ソフトウエア工学 93−20
（1993．7．9）

細粒度情報モデルに基づくＣＡＳＥの実用経験

西岡健自　平田陽一郎　渡邊多恵子　日野泰臣　秋本綾子

横河電機株式会社　オープンシステム研究所

開発工程を統一して，開発現場のソフトウェアプロセス成熟度を向上させるため，C言語向けプログラミング環境idtを構築し運用した．idtは細粒度情報モデルに基づくデータ統合によって，フェーズ，モジュールにわたるドキュメントやソースプログラムの整合性を保証し，機械的な記述作業，変更反映作業を自動化する．この機能によって，開発工程の統一に伴う偶発的な問題が解決し，工程統一の直接的効果を引き出して，開発工程を定着させることができた．idtは1年前に運用を開始し，現在，約50名のユーザが使用している．idtの支援する開発工程もidtと共に複数の開発現場に展開しつつある．本報告では，idtの特徴と構築，運用の経験について述べる．

Practical Experience of CASE based on Fine-grained Information Model

Kenji Nishioka, Yoichiro Hirata, Taeko Watanabe
Yasuomi Hino, Ayako Akimoto
Open Systems Laboratory, Yokogawa Electric Corporation
email: nishioka@crd.yokogawa.co.jp

We developed a programming environment for C language named idt to standardize a software development process and improve a software process maturity in software development divisions. The idt has an object management system (OMS) based on fine-grained information model. By the OMS, it guarantees consistency among documents for various development phases and modules. And it solves accidental problemss to standardize software development process. Currently about 50 engineers use idt, and work with software development process supported by idt.

In this report we describe overview of idt and its experience of building and using.

1 はじめに

中規模以上のソフトウェア開発プロジェクトでは，経験の裏付けのある反復可能な開発工程の確立が重要である．　ソースプログラムを含むドキュメントの種類と盛り込むべき内容，配置の統一は情報の漏れを防ぎ，読みやすく完結した情報の蓄積を実現する．そして，読みやすく完結したドキュメントやソースプログラムは，エンジニアが直接携わるソフトウェアの開発において，ソフトウェア製品の信頼性の基盤となる．

このような分かりやすさは高い保守性実現の鍵でもある．そして，高い保守性は要求の変化に応じて長い寿命を維持する条件であり，多大な工数を費やして開発したソフトウェアから十分な利益を回収するには，長い寿命が不可欠である．

また，記述や改訂の手順の統一によって進捗度を把握しやすくなり，効果的なプロジェクト管理が可能となる．そして，プロジェクト管理の確立はソフトウェアプロセスの成熟度を初期状態から引き上げるための最初の基本的対策である [1]．

しかし，統一した開発工程を実際に現場に定着させるのは容易なことではない．　経験の裏打ちがあっても，規約の形式でエンジニアの活動を拘束するのは彼らに精神的負担を強いることになる．ドキュメントの種類や書式，コーディングのスタイルまで規約のみで標準化するのは記述作業効率の上でも無理がある．また，記述結果は規約を遵守して始めて読みやすく完結したドキュメントやソースプログラムとなるが，規約の遵守には規約との食い違いを検査し，修正する新たな作業が必要となる．

Brooks[2] はソフトウェア開発工程に現われる問題を偶発的なものと本質的なものに分けている．本質的な問題とは仕様定義や設計の知的活動自体に付随する難しさで，偶発的な問題はそれらの表現や表現上の細かいテストに付随して発生するものである．

この観点から，開発工程の統一は未熟な開発プロセスに起因する偶発的問題を解決するものの，規約のみに頼る運用形態は開発作業上に新たな偶発的問題を発生させることになる．

しかし，ドキュメントの種類や書式の統一を逆手にとれば，開発工程遵守の自動チェック，あるいは，標準的開発工程からの逸脱防止を実現できる．また，記述作業の機械化等直接目に見えるメリットによって，エンジニアの精神的負担を緩和させることができる．こうして，統一した開発工程は彼らがソフトウェア開発の本質的問題に専念するための支援環境となり得る．

このような開発工程の統一を目指して，当社のソフトウェア開発事業を担当するシステム技術部では細粒度情報モデルに基づくオブジェクト管理システム TENSE OMS[3] を導入し，その周辺に C 言語向けのプログラミングシステムを構築した．このプログラミングシステムを idt と呼ぶ．なお，TENSE OMS は社内で研究開発したシステムで，idt の開発は事業部と研究開発部門が共同で行なった．

以下，idt の支援対象であるプログラミング工程の概要，ベースとなる TENSE の概要と開発工程定着に果たす効果，idt の構成と特徴，運用評価について述べる．

