ソフトウェア工学 42－3
（１９８５・６・２６）

設計用言語（ＳＬ）の言語仕様

稲田 満　岡本 務　渡辺 敏　中村 雄三
（ＮＴＴ　ソフトウェア生産技術研究所）

１．まえがき

プログラムの設計仕様を入力しＣＯＢＯＬプログラムを自動生成する設計用言語ＳＬ(Superb Data Oriented Language) の言語仕様について報告する。

プログラムジェネレータは①ジェネレータの入力自体にドキュメント性がある、②記述量が少ない、③処理系による記述内容の検証能力を充実できる、から飛躍的な生産性向上効果が期待できる。しかし従来のジェネレータは特定利用向けの専用性が強く融通性に乏しい、大規模プログラムの開発に必要な分業化した記述への配慮に乏しい、等の問題があり多くの工数を必要としている大規模プログラムの開発には適用がままならないのが現状である。専用ジェネレータを利用目的別に開発することはジェネレータの開発費用も含めたトータルの生産性を考えれば得策ではなく適用性の広いジェネレータの開発が現実的である。そこで我々は以下の特徴を持つ設計用言語の開発を進めた。

(1) 適用性向上のための特徴

(A) 設計仕様の記述に良構造プログラムを設計することを目的としたデータ構造設計法[1] をベースにしたプログラム仕様の記述法を用いているため特定問題向きのジェネレータに比べ汎用性がある。

(B) データ構造の表現には処理構造を示すデータ構造に加えアクセスパス等を示すデータ構造を新に導入したデータ構造の階層的表現方式[2],[3] を持つ。これにより木構造データベースを含む幅広いデータ構造の設計を可能にした。

(C) 入出力等のプログラムの走行環境に依存する機能、問題向き関数等の専用性の高い記述については、その実現法である目的プログラムの設計記述を利用者側で行なうことのできる目的プログラムの部品化機構[4],[5] を持つ。

(2) 大規模プログラムへの適用のための特徴

プログラムの設計仕様を大規模プログラムの共通設計者が先行設計する①データフォーマット等の共通仕様、入出力等の実現法を示す②実現仕様、及び個別の業務プログラム設計者が設計する③処理仕様、の３階層に分け分業化した記述を可能にする仕様書体系をもつ。

上記(1) の(A) を入力としてプログラムの自動生成を試みた類似の例としてMODEL[6]が報告されているが、その他の特徴を併せ持つ例はまだない。本報告ではＳＬの言語仕様の概要、特徴、評価について述べる。

２．設計法の背景と自動化範囲

２．１　データ構造設計法を選んだ背景

ＳＬの入力となる設計仕様の記述の考え方を与えるプログラム設計法としてはデータ構造設計法をベースとして発展させたものを使用している。以下にベースとしてデータ構造設計法を選定した理由を述べる。

(1) 現存するプログラム設計法の中で最も均質な設計結果が得られる設計法として普及している。またプログラミング言語を用いたプログラムの良構造化のための設計法のため馴染みが良い。

(2) データ構造設計法は事務処理向きではあるが、データ構造そのものは抽象概念であり、例えば特定の在庫管理とか、給与計算といった問題を直接記述するものではなく、汎用的な設計概念である。

(3) 大規模システムに利用されている木構造型CODASYL タイプデータベースのデータ構造とデータ構造設計法には整合性がある。

(4) 手書きプログラムのための設計法であり、自動化しても人手で作成したプログラムと比べて遜色のない能率のプログラムを生成する処理系の実現が期待できる。

２．２　設計手順と自動化範囲

本節ではデータ構造設計法の代表例であるジャクソン法[7] を例にとってその設計手順と生産物の関係及びＳＬでの自動化範囲についての概要を述べる。

ジャクソン法における設計手順と生産物の関係を表２．２－１に示す。ＳＬは表の(1) と(3) の生産物（但し、ＳＬでの表現法は異なる）から(2) と(4) の作業を自動化し(4) の生産物であるＣＯＢＯＬプログラムを自動生成する。データ構造はプログラムで扱うデータの集まり（一般にはファイル）からなり、そのデータ（一般にはレコード）を単位とした処理順序を表わしている。通常、一つのプログラムで複数個のデータ構造を扱うため、データ構造とプログラム構造とは異なる。このことから(2) を自動化することはいくつかのデータ構造を扱うプログラムの全体のロジック設計を自動化していることになる。(4) の作業は、必要な位置でかつデータの参

照関係から矛盾のない位置であることを確認しながら演算をプログラムに割りつけるコーディング作業であるが、ＳＬではデータの参照関係解析を行なうことにより自動化を図っている。

上述から、ＳＬ処理系への入力は設計過程での生産物でありプログラムではない。このことからＳＬでは処理系への入力ソースのことを仕様書と呼んでいる。

表２．２－１　データ構造設計法（ジャクソン法）の概要

設計作業	生産物	ＳＬ
（１）データ構造の設計	データ構造図	
（２）プログラム構造への変換	フローチャートまたは疑似コーディングによる概略制御フロー（プログラム構造）	○
（３）演算リストのリストアップ	データ項目への代入文，チェック文、入出力命令文等の演算リスト	
（４）プログラム構造への演算リストの割り付け	プログラム	○

