解　説

要求仕様の記述と検証†

大野　豊††　阿草清滋††

1. はじめに

計算機のハードウェア開発には，たとえば論理回路シミュレータ，布線設計自動化など各種の計算機援用設計システムが利用され，より複雑で高度なハードウェア・システムの開発を容易なものとしている．これに比して，ソフトウェア・システムの開発は，そのほとんどが人手により行われ，広く一般化された計算機による設計援助システムは存在していない．

ソフトウェアに対する需要は，ハードウェアの性能向上により軽減されるよりもむしろ，より厳しい条件が課せられるようになり，機能の複雑さ，高性能，高信頼性が求められている．このため，従来からの“工芸的な”手法では，ソフトウェアの開発は困難となり，特に大規模ソフトウェア開発では，開発コストや日程の大幅な増加，不充分な品質のソフトウェア，システム運用や保守の難しさなどの問題を含むことが多い[18]．

そこで，ソフトウェア開発を個人の能力に頼らず，ある一定の能力を有するものであれば，必ず所期の機能・性能を満足させるソフトウェアが得られるようなソフトウェア生産手法，特に計算機援用手法の確立が必要とされる．

従来，各種のソフトウェア開発手法や，開発支援ツール，プログラミング技法などが提唱されてきたが，いまだに計算機援用の体系化された実用的な手法はない．

ハードウェア開発に比して，ソフトウェア開発の計算機援用が困難である理由は，一つにはハードウェアの開発の各過程の出力が定型化されている（たとえばタイムチャート，論理回路図，布線表など）のに対しソフトウェアの開発の各過程の出力は，たとえば，システム仕様書，ソフトウェア基本設計書，ソフトウェア・モジュール設計書などはすべて，我々が日常用いる自然言語で書かれていることが多い．ハードウェアが1と0の論理の世界にとどまるのに対し，ソフトウェアはそのシステムの置かれる多種多様な環境を扱い，また，論理の複雑さも桁違いに大きいので，どのようなソフトウェア・システムに対しても適応でき，しかも計算機処理可能な形式化された一般的な記述様式が定められないからである．ソフトウェアを工業製品並みに規格化された品質で生産するためには，その各過程での品質管理，すなわち，規格を満たしているかどうかの確認（validation），検証（verification）が必要とされる[1]．この確認や検証の規準として役立てるためにも設計の各過程の出力として曖昧さのない，完全で形式化された記述の出力が望まれる．

では，設計活動という人間の最も知的な活動が，仕様，設計書を形式化することで不必要となるのか，或いは，設計の選択の自由度が小さくなるのであろうか．形式的仕様からのプログラムの自動合成に関する研究もあるが，その仕様記述言語は高レベルのプログラミング言語とも考えられる．そのための仕様記述に際して，モジュール分割，同期，資源割付けなどの設計作業が必要とされる．また，設計活動は，他の工学分野と同様に，完全に自由に行うより，確立された手法に沿って行うのが，より効率的で誤りも少ないため形式化が進むことは設計活動の助けとなっても，その選択の自由を奪うことは少ない．

たとえば，プログラミングの過程において，構造的プログラミング（Structured Programming）はプログラムの制御構造をある型にはめたが，それによる結果はプログラミング活動に規律が与えられたことにより出力であるプログラムの品質（機能，速度，オブジェクト・コードのサイズなど）が低下することなく，生産性の向上や保守性の向上が得られたと報告されている[2]．

高い信頼性と性能を有する大規模で複雑なシステムを開発するには，単にテスト技術のみでなく，システムのライフ・サイクル全体の統一的な管理と，その各段階での，検証，確認が必要とされる．各段階での出

† Requirements Description and its Verification by Yutaka OHNO and Kiyoshi AGUSA (Department of Information Science, Kyoto University).