2 開発工程の概要

当社は石油，製鉄，電力，水道等のプラントにプロセス制御システムを納めるＰＡ（プロセスオートメーション）分野を中心に，ＦＡ，ＬＡ，ＯＡ，さらには，ＣＩＭ，ＳＩＳの分野にわたる典型的な多品種小量生産のメーカである．また，近年はコンピュータ関連製品の大規模化，複雑化を反映してソフトウェアエンジニアの数が増大し，ソフトウェアの開発コスト削減が切実な問題となりつつある．特に，大規模プラントの制御，監視を行なうプロセス制御分野ではソフトウェアに高い信頼性が必要となる．

システム技術部ではこれらの要請から上流フェーズを中心に文書化システムを構築，運用してきた．そして，次の段階として詳細設計以降のプログラミングフェーズへの展開を図る必要があった．また，詳細設計フェーズで高度の信頼性を作り込むために市販の CASE ツールや PDL (Program Design Language) の導入を検討してきた．　従来の経験的手法にこれらの検討結果を加えた，C 言語向けのプログラミング工程は図 1 のとおりである．　基本はウォータフォールモデルで，詳細設計以降は以下の順に開発作業が進行する．

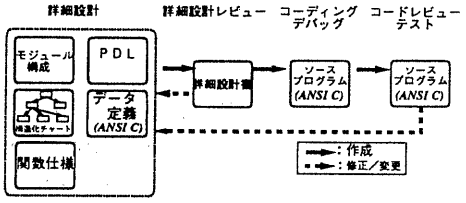

図 1 C 言語のプログラミング工程

1. 詳細設計：基本設計で明らかになった個々の処理内容について，モジュールに分解し，個々のモジュールの含む関数，データの詳細を定義する．
2. 詳細設計レビュー：1 の内容を詳細設計書として整理し，関係者間で調整，確認，修正を行なう．変更のある場合は 1 に戻る．
3. コーディング／デバッグ：詳細設計書に基づきソースプログラムを記述し，単体テストを行なう．大きな変更のない限り，作業効率上詳細設計には反映させない．
4. コードレビュー／テスト：ソースプログラムの机上チェックと結合テストを行いながら，ソースプログラムを改訂する．また，修正，変更内容を詳細設計に反映する．

この過程で，図 1 のように 7 種類のドキュメントが現われる．個々のドキュメントの概要は以下のとおり．なお，括弧内はファイル名の一般形．個々のドキュメント例は付録にまとめる．

1. モジュール構成（モジュール名.msp）：通常テキスト．モジュール毎に記述．内容は参照するヘッダファイルとモジュールを構成する関数の

説明付き一覧．なお，モジュールはコンパイル単位にあたる．

2. 構造化チャート（関数名.sce）：CASE ツールの出力テキスト．主な関数毎に記述．内容は関数の呼び出し関係と授受する情報．
3. 関数インタフェース（関数名.fsp）：通常テキスト．関数毎に記述．内容は仮引数等の説明．
4. PDL（関数名.pdl）：通常テキスト．関数毎に記述．当社で定義した疑似言語による手続きアルゴリズム．
5. 詳細設計書（モジュール名）：roff 形式テキスト．モジュール毎に記述．1～4 を第3者でも通読できるよう分かりやすくまとめたドキュメント．
6. データ定義（データモジュール名.h/.c）：通常テキスト．基本設計に基づき構造体等のデータ構造定義はヘッダファイルとして，大域変数等はソースプログラムとして C 言語で直接記述．
7. ソースプログラム（モジュール名.c）：通常テキスト．モジュール毎に記述．詳細設計書に基づく C 言語プログラム．コメントとして，PDL も含む．

ドキュメント体系としての特徴は関数インタフェース，PDL を関数毎に独立したファイルとしてテキストエディタの操作を単純化し，記述作業を行いやすくする点である．また，4 種類の詳細設計情報をモジュール単位で組み合わせて，全体をとおして読みやすくレビューしやすいドキュメントを提供する点である．このような特徴により，ソフトウェア製品の信頼性を上げ，効果的なプロジェクト管理を手際よく行なうことができる．

しかし，このドキュメント体系を支援環境なしで実際の開発作業で運用するには，作業効率や管理の上で次のような問題点がある．

1. 記述の2度手間：1～4 のドキュメントに基づいて，5 の詳細設計書を書く等記述作業の2度手間が多い．ドキュメント間で重複する情報は表 1 参照．
2. 煩雑な変更作業：個々のドキュメントの自己完結性を高めて読みやすくするために，表 1 のように異種のドキュメント間で重複する情報が多く，変更の反映作業が煩雑になる．
3. 情報の食い違いの発生：情報の重複は関数の定義，参照のように同種のドキュメント間でも発生し，記述や変更の負荷が増し，情報の食い違いも発生しやすくなる．
4. 新しい作業の発生：開発工程の確立には，個々のドキュメントがドキュメント体系や書式に準拠していることのチェックや準拠していないドキュメントの修正等，従来の開発では存在しなかった作業が必要となる．