備考　○：ＳＬでは自動化

３．仕様書体系の特徴

３．１　仕様体系の考え方

ＳＬの仕様書記述の体系は、大規模プログラムに適用したときの設計、記述の分業化を狙いとして、以下に示す３つの階層に分けて記述する方式とした。

（１）処理仕様層

個別の業務仕様を単位（ＣＯＢＯＬプログラムのコンパイル単位に相当）として、業務処理を表現するためのデータ構造（論理データ構造という）と演算リストを記述する層である。この層を記述する仕様書をプログラム仕様書という。

（２）共通仕様層

プログラム仕様仕様書間で共通に参照する仕様を記述するための層である。この層では、論理データ構造を設計する上での設計条件を与えるためにＳＬで新に導入したデータ構造（物理データ構造という）、フォーマット仕様、データ属性の定義等を記述する。この層を記述する仕様書を共通仕様書という。

（３）実現仕様層

演算リストの設計のうち、適用システム毎に異なる入出力処理、問題向き関数、例えば利息計算、預け入れ日数計算等、の処理を目的プログラム言語（COBOL ）で設計する。換言すればこの層はＳＬのコード生成部の適用システム向きチューニングのための仕様を記述する層である。この層を記述する仕様書を部品仕様書（単に部品ともいう）という。プログラム仕様書から自動生成するとき自動生成コードと部品に記述されたコードとを合成して完成したプログラムを生成する。

処理系では上記３つの層の仕様を入力してプログラムを生成する。本記述方式の特徴を以下に述べる。

（1）大規模プログラムの効率的開発に必要な分業化した記述のできる構成である。共通仕様層と処理仕様層はシステム共通設計者が先行設計して登録する。これを処理仕様層を設計する個々の業務プログラム設計者が参照する。共通仕様層についても登録時に内容妥当性の検証を行なうため、単なるマクロ機能ではない。

（2）適用性の高い記述方式である。実現仕様層は利用者が直接ＣＯＢＯＬの文を記述できる。このため、従来の専用ジェネレータの欠点であった目的プログラムのコードの固さの問題が解決した。

（3）ＳＬで記述したプログラムの再利用性が高い。実現仕様層のＣＯＢＯＬの文のみの差異であればそのまま処理仕様層の再利用ができる。処理仕様層での記述内容を変更して再利用するときにも情報隠蔽効果によって読解性が高いため修正が容易である。

３．２　仕様書の種類と参照関係

３．１で述べた考え方に基づいて、記述目的別に分類した表３．２－１に示す仕様体系を設けた。図３．２－１に各仕様書間の参照関係を示す。図３．２－１に示す仕様書で間接参照されるものについては詳細隠蔽効果が向上する。例えば、プログラム仕様書から入出力用の部品仕様書を参照するときデータ仕様書からの間接参照とするとプログラム仕様書

表３．２－１　仕様書体系

仕様書名		記述内容
プログラム仕様書		各プログラム対応の記述をする。 論理データ構造、演算リストを 主に記述する。
共通仕様書		プログラム中で規定する情報の中でプログラム間で一元的に管理すべきものをまとめた仕様書の総称。
	データ仕様書	物理データ構造、フォーマットを規定する。 データの使用目的に分けて複数の仕様書を作成することができる。
	データ属性仕様書	数え上げ属性を定義する。
	インタフェース仕様書	サブルーチンコールのパラメータ、生成プログラムの名前を記述する。呼び出し側、被呼び出し側共同一仕様書が参照できインタフェースミスを防止できる。
部品仕様書		適用システムごとにＳＬのコード生成部をチューニングするための仕様書。データベース、ファイル等へのアクセス処理、リカバリ対策等の特殊処理、 問題向き関数、等のための演算リストを記述する。

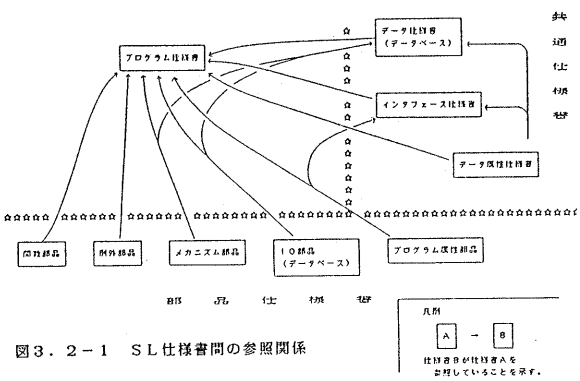


図３．２－１　ＳＬ仕様書間の参照関係

の設計者は部品利用に関する記述が不要になり専用ジェネレータと同等の効果が得られる。共通仕様書に記述する内容は小規模利用のときのためにプログラム仕様書に直接記述することもできる。

４．データ構造と演算リストの記述機能

ＳＬの設計ではベースとして利用したデータ構造設計法を基にして大規模バンキングシステムのアプリケーションプログラムの記述実験からの反映、処理系からの実現性検証を進め、設計法及びその生産物を記述するための言語仕様を発展させた。

