ソフトウェア工学 89−22
（1992. 12. 16）

概念データモデルと従属性制約を用いた言語による仕様再利用の実験

横田 和久　　橋本 正明　　佐藤 正和
ATR通信システム研究所

筆者らは，ソフトウェアの生産性を改善するため，手続き型言語のプログラムコードの再利用に代わる仕様の再利用のためのシステムを提案してきた．本稿では，そのシステムと，そのシステムを使って行なった実験の結果を述べる．再利用可能な仕様は，個々のプログラムではなく，一つの適用領域について，宣言的な仕様記述言語 PSDL で記述される．PSDL は概念データモデルと従属性制約に基づく言語で，階層的な概念スキーマも持っている．筆者らは，この機能を持つ SoftReuse システムと呼ぶ，プロトタイプシステムを構築し，実際の社会保険領域にこのシステムを適用し実験した．この実験により，仕様の再利用は，手続き型言語のコードの再利用より生産性が良いことを確認した．また，PSDL で記述された再利用可能な仕様と概念スキーマは，オブジェクト指向と機能分解アプローチの良いバランスを示した．さらにこの実験は，再利用可能な仕様の理解性が将来の重要な研究課題であることを明らかにした．

An Experiment on Reusing Program Specifications Described with Conceptual Data Model- and Dependency Constraint-based Language

Kazuhisa YOKOTA　　Masaaki HASHIMOTO　　Masakazu SATO

ATR Communication Systems Research Laboratories

2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan

The authors have already proposed a system for reusing program specifications instead of procedural program codes in order to improve software productivity, and this paper describes an experiment on the system. A volume of reusable specifications describes a domain but not individual programs with the declarative language PSDL based on the conceptual data model and the dependency constraints, and with hierarchically arranged conceptual schemata. Various program specifications are extracted from the reusable specification, and are transformed into C programs by the PSDL compiler. The authors implemented the prototype system, called the SoftReuse System, supporting those facilities, and tested the system with an actual social insurance domain example. The experiment estimated that specification reuse was more productive than procedural code reuse. The reusable specification described with PSDL and conceptual schemata showed a good balance between the object-oriented and functional decomposition approaches. The experiment also pointed out that the comprehensibility of the reusable specification was one of the most important issues to be studied in the future.

1 はじめに

ソフトウェアの生産性と信頼性は現在のソフトウェア工学の主たる問題であり，それらの改善にソフトウェアの再利用は有効である [1]. COBOL のような手続き型言語のプログラムコードの再利用は確かにそれらを改善してきた. 最近，ソフトウェア開発のための知識，たとえばドメイン知識や開発の経験，仕様などをプログラムコードの代わりに再利用する研究が増加している [2]. たとえば，オブジェクト指向で書かれたプログラムコードの再利用は，手続き型言語のコードの再利用より生産性が良いことが報告されている [3]. 筆者らはプログラムの仕様の再利用を研究してきており，PSDL (Program Specification Description Language) と PSDL コンパイラ，仕様再利用の支援系を含む SoftReuse システムを提案した.

通常，仕様を理解した後，新しい要求に合わせて仕様を変更するので，再利用可能な仕様の理解性と拡張性は非常に重要である. そのため，PSDL には概念データモデルである ER(Entity-Relationship) モデルを適用した. 仕様の再利用の支援系は，販売在庫管理や社会保険のような対象領域を PSDL で記述した再利用可能な仕様を蓄積し，その領域のさまざまなプログラム仕様をその再利用可能な仕様から抽出する. 抽出されたプログラム仕様は PSDL コンパイラによって C プログラムへ変換される.

大規模なソフトウェアの再利用の実験は，手続き型コードの再利用と違って報告例が少ない. そこで，筆者らは実際の社会保険の対象領域で SoftReuse システムを使った実験を行なった. この論文の目的はこの実験の結果と考察を述べることである. SoftReuse システムについては第 2 章で説明する. 実験と結果については第 3 章で述べる. 第 4 章では考察を述べる.

2 SoftReuse システム

この章では PSDL と再利用可能な領域の仕様，プログラム仕様抽出について述べる.

2.1 PSDL

PSDL プログラム仕様は，情報層，データ層，アクセス層の 3 つの層から成り立っている. それぞれの層は図 1の上，中，下に対応している. この図は販売管理プログラムの例を示している.