†† 京都大学工学部情報工学科．

力の記述は次の段階の仕様となるが，仕様には要求仕様，ソフトウェア・システム仕様，さらにプログラム仕様などがある．この中で要求仕様は要求（仕様）記述，要求定義，システム仕様などとも呼ばれるが，本文での"要求"とはシステムの持つべき性質，機能，性能などを指し，一般的な意味での要求ではない．仕様はその後に続くシステム開発過程の要求とも言えるが，ソフトウェア工学における要求工学，要求分析，要求記述などとして用いられる"要求"とは，システムのライフ・サイクルの最も初期の過程を対象としている．我々はモジュール構成，資源割付け，ハードウェア・システムの選択などは設計作業ととらえ，要求仕様はシステムをブラック・ボックスと考え，そのシステムと現実社会とのインタフェース条件を記述したものと考える．この最も初期にあたる要求分析，要求記述の段階での検証，確認，すなわち要求仕様記述の検証，確認は，ライフ・サイクルの他の段階での検証，確認と異なる点が多い．まず第一に，その活動の対象が必要性，要求，さらに利用者の嗜好などのように漠然としたものを形式化し，要求仕様書を作成することであり，他の段階のように，ある仕様書に基づいて行う活動でないことが挙げられる．次に，検証や確認のための"正しさ"の根拠とすべき記述が存在していないことが挙げられる．他の段階では与えられた仕様書の内容を充足するかどうかを，"正しさ"の根拠とできる．このため，要求記述の検証は主として記述の相互間に矛盾が存在しているかどうかを調べる"一貫性の検証"を指すことが多い．

以下，要求記述の支援システム，記述言語，及び検証について簡単に述べる．

2. 要求記述システム

大規模システムの開発経験から，開発されたシステムに含まれている誤りの多くは設計時に既に含まれていることが知れている[3]．このような誤りを修正するには多大の費用と労力を要するため，設計時の誤りを最小化する努力が払われている．また，目的システムに要求される機能，性能を正しく把握し，それを設計者に正確に伝えなければ，誤りのない設計は期待できない．このようなことから，"要求工学"（Requirements Engineering）と呼ばれる分野の重要性が認識されてきた*．これにはシステムの置かれる環境の分析，要求機能・性能の分析を通して，システムの要求仕様書を作成するための手法，および，支援システムなどが含まれる[6,7]．

2.1 要求記述の評価

システム要求記述において，なにをどのように記述すべきであるかについては，対象システムや開発環境などにより様々な主張があり，いまだ確定されたものはない．そのため，要求記述の評価は困難であり，ここでは要求記述の目的，要求記述の持つべき性質を各々 C. V. Ramamoorthy[10] と M. W. Alford[4] 等の文献から引用して述べるにとどめる．

システム要求定義の目的

（1） システムやソフトウェアに必要とされる機能，性能などの諸要求を正確に記述する．

（2） 要求の記述を正確に行う手法，記述の完全性，一貫性の検証のための解析的手法を与える．

（3） ソフトウェア・モジュール作製の正確な仕様を与え，各段階の設計者間の誤った解釈を防ぐ．

（4） 設計書，またはソフトウェアの検査のための"基準"を与える．

（5） システム要求に変更が生じた場合に，その影響の及ぶ範囲を把握し，余分な修正，検査を省いて保守を簡略化する．

要求記述の持つべき性質は，上記の（1）～（5）の目的に合致するものであるが，具体的には次の 11 の項目が掲げられる．これはプログラムの形式的仕様記述のもつべき性質に近い[5]．

要求記述に望まれる性質

（1） 伝達性 (Communicability)
明確で理解し易く曖昧さのない形で処理要求を伝える能力．

（2） モジュール性 (Modularity)
個々の項目を矛盾なく，最小限の努力で，付加，変更，削除できること．

（3） 追随性 (Traceability)
あるレベルでの要求仕様の各部分が，その1つ前のレベルで生じた要求に追随し，またそれに続く要求仕様に追随されること．

（4） 一貫性 (Consistency)
要求仕様として矛盾を含んでいないこと．

（5） 完全性 (Completeness)
すべての入力が処理され，すべての出力が生成され，処理のすべての条件が明らかにされていること．

（6） テスト可能性 (Testability)

* 我が国でもこの関係を特集した雑誌が出版されている[9]．

要求の各単位がそれを記述したもののテストで検証され得ること.

(7) 設計自由度 (Design Free)

要求記述はある特定の設計上の選定を強制しないこと.

(8) 正確性 (Correctness)

要求記述で示されたすべての要求を満たすとき，その生成物が元の要求をきっちり（余分なものも不足もなく）満たすこと.

(9) 必要性 (Necessity)

要求記述された各々の要求は，実際の要求を満たすために必要なものであること.

(10) 実現可能性 (Feasibility)

記述された要求を満たす生成物の設計が少なくとも1つ存在すること.

(11) 自動化可能性 (Automatability)

要求記述の形成と検証に役立て得るソフトウェア・システムを作製でき，利用できること.