その結果①データ構造に処理構造だけでなくデータ構造のアクセスパス等の仕様を説明する物理データ構造の概念を導入しデータベースを含む幅広い入出力の実現法をカバーする統一的なデータ構造設計法式を持つ、②演算リストの記述では処理仕様層には非手順型の記述、適用システム向きの入出力等の実現仕様層には目的プログラム言語であるＣＯＢＯＬの文が記述できる方式とし高生産性、高適用性が得られる、を特徴とする設計用言語方式を確立させた。

以下にＳＬを用いたデータ構造の記述、演算リストの記述についての主要機能の概要と特徴及び各仕様書との関係について述べる。

４．１　データ構造記述機能

ＳＬにおけるデータ構造の記法一覧を表４．１－１に示す。以下にデータ構造の記述についての特徴的な事項を述べる。

４．１．１　データ構造表現能力の向上

(1) 表記法の改善

データ構造の表現法に関してはジャクソン法の記法（ジャクソン木といわれる）が著名でありＳＬの表現法（但し、等価なテキスト記述）として利用できるかを考察した。ジャクソン木の構成はその基本単位となるコンポーネントとその処理順序を表わす３つの構造記号（順列、繰り返し、選択）から構成される。ジャクソン木の表記上の問題点としては１つのコンポーネントの意味がそのコンポーネントに付けられた構造記号だけではなく上下の階層関係を見て初めて明確になることがあり直感性に欠ける点がある。ＳＬでは処理対象となる実体を（ノードまたはリーフ）を①構成要素、それらの処理順序を表わすノードを②構造、として明確に分離して記述することにより木構造上のノード、リーフの役割が直感的に分かるようにした。

(2) 記述能力の向上

(A) 実体の階層

ジャクソン木はノードに実体を持つ表現ができない。しかし、データベースのデータ構造はノードに実体を持つことのできる実体の階層化が図られている。このためＳＬではノードにも実体を記述できるようにした。実体の階層はアクセスパスを表わし下位の実体をアクセスするには上位の実体のアクセスが必要であることを表わす。

(B) 同列構造

乱アクセスの可能なファイルを扱うとき、複数のレコードを同時にアクセスし相互に参照したいことがある。この処理はジャクソン法の３つの構造記号だけでは書けない。そこで複数の実体を同時または任意順序でアクセスし処理のできることを示す同列構造を設けた。

(C) プロセス構造

一度出力したファイルを入力モードに切り換えて読む処理等、１つのファイルについての多段の処理をプログラムを分けることなくできるようにするための構造である。ジャクソン法ではこのようなワークファイルが必要な問題の一つであるストラクチャクラッシュを解決する方法は述べられているがストラクチャクラッシュではなく本質的（あるいは業務の必要性から）多段の処理が必要なときのジャクソン木の書き方は述べられていない。ＳＬでは新たなプロセスという構造を設けることで解決した。

４．１．２　データ構造の階層化

ＳＬのデータ構造記述で最も特徴的な論理データ構造、物理データ構造及び両者の対応付けを行なう抽出文についてその背景、機能概要と特徴について述べる。

(1) 背景

データ構造（ジャクソン木）は抽象概念でありその実現法を問うものではないが以下説明の便宜上一つのデータ構造はファイルと対応し、その構成要素はレコードと対応していると仮定して説明する。ジャクソン木はファイルを単位としてそのレコードの処理内容による分類と処理順序を表わしており部分

的なプログラムの構造である。一方、データ構造の構造にはプログラムの処理とは無関係な入れ物としての構造（例えば逐次アクセスファイルであればレコードの並びだけの構造）がある。ジャクソン法では入れ物の構造が逐次アクセスファイルのみを仮定しておりその範囲で書ける部分のみジャクソン木を書かせ、それ以外の入れ物を扱うときは演算リストの設計の問題として片付けられている。

一方、我々が適用対象としているプログラムはではデータベース等の入れ物の構造が複雑なものとか、キー付きファイル等の乱アクセスのできるファイルを主体に扱っており、演算リストの問題としてしまうとデータ構造設計自体の意味が薄くなる。そこで我々はデータ構造の設計を処理の立場からみたデータ構造（論理データ構造）と入れ物の仕様としてのデータ構造（物理データ構造）の２階層に分けるとともにその対応をとる抽出文を設けることで解決した。

(2) 論理データ構造の記述

プログラム仕様書には一つ以上の論理データ構造が記述される。ジャクソン法では入力、出力等のデータ構造の用途を陽に示していないが、ＳＬでは入力、出力、更新等の用途に分類して記述する。この情報はプログラム生成上の情報になるとともに演算リストでのデータの参照関係の誤りチェックにも用いる。

ジャクソン法では入力と出力のデータ構造上の繰り返しデータ間の対応をとって両者のデータと処理の同期関係（ジャクソン法では対応関係という）を示す設計過程がある。ＳＬではこの情報を出力または更新用途のデータ構造の繰り返し構造（情報消費側）に同期対象となる相手（情報生成側）繰り返し構造名を指定することで対応付けを行なう。