アクセス層では，product_price_F や sale_F のような入出力ファイルへのアクセス方法を指定する. データ層では，product_price_data や sale_data のような入出力データの構造を指定する.

情報層では対象世界の情報構造を記述する. PRODUCT や SALE のような実体型は，対象世界の「もの」や「こと」を表す実体の集合である. SOLD や BUY のような関連型は，実体を相互に対応づけた関連の集合である. 個々の実体の性質は，属性値の組で表す. 実体型や関連型は，図 1の中では太い線の長方形や菱形で表されている.

プログラム中の計算は以下に述べる従属性制約で記述する. 属性値従属性制約は，関連で関係づけられた実体から属性値を得る. この制約は図 1では矢印で示されている. 関連存在従属性制約は，関連によって関係づけられなければならない実体から関連を得る. 実体存在従属性制約は，他の実体から新たな実体を導く. 同時に，それらの実体を相互に対応づける関連も得られる.

これらの制約は計算式と条件式で記述される. たとえば，属性値 SALE.amount と CUSTOMER.total は，図 1の中の矢印で示される計算式によって計算される.

2.2 再利用可能な仕様

前節では，PSDL の情報層がプログラムの情報構造を記述することを述べた. ところが，対象領域の情報構造とプログラムの情報構造には違いがないため，情報層は対象領域を記述することもできる. SoftReuse システムは，情報層の記述を再利用可能な仕様として蓄積する. もちろん，典型的なデータ層とアクセス層の記述も一緒に蓄積する. この再利用可能な仕様をモデルシステム仕様と呼ぶ. 図 2中の実体と関連，属性値，制約ウィンドウは，在庫管理プログラム，売上報告プログラム，請求書作成プログラムからなる販売在庫管理を対象領域としたモデルシステム仕様の例である.

対象領域のモデルシステム仕様は非常に量がおおくなるので，仕様全体を理解するのは難しい. そこで，モデルシステム仕様の理解性を向上させるため，階層的な概念スキーマを導入した. 概念スキーマはオブジェクト指向分析 [8] におけるサブジェクトに似た役割を果たす. 各々のスキーマは，実体型や属性値，関連型，制約を含みさらに compose-of の関係にある他のスキーマも含むことができる. さらに仕様の選択肢は概念スキーマ間の

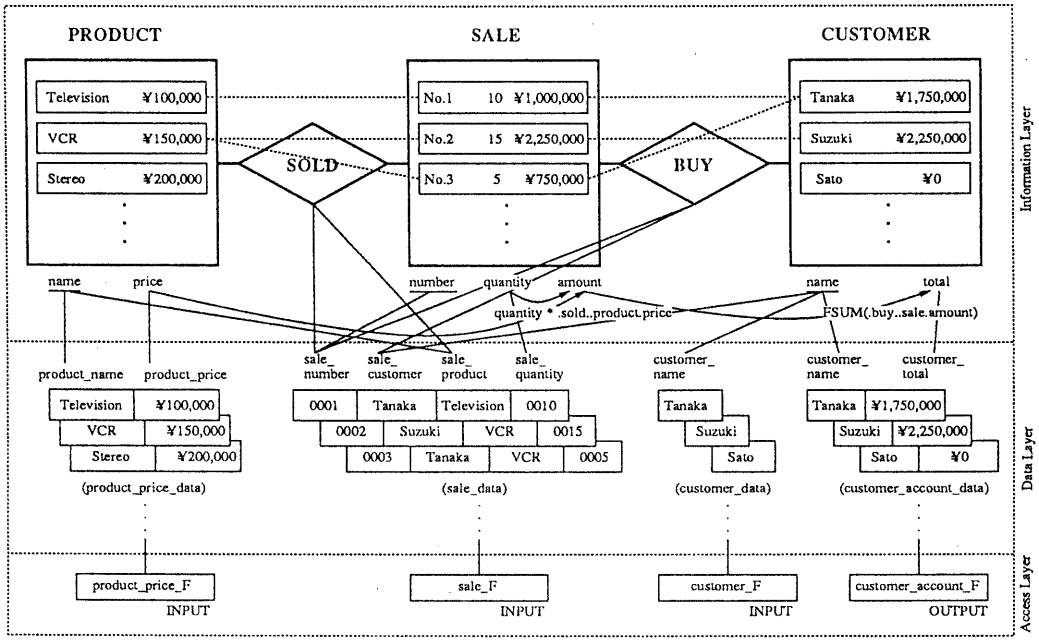

図 1: PSDL プログラム仕様

similar-to の関係によって管理される.

