ソフトウェア工学 83－7
（１９９２．２．６）

日常的世界観にもとづく言語ＮＡＩＶＥによる実行可能な仕様記述

日野克重

富士通㈱

日常的世界観にもとづく実行可能な仕様記述言語NAIVE を提案する．当言語は，実世界を自然かつ直接的に記述することを目標とした言語である．

本稿では，当言語について，その背後にある世界観，言語仕様，および記述例を中心に述べる．

The Specification Language NAIVE that is based on Everyday World View

Katsushige HINO

FUJITSU Ltd.

A executable specification language NAIVE is proposed. We can develop reactive and complicated software systems more easily using the language NAIVE.

In this paper, the world view behind the language NAIVE, the language specification of NAIVE, and some examples of description in NAIVE are presented.

1．はじめに

ソフトウェア開発の生産性および品質の向上にとって，ソフトウェア記述言語は最も根本的な役割を果たすものと考えられる．なぜなら，ソフトウェアの開発とは，つきつめるところ「問題（要求）自体の表現とソフトウェア記述言語との間の写像操作」と言うことができ，「ソフトウェア開発のあらゆる困難は，それら両者の間の階位的，構造的乖離からきている」と考えられるからである．ここでもし，「問題自体の表現に近いレベルで仕様を書けば，それがそのまま実行もできる」ような言語が実現できれば，設計からプログラミング，ならびにテスト工程が大幅に省略され，ソフトウェア開発の困難のおおかたは解消する．

こうしたねらいをもつ言語は，実行可能な仕様記述言語とよばれ，すでに種々のものが提案されている．それらは，論理型言語，関数型言語，代数型言語などと分類されることもある．また，オブジェクト指向が強調されるものもある．しかし，現時点では，これらはおうおうにして，記述・読解性の点で難解，動的システムや大規模システムの記述が困難，データベース操作などを含む実用システム全体を作る能力が十分でない，いまだ計算機概念が払拭しきれていないなどの難点を抱えており，その本来のねらいを十分達成したものは出現していない．

ここで提案する言語NAIVE は，日常的世界観にもとづき，自然語に近い形で問題対象を記述する言語である．具体的には，実体，関係，行為，変化，行為主体（エージェント）などの基本概念からなる日常的世界観にもとづいており，データ構造，ポインタ，プロセス，データ入出力，同期通信などの従来の計算機プログラミング上の概念を隠蔽した言語である．さらに，静的事態の記述についてはすでに実績のある述語論理をカバーしている上，並行処理などの実世界の動的側面の記述までを一つの論理体系の中に収めた言語である．データベースプログラム言語としての機能も備えている．すでにインプリメントされており，実用規模のソフトウェアシステムの記述実験も進められている．

本稿では，言語NAIVE について，その背後にある世界観，言語仕様，および記述例を中心に述べる．

2．言語NAIVE の世界観

仕様記述言語の設計において，その言語の背後に明確な世界観を設定しておくこと，そして，そのとき，いかに自然な世界観を設定するかが，決定的に重要である．言語NAIVE では，それが依拠する世界観として，日常的世界観を採る．

日常的世界観を採ることの妥当性については，およそ次のように考えれば納得されるだろう．すなわち，われわれが開発するソフトウェアシステムの多くは，われわれの日常生活の中に埋め込まれて使用される．したがって，このようなシステムに対する要求仕様も，その原初時点では，日常的概念や言葉で表現されているだろう．そもそも動的かつ複雑なシステムの開発が必要になるのも，われわれの日常生活の中ですでに高度な論理処理や並行処理が行われているからである．そうだとすれば，こうしたソフトウェアの仕様を自然に記述する言語を設計する上で，われわれが日常世界を眺めるときの世界観（日常的世界観）や自然語に規範を求めるのが妥当である．また，日常的世界観や自然語の中には，こうしたソフトウェアを記述するのに必要な要素はすべて備わっているはずである．

ところで，その自然語の論理的な側面を形式化した形式言語としては，すでに述語論理やその拡張としての内包論理（時間論理はその特殊形である）などがある．それらは，個体，関係，および時間の概念を基本的枠組みとした言語であり，その点で自然語の最も基底にある世界観を反映した言語であるといえる．言語NAIVE もその枠組みを引き継ぐ．しかし，日常的自然語をもう一度観察してみると，これらの既成の論理系では，実世界の記述あるいは動的システムの記述にとって重要ないくつかの概念がとり残されていることがわかる．それは，以下に示すような概念である．

(1) 実体に関する概念　－種，属性，実在性－

種：　自然語における普通名詞は，個体の属する種（クラス）を表すものである．この普通名詞は，自然語表現においては動詞や形容詞などとならんで中心的役割を果たしている．どの個体もなんらかの種に属しており，かつ，種の指定が人間が頭の中で行っているであろう情報検索においても効率上有利であることなどがその理由であろう．ソフトウェアの仕様記述においても，種概念は，プログラムの自動生成や仕様の検査において有効にはたらく．

属性：　これは，自然語における「～の住所」や「～の年令」のような属性名詞に相当する概念である．既成の論理系でも関数の概念がこれに相当するが，そこでは，関数は「一意的な関係」の別名として，いわば記述の便法として扱われている．言語NAIVE では，実体概念に付帯する本質的な概念として属性概念を導入する．実際上も，関係概念とは別に属性概念を導入することは，アクセス効率のよいデータ構造を自動設計する上で不可欠である．

