ソフトウェア工学 110－13
（1996．7．26）

パネル討論：ソフトウェア新工法へ向けて

パネリスト：鯵坂恒夫（京都大学）

井上克郎（大阪大学）

野呂昌満（南山大学）

ソフトウェア工学 110－13－1
（1996．7．26）

ソフトウェア新工法へ向けて

鯵坂恒夫
（京都大学工学研究科）

ソフトウェア開発の形態，方法がいま急速に変わりつつある．確固たるライフサイクルモデルに基づき，厳密な仕様の作成，維持を必要とする大規模建造物のようなソフトウェアは，今後も引き続き求められるだろう．しかし一方で，情報処理技術革新の進展により，エンドユーザコンピューティングに象徴される耐久消費材的ないし消耗品的なソフトウェアが爆発的に増えている．このようなソフトウェアに，重厚長大なソフトウェアと同じモデルが適用できるだろうか．従来のように，要求の定義，仕様に基づいて設計を行ない，そのすべてをプログラミングするという工程ではなく，異なるシステム形態や応用領域向きの基礎を与えるプラットフォームとその上でのコンポーネントの組み立てによる，ソフトウェアシステム新工法の推進が必要である．

このような状況を背景に，本研究会のもとに「ソフトウェア新工法」ワーキンググループの活動が今年度より認められた．ここでは，ソフトウェア工学の対処すべき新しい課題，試行を選択的に議論し，実験実証的研究成果を積み上げることを目標とする．対象とする課題例は次の通りである．

- コンポーネントの抽出と組立て
- プラットフォームとスクリプティング
- ソフトウェアアーキテクチャ
- 新工法のためのソフトウェアプロセス
- 新工法のためのメトリクス
- ネットワークソフトウェア

ソフトウェア工学では，ともすれば同じような目標，課題が，例えば再利用にも環境にもプロセスにもというように異なる小領域に現れ，それぞれが十分満足できる結果に至ってはいない，といった状況もあったように思われる．ここでまた新領域を起こして，これまでと同じ不満を託つことのないよう，本質的問題を捉え損ねないように注意を払いたい．

いわばソフトウェアシステムのプレハブ工法ともいえる新工法を成功させるためには，枠組みや構造設計が必要であるのはたしかである．ソフトウェアアーキテクチャやメソッドエンジニアリングは，それを支える研究開発領域であると考えられる．しかし，枠組みだけではなく，より重要なのがコンポーネント自体を充実させることである．枠組みがあっても部材が揃わなければ家は建たない．

ここでいう「部材」と，従来からよくいわれているソフトウェア部品やクラスライブラリとの違いは，その粒度や汎用性の違いである．部材は，穴埋めパラメタでカスタマイズするような適用範囲の狭いものではなく，共通理解の確立できる比較的大きな概念領域をもとにしている．

これはドメインエンジニアリングでいうドメインよりは広い．実際，そのような概念領域とそれに対応するソフトウェアシステムとして，表データ処理の領域に対する Excel，グラフィクスに対する Tcl/Tk など，いくつか（潜在的な）例を指摘することができる．

このようなシステムの多くは，対応する概念領域に適切な粒度の概念要素を終端記号としてもつスクリプト言語を伴っている．それが，堅固なライフサイクルモデルや仕様記述をベースとしない消費材的ソフトウェアの工法に要請されること，すなわち，不確定な要求をコードレベルの具体性をもって記述するという，矛盾する目的を両立させるのに有効に働いていると考えられる．

このように進行しつつあるソフトウェア新工法をさらに発展させるため，取り組むべき次のような課題があげられる．

- 新たな概念領域とそれに対応する基盤システムを確立する．例えば，イベント / 状態，すなわち時間概念が中心となるような対象領域は，現状ではまだ比較的手薄であると考えられる．
- その領域におけるコンポーネントの同定と蓄積．
- コンポーネント構成法と（細粒度）プロセスの確立．
- スクリプティング技術の確立．
  - 当該領域の「世界」を決めるエンジン（解釈・実行系）の研究開発．
  - スクリプティング方法論．
  - 複数のエンジンを連係，統合する方法．