(3) 物理データ構造の記述

物理データ構造は論理データ構造と同様の記法（但し、順列、繰り返し、同列の各構造のみ使用できる）で表現する。物理データ構造に含まれる繰り返し構造についてはその配下の構成要素が逐次アクセスできるのか、キーによる乱アクセスができるのか等を分類する以下の３種の属性（アクセス属性という）の中から一つを指定する。

① ＄キーナシ　－－－　逐次アクセスのみ可能。

② ＄キーツキ　－－－　キーによる乱アクセスのみ可能。

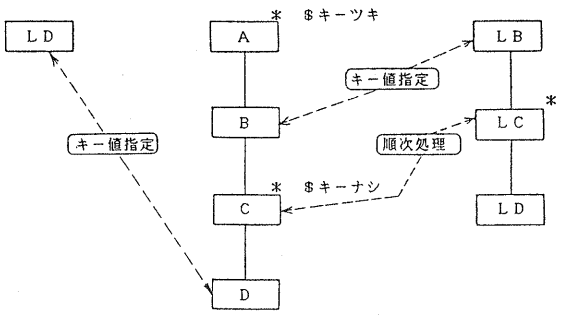
③ ＄キー　－－－　①、②の両者のアクセスの仕方が可能。

物理データ構造はファイルのアクセスの仕方についての仕様であり、論理データ構造設計者にどのような論理データ構造と抽出文が設計し得るかを示す機能をもつ。物理データ構造上の木構造はアクセスパスを表わしており、例えば構成要素に階層があるとき上位のものからアクセスしなければならない（論理データ構造上でアクセスすることを書かなければならない）ことを示す。また、例えば物理データ構造上のある繰り返し構造がキーなしの指定であるのにキー値指定の抽出文を書くと誤りとなる（図４．１－１）。物理データ構造によるアクセス仕様の表現法には①データ構造とアクセス属性のみの簡明な表現であり理解が容易である、②木構造の制約はあるが、メモリ上のデータからデータベースを含む幅広い入出力データのアクセス仕様を統一的表現で示すことができる、の特徴がある。

論理データ構造は処理を表わしているため同じ物理データ構造（ファイル）をアクセスするプログラムであっても処理内容が変われば構造が変化する。一方、物理データ構造はプログラム間に共通な仕様

表　４．１－１　ﾃﾞｰﾀ構造表現記号一覧

種別	構造					構成要素	
	① 順列	② 繰り返し	③選択	同列	ﾌﾟﾛｾｽ		階層
記号	。（句点）	＊	！	＋		ﾌﾟﾗﾝｸ	－
記述例	００。Ａ ０１　Ｂ ０１　Ｃ	００＊Ａ ０１　Ｂ	００！Ａ ０１　Ｂ ０１　Ｃ	００＋Ａ ０１　Ｂ ０１　Ｃ	００．Ａ ０１　Ｂ ０１　Ｃ	－	００　Ａ ０１＊Ｂ ０２　Ｃ
意味	配下のﾃﾞｰﾀ構造を ①記述順の処理 ②繰り返して処理 ③選択して処理 する。			配下のﾃﾞｰﾀ構造を任意順でアクセス／処理する。 実際の順序はﾃﾞｰﾀの参照関係で決まる。	配下のﾃﾞｰﾀ構造を記述順に処理する。 但し、順列と異なり、同一情報を再使用（２重書き、出力後入力等）する。	アクセスの実体（レコードﾞ等）を表わす。	実体にアクセスﾊﾟｽのあることを示す。 上位の実体をアクセスしてからでないと下位の実体をアクセスできない。



誤りの論理データ構造の例

●アクセスパスの無視がある（Bを読んでいない）。
●アクセス属性の無視がある（キー値指定の抽出文を使用している）。

物理データ構造（アクセス仕様）

●AファイルはDレコードの並びを配下にもつBレコードの並びである。
●Bレコードはキーによる乱アクセス、Dレコードは順アクセスのみできる。

正しい論理データ構造の例

●LBをキー値指定で1つ読み出し，配下のLDの並びを順次処理する。

凡例　<－－>
抽出文による対応づけを示す。（　）内は条件を示す。

図4．1－1　物理データ構造の役割

のため、その実現法を決める部品仕様書とともにシステム共通設計者が代表して設計し共通仕様書として登録する。

(4) 抽出文

論理データ構造と物理データ構造の対応をとるための文であり論理データ構造の構造、構成要素ごとに指定する。抽出文は上記の対応をとるとともに、①繰り返しの終了条件、②乱アクセス時のキー値の指定等のアクセス条件、③使用フォーマットの指定、等の役割をもつ。

抽出文にはこの他、スタック処理、コントロールブレーク等の処理を実現するためのものがある。抽出文は個々のプログラムでのデータ構造の取り扱い条件を示すためプログラム仕様書に記述する。

4．1．3　フォーマットの定義と参照

データ構造の構成要素には一つ以上のデータ項目を含むがその仕様をフォーマットとして定義する。フォーマットの仕様は物理データ構造同様に同一ファイルを扱うプログラム間で共用されるため物理データ構造と併せて共通仕様書に記述する。