上記のような作業の自動化にはドキュメント間の変換処理を考えることができる．しかし，個々のドキュメントの情報の重複は表 1 のように広範で複数の同種，異種のドキュメントに亘っているため処理効率や開発すべき変換ツールの量の上で現実的ではない．従って，システム事業部では細粒度情報に基づく TENSE OMS を採用した．

3 TENSE の概要

TENSE OMS のねらいはエラーの発生を防止し，機械的作業を自動化することによって，開発や管理における偶発的問題を解決することである．また，この解決によって，ソフトウェアエンジニアやプロジェクトマネジャを知的な仕事に専念できるようにすることである．

TENSE OMS の核は Prolog で記述したソフトウェアデータベース（TENSE データベース）とその管理処理群である．この OMS の特徴はフェーズやモジュールに亘って重複する情報を把握するために管理する情報の粒度を細かくして，情報の重複に起因する作業上のムダや人為的ミスを防止する点である．ここで，粒度とは情報の論理的な処理単位の粗さである．

表1　各ドキュメントの内容比較

開発フェーズ	詳細設計					詳細設計／コーディング･･･		コーディング
ドキュメント種類	ﾓｼﾞｭｰﾙ構成	構造化ﾁｬｰﾄ	関数仕様	PDL	詳細設計	ﾃﾞｰﾀ定義(.c)	ﾃﾞｰﾀ定義(.h)	ｿｰｽﾌﾟﾛｸﾞﾗﾑ
記述対象	システム	主な関数	関数	関数	モジュール	モジュール	モジュール	モジュール
情報内容 ・網掛部分を記述 ・他は上流より継承 ・いずれも書換は可 ・いずれも他ﾄﾞｷｭﾒﾝﾄの変更の自動反映を受ける 注) *名称のみ書けば良い **変更は下流ﾄﾞｷｭﾒﾝﾄからに限る． ***ｿｰｽﾌﾟﾛｸﾞﾗﾑ形式のﾃﾞｰﾀ詳細設計	各ﾓｼﾞｭｰﾙ： 名称 説明 include情報 各関数： 名称 PDL ﾏｰｸｱｯﾌﾟ	関数： 名称 ｲﾝﾀﾌｪｰｽ 呼出し関数	ﾓｼﾞｭｰﾙ： 名称 説明 関数： 名称 ﾀｲﾄﾙ 説明 ｲﾝﾀﾌｪｰｽ詳細	ﾓｼﾞｭｰﾙ： 名称 説明 関数： 名称 ﾀｲﾄﾙ PDL*	ﾓｼﾞｭｰﾙ： 名称 説明 include情報 関数： 名称 ﾀｲﾄﾙ 説明 ｲﾝﾀﾌｪｰｽ詳細 呼出関数 pdl	ﾓｼﾞｭｰﾙ： 名称 説明 RCSﾊﾞｰｼﾞｮﾝ ﾃﾞｰﾀ： 大域変数	ﾓｼﾞｭｰﾙ： 名称 説明 RCSﾊﾞｰｼﾞｮﾝ ﾃﾞｰﾀ： struct定義 union定義 typedef定義 extern宣言 ｲﾆｼｬﾙ宣言	ﾓｼﾞｭｰﾙ： 名称 説明 include情報 RCSﾊﾞｰｼﾞｮﾝ 関数： 名称 ﾀｲﾄﾙ 説明 ｲﾝﾀﾌｪｰｽ詳細 呼出関数 pdl(ｺﾒﾝﾄ形) 実行ｺｰﾄﾞ ﾛｰｶﾙ変数

TENSE では開発途上に現われるドキュメントの情報を分解して，論理処理可能な形でデータベースに格納する．この時，ドキュメントで重複する情報はデータベース上では各々1個の情報に一元化する．従って，ドキュメントと TENSE データベースの情報とが常に一致するような管理を行なえば，ドキュメント間の情報の食い違いを防止することができる．また，この管理を機械化すれば，記述の繰り返しや変更の反映作業を自動化することができる．
以下，TENSE の情報モデル，開発支援機能，カスタマイズ機能について概観する [3,4].