ソフトウェア工学の最大の弱みはその計量性の乏しさである．コンポーネントやスクリプトといった新たな道具だてで仕事を進めるにあたっては，それらに関わるメトリクスにも十分注意をそそぎたい．

コンポーネントウェアが複合文書として語られることがあるように，コンポーネントにはいわゆる情報コンテンツを主題とするものもある．CALS の進展などにより，円滑に流通させるべき電子化データ（マルチメディアデータを含む）の種類と量が今後ますます増えてくると，コンテンツへの指向はさらに重要なものとなる．情報の中身を混沌のまま，機械的に手の入れられない状態のまま，すなわち大きな「意味の塊」のまま放置することはできない．この塊にシンタクスおよび適度な粒度と共通性をもつ意味要素を導入し，いわゆる情報コンテンツのパラダイム（構成法と表現法）を確立することが要請される．

ソフトウェア工学 110－13－2
（1996. 7. 26）

ソフトウェアシステムの新工法に向けて
－ソフトウェアプロセスはどう変わるか－

New Approaches for Constructing Software Systems
- From Software Process View Points -

井上 克郎[1]
Katsuro Inoue

大阪大学大学院基礎工学研究科情報数理系

1 はじめに

ソフトウェアシステムの開発手法が大きく変わりつつある。今までの伝統的な方法、すなわち、プログラムの命令を一つずつ積み上げてシステムのを作成するのに対して、現在は以下のような例の開発が多いと聞く。

- 過去の類似のシステムのプログラムの一部を再利用する。使える部分を探し、一部手を入れる。
- 使えそうなプログラムを買ってきて、それに手を入れたり、マクロ定義やインターフェーススクリプトを書いたり、設定を目的のものに合わせる作業をする。
- 無料で公開されているプログラムを探し、ネットワーク経由で手に入れる。必要なものがあるかどうかを検索したり、ネットワークニュースで問い合わせたりする。

このような新たな開発形態に対し、今までのソフトウェア工学的の手法は、そのままでは適用しにくい。ソフトウェア工学の一分野であるソフトウェアプロセス（簡単にプロセスと呼ぶ）も、そのままでは、適用が困難であろう。以下、その理由やどう変わかについて私見を述べる。

2 旧工法と新工法

命令を積み上げて作成したシステム $S1$（旧工法と呼ぼう）と複数の部品を組み合わせて作成したシステム $S2$（新工法の代表として）について考える。表1は、それらを比較したものである。

[1]〒560 大阪府豊中市待兼山町1－3
Department of Information and Computer Sciences,
Faculty of Engineering Science, Osaka University,
Toyonaka, Osaka 560, JAPAN
06-850-6570 (Ph.) 06-850-6574 (Fax)
inoue@ics.es.osaka-u.ac.jp

$S1$ の構成要素は、プログラミング言語の文の長大な系列であり、非常に注意深く組み立てられている。個々の文の意味は比較的容易に把握でき、また、文と文との関係についても理解しやすい。従って、目的とするシステムに関して深い知識がない場合でも、順次作業をしながら知識を身に付けて構築できる場合があろう。しかし、知識の欠如が設計やコーディングの失敗につながり、うまく完成できない場合もあろう。

これらの開発を効率的に行なうための支援技術（いわゆるソフトウェア工学と呼ばれている技術）は、対象として均一な命令や文章、記号がかなりの分量で、いわゆる統計的な処理が可能になるくらい大きいことを仮定している場合が多い。仕様書や設計書、マニュアルなどのドキュメントのページ数や文字／記号の数、プログラムの行数、各種変数の数、サブルーチンの数や呼出数など、メトリクスデータとして活用されている。一方、統計的ではなく内部の意味や構文の詳細に関する情報に基づいて行なう支援技術は、その有用さが限られている。例えばプログラムの形式的な意味論に基づく設計や検証などの技術は、小規模例題から大規模実例へ拡大することが容易ではない。