2.2 要求記述用言語

要求記述は現に存在する環境から発生する必要性という概念的なものを，それに適合するシステムに写像するもので，その記述はきわめて人間的な創造活動である．そのため，従来，要求記述は自然言語を用いて行われることが多かった．新たなシステム開発には新たな概念を含むことが多く，一般のソフトウェア・システム記述言語（プログラミング言語も含めて）のような，その構文規則と意味が厳格に定義されている言語では，ある特定の問題向きの要求記述しかできないであろう．曖昧さのない記述ができ，しかも計算機援用による検証を可能とするためには形式的表現が必要である．設計者の読み易さ，記述し易さ，記述の正確さなどが考慮された要求記述言語が提案されている.

SA 図式[14),15),17)]

要求定義 (Requirements Definition) をうまく行うには，システムのアルゴリズム的な性質，すなわち，機能構造 (Functional Architecture) を正しく把握し評価しなければならない．機能構造の分析を自然言語で，簡潔にしかも曖昧さなく表現するために，建築などに用いられるような“青写真”を用いる．この青写真の図的な枠組と自然言語を組合せることにより機能構造の分析を簡潔に厳格に記述できる．SA 図式では適切に選ばれた図形記号を用い，正確な意志伝達のための記述の方法も指示されている．図形的表現は自然言語の表現能力を高めるためであり，各図形要素に名前が付加でき，必要ならば，脚注として自然言語での記述が可能である.

SA図式は階層化された記述に適し，分析の過程の記述に利用される．SA 図式における矢印は制御条件となる関係を示すが，これは必ずしも順序関係を表わすものではない．しかしSA 図式をソフトウェア設計にまで利用するために順序関係の導入も図られている．SA 図式は要求の分析過程の道具であり計算機援用は難しいが，分析の結果としての要求記述は計算機により，無矛盾性の検証，要求の配分の妥当性の検証などを行うことが望ましい．しかしSA 図式を言語とする要求分析手法である SADT (Structured Analysis and Design Technique) では，設計データベースを用いた計算機援用に際して，SA 図形をそのデータベースに入れるのはプロジェクト・ライブラリアンの仕事としており，内容の検証は要求定義者間での互いの記述の回覧により行われる.

PSL/PSA[19)-25)]

PSL (Problem Statement Language)/PSA (Problem Statement Analyzer) は米国ミシガン大学の ISDOS (Information System Development and Optimization System) プロジェクトで開発された言語であり，情報システムの要求記述，解析，文書管理を行うための形式言語である．これは情報システムの要求記述を計算機処理可能な形(PSL)で記述し，PSA を用いて，記述された関係の一貫性の検証やあるデータの利用頻度などについてのレポートを生成することができる．システムの記述は次の8つの面から行われる.

(1) システム・フロー

システムと環境との入出力関係

(2) システム構造

対象システムの機能や処理の階層構造

(3) データ構造

システムの要素に関する構造

(4) データ導出

あるデータ要素・グループが他のデータ要素・グループから生成される関係.

(5) システム・サイズ

システムの行うべき仕事量，要求する資源量

(6) システム動特性

ある事象が引き起こされる条件のリスト，その事象の結果としての動作

(7) 特　性

文書管理，コミュニケーション，解析などのた

めに用いられる機能

(8) プロジェクト管理

プロジェクトの組織化，管理を改善するための機能

これらの記述をもとに解析を行うが，この PSL/PSA による要求記述は，データを中心としたものであり，バッチ処理システムの記述に向いているが，オンライン実時間システムの要求記述には不適当である．

RSL 及び R-Net [4),11),26)]

RSL (Requirements Statement Language) はソフトウェア要求記述のための機能の“流れ (flow)”を意識した言語である．これは米国 TRW 社の SREM (Software Requirements Engineering Methodology) で用いられ，REVS (Requirements Engineering and Validation System) によって RSL で記述された要求対象システムの開発が支援される．RSL は実時間システムの要求記述に重点が置かれ，要求記述からシミュレーション・プログラムが生成され，実行時間などの性能面の要求記述の妥当性の検証を可能としている．このため，ある刺激に対する応答を発生するための処理系列として要求を記述する．この系列は R-Net (Requirements Net) と呼ばれ，システムの概念的なプロセスAlpha* の系列であり，必ずしもソフトウェア・モジュールに対応はしていない．しかし，そのAlpha のモデルは，手続き的なコード，具体的には Pascal[27)] で書くことができ，また Alpha の系列の制御構造も Pascal のプログラムに 1 対 1 対応させることが可能であり，要求記述段階での，しかも性能面の自動的なシミュレーションが可能となっている．