3.1 細粒度情報モデル

以上の目標を満たす TENSE データベースの情報モデルの特徴は以下のように4項目にまとめることができる．

1. 細粒度：内容の食い違いを検出できる程度以上の細かい情報粒度をもつこと．
2. 高抽象度：変更に伴う影響解析ができる程度以上に高い抽象度をもつこと．
3. 一元性：ドキュメント間で重複する情報を一元化できること．
4. 完全性：ドキュメントの内容を過不足なく，論理処理可能な形で保持できること．

特に，最後の特徴はソースプログラムのコメント情報まで TENSE データベースで保持することを主張している．この特徴によって，TENSE データベースから格納したすべてのドキュメントを完全に復元することができる． この特徴を実現する情報の単位が要素オブジェクトである．要素オブジェクトはプロセス／モジュール／プロシジャ／大域変数...といった，開発フェーズやドキュメントの書式に依存しない情報単位である．要素オブジェクトには，実体に関するインスタンス要素オブジェクト，分類に関するクラス要素オブジェクトがある．以下，両者をそれぞれクラス，インスタンスと略称する．

インスタンスは内部にその属性と他の要素オブジェクトとの関係を一括して保持している．属性とは書式に依存せず，ドキュメント間で重複する情報が形式的に同一視できるまでドキュメントを細分化した情報の断片である．これらの情報は図2のように，各々いづれかのインスタンスの属性として分類できる．この時，重複する情報は必ず特定のインスタンスの一個の属性となる．

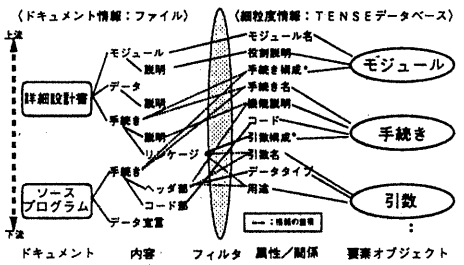

図2 要素オブジェクトの概要

クラスは同種のインスタンスに対応して一個ずつ存在しそれらに共通の情報を保持している．この情報としては属性，関係の種類等がある．なお，クラス構成，個々のクラスの保持する情報はソフトウェア開発工程やプロセス成熟度の進化によって変化する．

クラスは階層関係を構成し，上位のクラスの特性を下位が受け継ぐ．この継承関係により，情報の保守性が高まり，ソフトウェアプロセスに応じて，クラス情報のカスタマイズを効率的に行うことができる．

要素オブジェクトの保持する関係情報としては，継承関係（AKO）の他に，参照（REF），部分（BMO），記述（SPC）の4種類がある． 部分関係はインスタンス間の階層的な部分関係である．プロシジャとそれを含むモジュール，引数やローカル変数とそれを含むプロシジャが部分関係の例である．参照関係はインスタンス間の参照関係である．大域変数とその外部参照宣言を含むモジュール，プロシジャとそれを呼び出しているプロシジャの関係等が参照関係である．記述関係はドキュメントファイルのような物理インスタンスと，その記述対象となるモジュールのような論理的インスタンスの関係である．以上の関係を図3にまとめる．

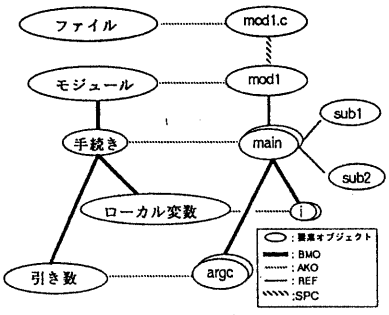

図3 要素オブジェクトの保持する関係情報

3.2 TENSE の機能

以上の情報モデルに基づいて，全ドキュメントと TENSE データベースの内容の整合性を保持するツール群をフィルタと呼ぶ（図2参照）．フィルタは格納対象となるドキュメントの種類毎に用意するもので，以下の3つのツールからなる．

1. 抽出ツール：ドキュメントを構文解析して，要素オブジェクト群に変換する．
2. 格納ツール：抽出結果をデータベース内容と照合しながら格納する．ドキュメントに変更のあった場合は，関係情報に基づいて影響解析を行い影響の及ぶドキュメントに変更通知を出す．
3. 逆生成ツール：データベースの内容に基づきドキュメントを生成する．変更通内を受けているドキュメントを逆生成すると変更内容を反映したものができる．

これらのフィルタを適宜使用することによって，TENSE OMS は以下のような機能を実現する．

1. 記述作業の機械化：上流側のドキュメントを格納すれば，逆生成によって下流や異なるモジュールのドキュメントはその情報を自動的に受け継いで部分生成できる．従って，同一の内容について記述を繰り返す必要がなくなる．
2. 変更作業の機械化：定義情報を変更すると，格納処理で上下フェーズのドキュメントや異なるモジュールのドキュメントから影響を受けるものを検出し通知を出す．それらを逆生成すると自動的に変更内容が反映する．
3. 情報サービス：ソフトウェアデータベースに蓄積した情報から，多様な情報サービスを実現する．たとえば，逆生成によって新たな様式のドキュメントや各種一覧表の自動生成，プロジェクトの進捗状況やフェーズにまたがる情報の静的解析レポートの発行等を行う．