実在性：　これは，自然語における「有る」や「無い」に相当する概念である．述語論理のように本来静的な世界を記述する言語においては，個体の生成や消滅は生じないので本概念は必要ないが，世界の動的側面を記述する言語では，この概念は必要となる．実際オペレーティングシステムやシミュレーションシステムなどでは，個体の生成・消滅に相当する処理は頻繁に現れる．また，「物（たとえば動物）は生成された瞬間から自発的に行動をはじめる」という日常的世界観に着目すれば，この実在性の概念は，後述の行為主体の概念とも有機的な関連をもつべきこともわかる．

(2) 行為に関する概念　－行為，主体，脈絡－

「世界は変化するものであり，世界の変化はかならず，なにがしかの主体による行為によってもたらされる」というのは，日常的世界観の中の基本的なものの一つであろう．この点から，実世界の動的側面を自然に記述するために行為の概念を導入することは自然なことであると考えられる．

さらに，行為の概念は，おのずと，行為主体の概念および行為脈絡の概念を導く．実世界の複雑な現象の系列も，いくつかの行為主体のおこなう脈絡のある行為の重ね合わせとして捉えるとき，はじめて整然と理解できる．同様に，動的並行システムの記述および検証に際しても，行為主体の概念と行為脈絡の概念の導入は，仕様全体を直和的に分割して把握する上で有効にはたらく．

この点，時間概念はあっても行為概念が導入されて

いない伝統的論理（内包論理や通常の時間論理）では，行為主体や行為脈絡の概念が現れる余地がないので，連接，反復，および入れ子構造などの世界変化の脈絡や，行為主体の概念に本質的に依拠する並行処理が自然には記述しにくい．行為概念をもたないこのような言語で動的システムを記述する場合，問題文には現れない多数の内部的な状態識別子を使って，状態遷移的プログラミングをせざるを得なくなる．その結果，仕様記述の自然な読解性が損なわれる．また，対象システムが少し大規模になれば，状態数が爆発的に増加し，システムの記述および検証が困難になる．

(3) 世界への外乱に関する概念　－神，天使－

日常的世界観においても，世界は完全に閉じた自律系とは考えられていない．世界を創成するものは，世界の外にあり，かつ，世界ができた後も，なんらか目にみえないものからの外乱があるものと捉えられている．ソフトウェアシステムの場合も，システム操作者の操作コマンドの投入や，当該システムの下位システムや隣接システムからの種々の事象通知などが，この外乱に相当する．記述した仕様の正当性が外乱の生起系列に依存する場合もあるように（無飢餓性の議論はその典型例である），動的システムの記述にとって，外乱が記述できることは本質的である．神と天使の概念は，それぞれ，システムへ外乱を発生させるものとその外乱自体とを日常的に言い表した概念である．

上に挙げた諸概念を従来の述語論理的世界観に融合的に付加することにより，新たに図－1および図－2のように要約できる統一的世界観が得られる．言語NAIVE は，基本的にこの世界観にもとづく．

なお，この世界観を構成する基本概念を個々に見ればさほど新奇なものはない．全体として，むしろトリビアルに見えるかもしれない．しかし，この世界観について筆者が強調したい点は以下にある．

（ｉ）この世界観の中では，それらの基本概念が，断片的な寄せ集めではなく，相互に有機的に結合された統一体をなしていること．

（ii）この世界観は，マルチエージェントによる並行協調動作やオブジェクトデータベース概念などの多くのソフトウェア概念を内に含んだ一つのソフトウェアモデルになっていること．

（iii）そして，それは，われわれの日常的世界観に近く，かつ，従来のプログラミング概念を完全に隠蔽したものであるがゆえに，ソフトウェアモデルとして，了解性，連想性，および抽象性の高いものになっていること．

3．言語仕様

言語NAIVE の基本詞を表－1に示す（これだけでは当言語の言語仕様を正確に述べたことにならないが，その点は，4章の記述例で補うことにする）．

(1) 意味論的観点からの特徴

意味論的観点からは，次の特徴がある．

(a) 当言語の言語仕様は，2章で示した日常的世界観を反映するよう設計されている，すなわち，述語論理を包含するとともに，行為主体（エージェント）やそれらの間の並行協調動作などを自然に記述するための基本詞を備えたものになっているなお，その世界観および言語仕様は，モンタギューの内包論理を拡張した「行為主体概念をもつ時区間型内包論理」と呼ぶべき一つの論理体系として形式化することもできる（この論理体系については別途詳細に報告する予定である〔7〕）．

(b) ポインタ，プロセス，データ入出力，および同期通信などの，計算機やＯＳに関わる概念は排除されている．

(2) 構文論的観点からの特徴

構文規則は，以下のような点に留意して設計されている．

(a) 基本詞は，日常談話においてよく使われる自然語の中の易しい語彙と対応がある．また，構文も自然語の構文に近い．

(b) 自然語と同様の品詞分類がある．そして，使用するコトバの品詞は文脈から判断されるので，明に宣言する必要がない．（これまでのプログラム言語では，データ構造やデータ型を手続き部とは別に宣言させるのが常であったが，日常談話の場合を思えば，それはやや不自然なことである．）