図 2で示された販売在庫管理領域の概念スキーマが図3である. 社会保険の対象領域の例を使った実験では, 概念スキーマの階層は, サブシステム, 業務処理単位, 業務単位中の計算処理, の 3 階層であった.

対象領域の分析後, モデルシステム仕様を定義するために設計者はスキーマウィンドウ中のスキーマの名前や階層を決めなければならない. それから, 設計者はあるスキーマのために開かれた ER ウィンドウ中に実体型や関連型を書き, 属性や制約などを書き加える. これらの記述はそのスキーマに含まれる. 既存の記述があれば, マウスで選択することによって特定のスキーマに含められる.

2.3 仕様抽出

設計者はまず始めに, ドキュメントを使い業務の概要を学び, SoftReuse システムのウィンドウを通して概念スキーマの階層をたどることにより, モデルシステム仕様全体を理解する. そして, 設計者は仕様を要求に合わせるために, 実体型や制約などを変更する. その後, プログラム仕様を以下の操作により抽出する.

始めに, プログラムが必要とする入出力ファイルをマウスで指定する. すると, 入力ファイルから出力ファイルへ至る従属性制約のパスが自動的に見つけられ, そのパス上にある仕様の記述が図 2にあるように反転した色で示される. これらの仕様は, その入出力ファイルを処理するプログラムの仕様を構成する. さらに, このパス検出アルゴリズムは, 出力ファイルは正しく指定されているが, 入力ファイルには間違いがありうることを仮定しているそのため, 最初は出力ファイルへ至る全てのパスが検出される. これらのパスの中には必要のない入力ファイルへつながる冗長なパスが存在しうる. さらに, 同じ入出力ファイル間でさえ, 複数のパスが反転色で示され, 冗長なパスが生じる.

そこで, SoftReuse システムは, 冗長なパスを, それが接続している点の色を薄い反転色にすることにより警告する. 接続点の冗長な従属性制約は, 色をマウスにより反転することで除去できる. なお, 一度に冗長性のない仕様を得ることはできないので, 上記の作業を繰り返すことが必要である. たとえば, 図 2中の反転色は,

