor.start_date;
}
```

PCTEではSDSのEリンクカテゴリによって実現される意味関係がCカテゴリによっても実現可能であるため，設計判断の正確な表現が困難であるという問題も指摘した．たとえば，図1のSDSで，worklistとworkloadの間の関連にCカテゴリを付与することで，存在依存性も同時に付随し，workloadのインスタンスが存在するためにはworklistインスタンスの存在が必要であることになる．つまり，個別にすでに存在するworkloadのインスタンスをworklistの構成要素とすることができないという問題が生じる．逆に，E，Cカテゴリの操作上の意味は，構成全体に対する特別な操作以外では同一のため，存在依存性を持つ関連を記述し操作するだけの目的にもCカテゴリを利用でき，Eカテゴリとの差異が設計結果に現れにくいという問題も生じる．一方，CDLで構成関係を表す予約語composeは，3.6節に示した操作上の意味を持ち，実体関連間の対応関係によって表現される存在依存関係とは独立に指定できる．このため，SDSの場合のような混同は起こりえない．

4. 制約管理を利用した意味指向型グラフエディタの実装

本章では，これまでに設計をすすめてきたリポジトリシステムのソフトウェアプラットフォームとしての有効性を確めることを目的とし，ソフトウェア開発ツールの一種である意味指向型グラフエディタ[7]の実装について説明する．このツールでは，リポジトリシステムの制約管理機構を利用することでグラフ編集操作を制御し，ユーザの誤った操作を抑制することが可能である．

4.1 意味指向型グラフエディタ

グラフエディタは，ソフトウェア開発をはじめとする様々なドキュメント作成作業において頻繁に利用されるツールである．代表的なものには，データフロー図，状態遷移図のエディタがあげられる．これらのグラフ図の編集を支援するうえで重要なことは，図の種類に応じて定められた描画規則があり，個々のグラフ操作はそれに従わなければならないことである．このため多くのソフトウェア開発支援システムにおいては，図の種類ごとの描画規則に従う操作のみを許すように作り込まれた専用エディタを提供している．

意味指向型グラフエディタとは，これらの描画規則が編集対象のプロダクトが持つべき意味から導出されることに注目し，意味定義をもとにグラフ操作を制御できるエディタである．つまり，このエディタはプロダクトの持つべき意味の定義を解釈し，矛盾が生じないグラフ操作のみを実行可能とするものである．

たとえばデータフロー図の場合，「実線円と実線円を実線矢印で結ぶ」というグラフ操作は，

- データ変換どうしはデータフローによって関係付けられる

という意味定義を解釈したうえで，

- データ変換＝実線円，データフロー＝実線矢印

という対応関係がある場合にのみ可能となる．

意味指向型グラフエディタは，与えられる意味定義集合を切り替えることで，様々な種類のグラフ図の編

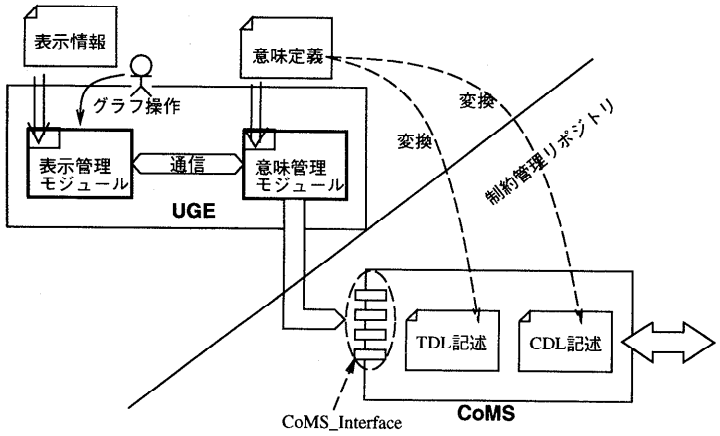


図 6 UGE のアーキテクチャ
Fig. 6 The architecture of UGE.