このように開発作業全体は、粒度の小さい要素を扱う、粒度の小さい作業で構成されている。すなわち、ドキュメントに文字を追加した、エディタでソースコードを変更した、など細かな作業内容の積み重ねである。ソフトウェアプロセスは、開発に関する作業を人間が把握や管理できるぐらいの粒度に分解することが必須である。小さな作業を基本単位とすることは、全ドキュメントやソースプログラムの命令を一つずつ管理するのと同じことで、これでは、役にたたない。従って、同種の作業を一まとまりにし、その内部は統計的な情報だけを利用して、そういうまとまり単位に全体を把握／管理するのが一般的であろう。

$S2$ については、素材は、すでに複雑な機能を持つ部品である。どのような部品を使うか、これを決定するためにかなりの知識を必要とする。現在流通しているしているものは何か、それは、どのような機

表 1: $S1$ と $S2$ との比較

性質	$S1$ (旧工法)	$S2$ (新工法)
構成要素	プログラムの命令語 (ソースコード)、自然語文 (各種ドキュメント)、四角や線 (設計図など)	部品、インターフェースプログラム、過去のプログラムなど
構成要素の粒度	小	大
開発支援技法	Metrics など統計的手法	それぞれの部品の性格に依存
開発に必要な知識	最低限作業ができるためには、基本的な事項 (日本語、設計図、言語仕様) を知っていればよい	部品の内容やインターフェースに関する比較的深い事項が要求される
開発自由度	大	小

能を持ちインターフェースはどうなっているか、どれを選べば目的にもっとも簡単に近付くことができるか、など、高度な知識の上での決断をする必要があろう。

$S2$ に関する支援技術は、$S1$ のように統計的な手法はあまり意味をなさない。通常、部品の内部に関しては、隠蔽されていたり、または、その詳細を (積極的に) 知ることなしに利用する場合が多い。利用の手がかりとなるのは、情報が抽象化して記載されているマニュアルや同類の使用をしている例プログラムなどであろう。種々の部品に対して一般的に通じる支援技法というのは考えにくく、それより個々の場合に応じた技術やそのツールが有効であろう。

$S2$ の開発のなかで最も重要な作業は、設計である。どのような部品を選びそれを組み合わせるか、で、ほどんどシステムの性質が決定する。また、開発に必要な作業の性質もここで決定されるであろう。従って、作業は、かなりその部品の構成に依存したものになる。たとえば、特定の部品が使えるようにするためのセットアップ作業、その部品に関するドキュメント作業など、設計で指定する部品に対応したプロダクト作成作業が基本となる。

このような場合では、統計的な値、例えば何行プログラムを書いた、何時間テストを走らせた、などは、作業の進捗を表すのにあまり役にたたない。それより、あるプロダクトが完成した、ある部品と別の部品が協調して動いた、などのマイルストンが重要である。

3 プロセス研究の方向について

このような $S2$ に対して、既存のプロセスの技術は有効ではない。既存の技術は、(プロセスプログラムを始めとして) 基本作業を粒度の低い構成要素と見て、ループで繰り返す、階層化を行なう、などと、個々の性質ではなく、構成方法について議論されていたように思う。マイルストンの考え方では、既存のプロセス技術は、対象の粒度が大きく、その長所を引き出せない。

では、$S2$ に対して有効なプロセス技術はあるのだろうか。開発作業を、上述のようなエンジニアリングの作業の他、教育やマネジメントまで広げて、種々の作業を取り込んで、部品に関わる作業の粒度を相対的に下げると、既存のプロセス技術の延長として扱うことはできよう。

一方、プロダクトの管理技術、ソフトウェアアーキテクチャ、そして設計の問題などと深く関連付けた作業の表現や管理技術としてプロセスを位置付けることも出来よう。それらの計画の結果、作業が定義され、それに基づいて進捗を知ることが出来る。しかし、この場合、あくまでプロダクトが主である。