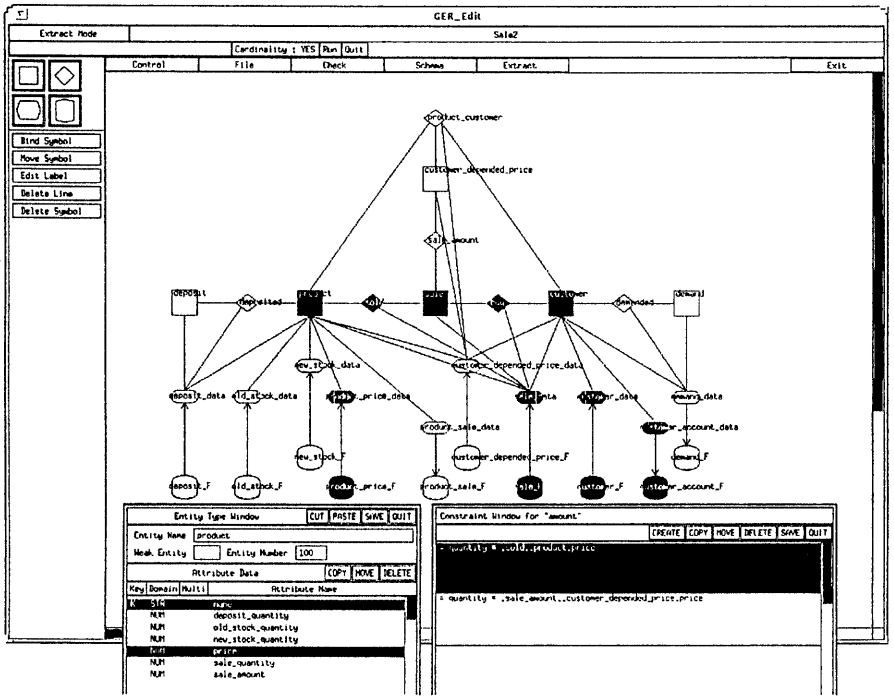

図 2: ER，属性値，制約ウィンドウ

図1中の売上報告プログラムを示す．反転色の部分は自動的にプログラム仕様として抽出される．プログラム仕様はSoftReuse システム上でさらに変更可能である．

もう一つのプログラム仕様の抽出の仕方は，マウスで，プログラムに対応したスキーマを選ぶことで行なえる．最後に，そのプログラム仕様はPSDL コンパイラにより，自動的にＣプログラムへ変換される．

3　実験

この章では，実験の手順と実験結果，被験者の感想を述べる．

3.1　実験の手順

実際の社会保険の領域を例として選んだ．社会保険システムの主要な部分はマスタファイルを処理する，1) 加入者の異動記録と保険料の計算，2) 受給資格の裁定，3) 受給額の調整，4) 支払の４つのプログラムからできている．これらのプログラムの仕様は，PSDL で別々に記述し，モデルシステム仕様に統合した．SoftReuse システムの完成後，この仕様をシステムに入力し，階層的概念スキーマを構築した．同時に，社会保険を説明する業務概要書も作成した．

モデルシステム仕様からプログラム仕様を得るのに，以下の３種類の抽出の要求を設定した．1) 既存のものと同じプログラム仕様，2) 既存のプログラム仕様を分割したり統合したプログラム仕様，3) 受給額の計算方法や支払の頻度のような仕様の細部を拡張した仕様．

被験者には，PSDL を知らず，社会保険システムの細部に詳しくない人を二人選んだ．一人は 20 年，もう一人は３年の COBOL の経験がある．

これらの準備の後，以下の仕様再利用の実験を行なった．

(1)PSDL の学習

被験者は，PSDL 言語の仕様書を使い，PSDL コンパイラの開発者の助けを借りて PSDL を習得した．

(2) 業務の理解

被験者は，社会保険の規定を業務概要書から学んだ．もちろん，彼らは業務概要書の作成者の助けも借りた．

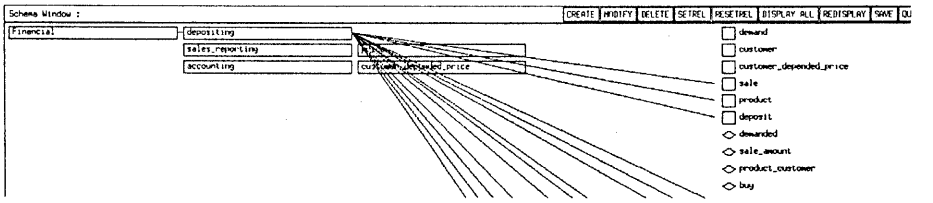

図 3: 概念スキーマウィンドウ

(3) モデルシステム仕様の理解

被験者は，モデルシステム仕様を SoftReuse システムと PSDL テキストを使って理解した．

(4) プログラム仕様の抽出

被験者は，さまざまなプログラム仕様を上記の 3 種類の要求にしたがって抽出した．2.3節で述べた 2 種類の抽出法が利用可能であるが，ここでは入出力ファイルを指定する方法を適用した．抽出されたプログラム仕様は，PSDL コンパイラがまだ完成していないため，他の PSDL 研究者によって検査された．

3.2 実験データ

以下のデータは実験により得られた．

(1) モデルシステム仕様の大きさ

実験で使われたモデルシステム仕様は PSDL で 2,020 行であった．一方，前章で触れたモデルシステム仕様の元になった 4 つのプログラム仕様はそれぞれ 1,200, 340, 490, 450 行であった．

(2) 工数

それぞれの再利用過程では図 4で示される時間が必要であった．

(3) 抽出された仕様の大きさ

プログラム仕様抽出法で用いられた方法は 2.3節で述べたように冗長性のある仕様を抽出する．およそ 5 回の繰り返しによりそれぞれのプログラムについて正しいプログラム仕様を得た．最初に抽出された仕様と正しい仕様の情報層の行数の比を図 5に示す．最初に抽出された仕様の行数は正しい仕様の行数の約 1.8 倍になっている．