一つの物理データ構造に複数種類のフォーマットを定義することができ、論理データ構造上の構成要素でどのフォーマットを用いて処理するかを抽出文で指定する。論理データ構造上の構成要素（論理構成要素という）は処理目的を表わしており必ずしもフォーマットの形式の種類と対応しないため同一フォーマット形式を複数の論理構成要素から使用できることにした。これに伴いフォーマット内のデータ項目を参照するときフォーマット名による修飾ではなく論理構成要素名による修飾によって行なうものとした。

4．1．4　数え上げ属性の定義[9]

データ項目の属性の一つであるデータの型については日本語処理のための日本語属性を含めてＣＯＢＯＬのデータ属性をＳＬの基本属性として全て網羅した。

ＰＡＳＣＡＬやＡｄａで採用されている数え上げ属性の概念はコード（定数）設計を問題向きにする上で効果が高い。ＳＬの数え上げ属性の定義では利用者でその実現法である基本属性の選択及び値の選択ができるようにした。これはコード設計の変更をしにくい既存システムへの適用を行なうとき処理系が自動決定する方式では結果として数え上げ属性が使えないことが多くなり効果が薄れるためである。

コード設計は通常適用システム内での標準設計事項となるため共通仕様書に記述するものとした。

4．2　演算リストの記述機能

ジャクソン法では演算リストについて、①データ構造で表わしたデータに対する業務記述のための演算リスト（代入文やチェック文が主に記述される）、②データ構造等の実現処理のための演算リスト（入出力用命令文が主に記述される）、とを区別せず単一の演算リストを作成することになっている。ＳＬ利用の設計では①は業務記述のための演算リストとしてプログラム仕様書に、②はデータ構造や問題向き関数等の実現法に関する演算リストとして部品仕様書に、それぞれ異なった記法で独立に記述する。

以下に各仕様書における演算リストの記述方式について述べる。

4．2．1　プログラム仕様書の演算リスト

個別の業務プログラムを記述する仕様書であり、この仕様書での演算リストの記述レベルの高度化を図ることは生産性向上に最も寄与する。以下に特徴を述べる。

(1) 抽出文や一時計算項目の定義文、代入文、チェック文等の細々とした文を、対象となる論理データ構造の構造及び構成要素ごとにまとめて整理記述する枠組みを設けた。これにより設計者のスキルによらない均質な設計結果が得られる。

(2) 代入文の記述は、1代入文則による非手順記述とした。これに伴い以下の工夫を図った。

(A) 代入式と代入条件を対にした代入文形式を設けた。一つの代入文には複数の対が記述できるようにした。

(B) 代入条件はいくつかの代入文で共通になることが多く、このときの記述量の削減、実行時性能の向上を狙いとして多分岐条件文（ケース文）を設けた。代入条件の評価順序は(A) 項の対のときも含めて実行時性能を重視して記述順とし、最初に満足したときの代入式の値を代入対象データ項目の値とした。

(C) 更新データ項目については旧値、新値の参照ができる。経験的な出現頻度を考慮して新値の参照のときにデータ項目名に修飾語（シン）をつけることにした。

(D) ＣＯＢＯＬ言語には関数がないが事務処理といえども関数は問題向き仕様記述のために有用である。部品仕様書で作成した利用者定義の関数を代入式等で引用できる。

(E) データ構造のアクセス結果等についての情報は組み込み関数を通じて問い合わせることができる。

４．２．２　部品仕様書の演算リスト

ＣＯＢＯＬプログラムの記述では、ある目的をもった処理を達成するためには手続き部の文ばかりでなく環境部、データ部での宣言等も必要である。部品仕様書は入出力等の部品種別毎に定められた目的を達成するために必要なＣＯＢＯＬ文で書かれたコードを一つの仕様書にまとめて記述する。例えばラインプリンタへ出力するために必要なＣＯＢＯＬコードは環境部のＦＣ句、データ部のＦＤ句及び手続き部のＯＰＥＮ文、ＷＲＩＴＥ文、ＣＬＯＳＥ文が必要である。

しかし大規模プログラムの走行環境配下では業務プログラムに必ずしもこのようなＣＯＢＯＬの標準入出力命令を使わせずに独自の外部サブルーチン等を用意してこれを使わせることが多い。このようなとき従来のジェネレータでは殆ど融通性がなかった。ＳＬでは部品仕様書で標準入出力命令の代わりに外部サブルーチン呼び出しのためのＣＡＬＬ文等をひな形として定義しておくこともできる。部品仕様書ではこれらの一連の宣言文、実行文をＳＬで定める部品定義用標準中間言語のオペレーション名（＜＜マエショリ＞＞，＜＜アトショリ＞＞，＜＜ＰＵＴ＞＞等）と対応付けて一つの仕様書に記述する。

部品仕様書は例えば入出力用部品であれば概ね媒体種別単位に作成する。部品仕様書を汎用的に記述するために部品仕様書にはパラメータが定義できる。部品仕様書ではパラメータを参照して展開時の制御を行なうことのできる展開時文、展開時関数を記述することができる。ＳＬのこの記述方式には以下の特徴がある。

(1) 関連のあるＣＯＢＯＬコードを一つの仕様にまとめて記述できるため設計、記述時の洩れや誤りを起こしにくい。またレビュが容易である。
(2) プログラム仕様書からの参照は殆どの場合、部品仕様書名や関数名、パラメータの記述だけで済み参照が容易で誤りを起こしにくい。