(c) 世界観を明に意識するようなメタ用語（例．class，method，precondition，procedureなど）は記述させない．（世界観自体は，明示すべきものではなく，言葉づかいの中に暗に含まれるべきものであろう．）

(d) 日常談話では省略しないようなコトバは，当言語でも明に記述させる．（たとえば，Prologとは異なり，all やsomeなどの限定詞は明に記述させる．）

こうした配慮は，従来のソフトウェア言語では閑却視されてきたことであるが，仕様記述言語の記述性，理解性，習得のしやすさ，および，実世界記述の快適性の点で効果があるものと思われる．

① 実体と関係　：　世界の中にはいくつかの実体が存在し，それらの間には種々の関係が成り立っている．実体とは，種が付与された個体のことである．実体は，実在性をもっている．実体はまた，いくつかの属性をもちえる．なお，成立している関係のことを事態とよぶ．世界の状態とは，事態の集まりのことである．

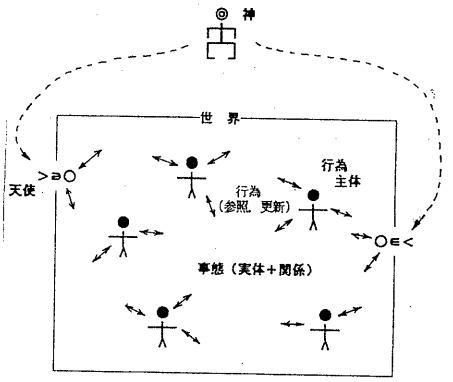
② 行為と変化　：　世界の状態は時間とともに変化する．そして，その変化は行為によってもたらされる．行為とは，世界の状態を参照しながら新たに事態を成立（非成立）させることである．行為には脈絡がある．

③ 行為主体　：　行為にはかならずその行為の主体が唯一存在する．行為主体になり得るのは，実体と天使だけである．行為主体たちは，それが実在しているかぎりその行動本性にしたがって，世界をながめながら，おのおの独立に行動する．

④ 神と天使　：　世界を創成するのは神である．神の意思は天使を介して世界に伝えられる．神は，世界の不変的性質，すなわち，どんな実体および関係が世界に現れ得るか，ならびに，実体および天使がどう行動するものか（それらの行動本性）を知っている．天使は行動するが，実在しない．

⑤ 論理　：　世界はつねに論理にしたがっている．

図－1　言語NAIVE の世界観



図－2　言語NAIVE の世界観（図解）

(3) ソフトウェアツールとしての特徴

ソフトウェア開発ツールとして見た場合，当言語には次のような特徴がある．

(a) この言語で書かれた仕様からCプログラムおよびデータ構造が自動的に生成され，かつそのCプログラムを実行させる実行系も備えていることから，「仕様記述するだけでソフトウェア開発の全過程が終わる」という理想をかなりの程度まで現実化するものである．（当言語の実現方式についても別途詳細に報告する予定である〔8〕．）

(b) 基本的に，開発者は，他のツールやＯＳや計算機に関する知識がなくてもソフトウェア開発ができる．その点で，ソフトウェア開発者の習得すべき技術や知識を軽減させる．その分，よりよい仕様設計などの本質的な作業に注力しやすくなる．

表－1　言語NAIVE の基本詞

品詞分類	基本詞	意　　味
定義詞	≡	行為主体の行動本性，ならびに，コマンドや単文の意味を定義するのに使う．自然語における「～とは～ということである」という表現に相当する．
叙述詞	＝ ≠	単文の肯定．自然語における「～である（is）」に相当する． 単文の否定．自然語における「～でない（is not）」に相当する．
普通名詞	物 数 字 （その他任意に定義可能）	種（クラス）を指定するものである．
実在詞	be nil	実在性．自然語における「有る（there is）」に相当する． 非実在性．自然語における「無い（there is not）」に相当する．
属性詞	（任意）	自然語における「～の父」や「～の持物」などの属性名詞に相当する．
関係詞	（任意）	自然語における形容詞や前置詞に相当する．
行為詞	（任意）	自然語における動詞に相当する．
限定詞	all some many the	自然語における「すべての」に相当する． 自然語における「ある」に相当する． 自然語における「いくつの」に相当する． 自然語における「その」に相当する．
論理詞	～ and or → not	文否定． 連言． 選言． 含意． 形容否定
接続詞 （様相詞）	☆ DO ⇨ IF WHEN WHILE UNTIL REPEAT and or WHERE FOR	行動本性．実体の行動本性を表す．（通常省略される．） 行為化．自然語における「～であるようにする」に相当する． 連接．自然語における「～して，その次に～する」に相当する．（通常省略される．） 判断．自然語における「もし～ならば」に相当する． 待機．自然語における「～になったとき」に相当する． 継続．自然語における「～であるあいだ」に相当する． 継続．自然語における「～になるまで」に相当する． 反復．自然語における「何回～する」に相当する． 同時実行． 選択実行． 修飾．自然語における「ただし～である」に相当する． 限定．
算術詞	＋，－，＊，／，＃（集合の要素数），sum(合計) など	
数詞	（任意の数字の列）	
助詞	: . ..	普通名詞と固有名詞あるいは指示詞との連結． 限定詞，関係詞（句），および普通名詞の連結． 属性詞の連結．
固有名詞	（任意）	個体を示す定数．
指示詞	（任意）	変数．
補助記号	括弧や引用符など．	