かなり悲観的なことを書いたが、やるべき研究テーマはたくさんあろう。

- オブジェクト指向技術を中心とした開発プロセスデータの蓄積。上述の $S2$ に最も近くかつ現実に利用されているのはオブジェクト指向関連技術であろう。数多くの実践データを得て、プロセス研究の方向性を確認することが必須である。
- 設計プロダクトに基づくプロセス管理の考え方の形式化と管理/支援手法の検討。開発作業の後のプロセスが、前のプロセスで作られたプロダクトに依存している。これは、一種の高階な作業と考え、それをどううまく扱うかが問題になろう。
- プロダクト主体のプロセスのモデル化、実働化の問題の検討。これには、マイルストンの考え方のモデル化も含まれよう。さらに、このようなプロセスモデルの有効性 (プロダクトモデルだけを利用するのに比べて) の検討も必須である。
- 設計作業のプロセスのモデル化。$S2$ では設計作業が重要である。$S1$ とは異なって部品の検索、知識の習得などの仕事が設計作業として加わっている。それらを含め、設計作業をある程度プロセスとして定式化できるであろう。

ソフトウェア工学 110－13－3
（1996．7．26）

ネットワークソフトウェアの新工法†

野呂昌満
南山大学情報管理学科

masami@iq.nanzan-u.ac.jp

1 はじめに

近年のコンピュータ環境の変化に伴い，ソフトウェアの開発形態や運用方法に変化が起こり，末端使用者による簡易プログラムの作成 (end-user computing) またはそれに類する，あつらえ可能で組み立て容易な部品の集合としての，ソフトウェア作成が注目を浴びるに至った．我々は，そのような状況の中でのソフトウェア開発が，

- 従来からソフトウェア工学が培ってきたソフトウェア開発・管理技術の外延上に位置するものかどうか，
- 新技術の発見・導入を必要するものなのか，
- 従来技術を改良することで対応出来るものか，

などを確認するためにソフトウェア工学研究会の下にソフトウェア新工法ワーキンググループを組織した．

今日のコンピュータ環境においてネットワークの存在は無視できないものになっていることは明白である．このようなネットワークコンピュータ環境において，ソフトウェアは，物理的・論理的に異なるコンピュータ上で複数プロセスとして実行される構成をとることになる．本稿では，このようなソフトウェアをネットワークソフトウェアと呼ぶ．ネットワークソフトウェアの身近な例としは，Unix に代表される，NFS, NIS, talk 等のインターネット応用ソフトウェアが挙げられる．筆者の関心はこれらのネットワークソフトウェアの新工法に関連するいくつかの話題である．

ここでは，インターネット応用ソフトウェアを例にとって，ネットワークソフトウェア開発の現状，問題点，およびその解決策を述べることで，ネットワークソフトウェアの新工法に関連する研究の位置付けを明確にする．以下，文章中現在の Unix 文化を否定するような表現を用いるが，Unix の存在を否定する気は毛頭ない．筆者は Unix の実用性を認めており，大半の仕事を Unix 上で行なっている．過激な表現が問題をはっきりさせるとの意図を持っての文章である．

†この予稿は日本ソフトウェア科学会研究会，ソフトウェアの新しい構成・統合技術に関するワークショップ WINGS の予稿を加筆修正したものである

2 ネットワークソフトウェア開発の現状

現在のネットワークソフトウェアの大半は Unix 上のもの，残りは Mac OS, Windows-95 などのパーソナルコンピュータ上のものといっても差し支えがないと思われる．ネットワークはインターネットに代表される IP 通信規約によるものが支配的であると考えられる．我々にとって不幸なことに (幸か不幸か本当のところはわからないが)，それらの多くは，ソフトウェア工学の成果を身に付け応用しているソフトウェア技術者の手によるものではない．多くはインターネットハッカーと呼ばれるスーパープログラマたちによって作られている．記述言語は C や C++ であり，極言すれば，“動けば勝ち” 式の開発法とそれによる製品がまかり通っている．