R-Net はグラフとして表現され REVS のグラフィックス機能を用いて入出力でき，テクスチャルな形での RSL による入出力と完全に互換性を持っている．

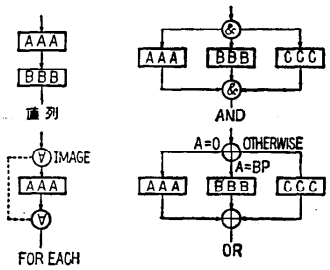

図-1 RSL の基本構造 (11)

* RSL, R-Net における Alpha は処理機能の単位を表わし，計算機処理におけるプロセスと必ずしも対応づけられないため特に Alpha と名付けている．

R-Nets は図-1 に示す 4 つの基本構造を持つが，これは構造化プログラミングにおける構造に類似している[28)]．しかし，その意味は少し異なる．たとえば，直列の構造はその処理の順序関係が保たれている限り，それを実現するソフトウェアはどのようにまとめられても構わない．AND 節はその間の Alpha が並列に，すなわちどのような順序ででも処理されうることを示し，AND 節の再結合点では同期がとられることを意味している．OR 節は条件分岐を意味し，条件を順に評価し真となるものを実行する．すべての条件が不成立とならないよう OTHERWISE 項が用意される．FOR EACH 節はそれに付けられた条件を満たすものに対して各々一回実行されるサブネット，あるいは Alpha を規定するものである．

RSL の拡張性を実現するため，柔軟な構造を持つトランスレータが用意され，また極めて単純な言語構造になっている．RSL の基本概念は次の 4 つである．

(1) 要 素

Alpha, R-Net, データなどの要素である．

(2) 関 係

要素間の 2 項関係を示す．

(3) 属 性

要素の性質を形式化したものである．

(4) 構 造

システムのフロー・モデルである．

システムの流れを記述するときの規律を与えるため構造の変更は許されないが，他の 3 つは利用者が自由に定義・拡張でき，新たな概念をその言語に付加することができる．

2.3 要求記述支援システム

大規模システム開発における要求記述では，膨大な文書の管理が要求され，また，一貫性などの機械的検証のためにも，要求記述支援システムが必要とされる．要求記述支援システムは，そのシステムの制御核となるモニタ，システム要求情報を蓄えておくデータベース，それに各利用者の必要とするレポートの生成や検証を行うための自動化ツール群からなる．これらを簡単に述べる．

モ ニ タ

要求記述支援システムの利用者と各種自動化ツールとの間のインタフェース制御を行う．利用者の入力したコマンドに従い，必要な自動化ツールの呼び出しを行い，レポートを作成する．開発対象システムの性格，および利用形態により，要求記述の検証項目・目的，

入出力レポート形式が変更されるため，それらの変更を受けつけ易い柔軟で拡張性に富んだ構造が要求される．

要求記述データベース

要求記述管理の中心となるデータベースであり，記述そのものの管理と，記述された意味を抽象化し，各種自動化ツールが操作可能な形での情報が管理される．ISDOS や REVS でのこのデータベースは関係データベースと呼ばれる．ここで言う“関係データベース”はデータベースの研究分野で使われている用語の意味とは異なる．このデータベースを用いデータの検索・更新がある観点から，またその補完的観点から検索・更新できる．たとえば，ある利用者はある機能に注目して要求記述を行い，その機能の入出力情報の記述を行ったとする．他の利用者は情報に注目し，その情報が入力または出力される機能を記述したとする．これらの記述は抽象化され，データベース内にその“関係”にもとづいて格納される．利用者は情報面，または機能面の各自の立場からデータベース内の記述にアクセスできる．また，ある機能にある情報が入力されなくなったとすれば，その機能と情報の間の入力という関係が削除される．このように関係を重視したデータベースを“関係データベース”と呼んでいる．

要求記述データベースへのアクセスは統一されたルーチンを介して行われ，各種自動化ツールによるデータ参照・変更に際して，構文，意味の検査が行われ，データの統一性を保証している．集中化されたデータベースは，ある要求定義者の活動，すなわち要求記述の追加・更新・削除が他のすべての要求定義者に通知することを可能とする．また，集中管理の他の利点として，システム全体を通して検査する必要のある“一貫性”の検証が容易に行えることがあげられる．