4．記述例

本章では，NAIVE による仕様記述例をいくつか示しそれらから見て取れる当言語の特徴について述べる．図－3，図－4，および図－5は，それぞれ，「飲酒する哲学者」，「図書館管理」，および「線型計画」の問題とそのNAIVE による仕様記述例である．

(1) 飲酒する哲学者

本問題は，哲学者の食事問題をさらに一般化したものであり，マルチエージェントによる並行協調動作や多少複雑な論理的処理の要素が盛り込まれた問題である．

この程度の並行・論理問題は，実世界では日常茶飯に見られるものであるが，それでもこれを，メッセージパシング方式にもとづくオブジェクト指向言語や，Prologのような既存の論理型言語で記述しようとすると，かなり複雑なプログラムになるであろう．

その点NAIVE ではこの問題を図－3に示したように記述できる．具体的には次のような利点が見て取れる

(a) 自然語で実世界を記述するのに近い感じで問題を記述できる．

(b) 問題文に現れる語彙と仕様記述に現れる語彙は，おおむね一致している．仕様記述上には，問題文に現れない状態変数や中間変数は現れないし，計算機やOSに関わる概念も一切現れていない．そのため，与えられた問題と記述した仕様との一致性が見やすくなっている．〔注〕ここで言った「問題と仕様の一致性」の確認は，原理的に形式化不可能な事柄であり，人間がやるほかに方法がないことである．しかし，それこそがプログラム（仕様）の検証の最終目標であることを思い起こせば，それが「しやすいことは，仕様記述言語の最重要の要件であろう．

(c) all やsomeを含む複雑な条件での待機（同期）処理が接続詞WHENを使って簡潔かつ宣言的に書けている．このことにより，仕様（プログラム）の正当性の検証が容易になる．たとえば，生存性（liveness property ；のどが渇いた哲学者はいずれ酒を飲めること）や安全性（safety property ； 同じ酒ビンを複数の哲学者が同時に飲むことはないこと）の要請が満たされていることは，仕様記述上からほぼ自明的に読み取れる．

(d) 仕様全体をいくつかの独立な部分仕様の和として直和的に記述できている．たとえば，この例題ではシステム実行中に動的に哲学者やテーブルや酒ビンが追加されてもよいようにコマンドが定義されているが，他の定義部分ではそのことを特に意識してはいない．

(e) 述語論理や並行処理になじみがないプログラマにとっても読み書きしやすい．

(2) 図書館管理

本問題は，集合操作をともなう問い合わせ処理やデータベースの検索・更新の要素を含んでいる点で，事務処理分野に典型的な問題の一つである．

先の例に現れた特徴に加えて，この記述例からは，以下のような当言語の特徴が読み取れる．

(a) このシステムに対する各操作コマンドの意味が宣言的に書けている．すなわち，いずれのコマンドについても，「それらが投入されたら，（瞬時に）こういう状態にせよ」という前件－後件の形で書けており，手続き型言語の場合のように途中の「計算過

程」を書いてはいない．

(b) 図書館管理システムの仕様に関するすべてが一つの仕様記述の中にもれなく記述されている．そして，各操作コマンドが投入されたときシステムが何をするかが，一箇所にまとめて記述されている．同じことをたとえばC言語のような従来型プログラム言語で記述する場合は，記述量がさらに多くなるとともに，1つのシステムが多数のモジュールから構成されることになるが，それら相互の関連の統制は言語記述の範囲外になっている．

(c) データベースへの入出力操作の意識が不要になっており，SQLなどのデータベースアクセス言語についての意識が不要になっている（にもかかわらず，本例の仕様記述はそのままで，データベースを検索／更新するアプリケーションとして動作可能である）．

これにより，当言語は，いわゆるホスト言語とデータベース言語のインピーダンスミスマッチを解消したデータベースプログラミング言語にもなっている．

(3) 線型計画問題

本問題は，先の2題のような動的な実世界を記述する問題とは異なり，静的，数論的問題と呼ぶべきものである．この例からは，以下のような当言語の特徴が読み取れる．

(a) 本来静的な問題は，そのまま宣言的に記述できる．

(b) 動的実世界の記述と数論的問題の記述という互いに性格の異なった問題が，当言語の一様な表現形式で記述できる．

なお，この問題の実行は現在はシラミツブシ方式（自然数のとり得る範囲は有限におさえておく）で行われ，実用的ではないが，それにしてもNAIVE ではこうした問題の表現および実行が可能である．

〔飲酒する哲学者〕

問題：哲学者達が飲酒する．哲学者の傍には何個かのテーブルが置ける．それらのテーブルは複数の哲学者によって共有されているかもしれない．テーブルの上には，いろんな種類の酒ビンが何個でも置ける．
哲学者達は，一度に何種類かの酒を飲みたくなり，そのときには，それらすべての種類の酒ビンが飲めるようになると飲酒をはじめる（他の哲学者が飲んでいる酒ビンは飲めない）．
渇きが癒えたら，飲んでいたビンは，元のテーブルに戻される．
なお，哲学者達は，自分の傍にあるテーブルの上の酒しか飲めない．
この饗宴のシミュレーションを行う．