WWW や NIS などの UNIX のネットワークソフトウェアは遠隔手続き呼び出し (RPC, Remote Procedure Call) を使ってモデル化されている．その実現には UNIX の socket が使われており，UNIX の socket は CSP を UNIX に採り入れたものと考えられる (図?? 参照)．RPC は呼びだし側のプロセス P1 が呼ばれる側のプロセス P2 と，呼びだし時と戻り値受け取り時にランデブし 2 度のランデブの間には何も行なわないものと考える．このように，RPC は CSP の計算モデルで説明できる．

図?? に WWW システム (WWW 関連のデーモンおよびデータベースを集合的にこう呼ぶものとする) や NIS に共通なアーキテクチャを示す．このアーキテクチャはクライアントのデーモンプログラムがサーバのデーモンプログラムに遠隔手続き呼び出しを使って情報を要求しそれを獲得する構成をとっている．図?? に WWW シス

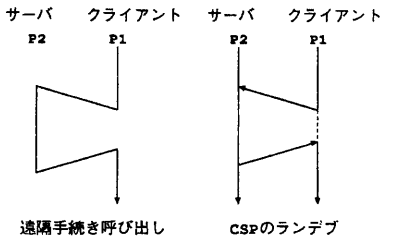

図 1: 遠隔手続き呼び出しと CSP のランデブ

テムの詳細アーキテクチャを示す. WWW システムの場合サーバ側で起動する処理プログラムをクライアントのブラウザで指定したり, 応答として送られてきたデータに応じてクライアント側で表示プログラムを起動したり出来る. これらの実現の欠点としては, 情報要求が集中するなどによりサーバ側での計算量が大きくなるとサーバの応答が劣化することが挙げられる. この問題点を解決するために tcl/tk[?] や Java[?] 等の言語で書かれた処理プログラムをクライアントに送り返し, クライアント側でプログラムを実行するための機構が考え出された. この機構を用いて多くのネットワークソフトウェアが設計・実現されている.

3 ネットワークソフトウェア開発の現在の問題点

現状のネットワークソフトウェア開発における問題点はすべて, これまでのソフトウェア工学の成果を踏まえずに開発されていることに起因するといっても過言ではない. 以下にいくつかの例を挙げる.

(1) 大域変数を多用する C プログラムが多い.

> 一部のスーパープログラマに開発のすべてを任せるというのであれば, 問題はないが, そうでない場合, このことの問題点は解説するまでもないであろう. Unix のコードは言うに及ばず, C コンパイラのコード, X 窓システムのコードと多くの場所にこのようなプログラム部分を見つけることは容易である.

(2) パラダイムを混在して作られた C++ プログラムが多い.

> これは C++ の言語の問題である. オブジェクト指向設計とは如何なるものか, オブジェクト指向ソフトウェアとはどのような構成を持つものか, 手続き指向とはどう違うかなどのソフトウェア工学の基本的な事項をわからずに C++ でソフトウェア開発を行なった結果得られるものである. 局所関数を持つクラスのメンバ関数, データメンバを持たないクラス, 大域関数を持つ C++ プログラム, 参照とポインタの両方を引数に持つメソッド等例の枚挙に暇がない. このようなソフトウェアの保守性を云々するのは意味がないと言える.

4 ネットワークソフトウェアの新工法に関する研究課題

上に述べた問題点の多くは C や C++ のプログラミング言語としての問題点に起因するものである. また同様の問題が yacc や lex などの Unix 上のソフトウェアツールについても指摘できる. yacc を考えた場合, しっかりした属性文法の処理系がそれにとって代わるべきであろう. このように, 第一の研究課題としては, プログラミング言語, ソフトウェアツール, 開発環境などの整備という古い問題がある. Java[?] はこの種の研究・開発の典型的な例であるが, 使用者イベントの取り込み処理の記述などで概念が整理されていない部分があり, ネットワークプログラミング言語として最良のものとは言えないのが現状である.