3.3 TENSEのカスタマイズ

多様なソフトウェアプロセスとその進化に対応するために，TENSE OMSはカスタマイズ機能を備えている．上記機能の3は逆生成ツールのカスタマイズ機能で実現できる．TENSEのカスタマイズ方式は図4のようにクラス情報とドキュメントの書式の定義から格納，逆生成処理等の管理処理を自動生成するものである．ここで，メソドとはフィルタ等のクラスに付随する手続きである．また，メタ属性は要素オブジェクトの種類毎に共通の特性情報である．

基づいて，整形して出力しながら，書式で指定したクラスの逆生成処理を階層的に呼び出す方式をとる．従って，逆生成処理は各クラスに付随したメソドとなる．

格納処理もクラス情報から生成し，クラスのメソドとなる．このようにTENSEではメタ属性をインタプリーティブに実行する方式を採らず，自動生成方式で管理処理を実現することにより，処理の速度化を実現している．

なお，現状では抽出ツールは部分的にクラス定義情報から生成できるが，構文解析部分等は個別に開発する必要がある．

4 idtによる開発の流れ

idtはTENSEのカスタマイズ機能によって構築した専用OMSの周辺に各種支援ツールを配したプログラミング環境である[6]．

idtの基本構成は図5のように，TENSEデータベースの周りに7種のドキュメントを一元管理する形態をとる．idtを使用するアプリケーション開発担当者は仕様定義，基本設計等の上流工程の終了後idtを使用する．idtによる一般的な開発作業の流れはおおよそ以下のようになる．

1. モジュール構成の記述：専用エディタで空欄記述式に作成．編集が終わると抽出，格納ツールが自動的に起動してTENSEデータベースに格納．

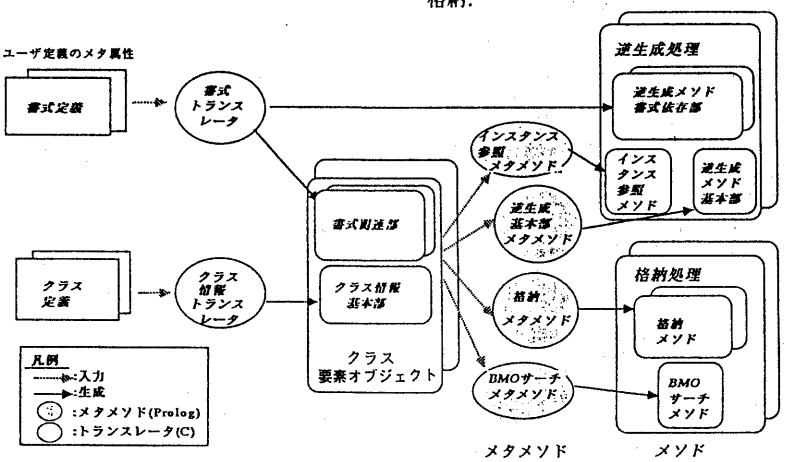

図4 TENSE OMSのカスタマイズ機能

属性や関係の種類と数，当該要素オブジェクトを参照する可能性のあるクラス名リストもメタ属性である．このリストはインスタンス値の変更の影響把握のために必要な情報である．この情報によって，影響把握の探索が高速化する．

各ドキュメントの書式情報もメタ属性の一つである．各クラスは各ドキュメントの種類毎に，要素オブジェクトレベルまで分割した書式情報を保持している．逆生成処理はこの書式情報から直接生成する手続きである．逆生成処理は各属性情報を書式に

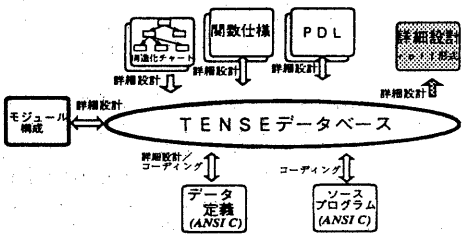

図5 idtの基本構成

2. 構造化チャート（SC）の記述：主な関数についてSCエディタを使用して作成．編集が終わると抽出，格納ツールが自動的に起動してTENSEデータベースに格納．

3. 関数仕様の記述：専用エディタを起動すると，1，2に基づいて関数仕様書が部分生成する．これを空欄記述式に完成させ，編集が終わると抽出，格納ツールが自動的に起動してTENSEデータベースに格納．