〔言語NAIVE での記述〕

```
the.哲学者 :p ≡                                  * 哲学者とは次のように行動するものである *
   WHILE  the.哲学者:p=be                         * すなわち，それが実在するかぎり          *
     DO                                           * 以下をおこなう                          *
        UNTIL the.哲学者:p=thirsty                * のどが渇くまで                          *
           the. 哲学者:p=tranquil                 * 平静であり，のどが渇くと，              *
        WHEN (for all.need(the. 哲学者:p,*).酒種:m * かれが欲するすべての酒種について      *
                some.(of-the.酒種:m).              * 少なくとも一本づつ，                    *
                  (near-the.哲学者:p).ビン=not-drunk * かれの近くにだれも飲んでいないビンが *
           DO                                     * あるという状態になるのを待って          *
             FOR all.need(the.哲学者:p,*).酒種:m  * それらのビンを                          *
                the.哲学者:p=drinking-            * 飲み始める                              *
                   some.(of-the.酒種:m).(not-drunk). *                                      *
                     (near-the. 哲学者:p).ビン     *                                       *
             WHEN  the. 哲学者:p=not-thirsty       * そして渇きが癒えると                    *
                the.哲学者:p=not-drinking-all.ビン * それらを飲むのをやめる              *
           END                                    *                                        *
     END                                          *                                        *
'飲みたい :p :x :y :z' ≡                          * 飲みたいコマンドが投入されると          *
     DO                                           *                                        *
        the.哲学者:p=need-the.酒種:x  and         * その哲学者p は，（のどの渇きを覚え）    *
        the.哲学者:p=need-the.酒種:y  and         * このコマンドで指定された酒種x,y,z を   *
        the.哲学者:p=need-the.酒種:z              * 欲するようになる                        *
     END                                          *                                        *
'飲みたくない :p' ≡                               * 飲みたくないコマンドが投入されると      *
     DO(the.哲学者:p=not-thirsty)                 * その哲学者は，のどの渇きが癒える        *
'哲学者生成 :p' ≡                                 * 哲学者生成コマンドが投入されると        *
     DO(the.哲学者:p=be)                          * 名前p をもつ哲学者を誕生させる          *
'テーブル生成 :t' ≡                               * テーブル生成コマンドが投入されると      *
     DO(the.テーブル:t=be)                        * 名前t を持つテーブルを生成する          *
'配置 :a :b' ≡                                    * 配置コマンドが投入されると              *
     DO(the.物:a=by-the.物:b)                     * 指定された物a と物b を隣同士に配置する *
'ビン生成 :m :t' ≡                                * ビン生成コマンドが投入されると          *
     DO                                           *                                        *
       some.ビン:b=be    and                     * 新たなビンを作る                        *
       the. ビン:b=of-the.酒種:m  and             * そのとき，そのビンの酒種をm とし        *
       the. ビン:b=on-the.テーブル:t              * テーブルt の上に置く                   *
     END                                          *                                        *
(the. ビン:b=drunk) ≡                             * ビンが飲まれているということは，だれか *
     (some. 哲学者=drinking-the.ビン:b)           * がそのビンを飲んでいるということである *
(the. 哲学者:p=tranquil)≡                         * 哲学者が平静であるということは          *
     (the.哲学者:p=not-thirsty)                   * かれののどが渇いていないということ      *
(the. 哲学者:p=thirsty) ≡                         * 哲学者ののどが渇いているということは    *
     (the.哲学者:p=need-some. 酒種)               * なにがしかの酒種を欲しているということ *
(the. ビン:b=near-the.哲学者:p) ≡                 * ビンが哲学者の近くにあるということは    *
     (the.ビン:b=on-some. テーブル:t  where        * そのビンが                              *
        (the. テーブル:t=by-the.哲学者:p  or       * その哲学者の傍にあるテーブルの上にある *
         the. 哲学者:p=by-the.テーブル:t))         * ということ                              *
```

図−3　飲酒する哲学者問題のNAIVE による仕様記述

〔線型計画問題〕

問題：次のような制約条件のもとで，x+2y+3z の値は最大いくつになりえるか．なお，x,y,z は，自然数の範囲を走るものとしてよい．

$$5x+6y+2z \leqq 80$$
$$3x+y+5z \leqq 65$$
$$2x+7y+3z \leqq 93$$

〔言語NAIVE での記述〕

```
'解け' ≡                                          * 解けコマンドが投入されると *
     DO                                           * 以下を行う，すなわち       *
       some.数:a=list                             * ある数a を表示する         *
       WHERE                                      * ただし，その数a は         *
         (solution(a)  and                        * 条件solutionを満足し，かつ *
          (for all.solution.数:n                  * 同じく条件solutionを満足す *
                a≧n))                             * る数の中で最大の数である   *
     END                                          *                            *
(solution(the. 数:n))≡                            *          def               *
     (for some. 数:x,some.数:y,                   * solution(n) ≡               *
          some. 数:z,the. 数:n                    *   ∃x ∃y ∃z ∈N              *
          (n = x + 2*y + 3*z)   and               *     {n=x+2y+3z ∧           *
          (5*x + 6*y + 2*z ≦80) and               *      5x+6y+2z≦80  ∧        *
          (3*x +   y + 5*z ≦65) and               *      3x+ y+5z≦65  ∧        *
          (2*x + 7*y + 3*z ≦93))                  *      2x+7y+3z≦93}          *
                                                  *                            *
```