５．記述例

記述例として簡単な元帳更新プログラムを示す。付図－１がプログラム仕様書である。付図－２はプログラム仕様書で参照している出力用DATAOUT 部品仕様書の例である。ＳＬの仕様書はテキスト形式で全て章節段落形式を用いて決められた箇所に決められた事項を記述する形式とし、記述者のスキルによらない均質な記述をガイドしている。プログラム仕様書については数え上げ属性の定数も含めて業務用語を適切に表現することのできる日本語名標が書ける。以下、各仕様書の主要な記述に対応する行番号を示す。

(1) プログラム仕様書

論理データ構造の定義	3-16
繰り返し構造の対応	8,13
構造、構成要素ごとの段落	18,34,39,...
抽出文	20,36,41,44,...
一時計算項目の定義	22,33,48
チェック文	38
代入文	54-58,66,...
例外処理定義	94-99
物理データ構造の定義	105-108,...
フォーマットの定義	109-115,...
入出力用部品参照	117,128,160
数え上げ属性定義	162-166

(2) 部品仕様書

部品とプログラム仕様書とのインタフェース定義	3-11
ＣＯＢＯＬコードひな形	
◎手続き部用	12-19
◎宣言部、データ部用	21-34

本例では共通仕様書機能を使用していないないがプログラム仕様書の３章については共通仕様書として記述できそのときのプログラム仕様書の記述はそれらの参照文だけとなり、記述量が大幅に減少する。

６．評価

(1) 記述能力

大規模バンキングシステムのアプリケーションプログラムの机上記述実験から約80%のオンラインプログラムに適用可能であることがわかった。データ構造記述能力上の問題点としてはデータベースの処理で逆順読みの必要なプログラムがありデータ構造

でうまく表現できない例があった。

部品仕様書の記述能力ではデータベースを含めた入出力について部品化できることがわかった。バンキングシステムの例では問題向き関数、特殊処理用部品も含めて、約 100部品、業務プログラムとの規模比で約 3%の部品を作成すればよいとの見通しを得ている。

(2) 記述量比較

手書きＣＯＢＯＬプログラムの記述行数と比べたＳＬプログラム仕様書の記述行数は、約 3分の１であった。

(3) 信頼性

手書きプログラムに比べた実行時バグ数が約 4分の１であった。

７．あとがき

適用性の向上及び大規模プログラムの開発に効果的に利用できることを狙いとした設計用言語ＳＬの言語仕様の主要機能の概要及び特徴について述べた。ＳＬではデータ構造の設計に物理データ構造と抽出文の概念を取り入れたことにより木構造データベースを含む幅広いデータ構造が取り扱えるようになった。また、３階層に分けた仕様記述方式は各層間の独立性があるため新規開発は勿論、ＳＬ記述の仕様書の再利用にも効果が期待できる。

開発中の処理系を内部で試用したところフラグ（構文エラー）取りを終わると殆ど実行時バグがなく好評であった。仕様書の均質記述を狙いとした章節段落構成及びデータ構造の構造、構成要素に着目した演算リストの記述の枠組みの考え方は、ＶＤＴ利用の入力支援システムがあれば、さらに効果的であるとの意見が多く、今後の主要課題である。

その他の今後の課題を以下に示す。

(1) 各種大規模リアルタイムシステム等のアプリケーションプログラムへの適用を進める。

(2) 現在ＳＬ向きドキュメント作成支援システムを作成中でありこれを利用すると、効果が一層向上する。またＳＬ向きのデバッガの開発を計画中である。

(3) ＳＬの目的プログラム言語は現在ＣＯＢＯＬとしているが目的プログラム及び処理系ともに汎用言語であるＡｄａとすれば一層適用域が広がる。このためＡｄａ化を予定している。

参考文献

［１］Bergland,G.D.,"A Guided Tour of Design Methodologies",Computer,Vol.14,No.10,1981

［２］岡本他，”設計言語におけるデータ構造の階層的表現について”，58.10，情処学会第26回全国大会ｐｐ457-458

［３］岡本他，”データ中心型設計言語を用いた効果的設計方法について”，59.3，電子通信学会総合全国大会ｐｐ1721

［４］稲田他，”設計言語における部品構成法の検討”，59.9，情処学会第29回全国大会ｐｐ563

［５］岡本他，”拡張可能な非手順型言語方式の一提案”，58.4，電子通信学会総合全国大会ｐｐ656

［６］Prywes,N.S.,"Automatic Generation of Computer Programs",Advances in Computers Vol.16, Academic Press pp.57-125 1977