自動化ツール

自動化ツールは要求記述データベースの情報の各種解析を行うツールである．支援システムの目的により，用意される解析ツールの種類は異なる．一般的に静的解析ツールとして，一貫性検証，完全性検証などを行うものが用意される．動的解析としてシステム動作状況の把握，性能要求の妥当性検証などのため，自動シミュレーションが用意されることがある．シミュレーション・モデルを人手により合成すると，そのモデルが要求記述されたシステムを正しく反映しているかどうかが問題となる．シミュレーションの自動化のためには要求を手続き的に記述することと，要求記述言語の構造がシミュレーション言語の制御構造にうまく適合できるものでなければならない．また，性能を規定する，そのシステムの環境，システム資源などのパラメータをどのように導入するかも難しい問題である．

3. 要求記述の検証

ここで扱う要求記述は形式化された自然言語風な記述であり，厳密な意味での正当性の検証は行えない．厳密な検証のためには構文，および意味が厳密に形式的定義された一階述語論理のような言語での記述が必要となる．そのような言語では，曖昧さがなく，要求定義者間および要求定義者と設計者の間での正しい伝達が行える．その反面，システムの置かれる環境の記述が繁雑となったり，誰でも簡単に参照できるとは限らないなどの問題もある．要求記述はシステム開発のあらゆる段階の開発者に参照され，システム開発の意図の確認のためにも用いられるため，“読み易さ”と言う要素は重要である．このため，現在の要求記述は形式を定めた自然言語，図，表などを用いて行われることが多い．

要求記述はシステムの置かれる実世界を，機能，性能などの概念的記述に写像することであり，また，それより以前に計算機処理可能な形で記述されたものがない段階であり，検証は一貫性，完全性などの検証に限られ，記述の不備の指摘にとどまる．要求記述がそのシステムの要求を正しく記述しているかどうかは要求記述者の判断に任され，計算機で指摘できるものではない．要求記述過程は要求記述のみならず，要求分析作業からいかに誤り，見落しをなくすかが重要となる．

要求記述の検証は以上のことから，きわめて限られた範囲でのみ行われる．自然言語風の記述から検証可能なモデルへの抽象化，簡略化を行い，そのモデルの検証で，要求記述を代行する．その抽象化，簡略化の過程でかなりの情報，とくにラベルとして付加された各種の意味情報は失なわれ，余り詳細な検証は行えない．たとえば，前述の ISDOS の PSL/PSA では計算機解析による検証は，

- 各入出力は何らかの処理を受け，或いは受けたものである．
- 各処理は何らかの入力を受け，何らかの出力を生成する．
- 定義された名前はどこかで用いられている．

といったものに限られている．ある情報が必要な処理をすべて受けるかといった意味的なものは計算機援用の対象になっていない．

要求記述からの検証可能なモデル化の例として，グラフ・モデルと有限状態機械モデルをあげ，簡単に説明する．また，筆者らが研究を進めているアサーション言語を用いた要求記述検証についても述べる．

3.1 グラフ・モデルによる検証

しばしばシステムは有向グラフ・モデルにより把えられる．これは次の理由による．

（1） グラフ理論での性質が応用できる．

（2） システムの情報の流れ，機能間の接続関係と対応づけが容易である．

（3） 抽象化の程度が高く，計算機処理が可能である．

（4） モデルの適応範囲が広い．

このような有向グラフを用いた情報システム記述にはペトリネット[29]がよく知られているが，これと同様な制御フロー図とデータ・フロー図を用いて，システム要求記述，設計，インプリメントを統合的に扱うシステムにLOGOS[30,31]がある．これはハードウェア／ソフトウェアの機能分担の決定を遅らせることで両者のよりよいトレード・オフを得ようとするもので，情報システム全体の設計を行う．制御構造の解析はうまく階層化されているが，データ構造には充分な解析が加えられていない．

検証グラフと称する機能要求間の関係を示すグラフを用いて検証を行う要求記述解析システムがBelford等によって提案されている[8]．ここで用いられるグラフは検証グラフと呼ばれ，記述された機能要求を