つぎに考えられるものが, ネットワーク対応のオペレーティングシステムの設計と実現がある. 共通のネットワークインタフェースを持ったオペレーティングシステムに関する研究が第一であると考えられるが, 異機種間での共有メモリ, プログラミング言語処理における実行時コード生成や部分実行の環境提供などいくつかの課題が挙げられる. 実現においてはマイクロカーネル等のあつらえ可能オペレーティングシステムの研究成果を応用発展させることが考えられる.

オペレーティングシステムに関連する研究課題として, より重要なものに通信規約に関するものが挙げられる. 現状では, OSI や IP などの規約の標準化 (事実上の標準も含めて) が盛んであるが, それらの規約を仕様の一部と考えた場合, それらを (とくに応用提示層の規約について), これまでのソフトウェア工学の成果を用いて再記述する必要性があろう. CSP 理論を用いての整理, State 図を用いての図式化, などが例として挙げられる.

最後に, この領域 (ネットワークソフトウェア) における応用枠組 (application framework) の整備が研究課題として挙げられる. これについてはアーキテクチャの確立

が不可欠と考えられる.

筆者の研究室では, 上記の課題を以下のような具体的な副課題に詳細化して研究を行なっている.

- 構文拡張可能なプログラミング言語の設計と試作.
- 分散共有メモリを用いた言語の実行時環境の設計と試作.
- 異言語ソフトウェア分散開発支援環境.
- 応用提示層の通信規約実現プログラムのオブジェクト指向による再設計.
- それらのオブジェクト指向プログラミング言語による再記述.
- 応用提示層の通信規約の共通枠組の定義.

以下では, これらの中でとくに重要なものであるネットワークソフトウェアのアーキテクチャおよび異言語ソフトウェア分散開発支援環境について述べる.

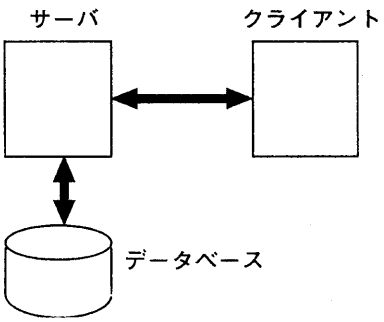

図 2: ネットワークソフトウェアのアーキテクチャ

5 ネットワークソフトウェアのアーキテクチャ

現在のネットワークソフトウェアは図?? にあるようなクライアント・サーバ型のものが主流である. しかし, これは, これからのネットワークソフトウェアの設計・実現を論じていく上ではあまりに一般的過ぎるものである. もう少し詳細なものでかつ一般性を失わないものとしては, 図?? の WWW のアーキテクチャのようなものが考えられるであろう. すなわち, あるノード上のインタフェースプログラム (ブラウザ) がネットワーク上に点在

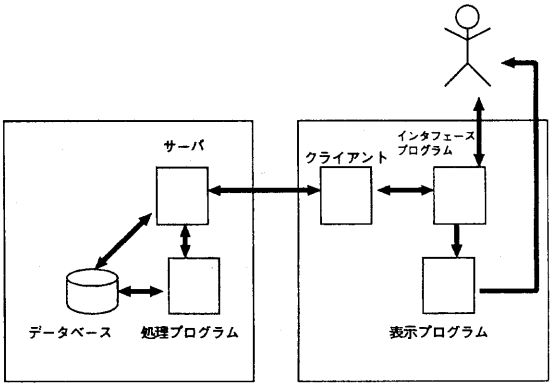

図 3: WWW システムのアーキテクチャ

するサーバに情報要求を行ない, 獲得した情報をデータの種類に応じてインターフェースプログラムを通して表示するものである.

これからのネットワークソフトウェアの開発を考えた場合, 図?? に示したようなアーキテクチャの下で有効なフレームワークを整備していくことが必要となると考える. MVC[?, ?] などはこれまでに提案されてきたフレームワークであり, 図 3 のアーキテクチャにおいてはインタフェース部分の設計・実現に有効なものであると考えられる. ネットワークソフトウェア特有のフレームワークとしては通信プロトコル, 通信データの構造, その処理に関するものを整理する必要があると考えられる.