4. PDLの記述：専用エディタを起動すると，1，2，3に基づいてPDLのヘッダ部分が生成する．これを完成させ，編集が終わると抽出，格納ツールが自動的に起動してTENSEデータベースに格納．

5. 詳細設計書の生成／レビュー：1～4を適宜完成させ，逆生成ツールによってroff形式の詳細設計書を生成．レビューを実施，その結果に基づいて1～4を変更．

6. コーディング：逆生成ツールにより1～4に基づくソースプログラムを部分生成．空欄となっているコード部分を記述．詳細設計でのアルゴリズムに変更を要す場合は，ソースプログラム内にコメントとして付随しているPDLを変更．

7. デバッグ，コードレビュー，テスト：大きな変更のない限りソースプログラムに基づいて実施．

8. 詳細設計への変更反映：全てのソースプログラムをTENSEデータベースに格納すると，7での変更が詳細設計に自動反映．

以上の手順により，必要最小限の記述量で整合のとれたドキュメント一式を完成させることができ，情報の食い違いから生ずるトラブルも未然に防ぐことができる．また，従来手作業で記述していたroff形式の詳細設計書を自動生成するなど，開発担当者の作業を直接低減させるメリットによって開発工程を速やかに定着させることができる．

さらに，TENSEデータベースへの格納時に構文解析を行なうため，書式からの逸脱を早期に発見でき，その場で修正することができる．

5 idtの実用経験

idtは1991年末より開発を開始した．1992年4月には詳細設計中心の最小機能版の試用が始まり，当社主力製品向けのC言語プログラムの開発に運用することとなった．　その後，事業部での運用と並行して機能の追加拡張を続け，現在所期の機能と安定した稼働を保証できる信頼性を実現した段階にある．ユーザ数は約５０名で，他の事業部での運用も始まっている．

しかし，構築の過程では現場の要求からTENSE OMS自体の改造も多く発生した．また，ユーザからのフィードバック情報は今後のidtの充実の課題として，実際的なソフトウェア開発支援環境の進むべき方向のヒントとして蓄積している．　以下，構築上の問題と対策，運用上の課題として主なものを紹介する．

5.1 idt構築上の問題と対策

TENSE OMSに基づく開発環境としては，モジュール単位の詳細設計書とソースプログラムを対象とする統合化Cプログラミングシステムがあった．しかし，idtは6種の異なる形式でモジュール単位，関数単位の混在するドキュメントを対象とするため，影響解析機構等の拡張が必要だった．また，書式に沿った記述作業を支援するためにemacsベースの各種エディタを開発する必要があった．特に，ソースプログラムエディタではコメントとして記載してあるPDLを編集するために，ユーザの要求に基づいてPDLの構造エディタを呼び出せる機構を持たせている．その他の主な拡張は以下のとおり．

1. フィルタ起動方式のモード化：統合化Cプログラミングシステムではドキュメントの編集開始，終了時にフィルタを自動的に起動した．しかし，idtではコーディング／デバッグフェーズでソースプログラムの格納を禁止する必要があるため，コーディング時には編集終了後，格納コマンドを手動で起動する方式をとった．

2. 条件コンパイル対応：濫用すると読みにくくなり，実現上も抽出時の構文解析が困難となるが，ユーザとの協議にもとづく制限付きで対応した．制限はステートメントの途中に条件コンパイルを入れない，条件のthen，else節に同一のインスタンス名が現われないようにする等．

3. 順序情報の組み込み：従来，大域データやマクロ等異種のデータ宣言が混在して現われることを想定していなかった．しかし，条件コンパイルを導入したこともあり，異種混在時の順序情報を保持できるようにした．

4. 変更通知方式の変更：従来，影響解析で検出したドキュメントが既存の場合は改名，不在の場合はダミーの空ファイルを生成する方式をとっていた．しかし，ユーザの誤操作を招くことから，OMS内部で隠蔽した形で通知情報を管理する方式とした．

5. 参照情報の格納：従来，extern宣言等の参照情報は対応する大域データ宣言に先行して現われた時大域データ宣言の代用として格納し，他の場合は照合にのみ用いていた．しかし，ユーザライブラリ等の使用では大域データ宣言が現われない場合もあり，参照情報の格納方式を拡張する必要があった．

5.2 idt運用上の課題

開発現場での運用を開始してからほぼ１年になる．信頼性の点でトラブルが多かったものの約５００項目のテスト仕様書を用意して安定な稼働を実現できている．idtの基本機能については，詳細設計からのソースプログラムの部分生成，roff形式詳細設計書の生成，変更の自動反映機能等に対してユーザの評価を得ている．これらの直接的メリットにより開発工程も反復可能なものとして定着しつつある．