集ツールとして機能でき，非常に汎用性の高いツールであるといえる．

以下，本研究において意味指向型グラフエディタの具体例として作成したツール UGE（Universal Graph Editor）の制約管理リポジトリシステム上での実装について説明する．

4.2 UGE の構成

図 6 に UGE の構成およびリポジトリシステムとの関係を示す．UGE は表示管理と意味管理の 2 つのモジュールから構成されている．

表示管理モジュール

ユーザによるグラフ操作を管理するモジュール．表示情報（意味要素と表示要素の対応関係．たとえば「データ変換 = 実線円」など）をもとにグラフ操作を意味操作に翻訳し，意味管理モジュールに可能な操作か否かを問い合わせる．意味管理モジュールが了承した操作のみを実行する．

意味管理モジュール

操作が意味的に正当か否かを調べ，正当と認められるもののみを実行するモジュール．操作の正当性は意味定義（意味要素間の関係記述）と照らし合わせて評価される．

意味管理モジュールはプロダクトの格納場所としてリポジトリシステムを利用する．ここでは，プロダクトを構成する意味要素をデータ処理の単位とし，データの生成，消去，更新といった編集操作をすべてリポジトリ操作で実現する．

さらにこの実装では，意味管理モジュールが解釈すべき意味定義を TDL 記述と CDL 記述に変換することで，意味的正当性の評価をリポジトリにすべて任せることを試みる．この結果，意味に関する制約をプログラムコードから独立させることができる．

したがって，意味管理モジュールの果たす機能は，

- 表示管理モジュールから渡された意味操作（データ生成，消去，更新）に対応する CoMS_Interface を呼び出す．
- CoMS からの実行結果（有効・無効）を検証結果として表示管理モジュールに返す．

といった非常に単純なものとなり，開発コストが大幅に削減される．このことが，意味指向型グラフエディタの実装において制約管理リポジトリを利用する最大のメリットであると考えられる．

4.3 グラフ図の意味を定義する TDL，CDL 記述

以下，グラフ図が持つ様々な意味が，TDL および CDL 記述によってどのように定義されるかを説明する．定義されたグラフ図の意味は，プログラムコードに埋め込まれることのない再利用可能な制約コンポーネントと見なすことができる．

(1)「要素 A と要素 B は要素 C によって関係付けられる」

グラフ図におけるノードとアークの接続を表現するには，この形式の意味定義が基本となる．この形式の意味は実体と関連によって記述することができ，次の TDL 記述 S1 に変換される．

```
schema S1 {
    ent A;
    ent B;
    rel C {
        role { r1 A; r2 B; }
    }
}
```

要素 A や B が別の意味定義によってすでに関連として認識されていた場合，要素 C はメタ関連（関連間

の関連）として定義する必要がある．TDL はメタ関連を概念要素としていないが，CDL の関連間制約により，以下のように要素 C の意味を記述可能である．

```
schema S1a {
    rel A {....}
    rel B {....}
}
cdl C_S1a(S1a) {
    constraint C {
        rel_rel A(r1), B(r2);
        ....description....
    }
}
```

要素 C の意味は，関連 A，B をターゲットとする制約として記述される．このとき，要素 C が表すことのできる意味は，要素数制約，射の関係，構成関係といった CDL の記述能力に従うことになる．

(2)「要素 B は要素 A の副要素である」

この形式の意味定義からは，ノードやアークがラベルを持つことやノード間の親子関係が導かれる．ここでは副要素 B が値域を持つか否かで記述が異なる．値域を持つ場合 B は属性であり，次の TDL 記述 S2 に変換される．

```
schema S2 {
    ent A { attr B; }
}
```

グラフ図において値を保持するのはラベルであるから，要素 B はラベルに対応付けられることになる．

一方，値域を持たない場合には属性とはならず，要素 A，B 間の関連を用いて表現される構成関係（3.6 節参照）となる．これは前述のスキーマ S1（実体 A を役割 r1，実体 B を役割 r2 とする関連 C が定義される）のもとで，

```
constraint C1 {
    rel C;
    compose(r2, r1);
}
```

という CDL 記述によって実現され，ノード間の親子関係を制御する．

(3)「要素 A は n 個以下の要素 B と関係付けられる」

あるノードに注目したときに，そこに接続されるアークや親子関係にあるノードの数が制限される場合がある．これに相当する意味定義をスキーマ S1 に対する CDL 記述によって行うと次の制約 C2 となる．

```