3.3 被験者の感想

理解性のような人間の要因を考察するには実験データの他に，被験者の感想は重要なので，被験者の感想を以下に述べる．

(1)PSDL の学習

単純な宣言型の文法を持つ PSDL は理解しやすかった．しかし，被験者は COBOL のような手続き型言語に慣れていたため，彼らの思考方法を変える必要があった．被験者はどのように実体型と関連型を使うのかがわからなかった．

(2) 業務の理解

業務概要書は簡単に理解でき，モデルシステム仕様の理解に役立った．しかし，以下のギャップが問題になった．業務概要書に書かれていた情報処理の目的は二つ以上の処理方法を持ちうるが，モデルシステム仕様には一つの方法しか書いていない．さらに，業務概要書は概要を書いているが，モデルシステム仕様は常識的なことを含み詳細に書いている．このため，業務概要書とモデルシステム仕様を関係付けるのは容易でない側面もあった．

(3) モデルシステム仕様の理解

モデルシステム仕様の実体型と関連型の記述は，設計者の思考に依存する．そのため，一般性に欠け，定義するのも難しい．さらに，たくさんの属性値を持つ実体型は，たくさんの制約と関連型を持つため，簡単に理解するには記述量が多過ぎる．

関連型の場合，名前や関連数のような少数の情報しか持たないため，その意味を読みとるのは難しい．複数の関連型の関係を理解するのは特に難しい．被験者は，関連型に 2 種類あるように感じている．一つは対象世界の情報構造を表現するのに不可欠なものである．もう一つは処理の方法を表現するために用いられるものである．それらは同じ記述方法をとっているが，ER ウィンドウ上では区別するのがよい．

従属性制約は宣言的に記述されている．しかし，仕様の意味を把握するには，データの流れを追ってそれらを

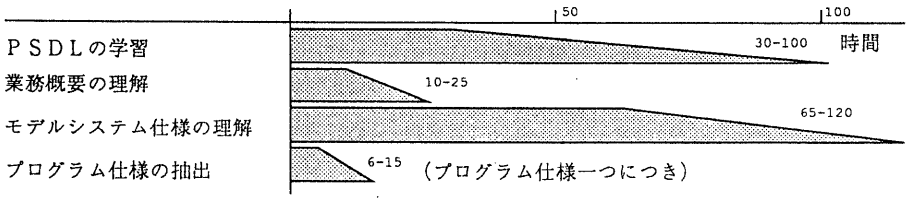

図 4: 工数

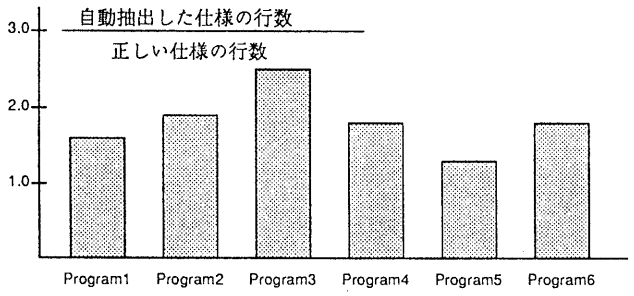

図 5: 情報層の仕様の大きさの比

手続き的に理解することが必要である.

各々の概念スキーマは違った大きさを持っており，大きなスキーマではどこから理解し始めたら良いかすぐにはわからない.

(4) プログラム仕様の抽出

2.3節で触れた，冗長なパスを削除するためには，モデルシステム仕様は正しいと仮定されているので，制約中の計算式や条件式の内容より従属性制約で作られたデータの流れのパスのほうが有用であった. 冗長な関連型を削除するには，関連型の名前が非常に重要であった. そのため，関連型の名前は処理内容を表すように正確に定義されなければならない.

カスタマイズにおいては，関連型を追加する方が実体型を追加するよりも難しかった. 3.1節で触れたように，3種類の抽出法を試したが，被験者はそれらの間に有意な差を見い出さなかった. もちろん，抽出されたプログラム仕様を検証するための解析方法を今後開発することが必要である.

4 考察

この章では実験結果について考察する.

4.1 生産性

実験工数の制限によって実験は仕様の再利用についてのみ実施し，プログラムコードの再利用については実施しなかったので，実験データからは生産性の向上を読みとることはできない. このため，以下のように生産性を見積もった.