図−5　線型計画問題のNAIVE による仕様記述

5．記述・実行実験と評価

当言語の記述性，実行可能性，ソフトウェア開発の生産性・品質向上への効果，ならびに，実用システム開発への適用性などを確かめるために，以下の実験を

〔図書館管理問題〕
問題：蔵書の保管および貸出しなどの管理を行う図書館管理システムを作る．
具体的には，次のようなコマンドを処理できるシステムでなければならない．
－本を（蔵書として）登録／抹消する．
－本を貸出／返却する．なお，どの会員も５冊以上の本を借りだすことはできない．
－本の題名を指定して，それが即時貸出可能か否かを表示する．もし否なら，それを借りだしている会員名をすべて表示する．
－特定の著者の本で当図書館に登録されている本の題名を表示する．
－特定の著者の特定の分野の本で即時に貸出可能な本の題名を表示する．
－会員を登録／抹消する．

〔言語NAIVE での記述〕

```
'本登録 :c :b' :a :r' ≡                    *  本登録コマンドが投入されると          *
    DO                                     *  以下を行う                            *
      the.コピー:c=be     and              *  指定された登録番号c のコピーを登録し，*
      the.本:b=be         and              *  指定された題名b の本を登録し，        *
      the.コピー:c=of-the.本:b    and      *  そのコピーは，本b のコピーであるとし，*
      the.本:b..著者= the.字:a    and      *  その本の著者名はa であって，かつ      *
      the.本:b..分野= the.字:r             *  その本の分野名はr であるとする．      *
    END                                    *                                        *
'本登録抹消 :c' ≡                          *  本登録抹消コマンドが投入されると      *
   the.コピー:c=nil                        *  指定された登録番号c のコピーを抹消する*
'貸出 :c :m' ≡                             *  貸出コマンドが投入されると            *
    IF  the. 会員:m=be  and                *  その会員m が登録されていて，かつ      *
        #((in-the. 会員:m).コピー) <5      *  まだ５冊以上借りだしていなければ      *
      THEN   the.コピー:c=in-the.会員:m    *  そのコピーc をその会員m に貸し出す    *
      ELSE   " 貸出不可" = list            *  そうでなければ，メッセージを出力する  *
'返却 :c' ≡                                *  返却コマンドが投入されると            *
   the.コピー:c=inlibrary                  *  そのコピーは図書館に戻る              *
'貸出状態表示 :b' ≡                        *  貸出状態表示コマンドが投入さると      *
    IF  some.(of-the.本:b).コピー=inlibrary *  指定された題名b の本のコピーが      *
      THEN                                 *  図書館の中にあれば                    *
        "貸出可能" = list                  *  メッセージ「貸出可能」を出力し        *
      ELSE                                 *  そうでなければ                        *
        ( "貸出不可" = list                *  メッセージ「貸出不可」を出力し        *
          all. 会員:m=list  WHERE          *  その本のコピーを                      *
            (some.(of-the.本:b).コピー =   *  今借りだしている会員の                *
                                in-the. 会員:m) )  *  名前を表示する                *
'特定著者蔵書表示 :a' ≡                    *  特定著者蔵書表示コマンドが投入されると*
  all.本:b=list                            *  本の題名をすべて表示する              *
    WHERE  the.本:b..著者=the. 字:a        *  ただし，その本の著者の名前はa であること*
'貸出可能書籍表示 :a :r' ≡                 *  貸出可能書籍表示コマンドが投入されると*
  all.本:b=list  WHERE                     *  次の条件を満たす本の題名をすべて出力する*
    ( the.本:b..著者= the.字:a    and      *  すなわち，著者がa であって，かつ      *
      the.本:b..分野= the.字:r    and      *  分野がr であって，かつ                *
      some.(of- the.本:b).コピー:c=inlibrary )  *  その本のコピーの少なくとも１冊はいま図書*
                                           *  館にあること                          *
'会員登録 :m' ≡                            *  会員登録コマンドが投入されると        *
   the.会員:m=be                           *  指定された会員m を生成する            *
'会員登録抹消 :m' ≡                        *  会員登録抹消コマンドが投入されると    *
   the.会員:m=nil                          *  指定された会員m を抹消する            *
(the.コピー:c=inlibrary) ≡                 *  コピーが図書館にあるということは，    *
    (the.コピー:c=be  and                  *  それが登録されていて，かつ            *
     the.コピー:c=not-in-all.会員)         *  それをだれも借りだしていないということ*
```

図－4　図書館管理問題のNAIVE による仕様記述

行った．

① 基本例題の記述・実行実験　：　種々の問題を当言語で自然に記述でき，かつ実行できることを確認するための実験．表－２に，それらの例題と当言語での記述行数を示す．それらの記述から自動的に展開出力されたＣプログラムの行数も合わせて付す．

② 他言語との比較実験：同一問題をNAIVE と他の言語で記述し比較する実験．表－３にその結果を示す．

③ 実用システムへの適用実験　：　実用システムへの適用性を確かめるための実験．表－４に対象となったシステムの概要と実験結果を示す．

【評価】

(1) 動的並行問題，データベース操作を伴う事務処理問題，および数論的問題などの広範囲の問題が当言語で自然に記述でき，かつ実行できることを確認した（それぞれどのような仕様記述になるかについては４章の記述例を参照されたい）．