constraint C2 {
    ent_rel A(r1), C;
    num(map(target1, target2)) <= n;
}
```

関連 C は要素 A と要素 B を関係付ける要素であり，実体 A から関連 C への射（3.4.4 項）の数に対して制限を加えている．

(4)「要素 A は要素 B によって識別される」

ノードやアークをラベルの値によって一意に識別できなければならない場合に必要な意味定義である．要素 B は識別値を持つことからも属性であることは明らかであり，したがって，この意味定義は属性から実体あるいは関連への射に関する制約（3.4.7 項，3.4.8 項）として記述される．スキーマ S2 のもとで，

```
constraint C3 {
    ent A;
    num(map(target1.B, target1)) == 1;
}
```

という CDL 記述が得られる．

(5)「要素 A，B，C の三者が要素 D によって関係付けられる」

グラフ図の種類により，3 つ以上のノードやアークを同時に関係付ける意味要素（多項関係）が存在する場合がある．TDL では，役割定義を並べることで多実体間関連を記述できる．

```
schema S3 {
    ent A;
    ent B;
    ent C;
    rel D { role { r1 A; r2 B; r3 C; }}
}
```

5. 考　　察

5.1 アプリケーションプラットフォームとしての制約管理リポジトリ

前章で述べたように，制約管理リポジトリをアプリケーションプラットフォームとして利用するメリットは，アプリケーションが扱うデータ間の意味制約に対する正当性がリポジトリによって保証される点にある．つまり，アプリケーションプログラムには手続き的なデータ操作を記述するだけでよく，結果として開発コストを大きく削減できる．また，アプリケーション論理から独立した構造定義や意味制約は，再利用可能な情報部品ととらえることができる．このような利用法においては，プラットフォームが提供する記述言語によってアプリケーションが必要とする意味制約をすべて記述できる場合，そのメリットが最大となる．言い換えれば，プラットフォームとしての有効性が意味制約の記述能力に大きく依存することになる．この点で本研究の制約管理リポジトリは，UGE における意味制約が十分に記述できていることからも，PCTE をはじめとする既存のリポジトリに比べての高い能力を持っているといえる．

たとえば PCTE の SDS では，メタ関連を直接記

述することはできない☆が，TDL と CDL では 4.3 節の（1）に例をあげたように関連間にメタ関連としての意味制約を記述できる．5.2 節で後述するように，記述できる条件が制限されているため，必ずしも任意の関連間関係を表すことはできないが，属性値制約，要素数制約，射の関係，構成関係などの言語要素を組み合わせて記述可能な範囲は，十分に広いものと考えられる．

PCTE に対する制約管理リポジトリの優位性として，4.3 節の（5）に示したように，関連を 3 実体以上の間に定義することが可能な点もあげられる．SDS のように 2 実体間の関連しか記述できない場合，多実体間の関連は複数の 2 実体間関連に分割して記述しなければならない．分割された関連間には「同時に操作の対象となる」という意味制約が必要となるが，SDS にはそれを記述する能力がない．つまり，分割された関連の管理はアプリケーション側で行わなければならない．

CDL の特長として，制約によって任意の属性を実体や関連の識別子として利用できる点もあげられる．既存のリポジトリではあらかじめキー属性として定義されている属性についてのみ，識別機能を利用することができるが，CoMS では構造定義から切り離して記述することで，自由度の高い識別機能が利用可能である．

意味制約の記述能力としては高いものを提供している制約管理リポジトリではあるが，提供すべきアプリケーションインタフェースにはまだ研究の余地があると考えられる．アプリケーションから依頼された操作が記述された制約に違反する場合，現在の仕様では操作を実行せず，制約違反であることをアプリケーションに通知するのみである．このとき，充足されない制約記述に関する詳細な情報が与えられなければ，アプリケーション側でその回避や代替のための操作を行うことができない．たとえば，制約違反がデータの生成や削除の順序によるものであることが分かれば，適切な並べ換えが可能となる場合もある．このようにデータ操作における不正回避という点からインタフェースをとらえることで，新たなインタフェース仕様を加えていく必要がある．また，制約違反の回避方式に関する記述言語を新たに設計し，記述された範囲の不正回避を自動的に行うサブシステムを構築することも必要となろう．

☆ 実体との接続関係とリンクカテゴリを用いて間接的に親子関係を表現することが可能である．

5.2 仕様記述方式としての TDL および CDL