再利用は以下の3つの部分に分けることができる. 1) 再利用する仕様やプログラムを理解する. 2) それらの拡張部分を設計する. 3) それらを書き換える. そこで，PSDL と COBOL との比較を行なった. PSDL は宣言型言語であるため，手続きのコントロールフローや中間データを保持する変数を考える必要がない. さらに，PSDL は概念的に記述されるので思考過程の個人差が COBOL よりも少ない. これらの特徴を考慮して，PSDL の生産性は COBOL よりも上記の 1),2),3) の部分に関して，6,3,2 倍，高いと推測した. また，それぞれの部分は全体の 50%，30%，20% の工数を必要とすると推測した. これらの推測は，被験者の COBOL の経験と，30,000 行以上におよぶ PSDL コンパイラの自己記述の経験に基づいている. これらの推測から，PSDL プログラム仕様の再利

用はCOBOLコードの再利用より約4倍生産性が高いと見積もった.

4.2 理解性

生産性は改善されたが，モデルシステム仕様の理解には図4に示すように多くの時間がかかった．このことについて以下に述べる.

(1) 処理の目的と理由

この実験では，処理の方法を記述したモデルシステム仕様を補うのに処理の目的や理由を記した業務概要書が必要だった．しかし，これらの間のギャップは3.3節(2)で指摘した.

当面は業務概要を概念スキーマのコメントに書くことや，モデルシステム仕様上に業務概要のハイパーテキストを構築することが考えられる．将来的には業務の目的や理由も表せるモデルを構築することが必要である.

(2) 宣言的仕様の手続き的理解

仕様の宣言的な記述は，再利用の際の切り貼りに不可欠である．一方，仕様の意味の把握や検証には，データの流れに沿った仕様の手続き的な理解も必要である.

当面の解決法は，自動的にデータの流れを追うことにより手続き的な理解を支援すべきである．また，将来の研究では，宣言的記述と手続き的理解との関係を明確にする必要がある.

(3) 概念とインスタンス

PSDLの情報層は，対象世界の概念を，従属性制約と実体型と関連型だけで記述する．しかし，実体や関連のインスタンスは記述していない．しかし，被験者は再利用のとき，はじめに典型的な概念のインスタンスを想像し，要求に合うようにインスタンスを変更する．そして，変更されたインスタンスを参照して概念を書き換えた.

関連を思い浮かべることは実体より難しいので，当面の解決策では，関連型のカーディナリティから自動的に典型的な関連の図を生成することが考えられる．将来の研究では，概念とインスタンスの関係を明確にする必要がある.

(4) 概念スキーマ

モデルシステム仕様は，関連型を通じてあらゆる方向に伸びているので，それを理解することは難しい．ただし，階層的に構成された概念スキーマは，モデルシステムの理解に役立った.

理解性を改善するには，概念スキーマ間のデータの流れをデータ・フロー・ダイヤグラムのように見せることである．というのは，階層的スキーマはデータ・フロー・ダイヤグラム中の機能分割を表すことができるからである．将来的には，スキーマの意味は(1)などとの関連において明確にしなければならない.

4.3 拡張性

カスタマイズを伴う場合と伴わない場合について，二つのプログラム仕様抽出を3.1節で述べたように実験した．両方の抽出についてかかった時間は図4に示すように非常に少なく，大きな差は見られなかった．被験者も3.3節(4)で述べたように，大きな違いを感じなかった．これは，PSDLがカスタマイズに要求される拡張性を持っていることを意味する.

4.4 仕様の抽出

2.3節で述べたように，実験で使ったプログラム仕様抽出法は，最初は冗長な仕様を抽出する．図5で示されるように，実験では正しい仕様に比べて1.8倍大きなプログラム仕様が最初に抽出された.

問題は，大きな冗長性を持った仕様が抽出されることである．そこで，元来不要な入力ファイルにつながるデータの流れのパスは，自動的に取り除くことが考えられる．さらに研究しなければならない課題は，同じ入力ファイルと同じ出力ファイルの間の複数のデータの流れのパスを取り除くための知識やアルゴリズムを見つけることである.