［７］Jackson,M.A.,"Principles of Program Design",Academic Press,New York,1975

［８］岡本他，”データ構造とオペレーションに基づくプログラム自動生成法の一提案”，58.10，情処学会第26回全国大会ｐｐ503-504

［９］岡本他，”設計言語におけるデータ属性の階層的定義について”，57.10，情処学会第25回全国大会ｐｐ495-496

```
1  1 ﾌﾟﾛｸﾞﾗﾑｼｮｳｼｮ(預金処理);
2  2 ﾛﾝﾘｷﾃｲ;
3   2.1 ﾛﾝﾘﾃﾞｰﾀｺｳｿﾞｳｷﾃｲ;
4    〈ﾆｭｳﾘｮｸ〉:
5     00* 入力電文,
6      01 入力レコード;
7    〈ｺｳｼﾝ〉:
8     00* 預金元帳=入力電文,
9      01 預金レコード;
10   〈ｼｭﾂﾘｮｸ〉:
11    00。出力情報,
12     01 表題行,
13     01* 明細行群=入力電文,
14      02 明細行,
15     01 日付行,
16     01 合計行;
17  2.2 ｼｮｳｻｲｼｮﾘｷﾃｲ;
18   〔入力電文〕:
19    〈ｾﾝﾃｲ〉:
20     〈1〉 @<=ＩＮＰＵＴ電文;
21    〈ﾎｿｸ〉:
22     00 入金件数計 $DEC(2) $ｼｮｷﾁ(0)
23          :=ｾﾝ.@+1:入力レコード.入出金区分="入金;
24     00 出金件数計 $DEC(2) $ｼｮｷﾁ(0)
25          :=ｾﾝ.@+1:入力レコード.入出金区分="出金;
26     00 入金金額計 $DEC(10) $ｼｮｷﾁ(0)
27          :=ｾﾝ.@+入力レコード.金額:入力レコード.
28               入出金区分="入金;
29     00 出金金額計 $DEC(10) $ｼｮｷﾁ(0)
30          :=ｾﾝ.@+入力レコード.金額:入力レコード.
31               入出金区分="出金;
32     00 西暦取引日 $DEC(6) $ｼｮｷﾁ(0)
33                :=ﾍﾝｶﾝ(入力レコード.取引日);
34   〔入力レコード〕;
35    〈ｾﾝﾃｲ〉:
36     〈1〉 @<=ＩＮＰＵＴレコード;
37    〈ﾁｪｯｸ〉:
38     〈1〉 金額 > 99999999 ? %ｴﾗｰ (101);
39   〔預金元帳〕:
40    〈ｾﾝﾃｲ〉:
41     〈1〉 @<=元帳ファイル;
42   〔預金レコード〕:
43    〈ｾﾝﾃｲ〉:
44     〈1〉 @<=ｾﾚｸﾄ(元帳レコード) ｷｰ (入力レコード.
45                    口座番号);
46     〈1〉 ｺｳｼﾝ;
47    〈ﾎｿｸ〉:
48     00 計算値 $DEC(10) $ｼｮｷﾁ(0)
49          :=残高-入力レコード.金額:入力レコード.
50               入出金区分="出金;
51    〈ﾁｪｯｸ〉:
52     〈1〉 計算値 < 0 ? %ｴﾗｰ (102);
53    〈ﾖｳｿｾｯﾃｲ〉:
54     〈1〉 残高:=@+入力レコード.金額:入力レコード.
55                    入出金区分="入金
56               :=@-入力レコード.金額:入力レコード.
57                    入出金区分="出金;
58     〈1〉 取引件数:=@+1;
59   〔出力情報〕:
60    〈ｾﾝﾃｲ〉:
61     〈1〉 @<=ﾚﾂ(出力ファイル);
62   〔表題行〕:
63    〈ｾﾝﾃｲ〉:
64     〈1〉 @(タイトル１)<=出力レコード;
65    〈ﾖｳｿｾｯﾃｲ〉:
66     〈1〉 出力項目１:='   ***ﾄﾘﾋｷｼﾞｮｳﾎｳ ***   ';
67   〔明細行群〕:
68    〈ｾﾝﾃｲ〉:
69     〈1〉 @<=出力ファイル;
70   〔明細行〕:
71    〈ｾﾝﾃｲ〉:
72     〈1〉 @(明細)<=出力レコード;
73    〈ﾖｳｿｾｯﾃｲ〉:
74     〈1〉 口座番号:=入力レコード.口座番号;
75     〈1〉 取引金額:=入力レコード.金額;
76     〈1〉 入出金区分:=1:入力レコード.入出金区分
77                         ="入金
78                      :=2:入力レコード.入出金区分
79                         ="出金;
80     〈1〉 残高:=ｼﾝ.預金レコード.残高;
81   〔日付行〕:
82    〈ｾﾝﾃｲ〉:
83     〈1〉 @(タイトル２)<=出力レコード;
84    〈ﾖｳｿｾｯﾃｲ〉:
85     〈1〉 出力項目２:=ｾﾝ.入力電文.西暦取引日;
86   〔合計行〕:
87    〈ｾﾝﾃｲ〉:
88     〈1〉 @(合計)<=出力レコード;
89    〈ﾖｳｿｾｯﾃｲ〉:
90     〈1〉 入金件数計:=ｾﾝ.入力電文.入金件数計;
91     〈1〉 入金金額計:=ｾﾝ.入力電文.入金金額計;
92     〈1〉 出金件数計:=ｾﾝ.入力電文.出金件数計;
93     〈1〉 出金金額計:=ｾﾝ.入力電文.出金金額計;
94    2.3 ﾚｲｶﾞｲｼｮﾘｷﾃｲ;
95     〔ｴﾗｰ〕;
96      〈ﾊﾟﾗﾒｰﾀ〉;
97       00 ｴﾗｰｺｰﾄﾞ  $DEC(3);
98      〈ﾚｲｶﾞｲｼｮﾘ〉;
99       〈1〉 ｲｼﾞｮｳｼｭｳﾘｮｳ;
100  3 ﾌﾞﾂﾘｷﾃｲ;
101   3.1 ﾌﾟﾛｸﾞﾗﾑｷﾃｲ;
102     ﾌﾟﾛｸﾞﾗﾑﾒｲ=YOKIN;
103     ﾌﾟﾛｸﾞﾗﾑｹｲｼｷ=ﾒｲﾝ;
104   3.2 ﾃﾞｰﾀｷﾃｲ;
105    〔ＩＮＰＵＴ電文〕;
106     〈ﾌﾞﾂﾘﾃﾞｰﾀｺｳｿﾞｳ〉;
107      00* ＩＮＰＵＴ電文  $ｷｰﾅｼ,
108       01 ＩＮＰＵＴレコード;
109     〈ﾌｫｰﾏｯﾄ〉:
110      00 ＩＮＰＵＴレコード,
111       01 取引日     $DEC(6),
112       01 口座番号   $DEC(6),
113       01 入出金区分 $区分コード,
114       01 金額       $DEC(10),
115       01 領域       $CHR(113);
116     〈ﾒｶﾆｽﾞﾑ〉:
117      @DATAIN=();
118    〔元帳ファイル〕;
119     〈ﾌﾞﾂﾘﾃﾞｰﾀｺｳｿﾞｳ〉;
120      00* 元帳ファイル   $ｷｰﾂｷ,
121       01 元帳レコード;
122     〈ﾌｫｰﾏｯﾄ〉:
123      00 元帳レコード,
124       01 口座番号   $DEC(6) $ﾎｽﾄﾒｲ(KOBAN),
125       01 残高       $DEC(10),
126       01 取引件数   $DEC(2);
127     〈ﾒｶﾆｽﾞﾑ〉:
128      @UPDATEK=();
129    〔出力ファイル〕;
130     〈ﾌﾞﾂﾘﾃﾞｰﾀｺｳｿﾞｳ〉:
131      00* 出力ファイル   $ｷｰﾅｼ,
132       01 出力レコード;
133     〈ﾌｫｰﾏｯﾄ〉:
134      00 出力レコード,
135       01 出力エリア   $CHR(128);
136      00 タイトル１,
137       01 出力項目１   $ﾍﾝｼｭｳ(X(30));
       :
159     〈ﾒｶﾆｽﾞﾑ〉;
160      @DATAOUT=();
161   3.3 ﾃﾞｰﾀｿﾞｸｾｲｷﾃｲ;
162    〔区分コード〕;
163     〈ﾛﾝﾘｷﾃｲ〉;
164      00 区分コード=["入金,"出金];
165     〈ﾌﾞﾂﾘｷﾃｲ〉:
166      00 区分コード< $区分コード:$DEC(1)> =[1,2];
167  4 ｼｮｳｼｮｵﾜﾘ;
```