機能要求＝入力/処理/出力
＝刺激/機能記述/応答

の関係で定まるフローから得られる有向グラフである．要求記述から検証グラフの導出は計算機により実行される．

この検証グラフに対する一貫性，追随性，完全性は次のように定義されている．

（1） 一 貫 性

検証グラフの各節点は機能的，あるいは論理的な節である．すべての弧は少なくとも一端が節に結ばれている．すべての弧・節点の集合は連結されている．

（2） 追 随 性

下位のレベルの検証グラフのグローバルな入出力の弧は，その数，方向，値について，その上位のレベルの検証グラフの弧に1対1写像される．

（3） 完 全 性

下位レベルのすべての検証グラフは，その上位レベルの検証グラフの機能節点に追随性を持つ．

これらの検証の他にシミュレーション，性能要求の処理，テストポイントの設定などのために，検証グラフの接続行列の巾行列が有効であるとしている．また，1入力1出力で連結された部分グラフを1つの節点に見なすことにより，シミュレーション・モデルの詳細度を必要に応じて変化できる．

3.2 有限状態機械モデル

情報システムの要求記述を考えると，ソフトウェアを実行するための機械が必要であり，ある種の抽象機械の振舞いとして要求記述することは素直な考え方である．

Parnas はソフトウェアの各モジュールを1つの抽象機械に対応させ，その機械の状態記述でシステム仕様を記述させる手法を提唱した[32]．これはモジュール間の構造が決定された後のシステム仕様記述に適するが，システムの全体的把握には不向きである．

また，Salter はシステム要求をデータ処理サブシステム (DPS) に対する要求に変換する手法の中で，DPS の制御要素を有限状態機械 (FSM) に対応づけている[9]．この中で，DPS は3つの要素

（1） 制御—必要とされる系列で機能を起動する機構

（2） 機能—DPS の数値処理，入出力処理

（3） データー機能の入出力，制御への入力

からなり，（図-2 参照）の DPS の動作は“機能フロー”と呼ばれる機能系列で表現される．ある要求記述の制御構造が一貫性を持っていることは，それに対応する FSM が決定的であるかどうかを調べることにより検証される．完全性は FSM が完全に定義されていること，また，到達性は FSM の各状態が START から END までの経路上にあることとして定義されている．

詳細化のあるレベルの要求記述は，FSM の直列接続，並列接続で表現されるが，さらに詳細化するため

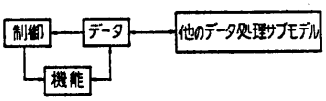

図-2 データ処理サブシステム (DPS) の有限状態機械モデル

に FSM の1つの状態はある FSM で表現される．これは FSM の階層化である．

3.3 アサーションを用いた検証

要求記述の一貫性，完全性の検証は意味的というより構文的なものに限られることが多い．これは対象システムを表現している他の記述が存在せず，正当性の根拠がないためである．プログラムの正当性検証のため，プログラムの入出力関係をアサーションとして付加することがある[33]．計算機の手続き実行を指示するためにはプログラムだけでよいが，その正しさの保証を与えようとする場合には他の観点からの記述（この場合は手続きからの観点と入出力関係という観点）をアサーションの形で記述することが有効である．

筆者らは要求仕様記述におけるアサーションを提案し，その言語定義，処理系の開発を進めている[34]．これはプログラムの検証と同様に，要求定義者にその要求記述の意図，要求記述対象システムについての知識をアサーションとして記述させることにより，要求記述の意味的解析，検証を行おうとするものである．

要求記述では脚注とか注釈として自然言語での記述が併記されていることが多い．また使用されるラベルの意味も重要である．これらは要求記述者間，また要求記述者と設計者の間での正確な伝達の補助になり得るが，計算機処理の対象にはならない．我々はこのような情報を検証のために有効に利用する．すなわち，脚注や注釈で伝えられる情報をアサーションとして記述し，計算機処理するものである．

アサーションのために次の3つの概念が与えられている．(図-3 参照)

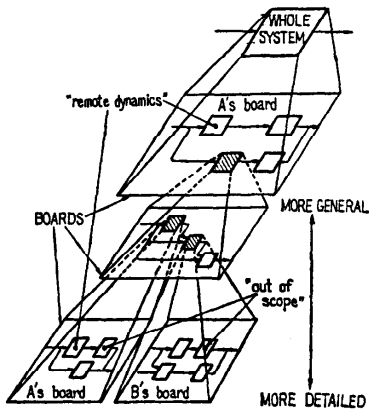

図-3 要求の段階的詳細化とそれに対するアサーション

(1) 負の要求 (negative requirements)