ソフトウェアの情報構造を記述する道具として，様々なオブジェクト指向分析設計方法論で実体関連モデルが採用されており，各方法論では，実体関連モデルを特定の意味制約でそれぞれに拡張している．Coad & Youdon 法[8]では，オブジェクトの間に「汎化・特化」関係と「全体・部分」関係を導入し，OMT（Object Modeling Technique）法[9]では，汎化関係に分離制約を加えた制約を導入している．これらの方法論で導入されている汎化・特化関係は CDL の包含関係制約，汎化の分離制約は包含関係制約を持つ関連の間の射の存在性に関する制約式を用いて記述することができる．

OSA（Object-oriented Systems Analysis）法[10]では，継承関係に和・排他・分離・積の制約を導入し，集約関係やカージナリティ制約に加え，共存関係など，高度な意味モデルを提供している．このモデルに関しても CDL の関連間関連を用いて記述することができると考えられる．これらの方法論で提供される特有のデータモデルに含まれる構造と意味制約とを分離し，TDL と CDL で記述することで，UGE によってそのプロダクトエディタを構築することも可能である．

CDL の 1 つの制約単位として記述できる意味制約の範囲は，3 章の図 3 にあげた，一定の接続関係にある実体，関連，属性の間に存在しうる以下の性質を組み合わせたものに限られている．

- インスタンスの要素数
- 属性の値域，属性値間の関係
- 集合間の射：実体・関連間，実体・属性間，関連・属性間，関連間（循環性制約を含む）
- 集合の包含関係
- 集合の構成関係

この範囲を超えるデータ要素間に制約を記述できない．たとえば，ある実体の属性値と，異なる種類の関連を 3 つ以上たどった先の実体の属性値との間に制約を指定できない．この範囲は，制約管理システムにおける制約チェックの実現性を考慮したうえで決定されたものであるが，4 章で UGE において実現された各モデルのほかにも，今後，上述の各方法論が提供する意味制約の管理を実現することで，CDL の妥当性を実証的に示していく予定である．

PCTE のデータモデルの拡張を行う研究分野において，PCTE オブジェクトにオブジェクト指向性を付加する試みが行われている[11]．ここでは，オブジェクトに対する操作（メソッド）の前提条件，帰結条件として制約の記述が行われ，オブジェクトやそれらの間に付与される条件としてつねに成り立つ静的な制約は

考慮されていない．しかしながら，ある制約が一定の時刻や期間に対して成立するという考え方は，たとえば，ソフトウェアプロセスのモデル化などに有効であるので，本研究でも今後採り入れる必要があるだろう．

6. おわりに

本論文では，ソフトウェアリポジトリにおいて，構造と意味を明確に分離して記述し管理することのできる制約管理システムについて説明した．制約管理システムでは，構造定義および意味制約を記述するための言語（TDL，CDL）の設計を行い，それらを管理するリポジトリシステム ToMS，CoMS のアーキテクチャと実装について説明した．さらに，システムのプログラミングインタフェースの有効性を示すために，UGE と呼ばれる意味指向グラフエディタの構築について説明した．

リポジトリシステムは，現在 UNIX オペレーティングシステム上に C++を用いて記述され，ToMS，CoMS のアプリケーションインタフェースは C++のクラスライブラリとして提供されている．今後は，3章にも述べたように，CDL 記述とリポジトリの内容を対話的に操作できるデバッガの構築，TDL，CDL 記述のための GUI ツールの構築☆など，関連ツールの整備を行う必要がある．また，制約記述と構造定義との間の対応を CDL のターゲットとして直接記述するのではなく，テンプレートの形で一般化し，頻出する制約記述を容易に部品化できるように言語仕様を改良することも今後の課題である．

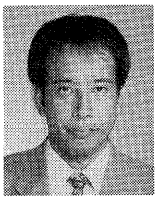
実体関連モデルを用いたソフトウェア分析・設計では，関連を実体に変更するといった，各モデル要素の間の相互変換がしばしば見られる．このような相互変換は，分析・設計の過程で意味制約が詳細化されるなどした結果，異なる要素への変換によってより適切なモデル化が可能となる場合に発生すると考えられる．今後は，分析，設計段階におけるモデル要素の相互変換とその原因となる制約とのかかわりを分類することによって，意味制約を考慮したデータ設計の方法論を確立することも課題となる．また，アプリケーションごとに異なる視点から共有されるデータ構造や意味制約に関する再利用方式やその枠組みの検討も今後の課題である．

☆ TDL，CDL に対する GUI は UGE を用いて実現できると考えている．

参考文献

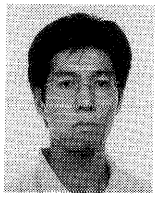
1) International Standardization Organization (ISO): *Portable Common Tool Environment (PCTE)*, ISO/IEC 13719 (1995).