この間，idtの運用に関する1992年度の事業部からの要求は約３０件にのぼった．この内約半数は

上記構築上の問題点を含み対応済みだが，主な未対応の要求を以下に例示する（発生順に列挙）．

- 対応するドキュメント間の渡り歩き機能が欲しい（編集対象名前指定の自動化）．
- 空欄の残っているドキュメントの逆生成で空欄位置を知らせて欲しい．
- 事業部のコーディング規約にはづれたコードに対して警告表示して欲しい．
- データ宣言をグループ化して見出し行にコメントを入れたい．その他，コメント位置の制限を緩和して欲しい．
- コーディングスタイルを K&Red2 対応にして欲しい．(}else{,return yew 等)
- エラーメッセージの内容を分かりやすくして欲しい．(より分かりやすくしたが，不満は解消していない．エラー処理としてより詳細な状況把握を実施中)
- 格納時間をスピードアップして欲しい（逆生成は 1 sec 以内で問題なし）．
- 事業部で行なっている RCS による製品管理とリンクさせて欲しい．
- TENSE データベースの内容をそのままアクセスできるようオープンにして欲しい．
- 逆生成ツールのカスタマイズ方式をオープンにして欲しい．

要求の多くはユーザインタフェース（UI）に関するものである．しかし，時間の推移に沿って眺めると，ユーザの要求は個別の書式やツールの使い勝手等から，逆生成ツールのカスタマイズ方式，TENSE データベースのオープン化や製品管理とのリンクへと idt の特徴を活用する方向に順調に推移している．

UI に関する要求では３０件中１０件と書式に関する要求が多い．ソースプログラム以外の書式はともかく，ソースプログラムのコメント部分については書式を記述上の制限とみる観点が強い．ユーザの記述作業の負担を軽減するための構造エディタも提供している．しかし，制約のない自由な記述に対するユーザの欲求を充分満たしきれていないと考えられる．

対策としては，ドキュメントの書式を納得のゆく自由度の高いものとし，記述作業自体の機械化を促進する，書式をユーザ毎に定義可能とする等が考えられる．また，スマートなグラフィックユーザインタフェース（GUI）を持つ CASE ツールではユーザは記述上の制限を感じにくいことから，ドキュメントの表現主体を文字から図へと移行させて行く方向も考えられる．この方向はハードウェア環境の推移からも自然で，ユーザに記述上の制約を感じさせないで高度な機械化支援を行なう上で有望である．

6 まとめ

細粒度情報モデルに基づく TENSE OMS をカスタマイズして，C 言語向けプログラミング環境 idt を構築し，ソフトウェア開発現場で運用した経験について述べた．

TENSE OMS のねらいは細粒度情報に基づくデータ統合である．この統合によって，フェーズ，及び，モジュールにわたるドキュメントやソースプログラムの整合性を保証できる．具体的には上流フェーズで記述したドキュメントからの下流ドキュメントの部分生成や，ドキュメントの内容変更に伴う他ドキュメントへの自動反映，各種一覧表の自動生成による情報サービス等を実現する．

一方，開発現場では経験に基づく開発工程を統一して，ソフトウェアプロセス成熟度を向上させることによる生産性改善を図っていた．idt は上記の TENSE OMS の機能により，開発工程の統一に伴って発生する工程遵守のための負荷等偶発的な問題を解決し，工程統一の直接的効果を引き出して，開発工程の定着を促進した．

idt は１年前に運用を開始し，機能，性能の充実を計ってきた．現在，約５０名のユーザが使用しており，安定した稼働状態にある．また，idt の支援する開発工程も idt と共に複数の開発現場に展開しつつある．

課題としては，ユーザからの約１５件の未対応案件の解決がある．この案件は OMS の中心となるデータベースのオープン化や製品管理とのリンク等，OMS をさらに活用する方向の要求を含んでおり，細粒度情報に基づく idt が現場に定着し，順調に推移していることを示している．

また，これらの案件の多くは UI に関するもので，書式などの制約のない自由な記述への欲求が高い．しかし，文字による自由な記述と機械的な支援の間のギャップは大きい．ハードウェア環境の進展により直観に訴える図形への期待が高まっていることから，むしろ文字にとらわれず，図形で自由にソフトウェアに関する情報を記述，参照できる環境を提供することを検討している．

今後は idt の定量的な評価を試みるとともに，そのフィードバック，OMS のオープン化，GUI に基づくユーザインタフェース統合，オブジェクト指向ソフトウェアの開発支援，OMS を活用した部品化／再利用へと展開してゆく予定である．

[参考文献]

[1] W.S.Humphrey, Managing the Software Process, Addison Wesley, 1989　藤野喜一監訳, ソフトウェアプロセス成熟度の改善, 日科技連, 1991

[2] F.P.Brooks, No silber Bullet -Essence and Accidents of Software Engineering, IEEE Comp., pp.10-19, April 1987