(2) 複数の要求定義者間の関係 (out of scope)

(3) 離れた要求要素間の関係 (remote dynamics)

要求記述の対象とするものは一般に限定された範囲にとどまるものではなく，要求という集合の補集合は明確でない．このため，要求定義者は“負の要求”の記述で，“正の要求”を定義することはできない．要求定義には正の要求を正しく行うしかないが，検証という観点からは，“負の要求”は非常に有効な武器となる．たとえば，ある機能が用いられるシステム環境下では，ある要求機能の系列Sは起動されることがないことを要求定義者が知っていたとする．これをアサーションとして記述することにより，Sに含まれるすべての機能がなくとも機能Fを含む処理系列が要求システムに存在するかどうかを検証することが可能となる．

複数の要求定義者間の関係は，他の要求定義者の記述内容と自分の記述内容との関係の記述であり，単に各定義者の定義しているサブシステム間のインタフェース的記述でなく，そのサブシステムと直接外部インタフェースを持っていない要求要素との間の関係をアサーションとして述べる．我々の扱う要求記述は自然言語的な言語で書かれたものを対象としており，他の定義者の記述を容易に理解できることが多く，このアサーション記述は難しくない．

離れた要求要素間の関係とは，要求定義の詳細化の階層における異なる階層に属する要求要素間の関係の記述であり，ある要求定義者が詳細化している親の要求要素以外の上位レベルの要求要素と，詳細記述中の要求要素の関係をアサーションとして述べるものである．

これらは静的な要求要素構造の解析に属すが，動的な意味を付加した解析をも考えられており，シミュレーションのように多大な費用と時間を必要としない，動的なシステム要求記述の機能的検証を意図している．

我々はこのようなアサーションを記述するための言語 ASL (Assertion Statement Language) とその解析システム ASA (Assertion Statement Analyzer) の開発を進めている．また，一般的に行われている一貫性検証，完全性検証で検証されたシステムの持つ性質と我々の ASL での記述の関連も明らかにしている．

4. おわりに

システム開発の初期段階であればあるほど，開発作

行の誤り修正は容易であり，誤りを見逃がした場合の影響は大きい．それゆえ，要求定義や基本設計のようなシステム開発初期段階に充分な時間をかけ，またできる限り詳細な検証を行う必要がある．しかし，要求記述の検証は，要求記述が手続きでないため計算機の動作で意味を調べることができないし，また，要求記述の多くが現実世界とのインタフェース記述であるため計算機による抽象化が難しいなどの問題が含まれている．

今後，要求記述手法の確立とともに，読み易さ，記述性のよさなどを備えた要求記述言語の開発が望まれる．要求記述はプログラミングにおけるアルゴリズム記述とは異なり，情報システムの入出力関係，データ間の関係，システムと外部環境の関係などの非手続き的記述であり，その言語の設計思想は自ずから異なる．優れた要求記述言語を用い，検証された要求記述を基礎とすることによりシステム開発は容易なものとなり，設計活動，プログラミング，保守まで簡略化されるであろう．

適当な知識データベースと形式的に記述された要求仕様定義を用いて，自動的に目的システムを構成することが究極的な目的であるが，それまでの道程は実際の大規模システムを考えるとまだかなりある．そこで現実的なサブゴールとして，複雑な知識・判断の必要な設計作業は人手に託し，その設計に対する仕様の管理，要求の意図の伝達などを正確に行うことを要求定義技術では目的としている．このような管理・伝達を目標と考えるとき，日本語による要求記述，その解析手法の確立もまた重要な問題として提起される．