ただし，文字やビットの操作を主体としたソフトウェア（例：エディタやコンパイラのようなもの）については，それをNAIVE で開発することは現実的には適当でないと考えられる．これは，当言語が実世界の実体や関係のレベルで問題を記述する言語であり，ポインタやデータ構造を直接操作する言語ではないことによる．

(2) ソフトウェア開発の生産性および品質に及ぼす効果については，次のようなことがいえる．

(a) NAIVE による問題の記述量は少ない．このことは，ソフトウェアの生産性および品質の向上にとって全ての局面で好影響を及ぼすであろう．実際，今回の実験でもその効果は現れている．

(b) NAIVE を使ったソフトウェア開発では，開発作業のほぼすべてが，当言語による仕様記述という一点に集約される．これにより，ソフトウェア開発の過程自体が単純化され，全体的見通しがよくなる．

(c) 実行テストで検出されるバグが少ない．その理由としては，以下のようなことが挙げられる．

・記述量が少なくなるため，バグ作り込みの母体そのものが小さくなる．

・記述レベルが高い（問題記述のレベルに近い）だけ，間違いを犯しにくいし，見つけやすい．また，その言語仕様から，同期バグや，プログラムの異常終了につながるような低レベルの誤りは，そもそも犯す余地がない．

・与えられた問題に関するすべての情報が１つの仕様記述の中に盛り込まれること，従来型言語での開発の場合のようなデータ構造やモジュール関連の設計の自由度が当言語では無くなる（自動的に行われる）こと，ならびに，変数への破壊代入がないことなどにより，有効な静的解析がやりやすい．

(d) 従来型言語による場合に比べて，全実行ルートの数が少ない．さらには，動的並行システムの場合でも，全実行ルートを静的に数え上げることが可能である．これらにより，テストの網羅度についての確信度が格段に向上する．

たとえば，3章で述べた飲酒する哲学者問題の場合の実行ルート数は高々11個であり（各定義ブロック毎に1個），かつ，それらは相互に独立的であるので，それらの組み合わせのケースを考慮する必要がない．

(3) 実用システムへの適用実験（表－4参照）の結果，対象システムやそれが扱うデータベースが大規模になっても，中小規模のシステム記述で現れた当言語の有効性は，そのまま保存されるという感触を得た．実際，表－4のデータは，従来言語による開発に比して，開発効率は少なくとも数倍であったことを示している．

表－2　言語NAIVE の記述例題

NO	例題	概　要	記述行数 (in NAIVE)	展開行数 (in C)
【並行処理，シミュレーション問題】				
1	迷路	迷路を自由に設定し，それを脱出する．	43	465
2	哲学者の食事	円卓に座った5人の哲学者がフォークをとり合いながら食事をする．	37	371
3	パッケージルータ	複雑な通路の中を複数の荷物がぶつかりを避けながら目的地へ流れていく．	62	920
4	酒類販売店	酒屋の在庫（倉庫，コンテナ，銘柄，本数）を管理し，顧客の注文に機敏に対応する．	37	510
5	座席予約システム	新幹線の座席予約システム．（区間単位での予約，キャンセル自動待ち，代替列車表示等）	155	1541
6	エレベータ	複数台のエレベータが全体として円滑に動作	92	1047
7	ミニ通信管理	ネットワーク上での動的経路選択，流量制御	50	843
8	洗車問題	ランダムに入ってくる車を複数の洗車係が公平かつ効率的に洗う．全体の管理者はいない	31	570
9	喫煙者問題	タバコ，紙，マッチの各資源を分け合いながら，複数の喫煙者が円滑に喫煙を続ける．	33	420
10	分散型ジャンケン	複数人がジャンケンを繰り返し，最後に一人勝ち残る．審判はいない．	49	590
11	Handshake Daisy Chain Arbiter	ループをなしたセル群が権利書をもちまわる各セルは，その権利書を獲得したときのみ，対応するプロセスに資源使用権を与える．	47	624
12	ジョブショップ問題	ランダムに注文される多種類の仕事を多種類の機械で処理していく．同種類の機械でも何台かづつある．	50	720
13	論理回路シミュレータ	加算器回路の設定及び動作シミュレーション	349	2550
14	自動ロック式ドア	自動ロック式ドアの開閉操作の模擬	55	330
15	電話交換機	発呼，接続管理，話中処理などの交換機処理	58	600
16	飲酒する哲学者	哲学者達が何種類かの酒を競合しつつ饗宴をくりひろげる．テーブル配置なども自由．	39	550
17	ガス給油所	客と係とガスポンプが連動してガスを給油する．代金支払い処理もある．	60	947
18	自動販売機	飲料の自動販売機のシミュレーション	115	1030
19	車輪駆動	複数の車輪を順々に回転させていく．	5	156
20	有限バッファ	有限なバッファを介して複数人が並行動作しつつ，ボールの受け渡しをする．	17	250
【事務処理，データベース操作問題】				
21	図書館管理	図書の貸出管理／表示サービスなどを行う．	41	590
22	書店売掛管理	書店の簡単な売掛管理．	44	450
23	部品供給管理	部品データベースの検索処理	13	240
【数論的，論理的問題】				
24	多元多次連立方程式	多元多次方程式／不等式を解く．	30	361
25	線型計画問題	宣言的に記述された線型計画式を解く．（最大化，最小化の表現）	20	280
26	ハノイの塔	円盤の大小関係を維持してそれらを動かせる	8	180
27	福祉国家判定	動的に変化する事態をみながら，その時点での福祉国家を判定する論理問題．	20	325
28	KWIC	本のタイトル群からKWICリストを作る．	58	490