[3] 西岡, 平田, 渡邊, TENSE OMS における細粒度情報モデル, 情報処理学会論文誌, Vol.34, No.3, pp.501-510 (1993)

[4] 西岡, 他 , 統合開発支援環境 TENSE のソフトウェアデータベース, 情報処理学会「アドバンスト・データベースシステム」シンポジウム '90 予稿集,pp33-42(1990)

[5] 有澤誠, ソフトウェア工学, 岩波書店,(1988)

[6] 西岡, 平田, 杉山, 統合化詳細設計ツール, 横河技報,Vol.36,No.2,pp.93-96,1992

付録：idtのドキュメント記述例

```
[1．プロセス名]
  member ： メンバ情報管理
[2．著者]
  桑原修二
[3．プロセス概要]
      1. メンバ情報の各種管理。
[4．モジュール構成]
  mmain ： メンバ情報処理プログラムメイン
  mlist ： メンバ情報の構造体のリニアリスト操作
[5．各モジュールの関数構成]
mmain ： メンバ情報処理プログラムメインモジュール
  [モジュール属性]
      [parts no.]    : K41-01
      [origin]       : *
  [概要]
      1. コマンドラインの引数のチェック
  [使用ヘッダファイル]
    [システムヘッダファイル]
      stdio.h  ： 標準入出力ヘッダ
    [共通ヘッダファイル]
      member.h ： マクロとstuct personの型定義。
  [関数構成]
      main     ： メイン関数
      getoption ： コマンドラインの引数のチェック
```

凡例
下線：自動生成
網掛：ｲﾝﾃﾞﾝﾄ記述

［モジュール構成設計の記述例（一部）］

```
[名前]
 main ： メイン関数
[プロセス呼び出し形式]
 member para1 [-para2]
 para1          ： 情報ファイル名
 para2          ： ディレクトリ名（省略時カレント）
[関数形式]
 int main(argc ,argv )
 int argc       ： コマンドラインの引数の個数。
 char *argv[]   ： コマンドラインの引数ポインタ。
[機能説明]
 1. コマンドラインの引数のチェック。
 2. メンバ情報処理関数のコール。
[注意事項]
 1. なし
```

［関数インタフェースの記述例］

```
// *PDL*****************************************
[名前]
  main ： メイン関数
[関数形式]
  argc: コマンドラインの引数の個数
  argv: コマンドラインの引数ポインタ
[機能説明]
  1. コマンドラインの引数のチェック。
  2. メンバ情報処理関数のコール。
[注意事項]
  1. なし
// *LDP*****************************************
コマンドラインの引数をチェックする。
switch( コマンドオプション )
    case( 'i' ):
        新規メンバのメンバ情報ファイルへの登録
    case( 'd' ):
        指定会員番号のメンバ情報の削除
ends
```

［pdlの記述例］

```
/*
*<<MAH>>*****************************************
* program name  : main
* description   : メイン関数
*  <function>
*    1. コマンドラインの引数のチェック。
*    2. メンバ情報処理関数のコール。
*  <usage>
*    member para1 [-para2]
*    para1     : 情報ファイル名
*    para2     : ディレクトリ名（省略時カレント）
*    (RET)     :
*  <remarks>
*    1. なし
*>>HAM<<*****************************************
*/
int main(argc ,argv )
int argc;      /* コマンドラインの引数の個数。*/
char *argv[]; /* コマンドラインの引数ポインタ。*/
{
/* [実行コード] */
    /*P* コマンドラインの引数をチェックする。
    /*P* switch(コマンドオプション)
    /*P* case('i'):
    /*P*      新規メンバのメンバ情報ファイルへの登録
    /*P* case('d'):
    /*P*      指定会員番号のメンバ情報の削除
    /*P* ends
     *P*/
        switch(/* コマンドオプション */) {
        case 'i':
            ユーザ記述コード
        case 'd':
            ユーザ記述コード
        }
}
```

［ソースプログラムの記述例（関数部分抜粋）］

プロセス一覧		
プロセス名/ライブラリー名	日本語名称	機能概要
member	メンバ情報管理	メンバ情報の各種管理
モジュール一覧		
プロセス名/ライブラリー名	モジュール名	概要説明
member	mmain	コマンドラインの引数のチェックと各メンバ情報処理関数のコール
	mlist	メンバ情報の構造体のリストの各種操作
	mfile	メンバ情報ファイルに対し、指定された操作を行う
	mstring	文字列の操作
	mmalloc	メンバ情報の構造体の記憶領域を確保する
関数一覧		
プロセス名/ライブラリー名	モジュール名（部品番号）	関数名
member	mmain (K41-01)	main getoption usage
	mlist (K41-02)	file_to_list end

詳細設計書の自動生成例