参 考 文 献

1) Boehm, B. W.: "Software and its Impact: A Quantitative Assessment", Datamation, Vol. 19, pp. 48-59 (May 1973).
2) 伊藤, 岡: "ストラクチャード・プログラミングを採用した効果" IBM REVIEW, 58, pp. 56-63 (Jan. 1976).
3) Thayer, T. A. et al.: "Software Reliability Study: Final Technical Report", TRW Report 75-2266 (Mar. 1976).
4) Alford, M. W. and Burns, I. F.: "R-NETS: A Graph Model for Real Time Software Requirements", Proc. MRI Symposium on Computer Software Engineering, pp. 97-108 (Apr. 1976).
5) Liskov, B. and Zilles, S.: "An Introduction to Formal Specification of Data Abstractions", in "Current Trends in Programming Methodology", pp. 1-32 (1977).
6) Bell, T. E. and Thayer, T. A.: "Software Requirements: Really a Problem?", Proc. 2nd International Conf. on Software Engineering, pp. 66-68 (Oct. 1976).
7) Boehm, B. W. et al.: "Quantative Evaluation of Software Quality", ibid pp. 592-605.
8) Belford, P. C. et al.: "Specifications: A Key to Effective Software Development", ibid, pp. 71-79.
9) Salter, K. G.: "A Methodology for Decomposing System Requirements into Data Processing Requirements", ibid, pp. 91-101.
10) Ramamoorthy, C. V. and So, H. H.: "Software Requirements and Specification: Status and Perspectives" (Aug. 1977).
11) Bell, T. E. and Bixler, D. C.: "A Flow-Oriented Requirement Language" Proc. MRI Symposium on Computer Software Engineering, pp. 109-121 (Apr. 1976).
12) Boehm, B. W. et al.: Some Experience with Automated Aids to the Large-scale Reliable Software", IEEE Trans. on Software Eng., Vol. SE-1, pp. 125-133 (Mar. 1975).
13) Bratman, H. and Court, T.: "The Software Factory", Computer. pp. 28-37 (May 1975).
14) Ross, D. T.: "Structured Analysis (SA): A Language for Communicating Ideas", IEEE Trans. on Software Eng. Vol. SE-3, No. 1, pp. 16-34 (Jan. 1977).
15) Ross, D. T. and Schoman, K. E.: "Structured Analysis for Requirements Definition", ibid pp. 6-15.
16) Ross, D. T. et al.: "Software Engineering: Process, Principles, and Goals", Computer, pp. 17-27 (May 1975).
17) Dickover, M. E. et al.: "Software Design using SADT" Proc. ACM National Conference (Oct. 1977).
18) Boehm, B. W.: "Software Engineering", IEEE Trans. on Comput, Vol C-25, pp. 1226-1241 (Dec. 1976).
19) Rigo, J. T.: "How to Prepare Functional Specifications", Datamation, pp. 22-24 (Aug. 1971).
20) Teichroew D. and Hershey, E. A.: "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing System", IEEE Trans. on Software Eng. Vol. SE-3, No. 1, pp. 41-48 (Jan. 1977).
21) ISDOS Project, "Problem Statement Language (PSL) User's Manual", (Mar. 1975).
22) ISDOS Project, "Problem Statement Language Version 3.0 Language Reference Manual", ISDOS Working Paper, No. 68 (May 1975).
23) ISDOS Project, "Problem Statement Analyzer

Reports (Version A2.1)" ISDOS Working Paper, No. 99 (Nov. 1975).
24) ISDOS Project, "Problem Statement Analyzer (Version A2.1) Users Manual" ISDOS Working Paper, No. 90 (Oct. 1975).
25) ISDOS Project, "Problem Statement Analyzer Command Descriptions (Version A2.1)" ISDOS Working Paper, No. 91 (July 1975).
26) Bell, T. E. et al.: "An Extendable Approach to Computer Aided Software Requirements Engineering" IEEE Trans on Software Eng.
27) Wirth, N.: "Programming Language PASCAL", ACTA Information, Vol. 1, No. 1 (1971).
28) Dahl, O. J. et al.: "Structured Programming", New York, Academic Press, (1973).
29) Petri, C. A.: "Communication with Automata", Translation, RAD C-TR-65-337, Vol. 1, New York. (1966).
30) Rose, C. W.: "LOGOS and the Software Engineer" Proc. AFIPS, Vol. 1, pp. 311-323, (1972).
31) Rose, C. W. and Albanan, M.: "Modeling and Design Description of Hierarchical Hardware/Software Systems" Proc. Design Automation pp. 421-430 (1975).
32) Parnas, D. L.: "A Technique for Software Module Specification with Examples", CACM Vol. 15 No. 1, pp. 330-336 (May 1972).
33) Manna, Z.: "Mathematicol Theory of Computation" McGraw-Hill (1974).
34) Agusa, K., Ban, R. and Ohno, Y.: "Verification of Requirements Description", 12th HICSS, pp. 131-139 (Jan. 1979).
35) 鳥居, 二木, 真野: "プログラミング方法論の展望" 情報処理, Vol. 20, No. 1, pp. 22-43 (1979).
36) 國井: "ソフトウェア工学—要求仕様技術" bit, 8 (1978).

(昭和54年3月5日受付)