4.5 方法論

SoftReuseシステムにおいては，ある対象領域の情報構造を記述するモデルシステム仕様には，オブジェクト指向アプローチ[8]が適用されている．一方，階層的に構成された概念スキーマには，機能分解アプローチ[9]が適用されている.

後者のアプローチでは，大規模なソフトウェアを扱うために「分割統治」法が使われる．しかし，分割されたソフトウェアのモジュールには，無視できない程度の重複が含まれる．たとえば，3.2節(1)では23％の重複がある．別人によって設計された各々のモジュールは，重複部分が同じ機能を持つにも関わらず，違う設計が行

なわれる傾向がある．それゆえ，この少なからぬ重複部分が長大なソフトウェアを生み出す一つの原因になっている．

一方，前者の対象世界の情報構造を記述するアプローチには，この重複の問題は無い．しかし，大規模な対象領域を扱うと理解性の問題が生じるが，それは分割アプローチを適用した階層的な概念スキーマで解決される．

これは，両方のアプローチが必要なことを意味しており，SoftReuse システムにおいては両者の調和がとれている．この調和は，異なるスキーマが PSDL の情報層の同じ部分を共有することによっていた．このため，両者のアプローチの調和の視点から，SoftReuse システムを用いたソフトウェア開発法は研究されるべきである．その他に，実体型と関連型を定義する方法や仕様の解析法，仕様の統合法についても研究しなければならない．

5 まとめ

この実験により，仕様の再利用は手続き的コードの再利用より生産性が高いと見積もることができた．PSDL が ER モデルと従属性制約とに基づいた宣言的言語であることが生産性につながっている．PSDL と階層的概念スキーマで記述された，再利用可能なモデルシステム仕様は，オブジェクト指向と機能分解アプローチのよいバランスを示していて，この両者は大規模なソフトウェアの開発には不可欠である．

この実験は，大規模な再利用可能な仕様の理解性は，今後研究すべき重要な問題であることを明らかにした．宣言的な記述の手続き的理解と，型の記述から推測するインスタンスのような問題は，認知科学の問題に近い．さらに，抽出されたプログラム仕様の冗長性を取り除く研究も必要である．また，再利用可能な仕様を用いたソフトウェア開発方法論も研究しなければならない．この方法論はモデルシステム仕様の構築法や仕様の解析法，仕様のカスタマイズ法を含むものになる．

謝辞

日頃御指導頂く葉原会長，寺島社長，太田室長に感謝いたします．また，SoftReuse システムの作成と実験にご協力いただいた日本電子計算株式会社の諸氏に感謝いたします．

参考文献

[1] F.P. Brooks, No Silver Bullet: Essence and Accidents of Software Engineering, *Computer*, Vol. 20, No. 4 pp. 10-19, 1987.

[2] T.J. Biggerstaff, A.J. Perlis (Eds), *Software Reusability*, Vol. I, ACM Press, 1989.

[3] J.A. Lewis, S.M. Henry, D.G. Kafura, R.S. Schulman, An Empirical Study of the Object-Oriented Paradigm and Software Reuse, *Conf. Proc. of Object-Oriented Programming Systems, Language, and Applications*, pp. 184-196, 1991.

[4] K. Okamoto, M. Hashimoto, On Real-Time Software Specification Description with a Conceptual Data Model-Based Language, *Proc. of Int. Conf. on Computing and Information*, pp. 186-190. 1990.

[5] M. Hashimoto, K. Okamoto, A Set and Mapping-based Detection and Solution Method for Structure Clash between Program Input and Output Data, *Proc. of IEEE Computer Software and Applications Conf.*, pp. 629-638, 1990.

[6] K. Okamoto, M. Hashimoto, Program Generation from Real-Time Software Specification Described with a Conceptual Data Model-based Language, *Proc. of Int. Conf. on Software engineering and knowledge engineering*, pp. 261-270, 1991.

[7] K. Okamoto, M. Hashimoto, Visual Programming with Reusable Specification Described by a Conceptual Data Model- and Constraint-Based Language, *Proc. of Int. Conf. on Human-Computer Interaction*, Vol. 1, pp. 587-591, 1991.

[8] P. Coad, E. Yourdon, *Object-Oriented Analysis*, Prentice-Hall, 1991.

[9] E. Yourdon, L. Constantine, *Structured Design*, Yourdon Press, 1978.