表－4　実用システムへの適用実験

対象システムの機能概要	工程情報データベースへの問い合わせ要求やデータ更新要求に対応するＤＢサーバを作る．なお，当システムが扱うデータベースのデータ件数は，約６万件である．
開発規模見積もり	COBOL プログラムとしては約20K 行．
NAIVE での開発実績データ	記述量　：　2K 行 所要工数　：　1人× 4ケ月 〔内訳〕仕様決定　：　1人月 仕様記述・テスト：　2人月 実行性能調整　：　1人月 検出バグ数：　23 件

表－3　他言語との比較

No	例　題	比較項目	NAIVE	他言語
1	迷路	記述量	43 行	Ada　：　117 行
2	自動販売機	記述量	115 行	Ada　：　359 行
3	書店の売掛管理	記述量	44 行	COBOL：　160 行 Ada　：　128 行
4	哲学者の食事	記述量	37 行	LOTOS：　106 行
5	飲酒する哲学者	記述量	39 行	LOTOS：　154 行
6	回転する車輪群	記述量	5 行	SIMULA：　19 行
7	部品供給者	記述量	13 行	PL1＋：　88 行 SQL
8	洗車問題	記述量	31 行	C　：　600 行
		所要工数	0.5 人日	14 人日
9	並行有限バッファ	記述量	17 行	C　：　198 行
		所要工数	1.5 時間 (PG：50分， TEST：40分)	10 時間 (PG：8時間， TEST：2時間)
		バグ数	0 件	2 件
10	福祉国家判定	記述量	20 行	C　：　452 行
		所要工数	1.5 時間 (PG：40分， TEST：50分)	40 時間 (PG：21 時間 TEST：19 時間)
		バグ数	0 件	3 件（9 ケ所） （コンパイルエラー： 4 種類28ケ所）

6．おわりに

言語NAIVE の特徴は，次の3点に要約できる．

①日常的世界観にもとづく自然な記述性

②行為や行為主体などの概念の導入によって内包論理を拡張した独自の論理体系

③C言語プログラムの自動生成機能およびデータベース操作機能を中心とした高い実行可能性

本稿では，このうちの①について主に述べた．

今後は，当言語の実用性をさらに高めていきたい．また，当言語を立脚点として，ソフトウェア工学上の諸問題を改めて再考してみたい．

謝辞

本研究について日頃御指導戴く東京大学　田中英彦教授ならびに当社技術担当部長　河田汎博士に深く感謝致します．

参考文献

〔1〕Chandy,K.M. and Misra,J. : The Drinking Philosophers Problem,ACM Transactions on Programming Languages and Systems, Vol.6, No.4, pp.632-646, 1984

〔2〕Dowty,D.R.,Peters,S and Wall,R.E. : Introduction to Motague Semantics, D.Reidel, Dordrecht, Germany, 1981

〔3〕榎本編著：ソフトウェア工学ハンドブック，オーム社，1986

〔4〕Gallin,D :Intensional and Higher-Order Modal Logic』,North-Holland, 1975
〔5〕日野: 言語NAIVE による並列処理システムの記述, 情処研報, Vol.89, No.107, pp.17-26, 1989
〔6〕日野: 日常的世界観にもとづく実行可能な仕様記述言語NAIVE, 日本ソフトウェア科学会第8回論文集, pp.333-336
〔7〕日野: 言語NAIVE の論理体系（仮題）,To appear
〔8〕日野他: 言語NAIVE の実現方式（仮題）,To appear
〔9〕佐伯: 実行可能な仕様記述, 情報処理, Vol.28, No.10,pp1346-1358, 1987
〔10〕桜川, 竹中, 中島, 新出, 服部 : RACCO:実時間プロセス制御システムのモデル記述のための様相論理プログラム言語, コンピュータ・ソフトウェア,Vol.5,No.3, 1988
〔11〕柴山, 米澤: 並列オブジェクト指向言語 ABCL/1, bit ,Vol.20,No.8,pp940-954, 1988
〔12〕上田: 並列プログラミングとＧＨＣ, bit,Vol.20, No.10,pp.1175-1184, 1988
〔13〕渡辺, 原田, 三谷, 宮本: 場とイベントによる並列計算モデル—Kamui88,コンピュータソフトウェア,Vol.6,No.1,pp.41-55,1989
〔14〕渡辺, 辻ケ堂, 吉岡: 「小さいプログラムの性質の考察（迷路のプログラムを例として）」, AIPPI 日本部会報, Vol.33, No.11, pp.685-706, 1988
〔15〕横田, 森田, 宮崎: オブジェクト指向データベース・プログラミング言語, 情報処理, Vol.32, No.5, pp.559-567, 1991
〔16〕米崎, 新, 蓬莱 : 時制論理プログラミング言語 Temprog, 日本ソフトウェア科学会第1回論文集