さらに, tcl/tk や Java を用いたネットワークソフトウェアのように, データ処理や表示のエージェントがネットワーク内を行き来するための機構も用意すべきである. これらを前提とした場合アーキテクチャは図?? のようになると考えられる.

6 ネットワークソフトウェア開発支援

先に述べたアーキテクチャに基づき, その構成要素であるサーバ, クライアント, データベース, エージェント等を記述するさいに, それらの記述言語を統一するという考えはあまり実用的とはいえない. それぞれには特有の記述方式と専用の言語が必要である. この観点からすれば, これからのネットワークソフトウェア開発においては異言語による共同開発が必要となるであろう.

これを実現する 1 つの手段としては CORBA[?] のような共通のオブジェクト交換形式を整備することが考えら

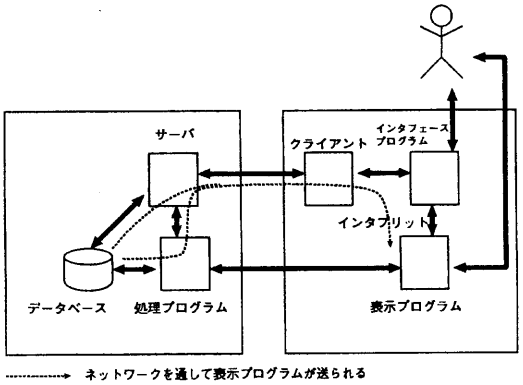

図 4: これからのネットワークソフトウェアのアーキテクチャ

れる. この方式の欠点としては, 各言語とのインタフェースをとるための費用や処理系の変更やインタフェースプログラムの記述の費用が(たとえ, IDL を用いたとしても)高いことが挙げられる. 筆者の研究室では, これらの欠点を補う方法として, 自己反映計算機能を持ち構文が拡張可能なオブジェクト指向プログラミング言語を基にした異言語プログラミング環境の研究を行なっている [?].

7 まとめ

"IP 接続で疎結合された複数のコンピュータから構成される環境" などをネットワーク環境と考え, "その環境内の物理的・論理的に異なるコンピュータ上で複数プロセスとして実行される構成をとるソフトウェア" をネットワークソフトウェアと定義し, その開発の現状を問題点を説明し, 以下の研究課題を挙げた.

- プログラミング言語, ソフトウェアツール, 開発環境などの整備.
- ネットワーク対応のオペレーティングシステムの設計と実現.
- 通信規約に関するもの.
- 応用枠組の整備.
- アーキテクチャの確立.

後半では, 筆者の現在の研究から, いくつかの着想を紹介した. すなわち, ネットワークソフトウェアのアーキテクチャおよび, 異言語プログラミング環境について言及した.

参考文献

[1] S. S. Adams:"The MVC Paradigm," *HOOPLA!*, vol.1, no.4, 1988.

[2] B. W. Boehm:"A spiral model of software development and enhancement," *ACM Software Engineering Notes*,vol.11, no.4, pp. 22-42, Aug. 1986.

[3] G. E. Krasner, S. T. Pops:"A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80," *Journal of Object-Oriented Programming*, vol.1, no.3, pp. 26-49, 1988.

[4] Masami Noro, Masahito Nakahara: "The Design of Reflective Object-Oriented Programming Language for Syntax Extension: A First Step towards Multi-Lingual Software Development Environment," *Proceedings of the International Symposium on Software Engineering for the Next Generation*, pp. 132–136, Nanzan University, 1996.

[5] Object Management Group: The Common Object Request Broker, Architecture and Specification Revision 2, *OMG documents*,1995.

[6] John O. Ousterjout: *Tcl and the Tk Toolkit*, Addison-Wesley,1994.

[7] Sun Microsystems,Inc.: *http: // java.sun.com/ doc.html*, 1995.