付図－１　プログラム仕様書の記述例

```
1   1 ﾌﾟﾋﾝｼｮｳｼｮ(DATAOUT);
2   2 IOﾌﾟﾋﾝｷﾃｲ;
3    2.1 ｲﾝﾀﾌｪｰｽｷﾃｲ;
4     <<IOｼｭﾍﾞﾂ>>;
5      <<1>> IO=(O);
6     <<ｺｳｿﾞｳ>>;
7      00* @OUTF,
8       01 @OUTR;
9     <<ｺﾝﾍﾞﾝｼｮﾝ>>;
10     <<1>> ｼｮﾘｹｯｶ=@GEN(1),$CHR(2),'00','10';
11     <<1>> ﾊﾞｯﾌｧ=@OUTR,FD;
12    2.2 ﾃﾝｶｲｼｮｳｷﾃｲ;
13    <<ﾏｴｼｮﾘ>>;
14           OPEN  OUTPUT  C-FILE.
15    <<PUT>>;
16           WRITE  @ﾎｽﾄﾒｲ(@OUTR).
17    <<ｱﾄｼｮﾘ>>;
18           CLOSE  C-FILE.
19    <<END>>;
20  3 ｷｮｳﾂｳﾃﾝｶｲｷﾃｲ;
21   3.1 ｾﾝｹﾞﾝｷﾃｲ;
22    <<FC>>;
23           SELECT C-FILE ASSIGN TO SYSOUT;
24           FILE  STATUS  @GEN(1).
25    <<FD>>;
26        FD  C-FILE LABEL RECORD IS STANDARD.
27    <<USE>>;
28        U-ERR-2  SECTION.
29           USE AFTER STANDARD ERROR PROCEDURE ON C-FILE.
30    <<WS>>;
31        77  @GEN(1)  PIC XX.
32    <<IOC>>;
33           APPLY CONTROL-CHARACTER ON C-FILE.
34    <<END>>;
35  4 ｼｮｳｼｮｵﾜﾘ;
```

付図－２　部品仕様書の記述例