2) Wakeman, L. and Jowett, J.: *PCTE The Standard for Open Repositories*, Prentice Hall (1993).
3) Chen, P.P.-S.: The Entity Relationship Model – Toward a Unified View of Data, *ACM Trans. Database Syst.*, Vol.1, No.1, pp.9–36 (1976).
4) Commissions of the European Communities PCTE Interface Group (PICG): PCTE: A Basis for a Portable Common Tool Environment, Functional Specification, Version 1.5 C Vol.1 (1989).
5) GIE Emeraude: The Emeraude Environment, Manual of the Emeraude Version 12.3.1 (1992).
6) 鯵坂恒夫，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, pp.65–77 (1993).
7) Mitsuda, N., Ajisaka, T. and Matsumoto, Y.: A Semantic-Directed Graph Editor on PCTE, *Proc. IPSJ and KISS Joint Conference on Software Engineering'93*, pp.69–76 (1993).
8) Coad, P. and Yourdon, E.: *Object-Oriented Analysis*, 2nd Edition, Yourdon Press (1991).
9) Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W.: *Object-Oriented Modeling and Design*, Prentice Hall (1991).
10) Embley, D.W., Kurtz, B.D. and Woodfield, S.N.: *Object-Oriented Systems Analysis: A Model-Driven Approach*, Yourdon Press (1992).
11) Charoy, F.: An Object-Oriented Layer on PCTE, *Proc. PCTE'93 Conference*, Paris, pp.379–389 (1993).

(平成 9 年 8 月 25 日受付)
(平成 10 年 9 月 7 日採録)



沢田 篤史（正会員）

1990 年京都大学工学部情報工学科卒業．1992 年同大学院工学研究科情報工学専攻修士課程修了．1995 年同専攻博士後期課程研究指導認定退学．同年，奈良先端科学技術大学院大学情報科学研究科助手．1997 年京都大学大学院工学研究科助手．同年，同大学大型計算機センター助教授，現在に至る．京都大学博士（工学）．ソフトウェア工学の研究に従事．特にソフトウェア生産環境，情報共有および統合技術，ソフトウェアリポジトリに関する研究を行っている．日本ソフトウェア科学会会員．



満田 成紀（正会員）

1991年京都大学工学部情報工学科卒業．1993年同大学院工学研究科情報工学専攻修士課程修了．1996年同専攻博士後期課程研究指導認定退学．同年，和歌山大学システム工学部デザイン情報学科助手，現在に至る．京都大学博士（工学）．専門分野はソフトウェア工学．特にソフトウェア開発環境，ミドルウェア，ユーザインタフェースサービスに関する研究を行っている．

鯵坂 恒夫（正会員）

1980年京都大学理学部物理学系卒業．1985年同大学院工学研究科情報工学専攻博士課程研究指導認定退学．同年，京都大学工学部情報工学科助手．1990年同助教授．1997年和歌山大学システム工学部デザイン情報学科教授，現在に至る．工学博士．1993年度，文部省在外研究員として英国Durham大学計算機科学科に滞在．専門分野はソフトウェア工学，情報システム工学．ソフトウェア設計自動化，再利用技術，ソフトウェアプロセスなどの研究を経て，最近は特にソフトウェアリポジトリやミドルウェアに関する研究を行っている．日本ソフトウェア科学会，IEEE Computer Society，ACM，電子情報通信学会，システム制御情報学会各会員．